DISSERTATION

ACCURATE PREDICTION OF PROTEIN FUNCTION USING GOSTRUCT

Submitted by

Artem Sokolov

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2011

Doctoral Committee:

Advisor: Asa Ben-Hur

Chuck Anderson
Ross M. McConnell
Haonan Wang

ABSTRACT

ACCURATE PREDICTION OF PROTEIN FUNCTION USING GOSTRUCT

With the growing number of sequenced genomes, automatic prediction of protein function is one of the central problems in computational biology. Traditional methods employ transfer of functional annotation on the basis of sequence or structural similarity and are unable to effectively deal with today's noisy high-throughput biological data. Most of the approaches based on machine learning, on the other hand, break the problem up into a collection of binary classification problems, effectively asking the question "does this protein perform this particular function?"; such methods often produce a set of predictions that are inconsistent with each other.

In this work, we present *GOstruct*, a structured-output framework that answers the question "what function does this protein perform?" in the context of hierarchical multilabel classification. We show that *GOstruct* is able to effectively deal with a large number of disparate data sources from multiple species. Our empirical results demonstrate that the framework achieves state-of-the-art accuracy in two of the recent challenges in automatic function prediction: Mousefunc and CAFA.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The need for automatic function prediction has been a growing concern in the field of genomics. The number of sequenced genomes has been growing rapidly and the recent advent of high-throughput sequencing technology is bound to accelerate it further. Experimental annotation of protein function, on the other hand, remains expensive and time consuming. This calls for computational methods capable of accurately predicting protein function from the protein's sequence and other genomic data.

The Gene Ontology (GO) [31] is the current standard for annotating proteins. GO is comprised of a set of keywords that specify three namespaces: the gene product's molecular function, the biological processes in which it participates, and its localization to a cellular component. Each of the three namespaces imposes a hierarchy over its set of keywords, as depicted in Figure 1.1; the keywords deeper in each hierarchy provide more detail. An annotation can be represented by a vector of binary variables that denote association with the corresponding GO keywords. Since a protein may have multiple functions in each GO namespace, the problem of protein function prediction can be formulated as hierarchical multi-label classification [6]. Hierarchical multi-label classification problems often occur in text categorization, where a document needs to be assigned to one or more topics, and the topics themselves belong to a hierarchy [68]. In a way, a protein can be thought of as a document and the corresponding GO terms a set of topics associated with it.

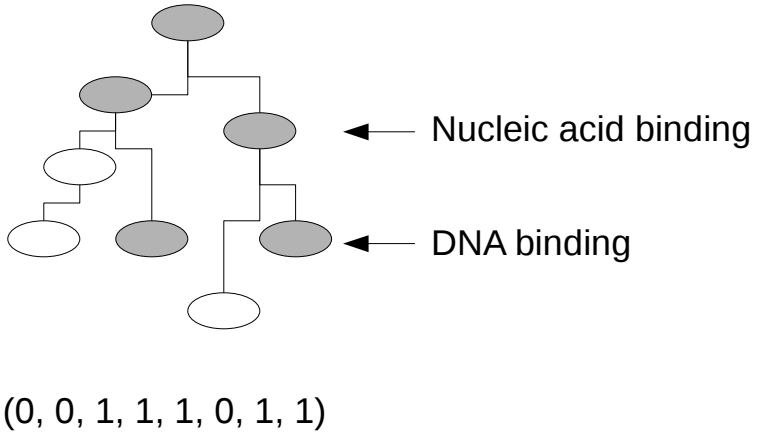


Figure 1.1: Example hierarchy of GO keywords. Nodes deeper in the hierarchy provide more detail. A protein may perform multiple functions, which would then be captured by nodes in two distinct subtrees. Association of a protein with GO keywords can be expressed as a vector of binary variables.

1.1 Transfer of annotation

Proteins are sequences over the amino acid alphabet that get folded into a 3D structure; the sequence and structure, therefore, define a protein and its function. For a long time, the predominant approach to inferring GO function for newly sequenced proteins has been *transfer of annotation* [52], where the keywords are transferred from proteins with known function on the basis of sequence or structural similarity, usually with the help of sequence alignment tools, such as Basic Local Alignment Search Tool (BLAST) [1]. The intuition behind transfer-of-annotation methods is based on the assumption that similar proteins that arise from a common ancestor, known as *homologs*, often share similar function. Many studies have shown that this assumption is limited in validity [23, 12, 29, 67]. Specifically, single point mutations and gene duplications can lead to different protein function while maintaining high sequence similarity [12, 29]. Other problems arise from erroneous annotations in databases [23] as well as from the fact that proteins tend to perform multiple functions, only a portion of which may be conserved across homologs [67], yielding predictions that are only partially correct.

Nevertheless, a number of methods employ sequence and structural similarity to make

functional annotation predictions with varying degrees of accuracy [94, 37, 54, 34]. Methods that use sequence similarly do so by aligning the query sequence against a database of annotated protein sequences with the help of alignment tools, such as BLAST. BLAST, specifically, produces a set of similarity scores between the query and significant matches in the database. These scores, known as "*e*-values", can be thought of as a reflection of the expectation that the corresponding alignment was produced by chance. The *e*-values can then be thresholded, as done by GOblet [37], to identify the closest homologous sequences from which the GO keywords are then transferred to the query. In the most extreme case, the single closest match is used, which provides a baseline method termed TOPBLAST by Martin, *et al.* [54]; we refer to it as "BLAST nearest neighbor", or *BLAST-NN*, in this work. Alternatively, the *e*-values themselves can be used as weights for the associated GO terms retrieved from the annotated database matches. This is the approach taken by OntoBlast [94] and GOtcha [54]. GOtcha further converts these weights to *p*-values by computing a background distribution from a set of 518,226 annotated sequences in SwissProt [11]. New score recombination schemes are still being proposed today, an example being the algorithm by Hamp, *et al.* that was used in the 2011 Critical Assessment of Function Annotations (CAFA) challenge [34].

In addition to sequence similarity, structure-based features, such as a protein's 3D fold, its structural motifs and domains, can be used to further establish protein homology. A web-based service ProKnow combines structured-based and sequence-based features to score a protein against a database of annotated proteins [63]. Rather than using *e*-values, the query protein is scored by ProKnow using a Bayesian framework that combines a number of factors including the quality of annotations in the database.

1.2 Function as a collection of binary variables

The nearest-neighbor behavior of the transfer-of-annotation approach is unable to effectively deal with today's noisy high-throughput biological data. This has led to the recent development of machine learning approaches that typically address the problem as a set of binary

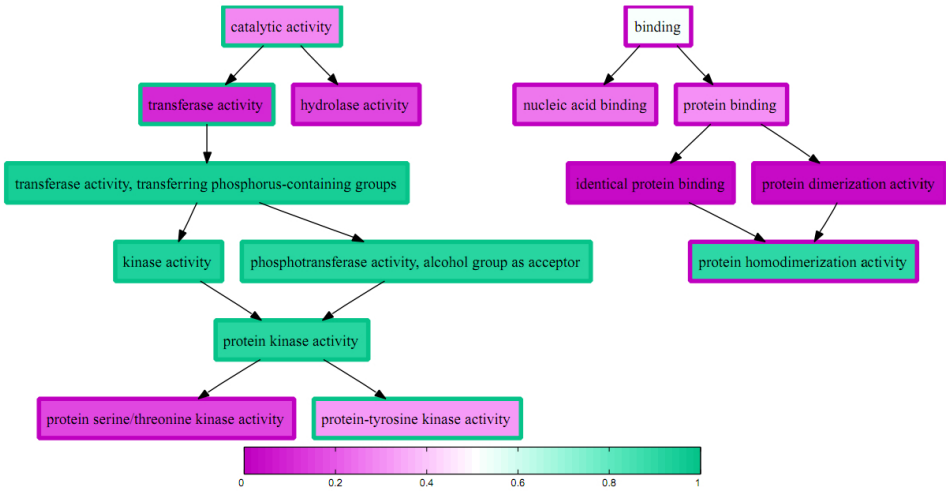


Figure 1.2: A figure taken from the paper describing the algorithm by Obozinski, *et. al* [62]. The algorithm trains a collection of SVMs to make predictions for individual GO terms. The prediction scores correspond to the background shading, while the outline represents the true label: "protein-tyrosine kinase activity" and its ancestors. Note that many of the intermediate nodes were predicted to be associated with the query protein, but not their ancestors "transferase activity" and "catalytic activity" — an inconsistency.

classification problems: whether a protein should be associated with a given GO term (see e.g., [57]). The methods used to solve each of the binary classification problems generally fall into two categories: those that use the Support Vector Machine (SVM) classifier, and those that employ guilt-by-association techniques by embedding the data in a graph with nodes corresponding to proteins and edges representing some evidence of functional similarity. Many of the methods outlined here were used in the Mousefunc competition, which we detail below.

The problem with breaking the problem up into a collection of binary classification problems is that the predictions made for individual GO terms will not necessarily be consistent with the hierarchical constraints: a method may assign a positive prediction score to a GO keyword and a negative prediction score to its ancestor. An example of this is presented in Figure 1.2. As such, some methods choose to reconcile the predictions with the hierarchy using Bayesian networks or logistic regression [62, 6, 32], to produce full annotations. Other methods employ inference algorithms on graphs to directly produce a hierarchical label, such as the method by Mostafavi and Morris [56]. Yet other methods choose to forgo the recon-

ciliation step entirely, because the predominant approach to measuring prediction accuracy for this problem is to use precision/recall and area-under-ROC metrics on a "per GO term" basis [64]. In this case, the interpretation of potentially conflicting binary predictions is left up to the user.

1.3 Function as a structured label

This project explores a different approach. Rather than treating the task as a collection of binary classification problems ("is a particular GO keyword associated with a particular protein?"), we train a predictor to infer the full annotations directly ("what GO keywords are associated with a particular protein?"). We accomplish this by learning a *compatibility function* $f(\mathbf{x}, \mathbf{y})$ that accepts a protein $\mathbf{x}$ and a full annotation label $\mathbf{y}$ as its arguments and returns a compatibility score associated with the two. Inference of annotations for novel proteins is then accomplished by finding the most compatible label:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}).$$

A good compatibility function will always score the correct label for a particular protein as being more compatible than all other labels, and the learning objective is to achieve this condition on the training data.

An algorithm aimed at directly inferring a complex label, such as a GO annotation, is called a *structured-output* method. Structured-output methods have been introduced to the field of machine learning fairly recently and span a number of discriminative and probabilistic approaches [5]. The most popular of these is the structured SVM [85], which shares many of the advantages of its binary counterpart [88], such as robustness to noisy data. Structured SVMs have been successfully applied to a variety of problems, including text categorization [85, 68], prediction of disulfide-bond connectivity [82], and prediction of enzyme function [2], but are still not as widely used as binary SVMs due to their higher level of conceptual complexity and the required understanding of the implementation details that prevent the user from treating the training algorithm as a black box.

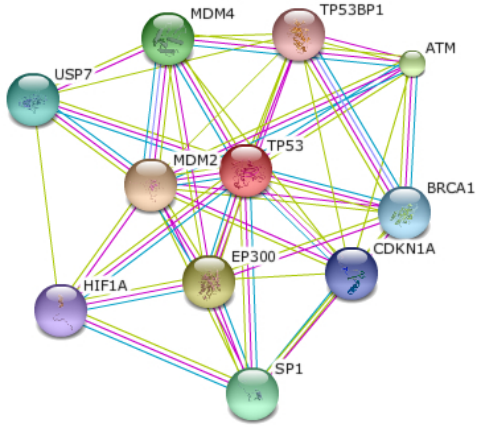
We propose *GOstruct* — a framework for applying structured SVMs to the task of GO term prediction for proteins and gene products. In chapter 3, we describe the basic GOstruct method in detail and demonstrate that it achieves state-of-the-art performance. We do so by first demonstrating that *GOstruct* outperforms the traditional method of annotation transfer on the basis of sequence or structural similarity. We then show that *GOstruct* also achieves highly competitive performance on the Mousefunc dataset [64] compared to other state-of-the-art methods.

1.4 Data

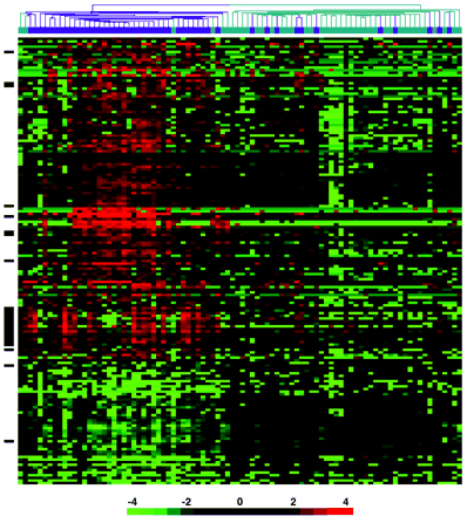
The sequence and structure of a protein are not the only sources of data relevant to the problem of function prediction. A number of studies demonstrated that high-throughput biological data such as gene expression and protein-protein interactions (PPI) can be predictive of protein function [64, 93, 26, 52]. In this section, we give an overview of several types of data and present an intuition behind how they relate to protein function. We leave the details of feature representation for later, when we use the data to make predictions.

The interaction of two or more proteins provides "guilt by association" evidence that the proteins share similar function; particularly, it is an indication that the proteins participate in the same biological process and are localized to the same part of the cell [52, 93]. Today, high-throughput interaction data is widely available through databases such as the General Repository for Interaction Datasets (BioGRID) [80] and the Search Tool for the Retrieval of Interacting Genes/Proteins (STRING) [43]. Information about protein interactions is often represented as a graph with nodes corresponding to proteins and edges representing some evidence that two proteins interact [56]. An example of such representation is given in Figure 1.3(a).

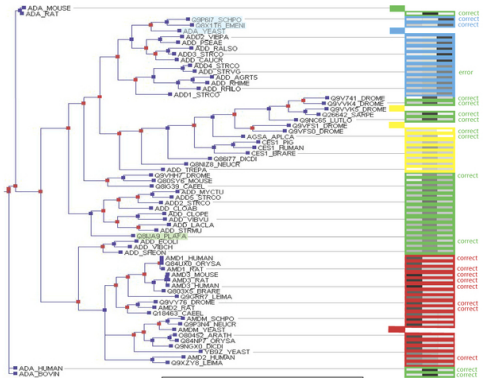
Another source of evidence that two proteins may be participating in the same biological process is the extent to which two genes are co-expressed [93]. Gene expression measures the level of RNA that codes for a specific protein being present under certain conditions. One of the most common ways to represent gene expression data is with a heat map, where the



(a) Example of a protein-protein interaction subgraph, constructed by STRING [43] for the human tumor-suppressing gene 'p53'. Nodes in the graph correspond to proteins and edges denote some evidence that two proteins interact.



(b) Example of a microarray heat map taken from a paper by Baechler, *et al.* [3].



(c) Example of a phylogenetic tree taken from the SIFTER paper [26].

Figure 1.3: Examples of common data representations for (a) protein-protein interactions (b) gene expression and (c) phylogenetic profiles.

rows correspond to individual genes and columns represent specific conditions under which the expression level was measured; the rows and columns are generally clustered according to some criterion. The underlying data that heat maps represent is often referred to as "microarray data" because the associated expression levels are measured through the use of DNA microarray chips [70]—a set of complementary DNA (cDNA) strands, specifically designed to bind to the target genes, attached to glass using robotic printing; the use of fluorescent labels and a laser allows the user to measure the number of bindings for all genes in parallel. An example of microarray data is given in Figure 1.3(b). Gene expression data tends to be noisy [42, 91], and we chose to forgo collecting these features in all experiments except Mousefunc where it was provided by the competition organizers.

As mentioned earlier, the intuition behind transfer of annotation comes from the assumption that proteins that evolved from a common ancestor will perform similar function. The related field of phylogeny studies the relatedness of organisms in the evolutionary tree in the context of function prediction [25]. Phylogenetic information is usually represented as an evolutionary tree, as depicted in Figure 1.3(c), with proteins residing at the leafs, internal nodes corresponding to speciation and duplication events and edge lengths being representative of the amount of time between the events. Algorithms, such as SIFTER [26], can use the phylogenetic trees directly as part of the inference; specifically, SIFTER treats internal nodes as hidden variables in a Bayesian framework [26]. Alternatively, the relatedness of proteins can be represented with a phylogenetic profile — a binary vector, where each variable to corresponds to a fully-sequenced species. The value of each binary variable is set to 1 if the species contains a homolog of the protein in question.

1.5 Data heterogeneity and multi-view learning

The availability of a large variety of genomic data relevant to the task of protein function prediction has led to the development of a variety of methods for integrating those disparate data sources. Approaches include kernel methods [51, 62] and label propagation on a network whose nodes are proteins and edges indicate similarity according to some data source [22,

57, 86]. However, all these methods perform data integration in a given species, and are not able to take into account the labels of annotated proteins in other species. The challenge in doing this integration is that examples are heterogeneous — examples representing proteins in the given species have features that capture diverse data: gene expression, protein-protein interactions, and sequence similarity. Most of this data, except for sequence, is species-specific: protein interactions are probed experimentally in a given species, and the expression of a given gene measured in one set of experiments is difficult to compare meaningfully to expression measured in another species, under possibly different conditions.

In Chapter 4 we extend *GOstruct* to combine heterogeneous data sources across several species. We do so by employing *multi-view learning*, which is an approach for dealing with multiple independent feature sets and unlabeled data. In multi-view learning, the input-space features are separated into two or more groups ("views") and a separate model is trained for each view with the goal of maximizing the accuracy on the labeled data and minimizing view disagreement on the unlabeled data [10]. The application of this technique to structured output spaces is fairly recent and several algorithms exist that either minimize the disagreement explicitly [30, 53] or use a more heuristic co-training approach where each view suggests labels for its peers [13].

Multi-view learning has been applied to natural-language processing [13], document categorization [30, 53] and signal processing [18]. However, all these applications make an emphasis on using unlabeled data and maintain an implicit assumption that every example can be represented in every view. *GOstruct* breaks away from this assumption by treating all *cross-species* features, such as sequence similarity, as one view and all *species-specific* features, such as protein-protein interactions, as another. We explore co-training [10, 13] and transductive learning [98] as the two approaches to assigning labels to unlabeled data. Empirical results demonstrate that the multi-view framework, that combines all available sources of data, outperforms all single-view formulations. In other words, combining all available features from all available species leads to the highest level of accuracy in predicted functional annotations.

1.6 Critical assessment

Automatic function prediction is widely recognized as an important problem in bioinformatics, and several experiments have been set up to perform critical assessment of the algorithms developed to solve the problem. The two prominent experiments that we describe in this section are Mousefunc [64] and Critical Assessment of Function Annotations (`http://biofunctionprediction.org/`). We used the *GOstruct* framework to make predictions for both experiments and outperformed all other algorithms in both cases.

1.6.1 Mousefunc

The goal in the Mousefunc challenge was to generate GO term predictions for a set of genes in the *M. musculus* species [64]. The training and test data was provided by the organizers and consisted of two sources of gene expression data [95, 81], two sources of protein domain data (Pfam and InterPro), protein-protein interactions, phylogenetic profiles and functional annotations for the training data. The gene IDs were masked to prevent the participants from augmenting the training set with additional sources of data. We describe the details of data representation in the next chapter, where we apply *GOstruct* to it.

The assessment of the algorithms has since been published [64] and the test labels made available for public use. The performance evaluation was performed by the organizers on a per-GO-term basis, where the GO terms were split into four categories based on their representation in the training data; the most specific category consisted of GO terms that occurred in fewer than 10 training examples. We now review several of the algorithms that were entered into the Mousefunc challenge.

The algorithm by Obozinski, *et al.* [62] trained a separate binary SVM for each GO term / data source pair; the data sources were processed to generate appropriate kernels for each. The outputs of individual SVMs were then combined using logistic regression to produce a single numerical score for each GO term. Recognizing that the scores may be in disagreement with the GO topology, the final step of the algorithm reconciled the predictions made for individual GO terms such that the scores assigned to ancestors of a GO

term were always higher than the score of the GO term itself. Obozinski, *et al.* considered 11 different methods to perform the reconciliation ranging anywhere from simple heuristics to more complex Bayesian methods.

GeneMANIA [57] is a "guilt-by-association" algorithm that treated each source of data as a network with proteins comprising the set of nodes and evidence of co-functionality being captured by the edges. For each GO term, the networks were combined using a set of learned weights; a different set of weights was used for each GO term. For every training protein in the combined network, the algorithm then assigned a "discriminant value" that measured the amount of association between the protein and the GO term in question. The discriminant values were then propagated to the test proteins using the Gaussian field label propagation algorithm [97]. While the version of GeneMANIA entered into the Mousefunc challenge made predictions for individual GO terms, the algorithm has since been extended to utilize the GO topology and make predictions for multiple GO terms simultaneously [56].

These two algorithm make up a representative set of methods applied to Mousefunc. All other algorithms were variations on these two approaches: sometimes decision trees were applied in place of SVMs [84], while other times a different label propagation algorithm was used on the association networks [47, 17]. Multiple models were generally combined using logistic regression [32, 84].

1.6.2 CAFA

Critical Assessment of Function Annotations (CAFA) is the most recent challenge in automatic function prediction. Unlike the case with Mousefunc, CAFA organizers released test sequences only, without masking protein IDs. Each participant was free to build their own training set.

CAFA borrows much of its motivation from Critical Assessment of Structure Prediction (CASP) [59]. CASP has been one of the main driving forces behind the advancement of algorithms aimed at prediction of protein structure; the goals of the challenge are both to determine the progress being made and to identify specific bottlenecks in the current state-

of-the-art methodology [58]. While CASP has gone through eight iterations already, CAFA is still in its infant stages with the first assessment presented at the Automatic Function Prediction special interest group meeting of Intelligent Systems in Molecular Biology 2011.

The goals in the CAFA challenge of 2011 were to make functional predictions for a set of genes from seven eukaryote species. More than 30 algorithms have been entered into the challenge, including our multi-view *GOstruct* framework. Among the baseline algorithms was FANN-GO [19], which uses GOtcha scores described above to train an ensemble of multioutput neural networks to predict the association of proteins with every GO term. The authors present two variants of FANN-GO: one trained on proteins from multiple species, similar to our cross-species view, and another limited to sequences in the target species only [19]. Other algorithms included SIFTER [26], transfer-of-annotation methods [34] as well as approaches that utilized Bayesian networks and binary SVMs, similar to methods used for Mousefunc [92].

1.7 Publications associated with the presented work

The *GOstruct* framework was first introduced at the 8th Annual International Conference on Computational System Bioinformatics [76], where we demonstrated that it outperformed other competitors on the Mousefunc challenge dataset. The extended version of the paper with additional experiments was then published in the Journal of Bioinformatics and Computational Biology [77]. We introduced the extension of the framework to multi-view learning at the ACM Conference on Bioinformatics, Computational Biology and Biomedicine [78]. Finally, application of our multi-view framework to the CAFA challenge was presented at the Automatic Function Prediction special group meeting of ISMB2011 [79].

1.8 Overview of chapters

We start off by describing previous approaches to GO term prediction in Chapter 2. In Chapter 3, we review structured SVMs and present the basic *GOstruct* method that utilizes structured SVMs to make GO term predictions. We compare the performance of *GOstruct*

to a BLAST-based nearest-neighbor classifier as well as algorithms entered in the Mousefunc challenge. We extend *GOstruct* to the multi-view learning paradigm in Chapter 4; multi-view learning allows us to combine heterogeneous data sources from multiple species. Our empirical results demonstrate that this combination yields more accurate predictions than classifiers trained on a single species or a single source of data only. In Chapter 5, we present a summary of our contribution and note a set of open questions.

Chapter 2

Previous Work

In this chapter we review previous machine learning approaches to GO term prediction. We focus on methods that specifically predict GO terms, noting that some algorithms in the literature use other protein function vocabularies [89, 61, 71]. Several of the algorithms reviewed here were entered in Mousefunc challenge [64], which provided a critical assessment of their performance.

All of the methods discussed here approach the problem of GO term prediction as a collection of binary classification problems, where predictions are made on individual GO terms. This often leads to predictions that are inconsistent among themselves (c.f. Figure 1.2); a classifier may predict that a protein is a p53 binder (GO:0002039) but not a binder in general (GO:0005488). Therefore, when discussing machine learning algorithms for GO term predictions, we consider three aspects:

- how the algorithm makes predictions for individual GO terms;
- how the algorithm combines multiple sources of heterogeneous data;
- how the algorithm addresses the hierarchical constraints of GO namespaces.

While every algorithm reviewed here will have a way to make binary predictions, some algorithms do not combine multiple sources of data, focusing on a single set of features (usually protein-protein interactions [55]). Likewise, some algorithms (e.g., work by Kim, *et al.* [47]) give no consideration to the hierarchical inconsistencies.

2.1 Binary prediction methods

The two common ways of assigning single GO terms to proteins are discriminative algorithms such as SVMs [62, 32] and random forests [84, 36], as well as label propagation on graphs [57, 17, 55]. The SVM [88] is a classifier that is widely accepted as a state-of-the-art tool for binary classification. Given a feature map ϕ, the SVM learns a maximum-margin separating hyperplane between the positive and negative examples in the corresponding feature space. The hyperplane is chosen such that it has the largest distance (margin) to examples from both classes. SVMs are used in conjunction with kernels, which are binary functions that can be thought of as similarity measures between proteins [9]. A kernel has the property that it represents a dot product in some feature space. Whenever an algorithm depends on the data through dot products only, as is the case with SVMs, the dot product computations can be replaced with calls to the kernel, which effectively allows one to apply the algorithm in the corresponding feature space without explicitly mapping the data to it first. The use of kernels is particularly appealing in bioinformatics where the datasets are often comprised of non-numerical objects, such as protein sequences. Obozinski, *et al.* [62] and Guan, *et al.* [32] used SVMs with linear kernels (dot products in the raw feature space) as well the more sophisticated diffusion kernel [49] for the protein-protein interaction data.

Random forest methods employ an ensemble of decision trees to make predictions about whether a protein is associated with a particular GO term [84]. Decision trees recursively partition the data according to the value of one or more features; the value thresholds at each step are chosen such that they maximize the separation between the classes [35, 84]. Inference in random forests is based on the fraction of decision trees that classify the query protein as positive (i.e., having a particular GO term). Hayete, *et al.* [36] used the OC1 decision tree system [60] to make split decisions at each node according to linear combinations of real-valued features. They used the resulting trees to make inference about individual GO terms from protein domain composition data [36].

Label propagation methods make use of functional association networks to predict the association of GO terms for query proteins [57, 17, 55, 45]. A functional association network

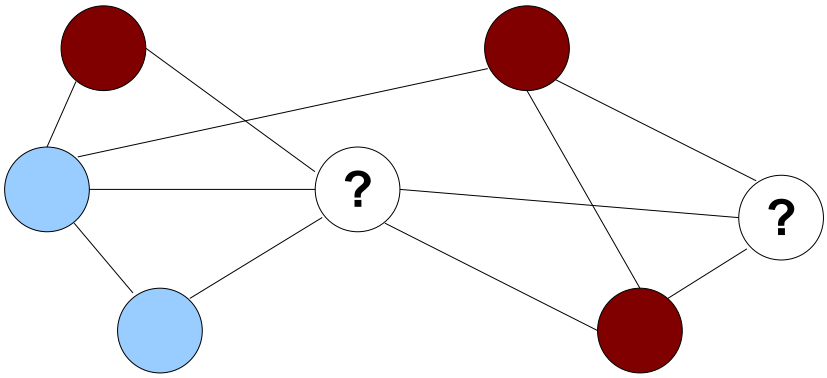


Figure 2.1: An example of a functional association network. The nodes correspond to proteins and edge denote evidence of co-functionality. Proteins annotated with a particular GO term are shaded dark, while the proteins not associated with the GO term are shaded light. The query proteins are presented with question marks. While the labels for some of the query proteins are obvious, such as the one on the right, the situation is usually more complicated as both positive and negative training proteins will be neighboring the query. The latter is demonstrated by the query in the middle of the example graph.

is defined as a graph $\mathcal{G} = (V, E)$, where the vertices V correspond to proteins and the edges E represent some evidence that two proteins perform the same function. The networks can be readily constructed from any protein similarity metric; some examples include binary interaction data [55] and Pearson correlation values between gene expression profiles [57, 17].

Given a functional association network, the set of training labels can be propagated to the test nodes according to the weight of the co-functionality links (Figure 2.1); the intuition is that two nodes that have high evidence of co-functionality are more likely to share the same binary label. Label propagation algorithms can be roughly separated into two categories: those that consider only the immediate neighborhood of a test node [47, 17, 84] and those that optimize some global criterion over the entire network [57, 17, 45]. If w_i is the edge weight between a query protein and its i^{th} neighbor, some local label propagation algorithms [47, 17]

compute the prediction scores according to

$$S = 1 - \prod_{i \in N^+} (1 - w_i), \tag{2.1}$$

where N^+ denotes the set of positively annotated nodes in the neighborhood of the query protein [47, 17]. In this case, the weights are treated as probabilities that two proteins perform the same function and the measure in Equation (2.1) is the probability that the query protein shares the function with at least one of its neighbors, where each neighbor is treated independently of all others.

The criterion optimized by global label propagation varies from method to method. Zhou, *et al.* [96] proposed to assign prediction scores such that the scores are consistent with known node labels and the score similarity for neighboring nodes is proportional to the edge weight between them:

$$\arg\min_{\mathbf{f}} \sum_i (f_i - y_i)^2 + \sum_i \sum_j w_{ij}(f_i - f_j)^2, \tag{2.2}$$

where f_i are prediction scores for individual nodes, w_{ij} is the weight between nodes i and j, and y_i are the labels specified as 1 for positive nodes, -1 for negative nodes, and 0 for unlabeled query nodes. From the machine learning point of view, the optimization criterion can be seen as error minimization (first term) and model regularization (second term). GeneMANIA [57] applied this method to GO term prediction, using $(n^+ - n^-)/n$ (the difference between the fractions of data being positive and negative) as the label bias for unlabeled nodes to account for the fact that only a small portion of all genes are labeled with a particular GO term. Karaoz, *et al.* [45] minimize the "energy" function given by

$$-\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j, \tag{2.3}$$

where w_{ij} is again the weight between nodes i and j, and s_i, s_j are the labels (in this case, -1 and 1) associated with nodes i and j. The labels are given by the training data from annotated nodes and inferred by the algorithm for unannotated nodes. Because the labels are set to 1 or -1, an inconsistent assignment of labels to two nodes that share a strong link will make a positive contribution to the objective function in Equation (2.3) and the

problem can be viewed as minimizing the effect of such inconsistent assignments based on the functional links w_{ij}.

Several of the methods combine several of the models discussed above. For example, Kim, *et al.* combine a naive Bayes classifier with local label propagation [47]. Funckenstein combines random forest classifiers with local label propagation [84]. Given a model θ, prediction scores obtained using the model are often represented as log-odd ratios $\log \frac{P(C=1|\theta)}{P(C=0|\theta)}$ (where again $C = 1$ implies association with a particular GO term, and $C = 0$ denotes the opposite), and multiple models are combined using logistic regression [84, 62]:

$$log\frac{P_{joint}(C=1)}{P_{joint}(C=0)} = w \log \frac{P(C=1|\theta_1)}{P(C=0|\theta_1)} + (1-w) \log \frac{P(C=1|\theta_2)}{P(C=0|\theta_2)}, \tag{2.4}$$

where w is learned from the training data.

2.2 Data Integration

As discussed in the previous chapter, a number of different data sources are relevant to the prediction of protein function and most methods reviewed here make use of two or more of these data sources. The simplest way in which these data sources are combined is via simple concatenation of features from all datasets. When working with kernels, concatenation of feature vectors is equivalent to the summation of the corresponding kernels [32].

In functional association networks, data sources are generally combined at the edge level [47, 57]. For example, GeneMANIA constructs a separate network for each source of data and then combines them via a weighted summation, where the weights are learned through ridge regression [57].

Another way of using multiple sources of data is to train a separate model for each source and then combine individual model predictions, using, e.g., logistic regression in Equation (2.4), as done by Obozinski, *et al.* [62]. Alternatively, predictions can be combined using Naive Bayes [32], or a simple combination of posterior probabilities [17]:

$$P(C=1|\theta_1,\theta_2,...,\theta_p) = 1-(1-P(C=1|\theta_1))\cdot(1-P(C=1|\theta_2))\cdot...\cdot(1-P(C=1|\theta_p)), \tag{2.5}$$

where θ_i refers to the model trained on the i^{th} of p datasets.

2.3 Prediction Reconciliation

As discussed earlier, treating prediction of protein function as a collection of binary classification problems can lead to inconsistent predictions where a query protein is annotated with a particular GO term but not its ancestor. In this section, we discuss some of the ways in which methods reconcile predictions made for individual GO terms.

While the original GeneMANIA algorithm [57] performed no reconciliation across GO terms, an extension of the framework by the authors solves the optimization problem in Equation (2.2) for all GO terms simultaneously, while introducing an additional term to account for parent-child relationships between the GO terms [56]:

$$\arg\min_{\mathbf{f}} \sum_{k} \sum_{i} (f_{ik} - y_{ik})^2 + \sum_{k} \sum_{i} \sum_{j} w_{ij} (f_{ik} - f_{jk})^2 + \sum_{k,l} \sum_{i} h_{kl} (f_{ik} - f_{il})^2, \qquad (2.6)$$

where h_{kl} are indicator variables denoting parent-child relationships between GO terms k and l. The prediction scores f in the optimization problem in Equation (2.6) are now dual-indexed, with the first index iterating over the proteins and the second index iterating over the GO terms. The three terms in the optimization problem correspond to a) the deviation of the prediction scores from the bias labels (as defined above), b) the difference in prediction scores between neighboring proteins in a functional association network for a particular GO term, and c) the difference in prediction scores across all proteins for any two GO terms that have a parent-child relationship. While the solution to the new optimization problem does not necessarily produce a set of consistent predictions, the prediction scores tend to be more consistent with the hierarchical constraints [56].

The method developed by Guan, *et al.* [32], that uses SVMs to make predictions for individual GO terms, reconciles predictions with the hierarchical constraints by using the Bayesian networks approach: each node in the GO hierarchy is treated as a hidden variable, and is associated with a single observed node corresponding to the output from the corresponding SVM. The hierarchical constraints can then be enforced through conditional probabilities on the hidden nodes [6].

Perhaps the most extensive analysis of binary prediction reconciliation has been per-

formed by Obozinski, *et al.* [62], who compared 11 different algorithms. Three of these algorithms were simple heuristics where the models trained on individual GO terms were combined through logistic regression according to "max", "and" and "or" operators. Another four of the algorithms utilized Bayesian inference (similar to what was used by Guan, *et al.* [32]) with individual algorithms being distinguished by whether the Bayesian network edges were directed from parents to children or the opposite, as well as whether Bayesian log posteriors or logistic regression log posteriors were used during inference. Another algorithm, which the authors call *cascaded logistic regression*, fit a logistic regression model at every GO node using only proteins that were annotated with all the ancestor terms. The remaining three algorithms project the set of posterior probabilities obtained at every GO node to the closest set of probabilities that satisfy hierarchical constraints, where the "closeness" was measured either through squared Euclidean distance or Kullback-Leibler divergence. Obozinski, *et al.* concluded that the latter set of projection methods yielded the most accurate predictions, as measure by precision and recall [62]. The authors also note that reconciliation can yield a decrease in performance compared to when no reconciliation is performed [62].

2.4 Case study: Funckenstein

We now focus on one particular algorithm, Funckenstein [84], which achieved the best performance in the Mousefunc challenge, compared to all other competitors [64].

Funckenstein trained two separate models for each GO term, and combined those models using logistic regression as in Equation (2.4). The first model, which the authors refer to as *guilt-by-profiling* [84], used a random forest classifier [14] on all available features. Given a random forest produced in training, predictions were made by a simple majority vote across all trees. The second model, which is referred to as *guilt-by-association* [84], was given by a functional association network. The edges in the network were obtained by training a decision tree to answer the question "do these two proteins perform the same function?" and using the prediction confidence as the edge weights. Given a network, the prediction score of

a particular GO term being associated with a query protein was obtained by averaging the highest three edge weights between the query protein and proteins with known annotations.

Although Funckenstein yielded the best performance in the Mousefunc challenge, it did not account for predictions that were inconsistent with the hierarchical constraints.

Chapter 3

The GOstruct Method

In this chapter we present the *GOstruct* method and compare its performance to the current state-of-the-art algorithms for GO term prediction. We assume the training data is provided as $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n \in (\mathcal{X} \times \mathcal{Y})^n$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input space and the output space, respectively. Given this training data, the goal of a structured-output method is to construct an accurate mapping $h : \mathcal{X} \to \mathcal{Y}$. The standard approach to finding such a mapping is to minimize the empirical loss, while maintaining low model complexity [35]. The empirical loss is given by $\sum_{i=1}^n \Delta(\mathbf{y}_i, h(\mathbf{x}_i))$, where Δ is a loss function that returns a non-negative measure of disagreement between a pair of labels. Maintaining low model complexity is aimed at preventing overfitting of the model to the training data, with the specifics being dependent on the model itself.

Structured-output methods are centered around the compatibility function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}$ that scores pairs of input-space examples and labels [5]. The desired mapping h is obtained from f via the arg max operator:

$$h(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} f(\mathbf{x}, \mathbf{y}), \tag{3.1}$$

which selects the label $\mathbf{y}$ most compatible with the input $\mathbf{x}$. The learning objective is then to ensure that the correct label $\mathbf{y}_i$ yields the highest compatibility score with $\mathbf{x}_i$ for every training example.

While the output space $\mathcal{Y}$ is domain-dependent, its size is usually exponential in the number of output variables. This makes the explicit computation of Equation (3.1) intractable

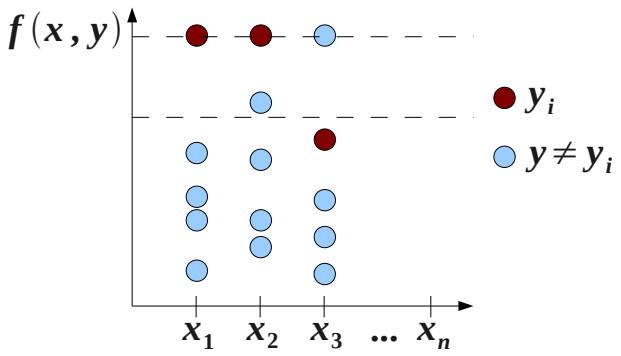


Figure 3.1: Graphical depiction of the constraints in Equation (3.3). Training examples are displayed along the horizontal axis. For demonstration purposes, we assume that the highest compatibility values for the three presented examples are all equal to each other. For every example, the aim is to have the compatibility values between the true label and all other labels separated by a margin, depicted with the two dashed lines. Example $\mathbf{x}_1$ satisfies this. Example $\mathbf{x}_2$, while correctly classified, has a margin violation. Example $\mathbf{x}_3$ is misclassified.

and requires an inference algorithm tailored to the structure of the output space. Some examples include dynamic programming [85] and graph inference algorithms [83]. The inference algorithm for the problem in Equation (3.1) is treated as a black box, often referred to as the *separation oracle*, by the training algorithm.

Structured-output methods differ in their definition of f, and in this project we focus on those that are linear in some joint input-output feature space: $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T \psi(\mathbf{x}, \mathbf{y})$. The feature map $\psi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}^d$ is user-specified, and the training objective is to find a weight vector $\mathbf{w}$ that yields the highest compatibility score for the correct label for all training examples, i.e.,

$$\arg\max_{\mathbf{y} \in \mathcal{Y}} w^T \psi(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_i \quad \text{for } i = 1, \ldots, n. \tag{3.2}$$

To ensure robustness, we further require that the compatibility values for all other labels are separated by a margin γ:

$$\mathbf{w}^T \psi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \mathbf{w}^T \psi(\mathbf{x}_i, \mathbf{y}) \geq \gamma \quad \text{for } i = 1, \ldots, n. \tag{3.3}$$

Examples that fail to satisfy these contraints are said to be *violating the margin*, which may or may not result in misclassification of those examples. The geometric intuition of the constraints is presented in Figure 3.1. The two algorithms we consider for this problem

are the structured perceptron [21], where the margin γ is specified by the user, and the structured SVM [85], which aims to maximize γ.

We consider both algorithms in the context of kernel methods. A kernel can be thought of as a measure of similarity between pairs of objects and formally requires an associated feature space where the kernel acts as the inner product [73]. Whenever an algorithm depends on the data through dot products only, each dot product can be replaced by a call to the kernel function K; this is known as the *kernel trick* [73] and has the effect of applying the algorithm in the feature space associated with the kernel. In the case of binary classification, the dot products are in the feature space defined by some input-space map $\phi : \mathcal{X} \to \mathcal{R}$. The corresponding kernel is then a function between the two data points in the input space: $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2)$. When dealing with structured-output problems, however, the dot products are in the joint input-output feature space, and the kernels are functions of both inputs and outputs: $K\left((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2)\right) = \psi(\mathbf{x}_1, \mathbf{y}_1)^T \psi(\mathbf{x}_2, \mathbf{y}_2)$.

Kernel methods possess many desirable properties that make their application to non-numeric data, such as biological sequences, natural [9]. One important property is that kernels don't require explicit access to the underlying feature maps, and in many cases computing the dot products can be done more efficiently without first mapping the data. For example, the local alignment kernel considers all possible local alignments of two strings $\mathbf{x}_1$ and $\mathbf{x}_2$ and computes the sum of scores associated with those alignments [69]; the value of the kernel can be efficiently computed using a modified Smith-Waterman dynamic programming algorithm [75]. Another appealing property arises from kernel arithmetic: the sum and product of two kernels is also a kernel [73]. This allows for a natural aggregation of heterogeneous data sources and feature maps. This is particularly appealing in biological applications, because the data sources are often disparate: a protein is described by its sequence, as well as its interactions with other proteins and the level of gene expression under various conditions.

3.0.1 Structured Perceptron

Algorithm 1 Structured-output Perceptron: $GOstruct_{p\Delta}$

Input: training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, parameter γ.
Output: parameters $\alpha_{i,\mathbf{y}}$ for $i = 1, \ldots, n$ and $\mathbf{y} \in \mathcal{Y}$.
Initialize: $\alpha_{i,\mathbf{y}} = 0 \quad \forall i, \mathbf{y}$. //only non-zero values of α 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 for x_i:
    $\delta \leftarrow f(\mathbf{x}_i, \mathbf{y}_i) - f(\mathbf{x}_i, \bar{\mathbf{y}})$
    if $\delta < \gamma$ **then**
      $\alpha_{i,\mathbf{y}_i} \leftarrow \alpha_{i,\mathbf{y}_i} + 1$
      $\alpha_{i,\bar{\mathbf{y}}} \leftarrow \alpha_{i,\bar{\mathbf{y}}} - \Delta(\mathbf{y}_i, \bar{\mathbf{y}})$
    end if
  end for
until a termination criterion is met

The perceptron algorithm is a simple linear classifier and its extension to the structured-output setting maintains this simplicity [21]. To make use of kernels in the case of perceptron we make an assumption that the weight vector $\mathbf{w}$ is a linear combination of training examples and labels:

$$\mathbf{w} = \sum_{i=1}^{n} \sum_{\mathbf{y} \in \mathcal{Y}} \alpha_{i\mathbf{y}} \psi(\mathbf{x}_i, \mathbf{y}). \tag{3.4}$$

We can now rewrite the difference in compatibility values from Equation (3.3) as

$$\begin{aligned} & \mathbf{w}^T \psi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \mathbf{w}^T \psi(\mathbf{x}_i, \mathbf{y}) & = \\ = & \sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} \psi(\mathbf{x}_j, \mathbf{z})^T \psi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} \psi(\mathbf{x}_j, \mathbf{z})^T \psi(\mathbf{x}_i, \mathbf{y}) & = \\ = & \sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} K\left((\mathbf{x}_j, \mathbf{z}), (\mathbf{x}_i, \mathbf{y}_i)\right) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} K\left((\mathbf{x}_j, \mathbf{z}), (\mathbf{x}_i, \mathbf{y})\right). \end{aligned}$$

The problem can now be reformulated as finding the coefficients α, such that

$$\sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} K\left((\mathbf{x}_j, \mathbf{z}), (\mathbf{x}_i, \mathbf{y}_i)\right) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \sum_{j=1}^{n} \sum_{\mathbf{z} \in \mathcal{Y}} \alpha_{j\mathbf{z}} K\left((\mathbf{x}_j, \mathbf{z}), (\mathbf{x}_i, \mathbf{y})\right) \geq \gamma \quad \text{for } i = 1, \ldots, n. \tag{3.5}$$

We present the structured-output perceptron variant considered in this work as Algorithm 1. This variant is characterized by the use of a margin, as per Equation (3.5). The desired value of the margin, γ, is a hyperparameter specified by the user. In our implementation, the termination criterion is taken to be a limit on the number of iterations.

Note that the standard version of the algorithm updates margin violations according to $\alpha_{i,\bar{\mathbf{y}}} \leftarrow \alpha_{i,\bar{\mathbf{y}}} - 1$ [21]. This assigns the same penalty for slight and gross misclassifications, which intuitively is not desired. We propose to scale the penalty by the loss instead, as presented in Algorithm 1; the results presented later indicate that this scaling yields better predictions than the standard update rules.

3.0.2 Structured Support Vector Machines

The structured SVM further aims to maximize the margin γ in Equation (3.3). Equivalently, this goal can be expressed as minimizing the norm of $\mathbf{w}$ while keeping the value of γ fixed [16]. The non-linear constraints in Equation (3.3) present an optimization challenge and the standard approach is to expand them into the corresponding set of linear constraints. The difference in compatibility values between the true label and another candidate is an integral part of this expansion and, for notational convenience, we define

$$\delta\psi_i(\mathbf{y}) = \psi(\mathbf{x}_i, \mathbf{y}_i) - \psi(\mathbf{x}_i, \mathbf{y}) \tag{3.6}$$

to represent this difference. Following the formulation where γ is fixed, we decompose the non-linear constraints in Equation 3.3 into the following set of linear constraints considered by the structured SVM:

$$\mathbf{w}^T \delta\psi_i(\mathbf{y}) \geq 1 \quad \text{for } i = 1, \ldots, n; \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\}. \tag{3.7}$$

If there exists $\mathbf{w}$ that satisfies all these constraints, then we say that the data is separable. Unfortunately, most real-world datasets are *not* separable and we have to allow for margin violations:

$$\mathbf{w}^T \delta\psi_i(\mathbf{y}) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, n; \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\}, \tag{3.8}$$

where $\xi_i \geq 0$ are called *slack variables* and measure the amount of violation. For a given solution $\hat{\mathbf{w}}$, we can compute the amount of margin violation for any particular training example directly from the corresponding set of constraints:

$$\xi_i = \left[1 - \hat{\mathbf{w}}^T \delta\psi_i(\mathbf{y})\right]_+, \tag{3.9}$$

where we use $[\cdot]_+$ to denote a function that returns its argument if the argument is non-negative and zero otherwise.

The structured SVM objective has two goals. One goal is to minimize the amount of margin violation, measured by $\sum_{i=1}^{n} \xi_i$. The other goal is to maximize the margin, which is equivalent to minimizing the norm of $\mathbf{w}$. Minimizing the norm of $\mathbf{w}$ plays the role of maintaining low model complexity in the context of linear models [35]. By maximizing the separation margin, we increase model robustness to noise and reduce overfitting to the training data. The two goals are generally competing with each other: larger margins are prone to introduce more margin violations. We use a user-specified parameter C to place more emphasis on either part, leading to the following structured SVM formulation [85]:

$$\min_{\mathbf{w},\xi_i} \tfrac{1}{2}\|\mathbf{w}\|_2^2 + \tfrac{C}{n}\sum_{i=1}^{n}\xi_i \tag{3.10}$$

$$s.t. \qquad \mathbf{w}^T\delta\psi_i(\mathbf{y}) \geq 1 - \xi_i \qquad \text{for } i = 1,\ldots,n; \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\} \tag{3.11}$$

$$\xi_i \geq 0 \qquad \text{for } i = 1,\ldots,n, \tag{3.12}$$

where $\|\cdot\|_2$ is the L^2 norm.

Equation (3.11) requires that all incorrect labels are separated from the true label by the same margin value. Intuitively, labels that are closer to the truth should be allowed to have higher compatibility values (and therefore, require less separation) than grossly incorrect labels. Two implementations of this intuition are *margin re-scaling* and *slack re-scaling*, which incorporate the loss function Δ into the constraints [85]. Margin re-scaling replaces the constraint in Equation (3.11) with

$$\mathbf{w}^T\delta\psi_i(\mathbf{y}) \geq \Delta(\mathbf{y}_i,\mathbf{y}) - \xi_i \;\text{ for } i = 1,\ldots,n; \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\}. \tag{3.13}$$

Similarly, slack re-scaling replaces it with

$$\mathbf{w}^T\delta\psi_i(\mathbf{y}) \geq 1 - \xi_i/\Delta(\mathbf{y}_i,\mathbf{y}) \;\text{ for } i = 1,\ldots,n; \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\}. \tag{3.14}$$

The optimization problem in Equations (3.10)-(3.12) is known as the *primal formulation* of the structured SVM [85]. Similar to the structured perceptron, we would like to make use

of kernels, which leads us to consider the *dual formulation* instead. The Wolfe dual [28] to the problem is given by the following [85]:

$$\max_{\alpha} -\tfrac{1}{2}\sum_{i,j,\mathbf{y},\bar{\mathbf{y}}} \alpha_{i\mathbf{y}}\alpha_{j\bar{\mathbf{y}}}\left[\delta\psi_i(\mathbf{y})\right]^T \delta\psi_j(\bar{\mathbf{y}}) + \sum_i \sum_{\mathbf{y}} \alpha_{i\mathbf{y}} \tag{3.15}$$

$$\text{s.t.} \quad \sum_{\mathbf{y}} \alpha_{i\mathbf{y}} \leq C/n \quad \text{for } i = 1, \ldots, n, \tag{3.16}$$

$$\alpha_{i\mathbf{y}} \geq 0 \quad \text{for } i = 1, \ldots, n; \mathbf{y} \in \mathcal{Y}, \tag{3.17}$$

where the only unknowns are the Lagrange multipliers $\alpha_{i\mathbf{y}}$. Note that since the constraints in Equation (3.11) involve both the training examples and the labels, the Lagrange multipliers, α, are dual-indexed. We can expand the dot product of the compatibility differences given by the $\delta\psi(\cdot)$ notation and replace the corresponding terms in the expansion with kernels. Similar dual formulations exist for margin and slack re-scaling [85]. In all formulations, the weight vector can be obtained from the Lagrange multipliers using

$$\mathbf{w} = \sum_i \sum_{\bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \delta\psi_i(\bar{\mathbf{y}}), \tag{3.18}$$

which is slightly different than the perceptron case (Equation (3.4)) in that the difference vectors defined in Equation (3.6) are used instead. The compatibility can then be computed according to

$$f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T \psi(\mathbf{x}, \mathbf{y}) = \sum_i \sum_{\bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \delta\psi_i(\bar{\mathbf{y}})^T \psi(\mathbf{x}, \mathbf{y}) = \tag{3.19}$$

$$= \sum_i \sum_{\bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \left[\psi(\mathbf{x}_i, \mathbf{y}_i) - \psi(\mathbf{x}_i, \bar{\mathbf{y}})\right]^T \psi(\mathbf{x}, \mathbf{y}) = \tag{3.20}$$

$$= \sum_i \sum_{\bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \left[K((\mathbf{x}_i, \mathbf{y}_i), (\mathbf{x}, \mathbf{y})) - K((\mathbf{x}_i, \bar{\mathbf{y}}), (\mathbf{x}, \mathbf{y}))\right]. \tag{3.21}$$

There are $n|\mathcal{Y}|$ variables in the dual optimization problem in Equation (3.15). For most applications, this is too many for solving the problem directly. Every dual variable is a Lagrange multiplier for a constraint in Equation (3.11), and by focusing our attention on the most violated constraints only, we can construct an approximate solution by working with a smaller subset of variables. This subset of variables is known as the *working set* and gets constructed incrementally by the training algorithm. At every iteration, the algorithm loops

Algorithm 2 Working-set-based approach to training a structured SVM

Input: training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$
Output: parameters $\alpha_{i,\mathbf{y}}$ for $i = 1, \ldots, n$ and $\mathbf{y} \in \mathcal{Y}$.
Initialize: $\alpha_{i,\mathbf{y}} \leftarrow 0$ for $i = 1, \ldots, n$, $\mathbf{y} \in \mathcal{Y}$, and $\mathcal{W}_i \leftarrow \emptyset$ for $i = 1, \ldots, n$.
repeat
  for $i = 1$ **to** n **do**
    Find the most violated constraint $\bar{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} H_i(\mathbf{y})$. (See text for the definition of H_i.)
    Compute the current slack $\xi_i = [\max_{\mathbf{y} \in \mathcal{W}_i} H(\mathbf{y})]_+$.
    if $H(\bar{\mathbf{y}}) > \xi_i + \epsilon$ **then**
      Add the new constraint to the working set: $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{\bar{\mathbf{y}}\}$.
      Optimize the dual objective in Equation (3.15) over $\alpha_{i,\mathbf{y}}$ for $\mathbf{y} \in \mathcal{W}_i$.
    end if
  end for
until No new constraints are added after a full pass through the data

over the training examples, and for every example it identifies the most violated constraint. The corresponding Lagrange multiplier is then added to the working set and the optimization problem in Equations (3.15)-(3.17) is solved with respect to those variables only. The algorithm terminates when any constraint not in the working set is violated by no more than ϵ when compared to the most violated constraint in the working set. It has been shown that even if $|\mathcal{Y}|$ is exponential in the number of variables that make up the output space, the training algorithm will still converge in a polynomial number of steps for any arbitrarily small ϵ [85].

We present the working set approach as Algorithm 2. We measure the amount of margin violation by

$$H_i(\mathbf{y}) = 1 - \mathbf{w}^T \delta\psi_i(\mathbf{y}) \tag{3.22}$$

for a particular training example $(\mathbf{x}_i, \mathbf{y}_i)$ when no rescaling is employed. Similarly, in the presence of margin or slack rescaling, the margin violations are measured by

$$H_i(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \delta\psi_i(\mathbf{y}) \tag{3.23}$$

and

$$H_i(\mathbf{y}) = (1 - \mathbf{w}^T \delta\psi_i(\mathbf{y}))\Delta(\mathbf{y}_i, \mathbf{y}), \tag{3.24}$$

respectively. These three definitions of function H_i can be derived directly from the constraints in Equations (3.11), (3.13), and (3.14), respectively.

The optimization step in Algorithm 2 is usually performed using a variant of Sequential Minimal Optimization (SMO) [65]. SMO-like algorithms alternate between selecting a subset of variables $\alpha_{i\mathbf{y}}$ and maximizing the dual objective (Equation (3.15)) with respect to that subset, while keeping all other variables fixed. The original SMO algorithm focused on optimizing two variables at a time, using a set of heuristics for choosing those variables [65]. The idea has since been extended to optimization of multiple variables [44] and the use of first and second derivative information in selecting which variables to optimize [46, 27]. The main advantage of all SMO-like algorithms is their scalability to large datasets. By considering a small number of dual variables at a time, the kernel matrix is never required in its entirety, making solution of large optimization problems tractable. In our case, the working set technique already limits the number of dual variables we consider during optimization. This allows us to precompute the entire kernel matrix involved in the optimization subproblem, and this allows us to employ a different optimization algorithm. In this work, we explore an alternative to SMO to solve the problem in Equations (3.15)-(3.17) over the dual variables $\alpha_{i\mathbf{y}}$ that correspond to the constraints in the working set $\mathcal{W}_i$. The details are in the Appendix.

Basic GOstruct

The *GOstruct* method requires the user to choose a joint kernel, an inference oracle, and a loss function. Once these three elements have been identified, the hyper-parameter C or γ can be selected via cross-validation on the training data.

The joint kernel used in our experiments is a product of the input space and the output space kernels:

$$K((\mathbf{x},\mathbf{y}),(\mathbf{x}',\mathbf{y}')) = K_{\mathcal{X}}(\mathbf{x},\mathbf{x}')K_{\mathcal{Y}}(\mathbf{y},\mathbf{y}'). \tag{3.25}$$

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.

In most of our experiments, we limited the output space $\mathcal{Y}$ to the labels that occur in the training set only, arguing that this allows the classifier to focus on combinations of GO terms that are biologically relevant. We experimented with other inference oracles that considered a larger portion of the output space (defined as the set of all labels that satisfy hierarchical constraints), and the empirical results of those experiments further motivate our choice to limit the inference to the labels that occur in the training set; we present these results later in this chapter.

The loss function plays two roles. First, it is used in margin and slack rescaling formulations (Equations (3.13) and (3.14)) to require less separation for labels that are close to the truth. The second role of a loss function is to measure algorithm performance. When working with binary classification problems, the accuracy of a prediction can be measured in several ways, one of which is with an indicator function that compares the predicted label to the true label. This is known as the 0-1 loss. Because we are dealing with complex labels in a structured-output problem, the 0-1 loss is no longer appropriate, as it makes no distinction between slight and gross misclassifications. A number of loss functions that incorporate taxonomic information have been proposed in the context of hierarchical classification [38, 68, 48]. These either measure the distance between labels by finding their least common ancestor in the taxonomy tree [38] or penalize the first inconsistency between the labels in a top-down traversal of the taxonomy [68]. Kiritchenko *et al.* proposed a loss function that is related to the F_1 measure which is used in information retrieval [87] and was used by Tsochantaridis *et al.* in the context of parse tree inference [85].

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 GO terms. 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

$$P(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}^T\hat{\mathbf{y}}}{\hat{\mathbf{y}}^T\hat{\mathbf{y}}}, \quad R(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}^T\hat{\mathbf{y}}}{\mathbf{y}^T\mathbf{y}}. \tag{3.26}$$

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 $\Delta_{F_1}(\mathbf{y}, \hat{\mathbf{y}}) = (1 - F_1(\mathbf{y}, \hat{\mathbf{y}}))$ [85]. 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*:

$$\Delta_{ker}(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{2K(\mathbf{y}, \hat{\mathbf{y}})}{K(\mathbf{y}, \mathbf{y}) + K(\hat{\mathbf{y}}, \hat{\mathbf{y}})}, \tag{3.27}$$

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

Another common way to assess predictor performance is to build a Receiver Operating Characteristic (ROC) curve, which plots the true positive rate as a function of the false positive rate. Many binary predictors output a score, which can then be thresholded by the user to produce either a positive or a negative prediction. The choice of the threshold corresponds to a point on the ROC curve. At the one extreme, a threshold of $-\infty$ will cause all predictions to be positive and yield 100% false positive and true positive rates. Similarly, a threshold of ∞ will cause all predictions to be negative and yield 0% false positive and true positive rates. Other threshold values will map to different points on the ROC curve.

It is common to report the area under the ROC curve (AUC) as a performance metric, and in the case of protein function prediction the metric is often averaged across individual

GO terms [64]. To meaningfully compare *GOstruct* to other algorithms, we compute the prediction confidence for each output-space variable y_i using

$$c_i(\mathbf{x}) = \max_{\mathbf{y} \in \mathcal{Y}_i^+} f(\mathbf{x}, \mathbf{y}) - \max_{\mathbf{y} \in \mathcal{Y}_i^-} f(\mathbf{x}, \mathbf{y}), \tag{3.28}$$

where $\mathcal{Y}_i^+ = \{\mathbf{y} \in \mathcal{Y} | y_i = 1\}$ is a subset of all labels that satisfy the hierarchical constraints and have the i^{th} variable set to 1. The subset $\mathcal{Y}_i^-$ is defined in a similar fashion, except with the i^{th} variable being set to 0. Given this measure, we generate an ROC curve for the variable y_i using the standard approach of computing $c_i(\mathbf{x})$ for all $\mathbf{x}$ in the test set. Note that, while we're able to measure the confidence associated with individual binary output variables, constructing an ROC curve for structured labels is less straightforward, because the notion of a "false positive" is not well defined: a p53 binder may be misclassified as an enzyme binder (slight error) or as a toxin transporter (gross error). An ROC curve that treats all misclassifications equally, no matter how slight or gross, will suffer the same drawbacks as the 0-1 loss.

Experiment 1: GOstruct vs. BLAST NN

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 [90], 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 structural similarity and would introduce bias into any classifier that used sequence similarity to make a prediction [66]. This was done by removing all annotations with the evidence codes: IEA, ISS, ND, RCA,

and NR. 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 terms.

The input-space features were obtained using the Basic Local Alignment Search Tool (BLAST) [1]. BLAST is a dynamic-programming-based method that can be used to align a query protein sequences to a database. The tool produces a set of e-values between the query and significant matches in the database, which we use as features.

We ran BLAST for each of the proteins in our dataset against all four species, removing the hits where a protein was aligned to other proteins from the same species. 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 [72]; 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. For the output-space kernel, $K_{\mathcal{Y}}$, we used a homogeneous linear kernel

$$K_{\mathcal{Y}}(\mathbf{y}, \mathbf{y}') = \mathbf{y}^T \mathbf{y}' - 1, \tag{3.29}$$

which accounts for the fact that all labels share the root node.

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 above.

The results for the leave-one-species-out experiments are presented in Table 3.1. We present loss values only, since the BLAST-NN algorithm does not have confidence scores associated with its predictions, preventing us from building ROC curves for it. We compare

Test on	*C. elegans*	*D. melanogaster*	*S. cerevisiae*	*S. pombe*
# proteins	926	1893	1907	939
BLAST-NN	0.61	0.46	**0.39**	0.37
$GOstruct_p$	0.61	0.43	0.40	0.41
$GOstruct_{p\Delta}$	**0.58**	**0.39**	0.42	**0.36**
$GOstruct_{svmm}$	0.60	0.43	**0.39**	**0.36**
$GOstruct_{svms}$	0.59	0.42	**0.39**	**0.36**
Random	0.78	0.86	0.88	0.84

Table 3.1: 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 $GOstruct_{svms}$). 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

two perceptron variants: one using the standard penalty update, and another using the proposed loss penalty update. We refer to these as $GOstruct_p$ and $GOstruct_{p\Delta}$, respectively. Additionally, we consider two SVM variants — $GOstruct_{svmm}$ and $GOstruct_{svms}$ — which use margin re-scaling and slack re-scaling, respectively. The results show that the various flavors of the *GOstruct* method performing on-par or better than the BLAST nearest-neighbor classifier (BLAST-NN). The only exception to this is 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 $GOstruct_p$ in all folds except *S. cerevisiae*. Furthermore, the SVM-based algorithm outperforms BLAST-NN and $GOstruct_p$, but not $GOstruct_{p\Delta}$, which is a significantly simpler algorithm. The difference in margin re-scaling versus slack re-scaling is negligible, and since the former requires slightly faster computations, we drop slack re-scaling from consideration in the experiments below.

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 3.1. 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, implying that the diferences are statistically significant.

In summary, the results in Table 3.1 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.

Experiment 2: GOstruct on Mousefunc data

As a further comparison of the *GOstruct* method we ran it on the Mousefunc dataset [64]. Mousefunc was a competition held in 2006 aimed at predicting GO terms for a collection of mouse proteins. The protein IDs were masked by the organizers and the data released to the competitors did not include protein sequences to prevent the inference of protein identity. The released data included two different sources of gene expression data, protein-protein interaction adjacency matrix, protein pattern annotation data from Pfam and InterPro, and phylogenetic profiles. The associated GO terms were also provided for the training proteins and the goal of each competitor was to produce predictions for the test set. Below is a description of features from each data source.

Gene expression Gene expression data came from two sources. The first dataset, by Zhang, *et al.* [95] had its values normalized, median-subtracted and transformed by the arcsinh operator. Any negative values were set to 0. The second dataset, by Su, *et al.* [81], was normalized by simply subtracting the mean and dividing by the standard deviation of each feature. We used a linear kernel on the processed features.

Protein domain The three-dimensional structure of a protein generally consists of several parts, where each part, referred to as *protein domain*, can function independently of the rest. Protein domains can be readily extracted from the sequence and further classified into families [7, 41]. The features provided for the Mousefunc challenge come from the Pfam [7] and InterPro [41] databases and consist of indicator variables that signify whether a protein belongs to a particular family. The linear kernel is then the number of families two proteins share in common.

Protein-protein interactions The interaction data for the Mousefunc challenge were obtained from human orthologs (homologs that arise through speciation), which in turn came from the OPHID database [15]. The fact that two proteins interact is not a kernel by itself. Rather, the features for a particular protein are binary values indicating the interaction with another protein. The linear kernel is given by the number of interactions two proteins have in common.

Phylogenetic profiles Phylogenetic profiles were obtained from BioMart [74] and represented by binary values that indicated the presence of orthologs in other species. The corresponding linear kernel was the number of species that were common to two profiles.

Data Integration 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 kernels $K_{\mathcal{Y}}$ and $K_{\mathcal{X}}$ (c.f. Equation 3.25), and the output space is limited to the labels observed in the training set.

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

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

Table 3.2: 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.

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.

We applied the SVM and perceptron-based *GOstruct* methods to the Mousefunc challenge dataset whose statistics are provided in Table 3.2. 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 3.3. 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 3.3). For algorithms that produce confidence scores, the kernel loss can be computed directly from the confidence scores, without thresholding the predictions. For all algorithms except GeneMania, thresholding the confidence measures leads to higher kernel loss.

In addition to the loss values, we computed the confidence of *GOstruct* predictions using

Equation (3.28). These confidence values allowed us to compute area under the curve (AUC) for each GO term. Comparison of averages across all GO terms is presented in Table 3.3. Overall, AUC values echo the relative performance assessment measured by the loss: algorithms that yield lower loss values also tend to yield higher AUC scores. The *GOstruct* method yielded the highest AUC values in all namespaces except biological process, where the value was slightly lower than the one obtained from Guan, *et al.*'s algorithm [32]. However, four of the algorithms produced AUC values in the range of 0.82-0.84 for the biological process experiment, suggesting that the algorithms may be hitting the ceiling of what's possible with the given training data.

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 3.3). The binary SVM experiment was performed using the SVM implementation in the `PyML` machine learning library available at `http://pyml.sf.net` run with the default parameters, and the same input-space kernel used to assess the *GOstruct* methods.

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 Equation (3.1) 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 3.2) 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

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**	0.53	**0.33**	0.42
	AUC	0.86	0.87	0.89	0.92	0.86	0.90	0.91	0.87	**0.94**	0.92
	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**	0.67	0.60	0.67
	AUC	0.76	0.74	0.77	0.83	0.76	**0.84**	0.82	0.76	0.83	0.79
	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
	AUC	0.79	0.77	0.78	0.84	0.75	0.83	0.80	0.82	**0.88**	0.82
	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 3.3: Prediction results on the Mousefunc dataset for molecular function (MF), biological process (BP) and cellular component (CC) namespaces. Reported are the mean kernel loss per protein, precision/recall and mean AUC per GO term. Lower values of the loss and higher values of other metrics 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 and AUC 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.* [47]. Alg 2 is an ensemble of calibrated SVMs by Obozinski, *et. al* [62]. Alg 3 is the kernel logistic regression, submitted by Lee, *et al.* [51]. Alg 4 is geneMANIA [57]. Alg 5 is GeneFAS [17]. Alg 6 is the work by Guan, *et al.* [32]. Alg 7 is Funckenstein [84]. $GOstruct_{p\Delta}$ 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.

competitive performance with the other entries. Whereas in the four species experiment the structured perceptron was on par with 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 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 [66].

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 3.4. The results labeled as "independent" are taken from Table 3.3 and are included for comparison. These correspond

GO namespace	Inference method	$GOstruct_{p\Delta}$ Loss	Prc.	Rec.	$GOstruct_{svm}$ Loss	Prc.	Rec.
MF	independent	0.53	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 3.4: 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.

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.

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. 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.

Inference

As discussed above, the size of the output space $\mathcal{Y}$ is exponential in the number of GO terms, making inference an expensive operation and the bottleneck of the algorithm. In this section, we explore an additional inference algorithm and its impact on predictor performance. There are two separate but related inference problems. The first is given by Equation (3.1) and yields the most compatible label for a query protein. The second problem finds the most violated constraint in a structured SVM using one of the functions H_i defined by Equations (3.22)-(3.24):

$$\arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} H_i(\mathbf{y}). \tag{3.30}$$

Note that when no rescaling is employed, the second inference problem reduces to a variant of the first one:

$$\begin{aligned}
\arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} H_i(\mathbf{y}) &= \arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} \left[1 - \mathbf{w}^T\delta\psi_i(\mathbf{y})\right] = \\
= \arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} &\left[1 - \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}_i) + \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y})\right] = \\
&\arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}) = \arg\max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} f(\mathbf{x}_i,\mathbf{y}),
\end{aligned}$$

but the additional loss terms in H_i prevents the same from happening in the cases of margin or slack rescaling.

In all experiments considered so far, we reduced the output space $\mathcal{Y}$ to the labels observed in the training set. Here we also consider an approximate inference algorithms that does not employ this limitation. The algorithm is based on dynamic programming and assumes that

the inference problem has a linear decomposition:

$$\arg\max_{\mathbf{y}} H_i(\mathbf{y}) = h(\mathbf{x})^T\mathbf{y}, \tag{3.31}$$

for some function h. The compatibility function f satisfies this requirement under the product joint kernel and linear output-space kernel. From Equation (3.19), we have

$$\begin{aligned} f(\mathbf{x},\mathbf{y}) &= \sum_i \sum_{\bar{\mathbf{y}}\neq\mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \left[K((\mathbf{x}_i,\mathbf{y}_i),(\mathbf{x},\mathbf{y})) - K((\mathbf{x}_i,\bar{\mathbf{y}}),(\mathbf{x},\mathbf{y}))\right] = \\ &= \sum_i \sum_{\bar{\mathbf{y}}\neq\mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} \left[K_{\mathcal{X}}(\mathbf{x}_i,\mathbf{x})K_{\mathcal{Y}}(\mathbf{y}_i,\mathbf{y}) - K_{\mathcal{X}}(\mathbf{x}_i,\mathbf{x})K_{\mathcal{Y}}(\bar{\mathbf{y}},\mathbf{y})\right] = \\ &= \sum_i \sum_{\bar{\mathbf{y}}\neq\mathbf{y}_i} \alpha_{i\bar{\mathbf{y}}} K_{\mathcal{X}}(\mathbf{x}_i,\mathbf{x}) \left[\mathbf{y}_i^T\mathbf{y} - \bar{\mathbf{y}}^T\mathbf{y}\right] = h(\mathbf{x})^T\mathbf{y} \end{aligned}$$

Since the two inference problems above are equivalent in the no-rescaling case and both rely on the compatibility function only, the structured SVM with no rescaling is the only classifier where this inference algorithm is considered.

The three GO namespaces (molecular function, biological process and cellular component) are disjoint, and the inference algorithm handles each one in an independent fashion. Each GO namespace is a DAG with a single source and multiple sinks. The function $h(\mathbf{x})$ defines an assignment of weights over the graph nodes, and the inference problem can be thought of as finding a subset of nodes that yields the highest weighted sum while satisfying the hierarchical constraints (every annotated node must have all of its ancestors annotated as well). The algorithm starts by ranking the nodes based on their shortest distance to the source. It then traverses the nodes from farthest to closest and for each node, the weighted sum of the node and its descendants is split up among the node's ancestors. At the end of the traversal, each node is annotated with some fraction of the weights contributed by the nodes deeper in the graph; the graph source is annotated with the full sum of weights. The algorithm then removes the node with the most negative annotation and all its descendants (to enforce hierarchical constraints), and recurses on the resulting subgraph. When no negative annotations remain, the algorithm terminates and returns the remaining set of nodes.

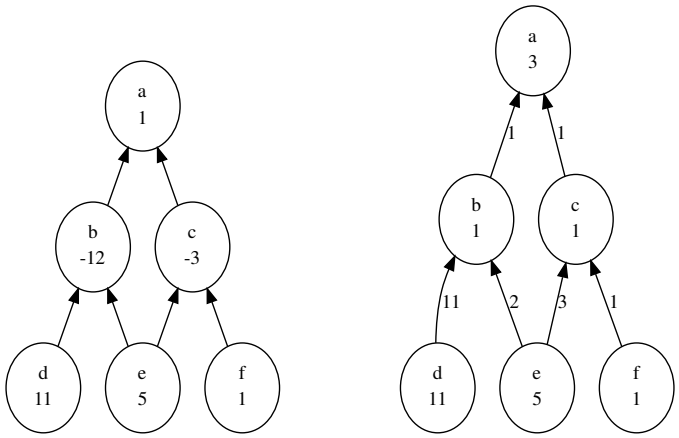


Figure 3.2: (Left) An example of a small GO graph and the corresponding set of weights defined by $h(\mathbf{x})$. (Right) A node traversal that results in exact inference; unfortunately, determining how to split the weight of node **e** among its ancestors is a non-trivial problem.

An important consideration is how the weighted sum of a node gets split up among the node's ancestors. Consider the graph in Figure 3.2. The optimal label includes all nodes and has the weight sum 3. However, deciding how to split the weight contribution of the **e** node between its two ancestors **b** and **c** is a non-trivial problem and the optimal solution requires consideration of the sibling nodes **d** and **f**, which has prohibitive complexity. Contributing the entire weight of 5 to either **b** or **c**, will yield at least one negative node that is going to be cut at the end of the traversal. In our implementation, we split the weight of a node proportionally between its ancestors. While this doesn't yield the optimal solution for the graph in Figure 3.2, we noticed that such weight assignments do not happen frequently and in most cases, this algorithm produces exact inference.

We present the performance of each inference algorithm on the molecular function namespace in Table 3.5. The performance was calculated using five-fold cross-validation on 2945 yeast proteins. We used a structured SVM with no margin rescaling for the two inference algorithm and compare the performance to the margin-rescaling variant used in the previous experiments.

The results demonstrate that while the inference algorithm based on dynamic program-

Rescaling	Oracle	Test Inference	Loss
No	Limited	Limited	0.535
No	Limited	Dynamic	0.842
No	Dynamic	Limited	0.506
No	Dynamic	Dynamic	0.770
Margin	Limited	Limited	0.480

Table 3.5: Performance of different inference algorithms, expressed as mean loss per example on the molecular function namespace. The oracle refers to the algorithm that finds the most violated constraint using the penalty function in Equation (3.22). The test inference refers to performing the argmax operation in Equation (3.1). When inference is restricted to the labels occurring in the training set only, we refer to the algorithm as "Limited". Inference that uses the dynamic programming algorithm described in the text is referred to as "Dynamic".

ming generally leads to better predictions, its restriction to the no-rescaling model limits the algorithm utility; a structured SVM that limits both inference subproblems to the labels occurring in the training set but using margin re-scaling outperforms all variants of the no-rescaling algorithm.

We also note that while dynamic inference yields more flexibility during training, limiting the output space during test inference is still preferred. This is signified by lower loss values when dynamic programming is used for oracle inference, but high performance degradation when used to solve the test inference problem.

Chapter 4

Multi-view Learning

In this chapter, we extend the *GOstruct* framework to handle heterogeneous sources of data using multi-view learning. We first describe the structured-output multi-view framework as it applies to labeled data only and then show how unlabeled data is incorporated. We consider two feature maps ("views"), $\phi^{(c)}(\mathbf{x})$ and $\phi^{(s)}(\mathbf{x})$. The first one is defined for all proteins and comprises a set of *cross-species features* that characterize the sequence of a protein. The second map, $\phi^{(s)}(\mathbf{x})$, is defined for proteins from a particular species only and comprises a set of *species-specific features* such as protein-protein interactions and gene expression data. The two maps are defined over the input space; the corresponding joint feature maps, which we denote with $\psi^{(c)}(\mathbf{x}, \mathbf{y})$ and $\psi^{(s)}(\mathbf{x}, \mathbf{y})$, are functions of both inputs and output as was the case in the previous chapter.

Our goal is to leverage information from both feature maps to make predictions about the function of proteins in the target species. We note that each view will contain a different number of labeled examples, but in the interest of keeping the notation simple, we use a single variable, n_l, with the understanding that its value will vary from one view to another.

In our multi-view setting we use two compatibility functions: $f^{(c)}$, which handles the cross-species view, and $f^{(s)}$, which handles the species-specific view. Inference is then performed according to

$$\hat{\mathbf{y}} = h(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \left(f^{(c)}(\mathbf{x}, \mathbf{y}) + f^{(s)}(\mathbf{x}, \mathbf{y}) \right). \tag{4.1}$$

Each compatibility function, $f^{(c)}$ and $f^{(s)}$, is associated with the corresponding joint feature

map, $\psi^{(c)}$ and $\psi^{(s)}$.

When working with labeled data only, each view is trained independently of the other using the margin-rescaling structured SVM formulation from Equations (3.10)-(3.13):

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C_l}{n_l}\sum_{i=1}^{n_l}\xi_i \tag{4.2}$$

$$\begin{aligned} \text{s.t.} \qquad & \xi_i \geq 0, && i = 1,\ldots,n_l, \\ & \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}_i) - \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}) \geq \Delta(\mathbf{y},\mathbf{y}_i) - \xi_i, && i = 1,\ldots,n_l, \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i \end{aligned}$$

where $\psi(\mathbf{x},\mathbf{y})$ is the feature map of the corresponding view. As before, the margin violations are measured by the slack variables ξ_i. The parameter C_l controls the trade-off between margin magnitude and the amount of margin violations for labeled examples.

In addition to the multi-view method outlined above, we investigate an approach we call the *chain* classifier. In this approach, the predictions made by the cross-species classifier are incorporated into the species-specific feature map by adding a feature for each GO term. In other words, $\arg\max_{\mathbf{y}} f^{(c)}(\mathbf{x}_i,\mathbf{y})$ becomes a set of features in $\phi^{(s)}(\mathbf{x}_i)$. The inference made by the species-specific classifier is then reported as the overall prediction. This is an alternative way of learning from the training information available in the two views and one of its advantages is that the user is not limited to structured SVMs for the cross-species view. While the multi-view approach requires a compatibility function for the joint inference in Equation (4.1), the chain classifier has no such requirement and a simple BLAST nearest-neighbor approach can be used to produce predictions from the cross-species information, which are then provided as input-space features to the species-specific classifier.

The cross-species view will contain proteins from other species and will generally have many more training instances than the species-specific view. Since the BLAST nearest-neighbor classifier scales well with the size of the training data, it provides a natural choice for the cross-species view. In our experiments below, we investigate the impact on performance when BLAST nearest-neighbor is used in place of a structured SVM to make cross-species predictions in a chain classifier.

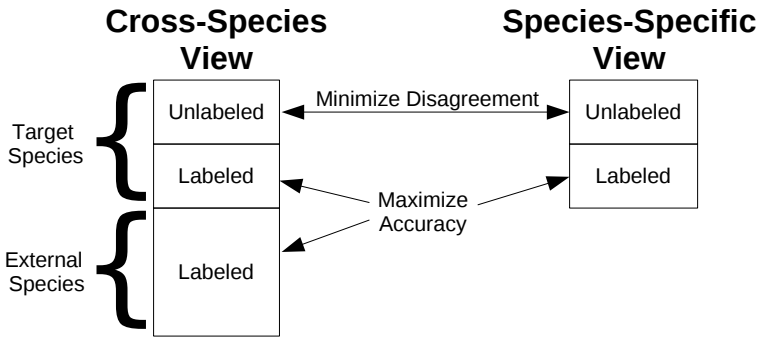


Figure 4.1: The multi-view approach. Data is separated into two views: a cross-species view that contains features computed from sequence, and a species-specific view that contains features computed from PPI data in the target species (*S. cerevisiae* or *M. musculus*). The objective is to maximize the accuracy on the labeled data and minimize the disagreement on the unlabeled data.

4.1 Unlabeled Examples

When producing labels is expensive and time-consuming, as is the case with experimental functional annotations, it is common to incorporate unlabeled example into the analysis [35]; this is known as semi-supervised learning. In our case, unlabeled data is abundant because many proteins in the target species are un-annotated and our intuition is that, by including it in our analysis, we can better learn the input space structure.

In addition to the labeled data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n_l}$, we are also given unlabeled data $\{(\mathbf{x}_i)\}_{i=n_l+1}^{n_l+n_u}$. The objective of multi-view learning now becomes two-fold: maximize the accuracy on the labeled data and minimize the disagreement between views on the unlabeled data [13]. Figure 4.1 presents the graphical overview of the approach. We require that all unlabeled examples span both views to make disagreement minimization possible in the absence of labels.

When dealing with labeled data, we aim to maximize the margin between the true label $\mathbf{y}_i$ and all other candidates. A similar principle holds for the unlabeled data. Given an unlabeled example, we would like to maximize the margin in compatibility between some

label $\mathbf{z}_i$ and all other labels. Formally, for each view we would like to optimize

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C_l}{n_l}\sum_{i=1}^{n_l}\xi_i + \frac{C_u}{n_u}\sum_{i=n_1+1}^{n_l+n_u}\xi_i \tag{4.3}$$

$$\begin{aligned}
\text{s.t.} \quad & \xi_i \geq 0, && i = 1,\ldots,n_l+n_u. \\
& \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}_i) - \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}) \geq \Delta(\mathbf{y}_i,\mathbf{y}) - \xi_i && i = 1,\ldots,n, \mathbf{y} \in \mathcal{Y}\setminus\mathbf{y}_i \\
\exists \mathbf{z}_i \quad & \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{z}_i) - \mathbf{w}^T\psi(\mathbf{x}_i,\mathbf{y}) \geq \Delta(\mathbf{z}_i,\mathbf{y}) - \xi_i && i = n_l+1,\ldots,n_l+n_u, \mathbf{y} \in \mathcal{Y}\setminus\mathbf{z}_i.
\end{aligned}$$

The label z_i in the last constraint in Equation (4.3) represents the unknown label associated with an unlabeled example x_i. We pursue two approaches that approximate a solution to this problem; the approaches differ in how z_i is obtained. The first approach follows the co-training algorithm, proposed by Brefeld *et al.* [13]. Each view suggests its most compatible label to be used as the "true" label $\mathbf{z}_i$ for the other view. The other view then updates its model based on the proposed label and makes its own suggestion to the first view. The process is repeated until consensus or until some number of iterations. The second approach is a generalization of the transductive structured SVM [98] to multi-view learning. The label $\mathbf{z}_i$ is simply inferred using the current model as $\mathbf{z}_i = \arg\max_{\mathbf{y}} \left(f^{(c)}(\mathbf{x}_i,\mathbf{y}) + f^{(s)}(\mathbf{x}_i,\mathbf{y})\right)$ [98].

4.2 Training and Inference

We have to address several issues associated with training of the proposed SVMs. As before, the size of the output space $\mathcal{Y}$ is exponential in the number of GO terms, and in view of the previous discussion we again focus only on those labels that appear in our training data.

As in the basic GOstruct method, we follow the working set approach [13, 85], where a set of active constraints is maintained, and is grown incrementally by adding the most violated constraint at every iteration. The outer loop of the algorithm iterates over the training examples, both labeled and unlabeled. The inner loop that performs the model update is presented as Algorithm 3. This algorithm addresses training of all the SVM formulations considered here. The inner loop signals whether a new constraint has been added to the working set of a particular training example, and the outer loop terminates when no new

Algorithm 3 Model update for a single example $\mathbf{x}_i$, for which a separate working set is maintained. The algorithm finds the most violated constraint using label $\mathbf{z}$, which is taken to be $\mathbf{y}_i$ for the labeled examples and inferred otherwise. If the new constraint is violated by a larger amount than the constraints already in the working set, it is added to the working set and the dual objective variables are updated using a projection algorithm.

Input: Training example $\mathbf{x}_i$, precision ϵ.
Output: Whether a new constraint has been added.
Define the current working set $\mathcal{W}_i = \{\mathbf{y} | \alpha_{i\mathbf{y}} \neq 0\}$.
if $\mathbf{x}_i$ is labeled **then**
  Define $\mathbf{z} = \mathbf{y}_i$.
else if using co-training **then**
  repeat
    Alternate between each view suggesting $\mathbf{z}$ [13]
  until Consensus is reached or r_{max} iterations.
else if using transduction **then**
  Define $\mathbf{z} = \arg\max_{\mathbf{y}} \left(f^{(c)}(\mathbf{x}_i, \mathbf{y}) + f^{(s)}(\mathbf{x}_i, \mathbf{y}) \right)$
end if
If $\mathbf{z}$ changed since the last iteration, clear the working set.
for each view $v = \{c, s\}$ **do**
  Find the largest margin violation and the associated slacks:
  $\bar{\mathbf{y}} \leftarrow \arg\max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{z}} f^{(v)}(\mathbf{x}_i, \mathbf{y})$
  $\xi_i \leftarrow \max^{+}_{\mathbf{y} \in \mathcal{W}_i} (\Delta(\mathbf{z}, \mathbf{y}) - f^{(v)}(\mathbf{x}_i, \mathbf{z}) + f^{(v)}(\mathbf{x}_i, \mathbf{y})$
  $\bar{\xi} \leftarrow \max \left\{ 0, (\Delta(\mathbf{z}, \bar{\mathbf{y}}) - f^{(v)}(\mathbf{x}_i, \mathbf{z}) + f^{(v)}(\mathbf{x}_i, \bar{\mathbf{y}})) \right\}$
  if $\bar{\xi} > \xi_i + \epsilon$ **then**
    Add the constraint to the working set: $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{\bar{\mathbf{y}}\}$
    Optimize the dual objective over the working set $\mathcal{W}_i$ keeping $\alpha_{j\mathbf{y}}$ fixed for $j \neq i$.
  end if
end for

constraints have been added after a full pass through the training data. Algorithms based around a working set are guaranteed to converge in a polynomial number of steps [13, 85]. In most of our experiments, the number of iterations did not exceed 50.

Algorithm 3 adds a new constraint to the working set only if it is violated by a larger amount than the current largest violation. We maintain a separate working set and a separate set of dual variables $\alpha_{i\mathbf{y}}$ for each view. As before, we optimize the dual objective using a projection method by first finding the optimal solution in an unconstrained space and then projecting it to satisfy the constraints.

The only place in Algorithm 3 where the two views interact is during the inference of the

label $\mathbf{z}$ for unlabeled examples. As mentioned above, we explore two ways of inferring the label $\mathbf{z}$. First is the transductive approach, which simply infers the most compatible label using the current model [98]. The second approach is the co-training algorithm proposed by Brefeld *et al.* [13]. There are several deviations from that algorithm, however. Brefeld *et al.*'s algorithm cross-assigns the labels suggested by each view as "truth" for the peer view. In our experience, after the weights are updated, each view will correctly infer the label suggested to it by the peer view, but those labels are still in disagreement. So, the labels get cross-assigned again and the algorithm continues to alternate between the two states of label assignment, neither of which yields consensus. To get around this problem, we replace cross-assignment with a one-way assignment where the label suggested by the first view is given to the second view and, after the second view updates its weights, we verify that the new inference matches the suggestion. If it doesn't, then the second view suggests its label to the first view and the update is performed analogously.

Another deviation from the original algorithm is in the number of constraints added at every iteration. Brefeld *et al.* proposed to keep adding constraints for a particular training example until all constraints outside of the working set are violated by no more than the constraints in the working set [13]. Instead, we choose to add constraints until a consensus between the two views is reached. Once the consensus label $\mathbf{z}$ is obtained, at most one additional constraint is added by Algorithm 3. Further constraints are not included until the example is revisited again. Our intuition is two-fold: focusing entirely on a single unlabeled example before moving on to the next one is likely to skew the model towards the examples considered earlier; and adding a single violated constraint per iteration is more consistent with how we treat labeled examples and inferences from the transductive SVM, which allows for a cleaner comparison.

The final implementation issue is the order in which the training examples are traversed. We alternate between a full pass through the labeled data and a full pass through the unlabeled data. Interspersing unlabeled data in such a way prevents overfitting of the model to the labeled data and "guides" the model towards a state that better captures the general

structure of the data. Note that it's not viable to completely randomize the order of example traversal, since there's a different number of labeled examples in each view.

4.3 Experimental Setup

In the first set of our experiments we make predictions in a target species using data from that species and data from other species, which we call the external species. As target species we use *S. cerevisiae* and *M. musculus*. As external species for yeast we use *D. melanogaster* and *S. pombe*, and *H. sapiens* is used as an external species for mouse. We choose external species that are reasonably close to the target species and have a significant number of experimentally derived GO annotations. As in the previous chapter, we downloaded annotations from the Gene Ontology website (`http://www.geneontology.org`), and all annotations that were obtained by computational predictions were excluded, limiting the analysis to the following evidence codes: IDA, TAS, IMP, IGI, IPI, IEP, NAS, TC.

4.3.1 Cross-species Data

We used features based on protein sequence to construct the cross-species view. Protein sequences for all species were retrieved from the UniProt database (`http://uniprot.org`). In the cases where a gene has multiple splice forms, the longest one was used. Sequence features were extracted as follows.

BLAST hits We represented a protein in terms of its BLAST similarity scores against a database of annotated proteins [1]. We performed all-vs-all BLAST and the output was post-processed by excluding all hits with e-values above 50.0. The remaining e-values were divided by 50.0 to normalize them. Any values below 1e-10 after normalization were brought up to 1e-10. We then use the negative log of the resulting values as features.

Localization signals Many biological processes are localized to various parts of the cell. Information about protein localization can, therefore, be indicative of the function those proteins perform [67]. For example, proteins that participate in translation of messenger

RNA to amino acids reside in the cell's ribosome. A number of computational methods can extract localization signal from a protein's sequence [39]. Here, we used the coefficients computed by the WoLF PSORT algorithm [40]. WoLF PSORT is an extension to PSORTII, which is a k-nearest neighbor framework for localization signal extraction [39]. The particular extension of WoLF is a feature selection algorithm that weighs PSORT features using a greedy neighborhood search.

Transmembrane protein predictions Many proteins are embedded in one of the membranes within the cell and tend to be associated with certain functions, such as cell adhesion and transport of ions. Transmembrane proteins weave in and out of the membrane, with some parts of the protein being inside of the cell, other parts being outside and yet a third set of parts being in the membrane itself. The composition of amino acids differs across the three parts, which allows one to predict the number of transmembrane domains — i.e., the number of times the protein weaves in and out of the membrane — from the protein's sequence using computational means, such as Hidden Markov Models [50]. For each protein, we estimated the number of transmembrane domains using the TMHMM program [50], and an indicator variable was associated with each number of transmembrane domains.

K-mer composition of N and C termini The two ends of a protein are known as the N and C termini. These contain signals that are important for protein localization, binding and other protein functions [4]. We computed features that represent the 3-mer composition of 10 amino acid segments in the N and C termini of each protein.

Low complexity regions Low-complexity regions in proteins are abundant, have an effect on protein function and are not typically captured by standard sequence comparison methods [20]. We scanned each protein with a sliding window of size 20, and a defined the low-complexity segment as the window that contains the smallest number of distinct amino acids. We used the amino acid composition of that segment as features.

Target Species	*S. cerevisiae*			*M. musculus*		
Namespace	MF	BP	CC	MF	BP	CC
# Target	3401	4332	4115	3150	2633	2125
# External	3917	3000	5000	5000	3000	5000
# GO terms	317	946	308	310	1697	240

Table 4.1: The number of proteins in the target and external species, as well as the number of GO terms considered in each dataset. Namespace designations are as follows: MF - molecular function; BP - biological process; CC - cellular component.

4.3.2 Species-specific Data

We used *S. cerevisiae* and *M. musculus* protein-protein interaction (PPI) data from STRING 8.3 [43] for species-specific information. A protein is represented by a vector of variables, where component i indicates the STRING evidence score of an interaction between protein i and the given protein.

In addition to PPI data, we obtained additional features based on natural language processing (NLP) for the *M. musculus* species. These were protein-protein and protein-GO-term co-occurrences that were extracted from PubMed abstracts with the help of an NLP pipeline developed by our collaborators in Karin Verspoor's group. Protein-protein co-occurrence within a given sentence provides loose evidence that two proteins are related. Similarly, if a protein and a GO term are mentioned in close proximity, this can be evidence that the corresponding function is associated to the protein. The pipeline consists of: 1) splitting of the abstracts into sentences. 2) protein name tagging using the LingPipe named entity recognizer (`http://alias-i.com/lingpipe`); 3) GO term recognition via dictionary lookup and 4) Extraction of term occurrence and within-sentence term co-occurrence counts. The counts are then used as another set of features for our experiments with the *M. musculus* species.

4.3.3 Data Statistics

The data pre-processing steps provided a certain number of target proteins that have features in both views and valid annotations. Five-fold cross-validation is performed on this set of

proteins. Additional proteins, with cross-species features only, were obtained from the external species *D. melanogaster*, *S. pombe* and *H. sapiens*. Table 4.1 provides several statistics about each dataset. In the interest of keeping the run times down, we randomly subsampled the external set down to 5000 proteins for molecular function and cellular component experiments and down to 3000 proteins for biological process experiments.

4.3.4 GOstruct Parameters

Similar to the experiments in the previous chapter, we use linear input- and output-space kernels:

$$\begin{aligned} K_{\mathcal{X}}^{(c)}(\mathbf{x}_1, \mathbf{x}_2) &= \phi^{(c)}(\mathbf{x}_1)^T \phi^{(c)}(\mathbf{x}_2) \\ K_{\mathcal{X}}^{(s)}(\mathbf{x}_1, \mathbf{x}_2) &= \phi^{(s)}(\mathbf{x}_1)^T \phi^{(s)}(\mathbf{x}_2) \\ K_{\mathcal{Y}}(\mathbf{y}_1, \mathbf{y}_2) &= \mathbf{y}_1^T \mathbf{y}_2 - 1, \end{aligned}$$

and compute the joint kernel values using the product kernel from before:

$$K^{(c)}\left((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2)\right) = K_{\mathcal{X}}^{(c)}(\mathbf{x}_1, \mathbf{x}_2) K_{\mathcal{Y}}(\mathbf{y}_1, \mathbf{y}_2) \tag{4.4}$$

for the cross-species view and, similarly, as

$$K^{(s)}\left((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2)\right) = K_{\mathcal{X}}^{(s)}(\mathbf{x}_1, \mathbf{x}_2) K_{\mathcal{Y}}(\mathbf{y}_1, \mathbf{y}_2) \tag{4.5}$$

for the species-specific view.

All kernels were normalized according to

$$K(z_1, z_2) = \frac{K(z_1, z_2)}{\sqrt{K(z_1, z_1) K(z_2, z_2))}}$$

to ensure consistent contribution across different feature spaces. Multiple sets of features were combined via unweighted kernel summation. Kernel loss was used for margin re-scaling and model assessment.

In each target species, classifier performance was estimated using five-fold cross-validation on the proteins that have features in both views; folds were randomly selected such that no

two proteins from different folds have more than 50% sequence identity. To select appropriate values for the parameters C_l and C_u, we ran four-fold cross-validation on the training data in each experiment. The values of $\frac{C_l}{n_l} = 1$ and $\frac{C_u}{n_u} = 0.1$ yielded the highest accuracy on the validation set almost universally.

4.4 Experiment 1: Impact of Cross-Species Information

The first experiment is designed to determine the change in prediction accuracy we obtain by introducing information from other species in the absence of unlabeled data. The multi-view SVM in this case combines the cross-species and species-specific SVMs that are trained separately; both models are used together for inference, as per Equation (4.1). The chain classifier first trains a structured SVM on the cross-species view; it then incorporates the predictions made by this SVM into the input-space feature map for the species-specific view. A second SVM, trained on these predictions combined with the PPI data, is then applied to the test set. As mentioned above, we also consider a variant of the chain classifier that uses a BLAST-nearest-neighbor (BNN) approach to perform the cross-species prediction instead of a structured SVM. We refer to this classifier variant as "BNN-Chain". While our experiments in the previous chapter showed that the structured SVM provided more accurate predictions than the BNN approach, the BNN approach is more scalable to the large datasets that can be used for the cross-species classifier. As a baseline, we trained a single structured-output SVM, which we call joint-SVM, on target species data only, combining the features from both views. Additionally, we trained two single-view SVMs: one using exclusively cross-species information and one using exclusively species-specific features.

The results in Table 4.2 demonstrate the advantage of the multi-view and chain approaches: these classifiers achieve the lowest loss and highest AUC of all the methods, and achieve higher performance than either view by itself. The BNN-chain classifier achieves slightly worse performance than the chain classifier that uses the structured SVM; however, the ability to use this classifier with much larger external species datasets makes it a highly

	Kernel Loss					
Target species	*S. cerevisiae*			*M. musculus*		
Namespace	MF	BP	CC	MF	BP	CC
Cross-Species	0.48	0.55	0.32	0.35	0.60	0.32
Species-Specific	0.44	0.35	0.21	0.38	0.55	0.30
Joint	0.34	0.35	0.20	0.32	0.54	0.28
Multi-view	0.34	0.34	0.22	0.30	0.53	0.27
Chain	0.33	0.34	0.20	0.31	0.55	0.27
BNN-Chain	0.33	0.35	0.20	0.32	0.55	0.28
	AUC					
Target species	*S. cerevisiae*			*M. musculus*		
Namespace	MF	BP	CC	MF	BP	CC
Cross-Species	0.87	0.79	0.78	0.89	0.67	0.80
Species-Specific	0.90	0.94	0.94	0.83	0.81	0.84
Joint	0.94	0.94	0.95	0.88	0.80	0.85
Multi-view	0.95	0.94	0.94	0.90	0.79	0.88
Chain	0.94	0.94	0.95	0.90	0.82	0.87
BNN-Chain	0.94	0.94	0.95	0.89	0.82	0.87

Table 4.2: Classifier performance in predicting GO terms, quantified by mean loss per example (top) and mean AUC per GO term (bottom) when no unlabeled data is used. Lower loss values and higher AUC values are better. The results were obtained via five-fold cross-validation on all proteins from the target species. Multi-view and cross-species SVMs were also provided with the training examples from external species.

viable approach. The species-specific classifier performs better in yeast than in mouse. For the cross-species classifier we see the opposite effect, with the mouse classifier exhibiting better accuracy than the yeast classifier. The accuracy of the cross-species view has to do with how well-annotated are closely related species (*S. pombe* in the case of *S. cerevisiae*, and *H. sapiens* in the case of *M. musculus*). Furthermore, the species-specific SVM outperformed the cross-species SVM in biological process and cellular component namespaces. This is expected since protein-protein interactions indicate that the two proteins participate in the same biological process in the same place in the cell.

We observe that in nearly all of the experiments, the performance values for the biological process namespace tends to lag behind molecular function and cellular component. We believe this is because the prediction of biological process is a harder problem, which is in part due to the larger number of GO terms in the namespace (Table 4.1).

To further investigate the interplay between cross-species and species-specific information, we ran additional experiments, reducing the number of target-species proteins while keeping the set of external proteins fixed. We present the results of these experiments on molecular function in *S. cerevisiae* in Table 4.3. As expected, the cross-species information becomes more important as the number of training examples in the target species decreases. In particular, the cross-species SVM outperforms both the species-specific and the joint SVMs when the number of *S. cerevisiae* proteins is 500, a scenario that more closely simulates annotating a newly-sequenced genome. We note that, in all cases, the multi-view and chain classifiers outperform all other methods, demonstrating their robustness in combining cross-species and species-specific information. We further observe that as the cross-species features become more relevant, proper utilization of those features becomes important; this is signified by the BNN-based chain classifier degrading in performance faster than the SVM-based chain classifier.

	Kernel Loss			
	# Training Samples			
Classifier	2720	1500	1000	500
Cross-Species	0.48	0.49	0.51	0.51
Species-Specific	0.44	0.48	0.52	0.56
Joint	0.34	0.40	0.44	0.52
Multi-view	0.34	0.37	0.40	0.43
Chain	0.33	0.36	0.39	0.45
BNN-Chain	0.33	0.38	0.41	0.48
	AUC			
	# Training Samples			
Classifier	2720	1500	1000	500
Cross-Species	0.87	0.86	0.84	0.84
Species-Specific	0.90	0.87	0.85	0.80
Joint	0.94	0.91	0.89	0.83
Multi-view	0.95	0.94	0.92	0.90
Chain	0.94	0.92	0.91	0.88
BNN-Chain	0.94	0.93	0.91	0.87

Table 4.3: Classifier performance in predicting molecular function GO terms, quantified by mean loss per example (top) and mean AUC per GO term (bottom) when no unlabeled data is used. Lower loss values and higher AUC values are better. The number of training examples refers to *S. cerevisiae* proteins that are represented in both views. Multi-view and Cross-Species SVMs were provided the additional 3917 proteins that only have BLAST features.

Cross-species dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0016773	140	0.96	0.87	phosphotransferase activity, alcohol group as acceptor
GO:0004672	97	0.99	0.90	protein kinase activity
GO:0016301	167	0.95	0.86	kinase activity
GO:0042578	93	0.93	0.87	phosphoric ester hydrolase activity
GO:0016772	247	0.91	0.87	transferase activity, phosphorus-containing groups
GO:0016787	621	0.86	0.82	hydrolase activity
GO:0016740	576	0.88	0.85	transferase activity
GO:0016462	247	0.87	0.85	pyrophosphatase activity
Species-specific dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0003702	121	0.85	0.94	RNA polymerase II transcription factor activity
GO:0016746	93	0.83	0.90	transferase activity, transferring acyl groups
GO:0005515	511	0.75	0.82	protein binding
GO:0005198	321	0.89	0.96	structural molecule activity
GO:0022890	102	0.93	0.98	inorganic cation transmembrane transporter activity
GO:0030528	293	0.88	0.93	transcription regulator activity
GO:0008324	113	0.93	0.97	cation transmembrane transporter activity
GO:0016879	106	0.88	0.92	ligase activity, forming carbon-nitrogen bonds

Table 4.4: A comparison of the cross-species and species-specific SVMs across general GO terms. For each classifier, we present eight GO terms for which that classifier outperformed the other by the largest margin. The second column displays the number of proteins in the dataset annotated with each GO term. The third and fourth columns display the corresponding AUC scores.

4.4.1 Performance comparison on individual GO terms

For further analysis of performance we examined the above classifiers in the context of individual GO terms. The analysis presented here is on the molecular function namespace for the *S. cerevisiae* species; we note that the results are qualitatively similar in *M. musculus*. We refer to a GO term as "general" if more than 90 examples in the dataset are annotated with it; otherwise, we refer to it as "specific". We picked this threshold by analyzing the distribution of all 317 GO terms across all 3401 *S. cerevisiae* examples and identifying a large break in the distribution.

The comparison of cross-species and species-specific classifiers on general GO terms is presented in Table 4.4. GO terms for which the cross-species classifier achieved better performance primarily correspond to enzyme activity. This is attributed to the fact that en-

Cross-species dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0003746	10	0.97	0.60	translation elongation factor activity
GO:0032561	13	0.86	0.63	guanyl ribonucleotide binding
GO:0005525	13	0.86	0.63	GTP binding
GO:0042803	10	0.84	0.62	protein homodimerization activity
GO:0019001	14	0.87	0.65	guanyl nucleotide binding
GO:0003724	31	0.97	0.80	RNA helicase activity
GO:0016651	12	0.95	0.79	oxidoreductase activity, acting on NADH or NADPH
GO:0042802	10	0.78	0.63	identical protein binding
Species-specific dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0000384	12	0.51	0.99	first spliceosomal transesterification activity
GO:0000385	18	0.51	0.99	spliceosomal catalysis
GO:0000386	12	0.59	0.99	second spliceosomal transesterification activity
GO:0003684	11	0.59	0.96	damaged DNA binding
GO:0042054	10	0.62	0.99	histone methyltransferase activity
GO:0018024	10	0.63	0.99	histone-lysine N-methyltransferase activity
GO:0030515	16	0.70	0.99	snoRNA binding
GO:0019213	24	0.71	0.99	deacetylase activity

Table 4.5: A comparison of the cross-species and species-specific SVMs across specific GO terms. The columns present the same type of information as those in Table 4.4.

zymes (proteins that catalyze chemical reactions) can be readily identified from sequence motifs [24, 8], and the cross-species view works with sequence-based features. The species-specific SVM, on the other hand, achieves better performance on a wider array of GO categories ranging from binding and transport to regulation and even some enzyme activity as well.

The same analysis is presented in Table 4.5 for the specific GO terms. One concern when working with specific GO terms is their representation across the fold split. In the extreme case, when all examples with a particular GO term are assigned to a single fold, the classifier has no training data for that GO term when that fold is chosen as the test fold. After analyzing the distribution of GO terms across the dataset, we noted that this was not the case; we observed that the distribution was fairly even and in all cases, the species-specific classifier had access to at least half of examples annotated with any specific GO term. We believe this indicates that the performance differences in Table 4.5 have more to do with each SVM being able to learn better from its features for a particular set of GO terms.

The cross-species classifier achieves better performance on a number of specific GO terms.

Multi-view dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0042803	10	0.79	0.71	protein homodimerization activity
GO:0019200	14	0.98	0.90	carbohydrate kinase activity
GO:0016780	14	0.98	0.91	phosphotransferase activity, substituted phosphate groups
GO:0016627	19	0.96	0.89	oxidoreductase activity, CH-CH group of donors
GO:0008374	13	0.97	0.91	O-acyltransferase activity
GO:0043169	26	0.87	0.81	cation binding
GO:0031267	10	0.88	0.83	small GTPase binding
GO:0017016	10	0.88	0.83	Ras GTPase binding
Chain dominant				
GO ID	n	Cross-sp.	Sp.-spec.	GO term
GO:0000166	38	0.66	0.74	nucleotide binding
GO:0051087	12	0.86	0.93	chaperone binding
GO:0043130	25	0.89	0.96	ubiquitin binding
GO:0032182	28	0.89	0.96	small conjugating protein binding
GO:0017076	37	0.77	0.83	purine nucleotide binding
GO:0003684	11	0.91	0.97	damaged DNA binding
GO:0030554	24	0.80	0.86	adenyl nucleotide binding
GO:0001671	11	0.85	0.90	ATPase activator activity

Table 4.6: A comparison of the multi-view and chain classifiers across specific GO terms. For each classifier, we present eight GO terms for which that classifier outperformed the other by the largest margin. The second column displays the number of proteins in the dataset annotated with each GO term. The third and fourth columns display the corresponding AUC scores.

However, there appears to be no common theme among the terms. The three top GO terms for which the species-specific classifier does better, on the other hand, are all related to spliceosomal activity. Spliceosome is a complex of proteins working together to perform splicing (removal of introns from pre-mRNA) and the corresponding proteins can be easily identified through protein-protein interaction data used by the species-specific classifier. Indeed, all 12 proteins annotated with GO:0000384 have very strong evidence of interaction with each other.

In addition to the two single-view classifiers, we also looked at the performance of the multi-view framework. The multi-view framework did better than the cross-species SVM in 286 out of 317 GO terms and outperformed the species-specific SVM in 280 out of 317 GO terms. In both cases, the multi-view framework achieved better performance across all general GO terms and, in specific GO term cases where it got outpeformed, the difference in the AUC scores was never above 0.06.

We also compared the multi-view and chain classifiers. The difference in performance of the two frameworks on general GO terms was negligible. Performance comparison on specific GO terms is presented in Table 4.6. Similar to the cross-species classifier in Table 4.4, we observe a lot of enzyme activity associated with the multi-view framework. We attribute this to the fact that multi-view classifier works with the sequence data directly, while the chain framework effectively converts the sequence data to cross-species view predictions that are then used as features in the final classifier stage. The chain classifier achieves better performance on GO terms associated with a number of binding functions, which is something that can be identified in the protein-protein iteraction data; we conjecture that because the sequence-based features become abstracted in the chain framework, the classifier places more emphasis on learning from the PPI data.

4.5 Experiment 2: Impact of Unlabeled Data

The second set of experiments is designed to measure the impact of unlabeled data. We ran five-fold cross-validation using the same test data in every fold as above. The *S. cerevisiae* training data for every experiment was split into labeled and unlabeled examples. Similar to experiments in the previous section, we include all labeled proteins that only have feature representation in the cross-species view.

As shown in Table 4.7 the addition of unlabeled data had negative impact on classifier performance. The co-training approach appears to be affected less severely, but both semi-supervised algorithms fail to learn from unlabeled data. We conjecture that this is due to the sparsity with which the joint input-output space is characterized by the labeled examples.

4.6 Experiment 3: CAFA challenge

We entered the multi-view *GOstruct* framework into the CAFA challenge (`http://biofunctionprediction.org/`). The challenge involved making molecular function and biological process predictions for a number of target proteins in seven eukaryote species: *A. thaliana*, *D. discoideum*, *H. sapiens*, *M. musculus*, *R. norvegicus*, *S. cerevisiae* and *X. laevis*. The targets were made

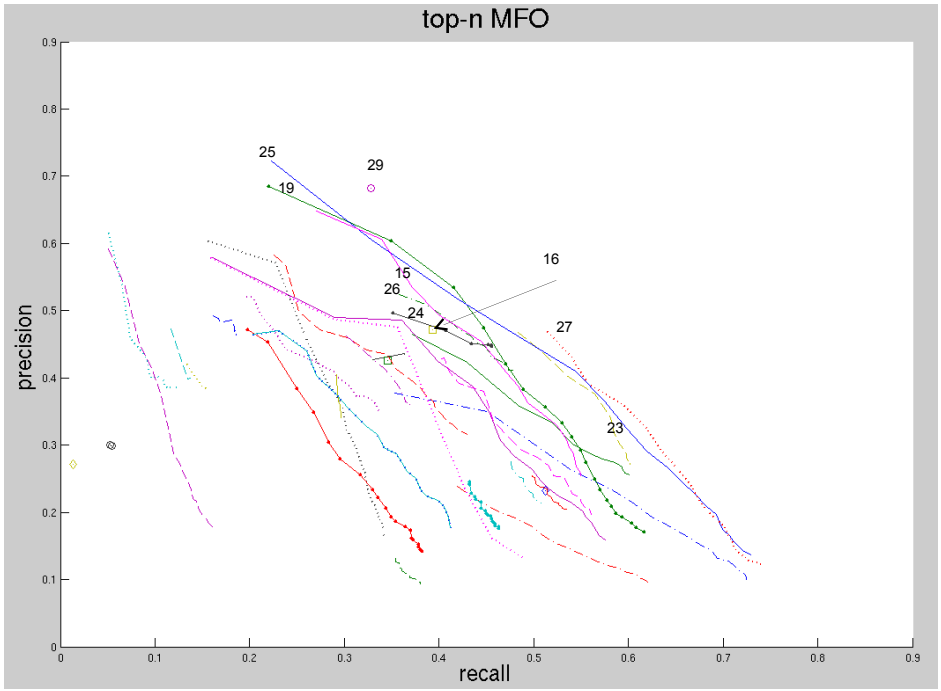


Figure 4.2: A figure presented at the Automatic Function Prediction special interest group meeting at ISMB2011 detailing the performance of classifiers used in the CAFA challenge. The precision and recall values were computed for the molecular function namespace using the top n GO terms retrieved for each test protein. *GOstruct* is identified with the label "29".

# examples		Loss		AUC	
# Lbld	# Ulbld	CO-tr.	Trans.	CO-tr.	Trans.
500	0	0.43	0.43	0.90	0.90
500	500	0.50	0.62	0.86	0.77
500	1000	0.65	0.66	0.75	0.74
500	1500	0.67	0.77	0.75	0.66
500	2000	0.71	0.77	0.72	0.63
1000	0	0.40	0.40	0.92	0.92
1000	500	0.42	0.44	0.91	0.89
1000	1000	0.43	0.57	0.91	0.81
1000	1500	0.42	0.62	0.91	0.77

Table 4.7: Mean loss per example for co-training and transductive SVMs computed for various numbers of labeled and unlabeled *S. cerevisiae* training examples. The number of non *cerevisiae* proteins was the same in all cases. The test data used in these experiments was identical to that used in Table 4.2.

available to the contestants in September, 2010, at which point they had no experimentally verified functional annotations. Over the course of the next few months, experimental annotations became available for around 700 of the target proteins (primarily human and mouse), which allowed the organizers to evaluate submitted predictions.

The assessment results were presented at the Automatic Function Prediction special interest group meeting of ISMB in July, 2011. One of the figures presented at the meeting is displayed here as Figure 4.2. For privacy reasons, the identity of all algorithms was masked through the use of numeric identifiers, with the label "29" corresponding to *GOstruct*. We note that because the predictions we submitted were in the form of binary vectors, the performance of the algorithm is captured by a single point on the precision-vs.-recall plot. Figure 4.2 demonstrates that *GOstruct* achieves higher level of precision for a fixed value of recall than all other participants in the molecular function namespace. The performance of *GOstruct* was closer to the median in the biological process namespace. We note that high precision is desirable for biologists who may want to experimentally verify a small number of GO terms returned for a query protein.

The CAFA challenge provided the target proteins but the training data was up to each participant. We retrieved protein sequences from UniProt and trained three cross-species

Predictions for	Cross-sp. view	Sp.-specific view
A. thaliana	*A. thaliana*, *S. cerevisiae*, *D. melanogaster*, *S. pombe*	PPI
D. discoideum	*A. thaliana*, *S. cerevisiae*, *D. melanogaster*, *S. pombe*	PPI
H. sapiens	*H. sapiens*	PPI
M. musculus	*M. musculus*, *R. norvegicus*	PPI, NLP
R. norvegicus	*M. musculus*, *R. norvegicus*	-
S. cerevisiae	*A. thaliana*, *S. cerevisiae*, *D. melanogaster*, *S. pombe*	PPI
X. laevis	*A. thaliana*, *S. cerevisiae*, *D. melanogaster*, *S. pombe*	-

Table 4.8: The breakdown of information employed by each model to make predictions for CAFA targets. The cross-species view used all of the sequence-based features described in the text.

	Loss			AUC		
Species	cross-sp.	sp.-specific	m.-view	cross-sp.	sp.-specific	m.-view
A. thaliana	0.36	0.46	**0.34**	0.92	0.87	**0.94**
D. discoideum	0.41	0.48	**0.39**	0.79	0.77	**0.84**
H. sapiens	0.37	0.41	**0.34**	0.83	0.83	**0.87**
M. musculus	0.34	0.40	**0.32**	0.84	0.83	**0.87**
S. cerevisiae	0.46	0.44	**0.33**	0.85	0.92	**0.94**

Table 4.9: Cross-validation results on the training data for five of the CAFA target species, for which species-specific features were available. Presented are mean loss per example and mean AUC per GO term.

SVMs using the sequence-based features described earlier in this chapter. We present the overview of which models were applied to make predictions for which species in Table 4.8. Note that the multi-view framework allowed us to incorporate information from well-annotated organisms that were not part of the challenge (*D. melanogaster* and *S. pombe*). In addition to the three cross-species models, we trained species-specific SVMs for five of the target species using protein-protein interaction data from the STRING database; *R. norvegicus* and *X. laevis* did not have species-specific models due to data availability and issues with protein ID matching. The *M. musculus* model also included NLP-based features.

In addition to submitting predictions for the CAFA challenge, we ran cross-validation on the training data to estimate the impact of combining the information from both views. The results of this cross-validation are presented in Table 4.9. We focused on the five target species for which we had species-specific features, limiting the cross-species SVMs to the same

set of proteins in each case. As expected, the multi-view framework provided a performance boost for all species, as signified by lower loss and higher AUC values in Table 4.9. The multi-view results for *S. cerevisiae* and *M. musculus* are on par with our previously obtained values in Table 4.2.

Chapter 5

Conclusion

This work presented *GOstruct*, a structured-output framework for automatic function prediction. We demonstrated that by predicting the full annotation directly, *GOstruct* outperformed both the traditional transfer-of-annotation methods as well as algorithms that break the problem up into a collection of binary classification tasks in two critical assessment challenges: Mousefunc and CAFA. Furthermore, we showed how multiple disparate sources of data across multiple species can be combined via the use of views. Our analysis showed that sequence-based features are highly predictive of enzyme function, while protein-protein interaction data is able to identify genes that participate in the same biological process and are localized to the same cellular component; we demonstrated that the multi-view framework was able to effectively combine both predictors, achieving better performance than either classifier.

5.1 Open problems

A number of issues remain to be resolved. In our experience, some data sources, such as gene expression, were not contributing to increased classifier accuracy and in some cases were even degrading the performance. However, a more sophisticated kernel could make better use of such features and further analysis of kernels used by the framework is required. Similarly, our semi-supervised learning results indicated that unlabeled data led to degraded performance, an issue that could potentially be resolved through a different choice of kernels.

A related issue is model interpretability. The current SVM models produced by the framework are sparse with respect to example/label pairs, but not features. While this allows us to identify important proteins and annotations in the dataset, we are currently unable to specify what features of those protein are responsible for the predictions. One of the future directions is to incorporate feature selection methods, such as filter and embedded methods used for binary classification [33], into the framework.

The final issue related to the framework is scalability. The running time of the working set training algorithm is quadratic in the number of training examples, which presents a problem for the ever-growing datasets. While we were able to partially address the issue by replacing the cross-species SVM with the BLAST nearest-neighbor classifier in our chain algorithm, more efficient training algorithms are required for structured SVMs to be applied to larger species-specific datasets.

In general, automatic function prediction is still an ongoing area of research. While designing more accurate and efficient algorithms is important, there is an orthogonal issue of critical assessment. Future iterations of the CAFA challege will help with identifying the current state-of-the-art as well as the bottlenecks in automatic function predictions. Similarly, The HumanFunc Challenge (`http://func.mshri.on.ca/human/challenge`) currently being developed will allow critical assessment of algorithms on the human genome, similar to what Mousefunc achieved with the mouse.

Appendix

In this chapter, we present the projection algorithm used by Algorithm 2 to solve the opimization problem in Equations (3.15)-(3.17) over the dual variables $\alpha_{i\mathbf{y}}$ that correspond to the constraints in the working set $\mathcal{W}_i$. For simplicity, we can rewrite the problem in its vector form

$$\max_\alpha -\tfrac{1}{2}\alpha^T J\alpha + b^T\alpha \tag{5.1}$$

$$\text{s.t.} \qquad \mathbf{1}^T\alpha \leq C/n, \tag{5.2}$$

$$\alpha \geq \mathbf{0} \tag{5.3}$$

where α is the vector of variables in the working set $\mathcal{W}_i$, J is the associated matrix of $[\delta\psi(\cdot)]^T\,\delta\psi(\cdot)$ values, $b^T\alpha$ is the linear term obtained from the product of dual variables in α with all other fixed variables, and $\mathbf{1}$ and $\mathbf{0}$ are vectors of all ones and zeros, respectively. The projection method first finds the optimal solution in the unconstrained space and then projects it to the subspace defined by the constraints. The unconstrained maximum $\alpha^{(u)}$ occurs where the objective derivative is equal to zero:

$$-J\alpha^{(u)} + b = \mathbf{0}. \tag{5.4}$$

To find the direction of projection we use the fact that the derivative will be perpendicular to the active constraints. Any vector $c\mathbf{1}$, where c is a scalar, will be perpendicular to the constraint in Equation (5.2). Thus, we would like to offset the unconstrained solution $\alpha^{(u)}$ along a direction $\mathbf{v}$, such that the derivative maintains orthogonality. If t is the amount of offset, then the derivative at the resulting point is given by $-J(\alpha^{(u)} + t\mathbf{v}) + b = (-J\alpha^{(u)} + b) - tJ\mathbf{v} = -tJ\mathbf{v}$. From here, it is easy to see that offsetting along the vector $\mathbf{v}$ that satisfies

$J\mathbf{v} = \mathbf{1}$ will always yield a point where the derivative is along the direction $\mathbf{1}$ (i.e., is of the form $c\mathbf{1}$). We then determine the amount of offset t by solving $\mathbf{1}^T(\alpha^{(u)} + t\mathbf{v}) = C/n$.

Similar computations result in projection directions that yield orthogonality with constraints in Equation (5.3). The matrix J is positive semi-definite and all linear systems can be solved simultaneously by performing Cholesky decomposition on J. In our experience, the projection method converges faster than the SMO-like algorithms.

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

[3] E.C. Baechler, F.M. Batliwalla, G. Karypis, P.M. Gaffney, W.A. Ortmann, K.J. Espe, K.B. Shark, W.J. Grande, K.M. Hughes, V. Kapur, et al. Interferon-inducible gene expression signature in peripheral blood cells of patients with severe lupus. *Proceedings of the National Academy of Sciences of the United States of America*, 100(5):2610, 2003.

[4] I. Bahir and M. Linial. Functional grouping based on signatures in protein termini. *Proteins: Structure, Function, and Bioinformatics*, 63(4):996–1004, 2006.

[5] G. Bakir, T. Hofmann, and B. Schölkopf. *Predicting structured data*. The MIT Press, 2007.

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

[7] A. Bateman, L. Coin, R. Durbin, R.D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E.L.L. Sonnhammer, et al. The pfam protein families database. *Nucleic acids research*, 32(suppl 1):D138, 2004.

[8] 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.

[9] A. Ben-Hur, C.S. Ong, S. Sonnenburg, B. Scholkopf, and G. Ratsch. Support vector machines and kernels for computational biology. *PLoS Computational Biology*, 4(10), 2008.

[10] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, page 100. ACM, 1998.

[11] B. Boeckmann, A. Bairoch, R. Apweiler, M.C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, et al. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic acids research*, 31(1):365, 2003.

[12] P. Bork and E. V. Koonin. Predicting functions from protein sequences - where are the bottlenecks? *Nature Genetics*, 18:313–318, 1998.

[13] U. Brefeld and T. Scheffer. Semi-supervised learning for structured output variables. In *Proceedings of the 23rd international conference on Machine learning*, pages 145–152. ACM, 2006.

[14] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees. Wadsworth*. Belmont, CA; Wadsworth International Group, 1984.

[15] K.R. Brown and I. Jurisica. Online predicted human interaction database. *Bioinformatics*, 21(9):2076, 2005.

[16] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[17] 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.

[18] C.M. Christoudias, R. Urtasun, and T. Darrell. Multi-view learning in the presence of view disagreement. In *UAI*, page 5, 2008.

[19] W.T. Clark and P. Radivojac. Analysis of protein function and its prediction from amino acid sequence. *Proteins: Structure, Function, and Bioinformatics*, 2011.

[20] A. Coletta, J.W. Pinney, D.Y.W. Solís, J. Marsh, S.R. Pettifer, and T.K. Attwood. Low-complexity regions within protein sequences have position-dependent roles. *BMC systems biology*, 4(1):43, 2010.

[21] 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.

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

[23] D. Devos and A. Valencia. Practical limits of function prediction. *PROTEINS-NEW YORK-*, 41(1):98–107, 2000.

[24] K. Dhwani et al. Modenza: Accurate identification of metabolic enzymes using function specific profile hmms with optimised discrimination threshold and modified emission probabilities. *Advances in Bioinformatics*, 2011, 2011.

[25] J.A. Eisen. Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis. *Genome research*, 8(3):163, 1998.

[26] B.E. Engelhardt, M.I. Jordan, K.E. Muratore, and S.E. Brenner. Protein molecular function prediction by Bayesian phylogenomics. *PLoS computational biology*, 1(5):e45, 2005.

[27] R.E. Fan, P.H. Chen, and C.J. Lin. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

[28] R. Fletcher. *Practical methods of optimization*. John Wiley and Sons, Inc., 1987.

[29] M. Y. Galperin and E. V. Koonin. Sources of systematic error in functional annotation of genomes: domain rearrangement, non-orthologous gene displacement, and operon disruption. *In silico Biology*, 1:55–67, 1998.

[30] K. Ganchev, J. Graca, J. Blitzer, and B. Taskar. Multi-view learning over structured and non-identical outputs. In *Proceedings of The 24th Conference on Uncertainty in Artificial Intelligence*. Citeseer, 2008.

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

[32] 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.

[33] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[34] T. Hamp, R. Kassner, S. Seemayer, E. Vicedo, et al. Nearest-neighbor approaches to predict protein function by homology inference alone. In *Automatic Function Prediction special interest group meeting at ISMB*, 2011.

[35] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[36] B. Hayete and J.R. Bienkowska. Gotrees: predicting go associations from protein domain composition using decision trees. In *Pac Symp Biocomput*, volume 10, pages 127–138, 2005.

[37] S. Hennig, D. Groth, and H. Lehrach. Automated gene ontology annotation for anonymous sequence data. *Nucleic Acids Research*, 31(13):3712, 2003.

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

[39] P. Horton and K. Nakai. Better prediction of protein cellular localization sites with the k nearest neighbors classifier. In *Proceeding of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 147–152, 1997.

[40] P. Horton, K.J. Park, T. Obayashi, and K. Nakai. Protein subcellular localization prediction with WoLF PSORT. In *Proceedings of the 4th annual Asia Pacific bioinformatics conference APBC06, Taipei, Taiwan*, volume 39, page 48. Citeseer, 2006.

[41] S. Hunter, R. Apweiler, T.K. Attwood, A. Bairoch, A. Bateman, D. Binns, P. Bork, U. Das, L. Daugherty, L. Duquenne, et al. Interpro: the integrative protein signature database. *Nucleic acids research*, 37(suppl 1):D211, 2009.

[42] R. A. Irizarry, B. M. Bolstad, Francois Collin, Leslie M. Cope, Bridget Hobbs, and Terence P. Speed. Summaries of affymetrix genechip probe level data. *Nucleic Acids Research*, 31(4), 2003.

[43] L.J. Jensen, M. Kuhn, M. Stark, S. Chaffron, C. Creevey, J. Muller, T. Doerks, P. Julien, A. Roth, M. Simonovic, et al. STRING 8.a global view on proteins and their functional interactions in 630 organisms. *Nucleic acids research*, 37(suppl 1):D412, 2009.

[44] T. Joachims. Making large-scale svm learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, 1999.

[45] U. Karaoz, TM Murali, S. Letovsky, Y. Zheng, C. Ding, C.R. Cantor, and S. Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2888, 2004.

[46] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 2001.

[47] 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.

[48] S. Kiritchenko, S. 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.

[49] R.I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 315–322, 2002.

[50] A. Krogh, B.È. Larsson, G. Von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden markov model: application to complete genomes1. *Journal of molecular biology*, 305(3):567–580, 2001.

[51] 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.

[52] 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.

[53] B. Long, P.S. Yu, and Z.M. Zhang. A general model for multiple view unsupervised learning. In *Proceedings of the 8th SIAM International Conference on Data Mining (SDM'08), Atlanta, Georgia, USA*. Citeseer, 2008.

[54] D. Martin, M. Berriman, and G. Barton. Gotcha: a new method for prediction of protein function assessed by the annotation of seven genomes. *BMC bioinformatics*, 5(1):178, 2004.

[55] J. McDermott, R. Bumgarner, and R. Samudrala. Functional annotation from predicted protein interaction networks. *Bioinformatics*, 21(15):3217, 2005.

[56] S. Mostafavi and Q. Morris. Using the Gene Ontology hierarchy when predicting gene function. In *Conference on Uncertainty in Artificial Intelligence*, 2009.

[57] 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.

[58] J. Moult, K. Fidelis, A. Kryshtafovych, B. Rost, and A. Tramontano. Critical assessment of methods of protein structure prediction - Round VIII. *Proteins: Structure, Function, and Bioinformatics*, 77(S9):1–4, 2009.

[59] J. Moult, J.T. Pedersen, R. Judson, and K. Fidelis. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 23(3):ii–iv, 1995.

[60] S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[61] E. Nabieva, K. Jim, A. Agarwal, B. Chazelle, and M. Singh. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21(suppl 1):i302, 2005.

[62] 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.

[63] D. Pal and D. Eisenberg. Inference of protein function from protein structure. *Structure*, 13(1):121–130, 2005.

[64] 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.

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

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

[67] B. Rost, J. Liu, R. Nair, K.O. Wrzeszczynski, and Y. Ofran. Automatic prediction of protein function. *Cellular and Molecular Life Sciences*, 60(12):2637–2650, 2003.

[68] 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.

[69] H. Saigo, J.P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682, 2004.

[70] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467, 1995.

[71] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski. Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC bioinformatics*, 11(1):2, 2010.

[72] 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.

[73] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge Univ Pr, 2004.

[74] D. Smedley, S. Haider, B. Ballester, R. Holland, D. London, G. Thorisson, and A. Kasprzyk. Biomart–biological queries made easy. *BMC genomics*, 10(1):22, 2009.

[75] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. Molecular Biology*, 147:195–7, 1981.

[76] 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*, 2009.

[77] A. Sokolov and A. Ben-Hur. Hierarchical classification of Gene Ontology terms using the GOstruct method. *Journal of Bioinformatics and Compuational Biology*, 8(2):357–376, 2010.

[78] A. Sokolov and A. Ben-Hur. Multi-view prediction of protein function. In *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2011.

[79] A. Sokolov, K. Graim, C. Funk, K. Verspoor, and A. Ben-Hur. Combining heterogeneous data sources for protein function prediction. In *Automatic Function Prediction special interest group meeting at ISMB*, 2011.

[80] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *Nucleic acids research*, 34(Database Issue):D535, 2006.

[81] A.I. Su, T. Wiltshire, S. Batalov, H. Lapp, K.A. Ching, D. Block, J. Zhang, R. Soden, M. Hayakawa, G. Kreiman, et al. A gene atlas of the mouse and human protein-encoding transcriptomes. *Proceedings of the National Academy of Sciences of the United States of America*, 101(16):6062, 2004.

[82] 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.

[83] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS*, 2003.

[84] 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.

[85] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

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

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

[88] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000.

[89] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction in protein-protein interaction networks. *Nature Biotechnology*, 21(6):697–700, 2003.

[90] 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.

[91] Claire L. Wilson and Crispin J. Miller. Simpleaffy: a bioconductor package for affymetrix quality control and data analysis. *Bioinformatics*, 21:3683–3685, 2005.

[92] Aaron K. Wong, Christopher Y. Park, Casey S. Greene, Yuanfang Guan, and Olga G. Troyanskaya. Predicting gene function through homology-driven functional genomics. In *Automatic Function Prediction special interest group meeting at ISMB*, 2011.

[93] G. Xiao and W. Pan. Gene function prediction by a combined analysis of gene expression data and protein-protein interaction data. *Journal of bioinformatics and computational biology*, 3(6):1371, 2005.

[94] G. Zehetner. Ontoblast function: From sequence similarities directly to potential functional annotations by ontology terms. *Nucleic acids research*, 31(13):3799, 2003.

[95] W. Zhang, Q. Morris, R. Chang, O. Shai, M. Bakowski, N. Mitsakakis, N. Mohammad, M. Robinson, R. Zirngibl, E. Somogyi, et al. The functional landscape of mouse gene expression. *Journal of biology*, 3(5):21, 2004.

[96] D. Zhou, Bousuet O., Lal T., Weston J., and Schoelkopf B. Learning with local and global consistency. In *Neural Information Processing Systems*, 2003.

[97] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919, 2003.

[98] A. Zien, U. Brefeld, and T. Scheffer. Transductive support vector machines for structured variables. In *Proceedings of the 24th international conference on Machine learning*, page 1190. ACM, 2007.