University of Alberta

HIERARCHICAL PREDICTION OF PROTEIN FUNCTION IN THE GENE ONTOLOGY USING GRAPHICAL MODELS

by

Peng Wang 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

in

Department of Computing Science

Edmonton, Alberta
Spring 2008

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-45905-8
Our file Notre référence
ISBN: 978-0-494-45905-8

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

High-throughput functional annotation of proteins is a fundamental task in functional proteomics. Protein functions are typically organized in the form of a general-specific hierarchy, such as the Gene Ontology (GO), which describes when one functional class is a specialization of its parent class. The hierarchical structure indicates that if a protein belongs to one class then it also belongs to all ancestor classes up to the root. Most previous work on protein function prediction has constructed independent classifiers for each function, which ignore the hierarchical information available in the GO. We develop a framework for combining the local independent SVM predictions with graphical models, both Bayesian networks (BNs) and Conditional Random Fields (CRFs), which are built upon the hierarchical structure in the GO. Our goal is to increase the overall predictive accuracy by exploiting this hierarchical information. Compared to the baseline technique (i.e. independent SVM classifiers), our techniques using BN and CRF yield significant improvement on two large data sets constructed from the Uniprot database.

Acknowledgements

I would like to thank my supervisors Russell Greiner and Duane Szafron for their great support, indispensable guidance, and helpful comments. I also thank the entire Proteome Analyst research group for their support and great times at the bioinformatics lab.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Thanks to rapid advancement in genome sequencing technology, a large quantity and variety of genomic and proteomic information has recently become available. The ever-increasing flood of diverse biological data from high-throughput processes, such as genomic and proteomic sequencing, and gene expression, can be used to study the characteristics and interactions of cellular components. It also pushes the elucidation of protein function to the center stage in computational biology. However, this remarkable speed of discovery has made it impossible to experimentally determine the function for most new proteins, and made it difficult to keep up with the influx of data produced by human-curated annotation. Thus, scientists have been turning to sophisticated computational approaches for annotating the huge amount of proteomic sequence data being produced. Since the protein's sequence characterizes its function, it is essential to design effective computational approaches to predict the protein functions based on protein sequence.

To better characterize protein functions, biologists have defined a hierarchy of protein functions, i.e. the Gene Ontology (GO). Previously, a number of approaches had been proposed to automatically predicting function using the GO hierarchy. Some of them [1, 16, 40, 59] treat the hierarchy as a flat ontology in which the functional classes have are not interrelated, while others utilize some of the hierarchical information encoded in the GO (only the parent-child dependencies [9, 30, 43, 55, 62] or only used to construct data sets [29]). Since protein functions are naturally organized as a hierarchy, the hierarchical structure presented by the GO should be exploited when computational function prediction is performed.

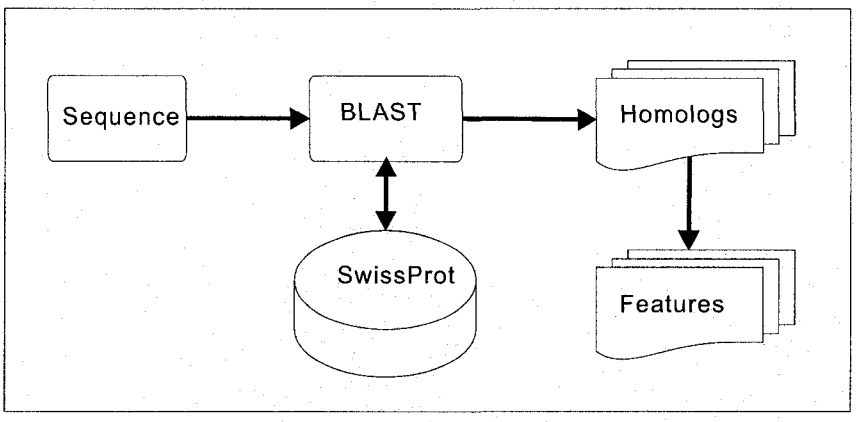


Figure 1.1: Feature extraction using the PA tool.

From a machine learning perspective, hierarchical protein function prediction is a task in which one tries to predict the labels in a structured graph for unknown instances given some observed evidence. This is a typical example of a problem that can be solved using graphical models. In fact, the hierarchical function prediction problem is similar to other prediction tasks with particular structures, such as information extraction from webpages with hyperlinks or image labeling on grids of pixels, which have been extensively studied by using graphical models, such as Bayesian networks (BNs) or Conditional Random Fields (CRFs). Graphical models prove to be one of the most effective tools for this kind of problems.

In this dissertation, we develop a framework, based upon CHUGO (Classification in a Hierarchy Under Gene Ontology), proposed by Eisner et al. [29], to predict a protein's set of functions within the GO hierarchy, using graphical models. Before introducing our framework, it is necessary to review the CHUGO system.

1.1 Revisiting of the CHUGO System

CHUGO constructs a function hierarchy by pruning the GO to include only those nodes that have sufficient positive training instances. Features used for training and prediction are extracted using the Proteome Analyst (PA) [59] tool, which are key words from main entries in the Swiss-Prot database for the most similar proteins obtained by using BLAST. This feature extraction process is illustrated in Figure 1.1. To build a robust predictive system, CHUGO addresses the issue in the GO hierar-

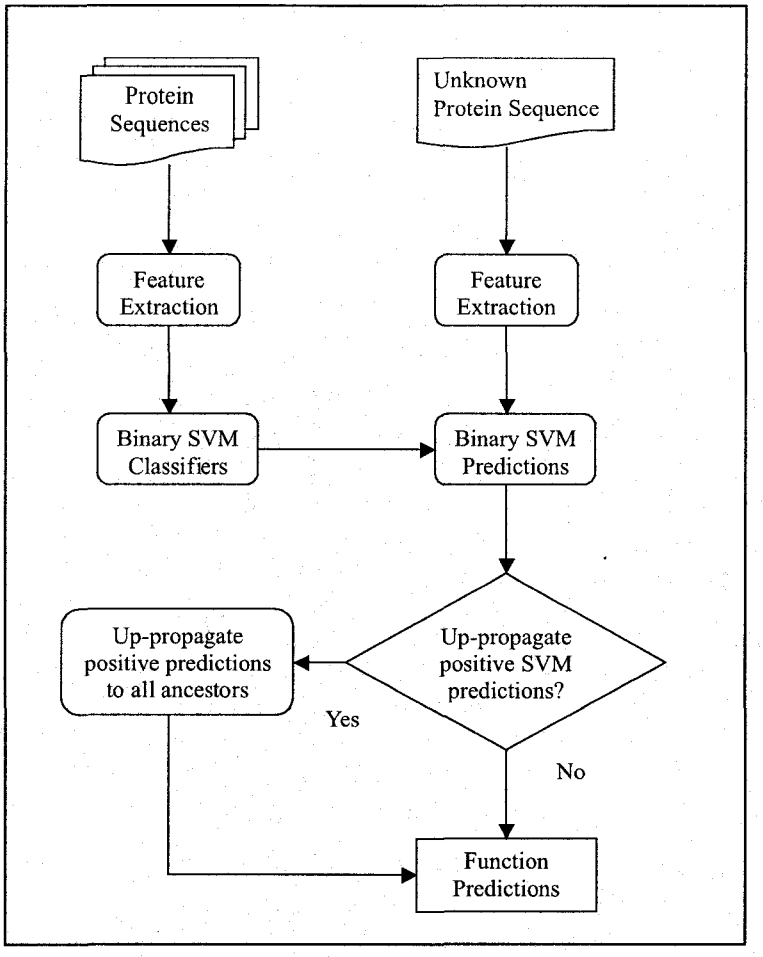


Figure 1.2: The CHUGO function prediction system.

chy from the following three aspects:

- Training set design: Since a protein annotated as a GO node N implicitly inherits functions from all the ancestors of N, functional annotation of each protein in the training set is expanded to include node N and all its ancestors. For example, if the function "neuropeptide receptor activity" is experimentally assigned to a protein, then all its GO ancestors, such as neuropeptide binding, receptor activity, etc, are also included as positive annotations. This expansion, called the *all inclusive* approach by Eisner et al., intuitively leads to an improvement in recall.

- Prediction model: The initial prediction for each protein is made by a set of independent binary support vector machine (SVM) predictors, where each is

trained on an individual GO node. Local positive predictions are then propagated up the hierarchy.

- Evaluation methodology: As in training set design, precision and recall are calculated using expanded labels for the test set as well.

Figure 1.2 summarizes the CHUGO prediction system. These techniques are simple but effective for function prediction, as the all *inclusive* approach increases the F-measure of hierarchical classification on a 5-fold cross-validated data set from 46% to 70%. However, the hierarchical relationship can be utilized to train a more accurate predictive model as will be explained in Section 1.3.

1.2 Motivation for Using Graphical Models

Before revealing the potential hazard in the CHUGO system, we must be clear about the rule of consistent labeling imposed by CHUGO. That is, if a protein is annotated as a GO node N, then it is explicitly annotated as all the ancestors of node N. For example, in Figure 1.3, if function GO0008188 is assigned to a protein, then all ancestors of node GO0008188, i.e. GO0030594, GO0008528, GO0001584, GO0004930, GO0004888 and GO0004872, are also included as positive labels. Both training and evaluation are carried out using the extended annotations.

This rule imposed for consistent labeling in the hierarchy implies two special properties:

1. If a node in the hierarchy has a negative label assignment, then all its descendants in the hierarchy must also have negative label assignment.

2. If a node in the hierarchy has a positive label assignment, then all its ancestors in the hierarchy must also have positive label assignment.

Either of these two properties implies that there does not exist a configuration that a node has a positive assignment and one or more of its ancestors has a negative assignment. These properties in the hierarchy leave us only one possible parent-child configuration probability to compute from the actual data, which is the probability of a node being positive given that *all* its ancestors are positive.

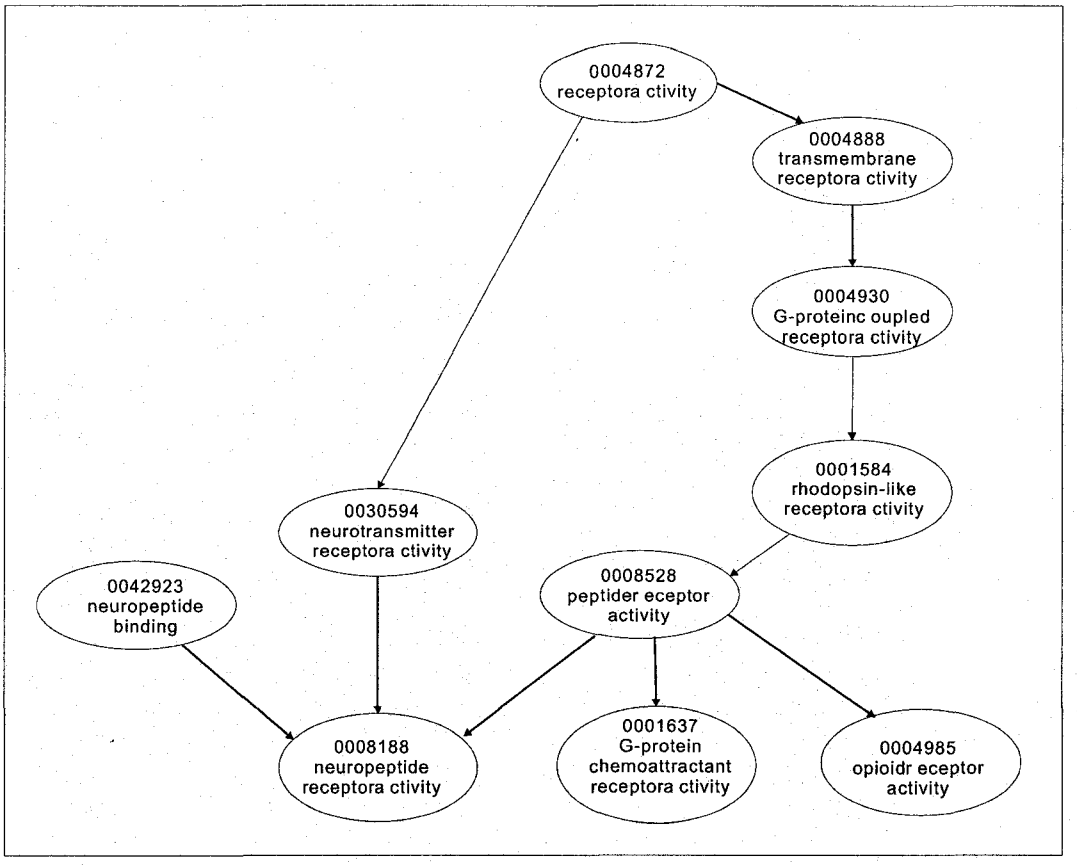


Figure 1.3: A subgraph of pruned GO hierarchy used in our experiments

Although CHUGO constructs the training and test data using the hierarchy, it builds an individual binary SVM classifier for each GO node and only exploits the hierarchical information in a simple way. The final result of the system is simply a combination of all binary SVM predictions plus "up-propagation" of positive results. They demonstrated that a propagation of all positive predictions up to the root slightly improves the result by about 0.2%. While the up-propagation approach is simple, it has critical drawbacks that may hurt overall precision and recall.

First, CHUGO propagates positive predictions obtained from the local SVM predictors for node N up to the root node. This could be problematic if node N's SVM prediction is inaccurate. For example, if a protein P is predicted by a local SVM to belong to node GO0008188 (i.e. neuropeptide receptor activity), could in fact be an incorrect prediction. If an up-propagation is applied, this local prediction error will be propagated to all ancestors of node GO0008188, as shown in Fig-

ure 1.3. In this particular case, a single misclassification would result in 8 prediction errors, which is disastrous. If we do not propagate a positive local prediction up, and instead, we look around the node and examine the status of all the neighboring nodes,i.e. classes that are connected to this node, a more accurate conclusion may be drawn by taking the neighbors in the hierarchy into account. For instance, consider a protein P that is predicted to belong to function GO0008188 but not to any of its three immediate parents of GO0008188. By tracing back statistics from the training data, one may conclude that the local SVM prediction on node GO0008188 may be more likely to be erroneous. This dependency problem between neighbors is a perfect candidate for graphical models, which learn statistics of neighboring nodes from the training data.

Second, CHUGO always trusts the positive local SVM predictions, and leaves negative local prediction to be determined by the up-propagation step. This bias is unjust since it favours positive up-propagation over negative down-propagation. Instead, we use a probabilistic SVM model to give a confidence measure to the local SVM prediction. This confidence prediction is integrated into the graphical model for making a final prediction, and we discuss in Chapter 4 and 5.

1.3 Summary of Thesis Work

In this dissertation, we propose a framework for hierarchical function prediction using graphical models. As Figure 1.4 shows, the CHUGO system (without positive prediction up-propagation) is integrated into this framework as a local predictive component, which gives a prediction from the local SVM predictor for each node. These local predictions and hierarchical information embedded in the GO are combined to train a graphical model that presents a global prediction on all nodes in the hierarchy.

During training of the graphical model, two sets of parameters are produced: parameters for hierarchy consistency and parameters for distributions of local SVM outputs. The former parameters ensure our final function predictions consistent to the hierarchy, and can be estimated from proteins' functions in the training set. By

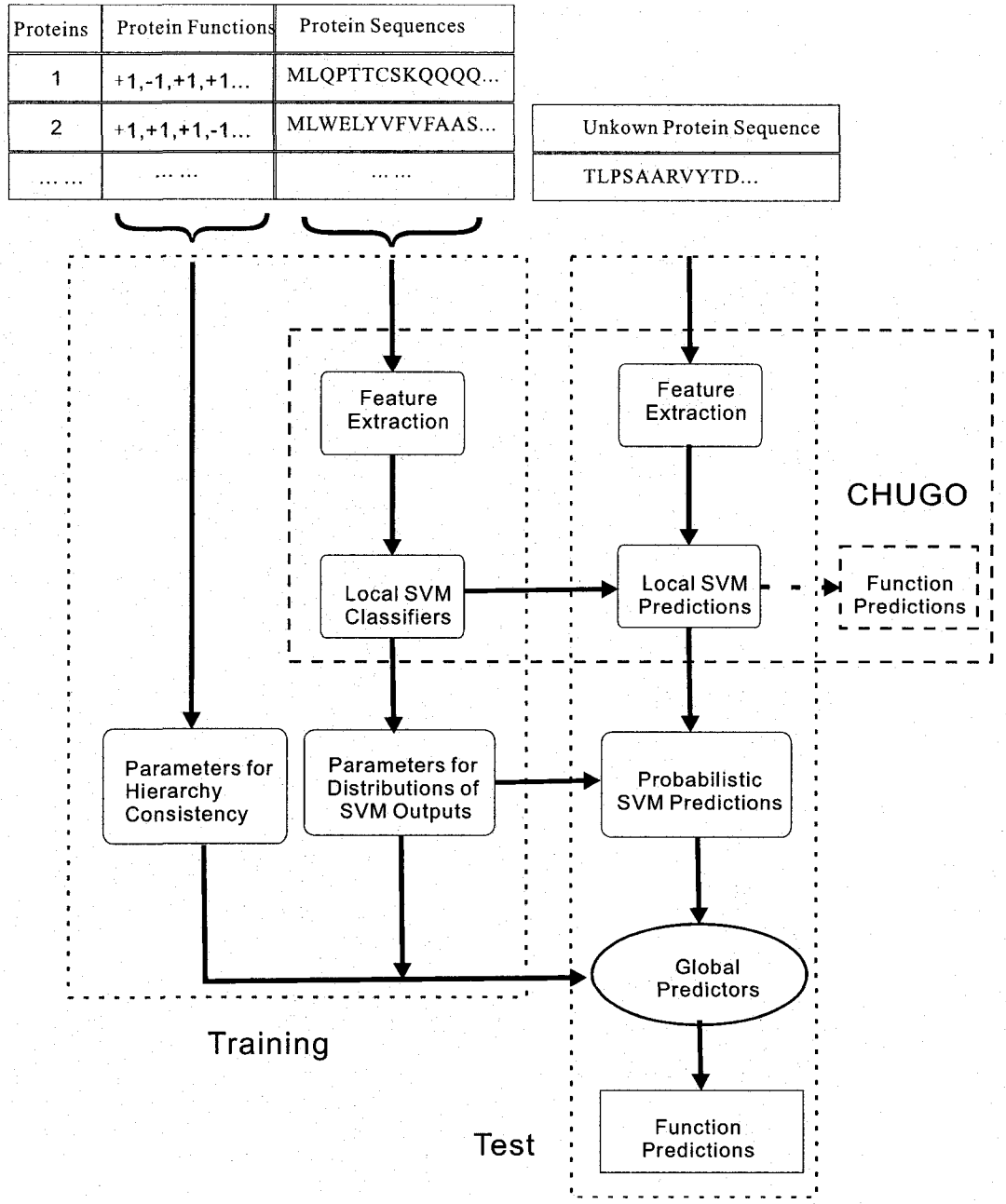


Figure 1.4: A framework for hierarchical function prediction using graphical models. During training, two sets of parameters are learned: parameters for hierarchy consistency and parameters for SVM outputs distributions. Given an unknown sequence, the graphical model can make consistent function predictions.

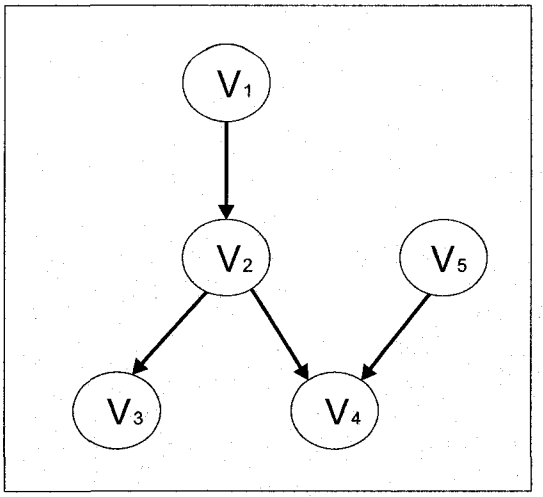
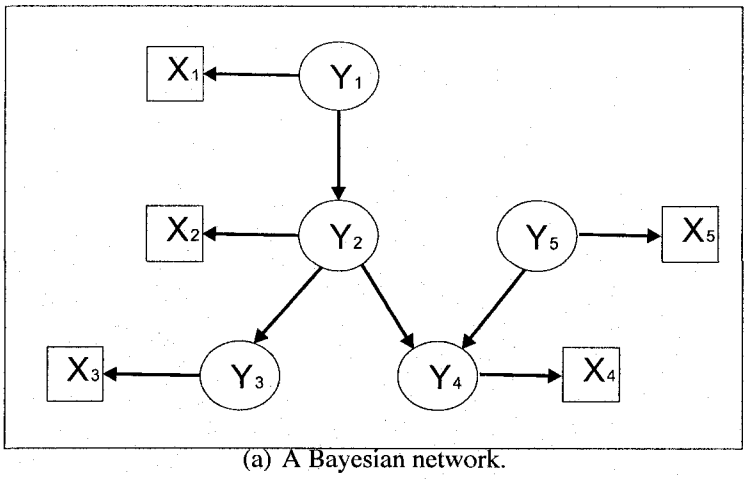


Figure 1.5: An abstract Gene Ontology hierarchy. Each V_i represents a GO term, and a directed edge indicate a parent-child relationship.

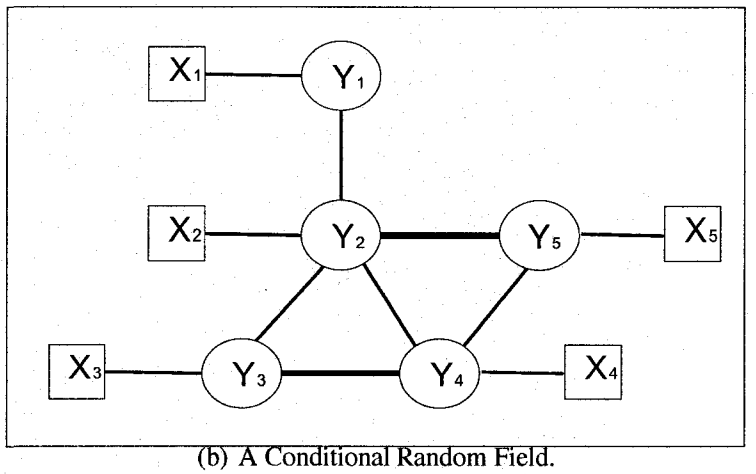
converting CHUGO's binary SVM predictions to real values, the latter parameters indicate the distributions of these real-valued SVM outputs, and can be estimated from SVM outputs for training data.

The hierarchical structure of the GO and probabilistic SVM outputs are used to construct graphical models. The first graphical model we built was a Bayesian network, which is a directed graphical model. The GO hierarchy was used as the structure of the BN, in which each directed edge represented a parent-child dependency, as shown in Figure 1.5. This graph was converted to a Bayesian network (Figure 1.6(a)) in which each node y_i represented a variable and each node in the hierarchy was also connected to an observation node x_i which represented the local SVM prediction for that node. The parameters of the Bayesian network were learned from the training data and from the probabilistic SVM outputs.

The second model we built was a Conditional Random Field (CRF), which is an undirected graphical model. The undirected graph was constructed by dropping arrows in the directed graph and adding connections between nodes that share the common child, also called "spouse" nodes, and nodes that share the common parent, also called "sibling" nodes. The resulting CRF graph is shown in Figure 1.6(b). Nodes y_2 and y_5 are spouses, and nodes y_3 and y_4 are siblings. To learn the parameters of the CRF, two types of cliques are defined, i.e. edge cliques and node-



(a) A Bayesian network.



(b) A Conditional Random Field.

Figure 1.6: The graphical models of the hierarchy. Graphical models converted from the hierarchy in Figure 1.5. Each y_i represents a GO term whose value is either $+1$ or -1, and each x_i represents a local SVM output whose value is in R.

observation cliques. The edge clique over two connected nodes encodes the hierarchical structure in the GO and the node-observation clique over a node and its SVM prediction indicates the confidence of a local SVM prediction.

Having parameters in the graphical models well defined, we can make consistent function predictions for an unknown protein sequence, using its BLASTed features. In our experiments, both BN and CRF have significantly improved the prediction accuracy.

1.4 Thesis Statement

In this dissertation, it is hypothesized that *the graphical models, i.e. BNs and CRFs, based upon the GO hierarchy are effective for hierarchical function prediction.* Specifically, the graphical models can make function predictions consistent with the hierarchy by incorporating the hierarchical information to learn the models. Based on the thesis research, we make the following claims:

1. The BN model can capture the hierarchy structure in the GO for function prediction.

2. The CRF model has the representation power to capture the GO hierarchy and statistical dependencies that are not shown in the hierarchy for function prediction.

1.5 Contributions

This thesis research makes three novel contributions:

1. Prediction accuracy is significantly improved by using the parent-child relationships in the Gene Ontology to build the Bayesian networks. While CHUGO takes the full hierarchical information into account when building the training and test data sets, it uses only the naïve positive up-propagation for prediction. Our BN approach fully exploits the parent-child relationship for making consistent function predictions. Compared to only using local SVMs, the BN system improves the F-measure of two data sets in our experiments by 2.61% and 0.91%. Compare to naïve up-propagation, the BN system improves the F-measure of two data sets in our experiments by 2.33% and 1.08%.

2. A Conditional Random Field model is applied for making hierarchical function prediction, and it increased the F-measure of two data sets by 2.94% and 1.19%, compared to only using local SVMs, and by 2.66% and 1.36% compared to naïve up-propagation. The CRFs utilize parent-child relationship in

the GO hierarchy, and also statistical information from the actual data that is not shown in the hierarchy.

3. We created an approach for estimating the SVM output distribution, i.e. a Laplace mixture distribution for SVM outputs from positive examples and a single Laplace for SVM outputs from negative examples. Our approach is more sophisticated than that proposed in Barutcuoglu et al. [9] which fits a single Gaussian distribution to SVM outputs obtained from both positive and negative instances. Our estimation technique for SVM outputs empirically works, because there are significantly a smaller number of positive instances for each functional class than that of negatives which possibly leads a negative prediction more favorable by the SVM predictor.

1.6 Thesis Outline

In this dissertation, our primary goal is to develop an effective hierarchical machine learning approach using graphical models for predicting protein functions in the Gene Ontology. Guided by this research goal, the rest of the thesis is organized as follows:

Chapter 2 introduces the necessary biological background on protein function and explains the terminology used in computational biology. It also presents related work in sequence-based protein function prediction. It concludes by reviewing hierarchical function prediction and hierarchical classification in general.

Chapter 3 reviews two main types of graphical models, i.e. directed and undirected models, and briefly describes inference approaches.

Chapter 4 revisits the binary local SVM predictors built by Eisner et al. and reviews probabilistic SVMs. It then presents a new approach to estimating the distribution of the local SVM outputs using a Laplace mixture and a single Laplace.

Chapter 5 applies two specific graphical models, i.e. BN and CRF, to make hierarchical function predictions. Model construction, parameter estimation and inference are described for each model.

Chapter 6 presents and discusses the results obtained by using the graphical

models on two different data sets.

Chapter 7 summarizes the thesis research and concludes the dissertation with possible future work.

Chapter 2

Background

A protein is a large complex molecule made up of one or more chains of amino acids. Accounting for the second largest segment of the cellular weight after water, proteins serve as building blocks and functional components of a cell [48]. They perform most of the important functions in a living organism, such as constitution of the organs (structural proteins), the catalysis of biochemical reactions (enzymes), receptors for hormones and other signaling molecules (receptors), and the maintenance of the cellular environment (transmembrane proteins). Therefore, discovering new proteins and understanding their biological functions is crucial in the development of new drugs and synthetic biochemicals.

Due to high-throughput techniques for various genome analyses, a huge quantity of genomic information has become available. The influx of biological data also makes manual annotation of protein function slow and tedious. This challenge paved the way for the emergence and popularization of automated function prediction.

This chapter will introduce protein function in general, and review previous research on protein function prediction using primary protein sequence data, annotation data from databases and GO hierarchy information.

2.1 Introduction to Protein Function

Before delving into the details of computational prediction, we start by describing protein sequences, the common understanding of protein function, and a standard-

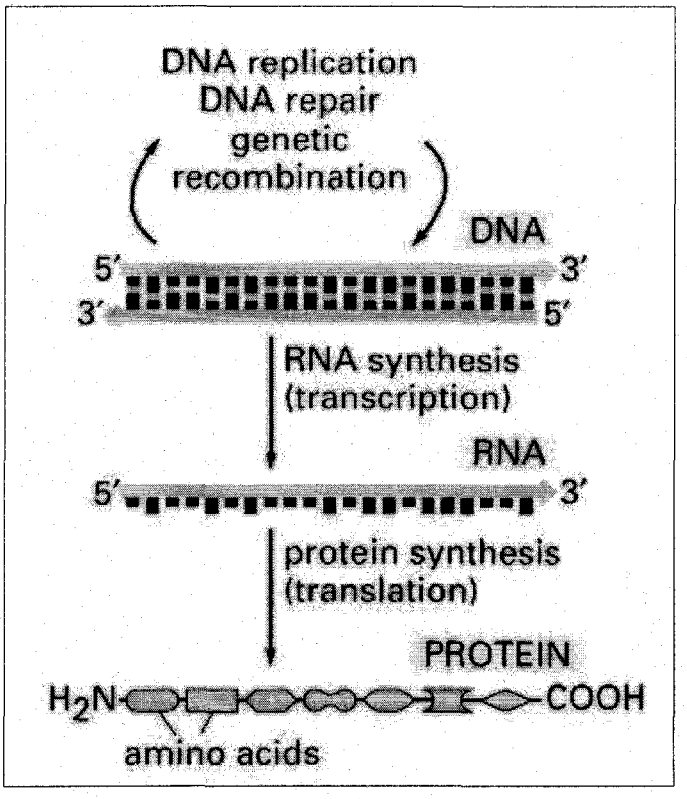


Figure 2.1: The Central Dogma of Molecular Biology. A schematic representation of the Central Dogma of showing the flow of information from DNA to RNA (transcription), and from RNA to protein (translation). Image courtesy of [2].

ized vocabulary, i.e. the Gene Ontology, which is widely used for describing protein function.

2.1.1 Protein Sequences

The Central Dogma of Molecular Biology [24] is a framework for understanding information transfer in a cell from DNA to RNA to protein, as shown in Figure 2.1. This process produces a protein sequence, constructed from a combination of 20 amino acids, each of which is represented by a letter of alphabet. The primary sequence is the most fundamental form of a protein since it determines different characteristics of the protein such as its structure, function and subcellular localization. An example of such a sequence is shown in Figure 2.1.1.

An enormous number of protein sequences have been identified by high-throughput sequencing techniques, and information about them has been collected and orga-

```
> P75245|ACKA_MYCPN Acetate kinase - Mycoplasma pneumoniae
MNDNKILVVNAGSSSIKFQLFDYHKKVLAKALCERIFVDGFFKLEFNEQKVEEKVAFP
DHHAAVTHFLNTLKKHKIIQELSDIILVGHRVVQGANYFKDSVIVDAEALAKIKEFIK

...
```

P75245:	SWISS-PROT accession number
ACKA_MYCPN:	SWISS-PROT entry name
Acetate kinase:	protein name
Mycoplasma pneumoniae:	origin of the protein

Figure 2.2: A protein sequence segment in FastA format from SWISS-PROT

nized in various standardized databases. Among these databases, the Swiss-Prot and TrEMBL databases [11] are most widely used, and they are together called the UniProt database. The Swiss-Prot is a comprehensive, human-curated database that provides a wide variety of information about proteins, such as their amino acid sequence, functional annotation, subcellular location, and other information in the form of keywords and features. As a supplement to Swiss-Prot, TrEMBL (Translated EMBL) is a computer-annotated protein sequence database that contains the translations of all nucleotide sequences present in the EMBL/GenBank/DDBJ databases [14]. As of October 2 2007, the Swiss-Prot Release 54.3 contains 285,335 entries and the TrEMBL Release 37.3 contains 4,932,421 entries, which together comprise the UniProt Knowledgebase Release 12.3.

2.1.2 Protein Function

The definition of "protein function" is ambiguous and highly context-sensitive, since proteins are involved in more than one type of activity, such as cellular, molecular and physiological activities. Bork et al. [12] have proposed categorizing protein functions into three types:

- **Molecular function:** The biochemical functions performed by a protein, such as binding sites, catalytic activity and conformational changes.
- **Cellular function:** Many proteins co-operate to perform complex physiological (cellular) functions, such as metabolism, signal transduction cascades

and structural association, to keep the various components of the organism working well.

- **Phenotypic function:** The totality of the physiological subsystems, consisting of various proteins performing their cellular functions, and their interplay with various environmental stimuli determines the phenotypic properties and behavior of the organism.

Therefore, when speaking of protein function, one must specify the context or function category, since a functional term can refer to different meanings in different functional categories. Throughout this dissertation, we consistently use the term "protein function" to refer to *molecular function.* For example, metal ion binding, transporter activity and kinase regulator activity are molecular functions of protein.

Since protein function assignment is somewhat subjective, different biologists may assign the functions of proteins differently. To make functional annotation consistent and available as input for computational processes, functional terms must be well defined and organized by some standard scheme, such as the Gene Ontology.

2.1.3 Gene Ontology

The Gene Ontology [6] is a functional classification system developed by the Gene Ontology Consortium. At the highest level, the GO controlled vocabulary is composed of three disjoint functional ontologies corresponding to *molecular function*, *biological process*, and *cellular component.* Although cellular location is not a functional aspect per se, it is included for functional annotation since proteins do not function in a vacuum but at certain locations in the cell. Each ontology is hierarchically structured and is implemented as a directed acyclic graph (DAG). This dissertation focuses on the molecular function sub-hierarchy.

The GO DAG starts with very general classes and becomes more specific at lower levels of the hierarchy. Each node in the DAG corresponds to a functional term. Terms are the building blocks of the Gene Ontology. Each term has a unique numerical identifier, which is a 7-digit sequence, and a term name, e.g. ion binding, receptor activity or catalytic activity. Each directed edge in the DAG corresponds to

either an *is-a* or a *part-of* relationship. The is-a relationship is a simple inheritance relationship, where A is-a B means that every A is a B. For example, "metal ion binding" is a child of "ion binding", since every metal ion binding protein is also an ion binding protein. The part-of relationship is a sub-component relationship, where C part-of D means that every C is a part of a D. For example, every nucleus is part of a cell. Since there are only 2 part-of edges (two children of GO0003720 telomerase activity) in the molecular function part of the GO, they are ignored in our experiments.

Since the GO forms a DAG, each node may have more than one parent, which is biologically appropriate since a specific function can be the specialization of of more than one general function. For example, auto transporter activity is a child of both porin activity and protein transporter activity. This multi-inheritance property is also taken into account when we build our model to make function predictions, as will be explained in more details in Chapter 5.

Due to its sophisticated design and wide coverage, the Gene Ontology has become the most popular functional classification scheme for protein function prediction studies. More importantly, the GO terms and hierarchical structures are constantly updated by the curators, who seek scientifically correct information to keep the ontology up-to-date with latest research. Since the GO became public, a large number of studies have used it for protein functional analysis. As listed on GO's website [1], there are 1654 publications describing studies following GO (as of November 12, 2007). For these reasons, we used a sub graph of the Gene Ontology in our function prediction experiments.

As the vocabulary in the GO grows and the volume of protein sequences increases steadily, there is a need to have a list that maps each protein to its biological functions using the GO vocabulary, and the Gene Ontology Annotation project was created to fill this need.

[1] http://www.geneontology.org/cgi-bin/biblio.cgi

2.1.4 Gene Ontology Annotation

The mission of the Gene Ontology Annotation (GOA) project [17] within the European Bioinformatics Institute is to annotate proteins in the UniProt database using the GO terms. Each of these annotations is accompanied by an evidence code, stating how the annotation was obtained. Based upon an evidence code, one can tell the reliability of the annotation: Is it an experimentally-derived annotation with high reliability, or is it inferred by a computational approach (possibly low reliability)? Evidence codes are organized into the following thirteen categories:

- **IC:** Inferred by Curator
- **IDA:** Inferred from Direct Assay
- **IEA:** Inferred from Electronic Annotation
- **IEP:** Inferred from Expression Pattern
- **IGC:** Inferred from Genomic Context
- **IGI:** Inferred from Genetic Interaction
- **IMP:** Inferred from Mutant Phenotype
- **ISS:** Inferred from Sequence or Structural Similarity
- **NAS:** Non-traceable Author Statement
- **ND:** No biological Data available
- **RCA:** inferred from Reviewed Computational Analysis
- **TAS:** Traceable Author Statement
- **NR:** Not Recorded

GO annotations should be used with caution and based on different annotation reliability requirements. To create a reliable data set, one should only consider manually-curated annotations, such as IDA, IEP, IGC, IGI, IMP, and TAS, and exclude electronically-curated annotations, such as IC, IEA, ISS, NAS, ND, RCA, and NR.

2.2 Review of Function Prediction Using Primary Protein Sequence Data

In the domain of computational function prediction, primary protein sequence data has been widely used since it is the most fundamental factor that determines protein function. Specifically, automatic techniques that use primary protein sequence data to predict function are grouped into three categories: homology-based approaches, subsequence-based approaches and feature-based approaches.

2.2.1 Homology-based Approaches

The homology-based approach is one of the first and most-often-used techniques for predicting protein function. It is widely believed that similar proteins in different species mutated a common ancestor sequence during evolution. Therefore, homology-based approaches predict function by comparing a query sequence to similar sequences contained in the databases that have known function and inferring that function from the known functions.

BLAST [3] is an automated tool for finding homologs and it is the foundation of most homology-based techniques. BLAST searches standard databases such as Swiss-Prot for sequences similar to the query protein using approximate sequence alignment algorithms. The result of a BLAST search is accompanied by an E-value for each match in the database, which denotes the quality of the alignment between the query sequence and the matched sequence. Smaller E-values mean higher similarity between the query sequence and the sequence match from the database. BLAST is now pervasively utilized in molecular biology, and in fact, it is widely believed that [4] there is rarely a study in molecular biology that does not involve BLASTing a gene or protein against a standard database.

Although homology-based methods such as BLAST have demonstrated promising results, their applicability is restricted mainly for two reasons:

- Limited protein coverage: No BLAST result will be returned if there are no similar sequences found in the database.

- Inconsistency of function between homologues: Sometimes a homologue

will have a different function in a different species in response to selective pressure during evolution [32, 66].

2.2.2 Subsequence-based Approaches

It is believed that some sequence segments are more important for determining protein function than others, and the subsequence-based approach identifies such critical segments as features of a protein and builds a computational model to predict function based upon those features.

The subsequence-based techniques extract features from *sequence motifs*, meaningful subsequences that are conserved across a set of protein sequences of a family [13]. Although subsequence motifs may be responsible for biological characteristics of a protein, it is difficult to identify these motifs that can best distinguish function. Although various strategies have been proposed for subsequence-based prediction [13, 37, 38, 47, 54], the results obtained so far are not as impressive as expected.

2.2.3 Feature-based Approaches

Feature-based techniques collect biologically meaningful features from each individual protein sequence, and use those features to construct models for predicting protein functions. They are similar to the subsequence-based approaches in terms of building a predictive model using features from the data, but differ in that they focus on biologically meaningful features while subsequence-based techniques extract features from sequence motifs, based on patterns.

In recent years, feature-based approaches have been shown to be successful in a number of studies [1, 16, 29, 40, 59], and it is widely believed that the inclusion of physical and functional features, such as subcellular location, post-translational modifications and residue-related features, creates a more robust model for the function prediction task.

Currently, the most cited work in the feature-based category is Jensen et al. [40]. They proposed a method, called ProtFun, for predicting function using sequence-derived protein features such as predicted post translational modifications (PTMs),

protein sorting signals and physical/chemical properties calculated from the amino acid composition. The ProtFun system [41] used 14 such calculated features to make predictions on 14 GO functions using neural networks, and achieved a sensitivity of at least 50% for all classes and 70% for the best category, namely hormones and receptors.

Another feature-based prediction tool is SVMProt [16]. SVMProt constructs its feature space using a set of residue-specific attributes such as normalized Van der Waals volume and polarity, and trains a binary SVM classifier for each functional family. The overall accuracy, defined as the proportion of true results (both true positives and true negatives) in the population, by applying SVMProt in the function prediction experiments ranged from 69.1 to 99.6% among 54 functional classes.

A trend of feature-based approaches is to integrate the homology-based technique such as BLAST into the feature extraction process, and Proteome Analyst [59] is one of these integrative tools. PA predicts functions of query proteins by training binary SVM classifiers using features from the Swiss-Prot main fields associated with most similar proteins that are obtained by BLASTing. PA constructs a function tree by selecting 14 nodes from the original GO hierarchy and makes predictions that are consistent with the constructed tree. This hierarchy construction differs from ProtFun and SVMProt, neither of which treats GO functions as a structured hierarchy but rather a set of independent labels, and makes a better use of the hierarchical information for training predictive models. Figure 2.3 shows the hierarchy of the 14 functions used in PA.

Another trend in sequence-based approaches is to produce an integrated predictive tool by unifying various data types. For example, InterPro [69] provides a single prediction system by integrating the commonly used signature databases. Each protein run through InterPro is assigned a variety of InterPro codes, some of which can be mapped to the GO terms if they represent functional families.

Besides protein sequence, there are a large variety of other kinds of data that have been used to predict function, such as protein structure, gene expression data, phylogenetic profiles and protein interaction networks, etc. Since this dissertation deals with sequence data only, review of other types of methods is not covered and

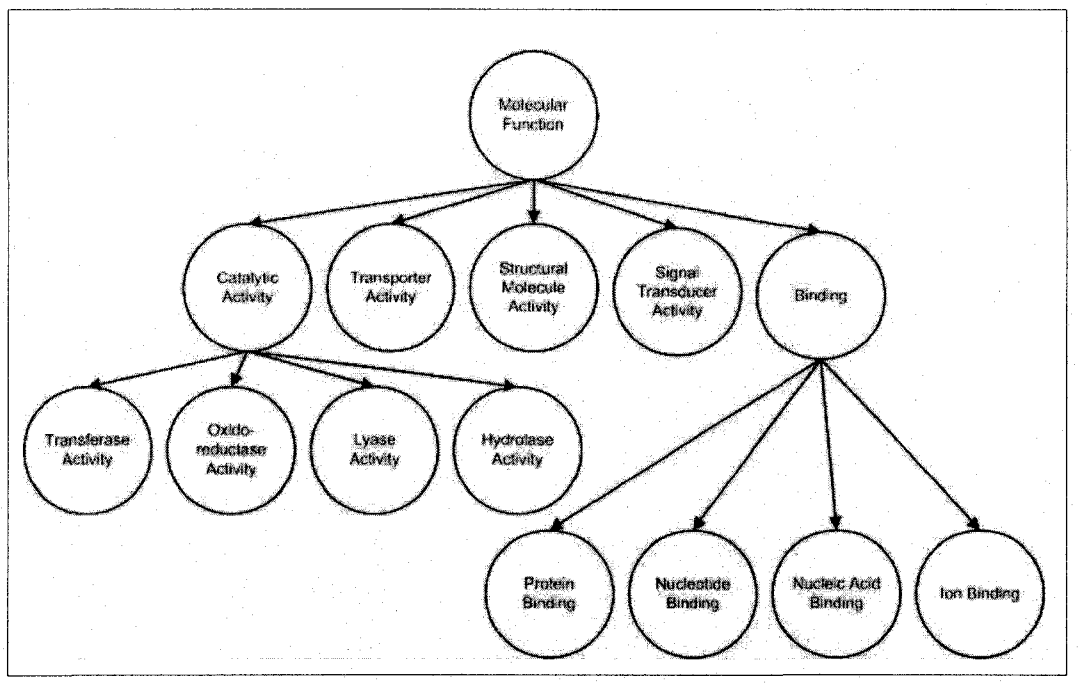


Figure 2.3: The pruned Gene Ontology used in Proteome Analyst function prediction. Image courtesy of [29].

we instead refer readers to [36, 48].

2.3 Review of Hierarchical Function Prediction

As described in Section 2.1.3, the Gene Ontology is a structured ontology that relates terms to each other. It is unwise to ignore these relationships since they provide additional information that can be used to improve the classification model. This section reviews a number of proposed computational approaches to function prediction, which explicitly incorporate the GO hierarchy into the prediction algorithm.

King et al. [43] used the additional information from the hierarchy of functional classes by simply using different decision tree models for each level of the hierarchy. They induced rules for predicting function using a variety of data sources, such as residue frequencies, phylogeny and predicted structure. In another study, Clare and King [22] proposed a modified decision tree model, in which a positive functional annotation to a node in the decision tree was propagated to all of its parent classes (i.e. up-propagation in Eisner et al. [29]). Their experiments on *Saccharomyces cerevisiae* data showed that the modified version was sometimes better

than the non-hierarchical model and sometimes worse.

Struyf et al. [55] suggested an alternative modification of using decision trees for hierarchical prediction that used distances derived from the hierarchy to train the model. Their approach makes use of the edge distances between nodes, which is a useful feature for making predictions in a hierarchy and should be considered in our future experiments. They evaluated their approach on different datasets available for *Saccharomyces cerevisiae*, and showed that their model outperformed the hierarchical model proposed by King et al. [43].

Tu et al. [62] proposed a learnability-based approach for predicting functions in the Gene Ontology. The basic idea of their approach is to focus on the prediction of "learnable" functional families, i.e. classes where the membership can be predicted with a reasonable accuracy given the available features. For each learnable class, a classifier is trained to predict the child class, which they called "further prediction". Through learnability-based predicting, functional annotations are made more specific. Although we do not explicitly define learnable classes in our approach, our graphical models can possibly extend an SVM annotation to a more specific class based on the training data, which will be discussed in Chapter 5.

Verspoor et al. [64] presented a system for functional annotation by analyzing collections of GO nodes obtained from annotations of protein BLAST neighborhoods. Those GO annotations are weighted according to their E-values. The weighted GO nodes are then imported using a ranking system, POSet Ontology Categorizer (POSOC), to identify the most representative nodes as predicted functions of the query protein. They evaluated their function-prediction method by presenting what they called the hierarchical precision and hierarchical recall. In fact, their evaluation technique is equivalent to that of Eisner et al. which evaluates prediction performance based on the expanded set of annotations.

Recently, there has been a growing interest in applying Bayesian networks to protein function prediction. Barutcuoglu et al. [9] developed a Bayesian network for combining a set of independent classifiers. The architecture of the GO hierarchy is used as the structure of the Bayesian network, and a single SVM is trained independently for each functional class in the GO. Thus, the outputs of individual

classifiers are combined in a hierarchical fashion. They fit local SVM outputs from both positive and negative instances using the same distribution, i.e. Gaussian. (We will show a superior model in Chapter 4.) They presented their experimental results based on 105 functional classes using 3,465 annotated sequences, while our experiments are carried out on a much larger scale, i.e. 399 classes using 14,018 proteins, and 792 classes using 45,956 proteins.

In a different manner from [9], Engelhardt et al. [30, 31] built a Bayesian system to model the probability for the transfer of protein function from a parent to a child class in a single phylogenetic tree, composed of GO terms. Given a query protein sequence, they first constructed a phylogeny based on a set of homologous proteins. They learn, by fitting a probabilistic model, the transition probability for any parent-child pair in the GO hierarchy, which was constructed by taking the union of all GO annotations associated with the proteins in the phylogeny. The final prediction was obtained by computing the maximal posterior probability of possible node assignments. It was reported as the best results in function prediction via phylogenetic analysis.

2.4 Review of Hierarchical Classification

A great amount of effort has been dedicated to studying hierarchical classification in general and to studying applications in domains where an organized ontology exists, such as text categorization and web content extraction. The use of hierarchical decomposition allows a classification problem to be addressed using a divide-and-conquer approach, which can be solved efficiently.

Before delving into reviews of hierarchical classification algorithms and measures, we must be clear about the structure of a hierarchy. In general, there are two main types of structures for a class hierarchy, a tree structure and a Directed Acyclic Graph structure. They both consist of root class(es), internal classes, and leaf classes. The root class(es) denote(s) the most general description of all categories in the hierarchy. Each internal class has its parent and child class(es). The main difference between these two hierarchical structures is that each class in a tree

has at most one parent, while each class node in DAG can have multiple parents.

Hierarchial classification algorithms can possibly be categorized into two major groups, *big-bang* and *top-down* [56, 57]. In the big-bang approach, a single complex model is trained from the training data, which takes the class hierarchy into account during a single run of the classification method. Given a test instance, the classifier can assign it to more than one category in the category tree, and makes assignments of classes at potentially every level of the hierarchy. The big-bang approach has been used by several studies in text mining, such as the rule-based classifier [52], Naive Bayes classifier [61], and methods built on association rule mining [65].

In the top-down approach, more than one classifier is built each level of the hierarchy during training, and each classifier functions independently as a flat classifier at that level. A test example is first classified by the first-level classifiers, and then is further classified by the classifiers of the lower level classes whose parent classes have been predicted at the higher level until the example cannot be further classified. For example, Dumais and Chen [28] explored the hierarchical structure for classifying a large collection of web content, using a top-town approach with SVM base classifiers. Their hierarchy of the web content consisted of two levels, 13 top-level and 150 second-level categories (for instance, sports/football, sports/soccer, computer/hardware, and computer/software). They trained SVM classifiers to distinguish a second-level category from other categories within the same top level. In one of their experiment settings, a classification process continued to the second-level categories only if the corresponding top-level category had a positive classification. Their top-down approach using the hierarchical information was shown to be effective and to be able to improve the overall classification performance slightly, compared to non-hierarchical classification. The top-down approach has been also implemented using other base classifiers such as ACTIONs(for Automatic Classification for Full-Text Documents) algorithm [25] and Bayesian classifiers [44].

The top-down approach has the advantage that the original big classification problem can be divided to smaller sub-problems at each level, and it is efficient in both training and classification phases. Compared to the top-down approach, the

big-bang approach builds a more complex classification model by considering the entire class hierarchy in the training phase, but not the classification phase. The big-bang classifiers assign the test instances to classes regardless of their locations in the hierarchy. Another problem with the big-bang approach is that the constructed classifier may not be flexible enough to adjust for changes to the hierarchy. The classifier must be re-trained if the hierarchy is changed.

On the other hand, the top-down approach has the disadvantage that a classification error at a parent class may mislead classifications at all the deeper levels. It requires some recovery procedure to reduce this kind of errors induced by the parent classifiers. In particular, the top-down approach has to deal with a special situation in a DAG hierarchy: for instance, given a class with two parent classes, if a top-down classifier at the parent level classifies one parent positive and the other negative, should the classification process continue on the next level? A tree hierarchy does not have such an issue since it only has a single parent, while a DAG-like structure may cause some contradictory predictions for the top-down approach.The top-down approach also requires more training instances since multiple classifiers have to be constructed and each requires a different training data set.

We consider our hierarchical prediction system using graphical models as a top-down approach, although it does not make classifications level by level. It takes into account the hierarchical structure for training SVM classifiers. Although level-based classification could be considered in our application, the issue that a protein may belong to classes in different branches in the hierarchy must be addressed first. The proposed graphical models serve as the remedy mechanism to recover from incorrect classifications at the shallower levels of the hierarchy. To have adequate training examples in our experiments, we set up a lower bound for the number of proteins belonging to each GO function before constructing a prediction for that function.

Another issue in hierarchical classification is how to measure the predictive performance of a classification algorithm. The performance of hierarchical classification can be measured in several ways [10, 56, 57]. The widely-used and easily-implemented approach is the measure for flat classification. Using this measure,

every classification error is assigned the same cost, regardless of the level of the classes in the hierarchy. The most commonly used performance measures in flat classification are *Precision* and *Recall*, which will be introduced in Section 6.1. However, the uniform cost measure is usually not ideal for measuring predictive performance in hierarchical classification tasks, because it ignores the fact that classes that are closer in the category are more similar to each other than classes that are further away. Hence, measures based on hierarchy properties are proposed. Distance-based and semantics-based (or category-based) measures evaluate predictive performance based on the distance in the hierarchy and category similarity of the predicted class and true class. The distance can be calculated based on depth or path in the hierarchy [10], and the category similarity can be defined by some measure of similarity between instances belonging to each class [57]. Those two approaches take the hierarchy properties into account for algorithm evaluation, but they are not as well accepted as in the flat classification measures so they need further study and discussion. Since we are most interested in measuring the accuracy of the final classification results of our system, we adopt the uniform cost measure for evaluations.

Chapter 3

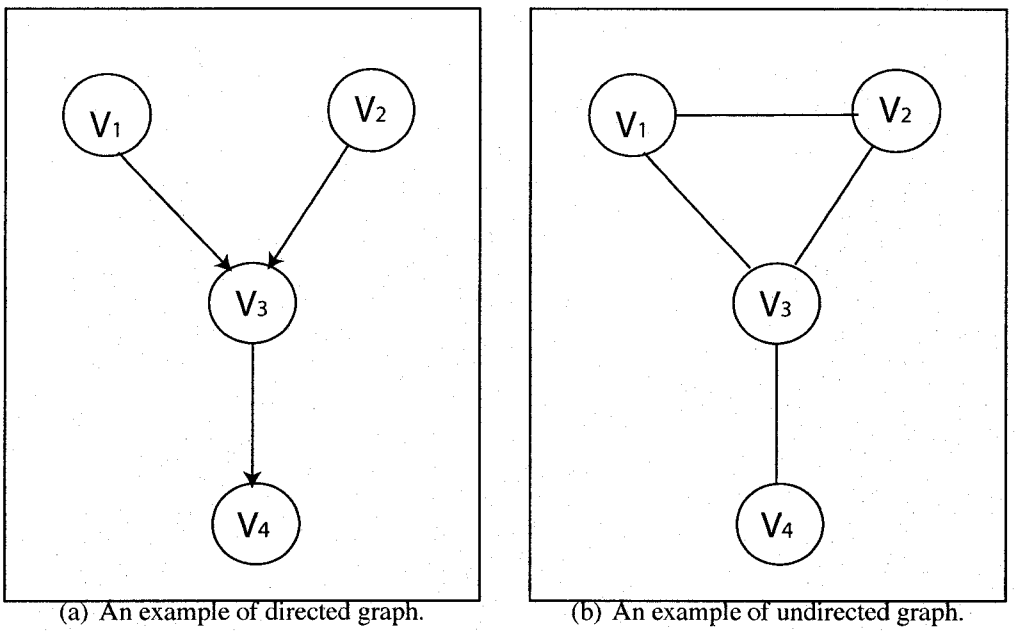
Introduction to Graphical Models

Graphical models are a graph-theoretic tool for dealing with conditional probability distribution problems. The nodes in the graph represent random variables, and the absence of edges represent conditional independence between random variables. The graphical model is a convenient way to represent joint probability distributions and to answer queries about them. The graph can be either directed or undirected. This chapter will introduce both directed and undirected graphical models, and the inference algorithms available for these graphical models.

3.1 Directed Graphical Model

A directed graphical model, also called a *Bayesian Network* or *Belief Network*, is based on a directed acyclic graph $G = (V, E)$, where V is a set of vertices, each representing a variable, and E is a set of directed arcs in the graph G. One can regard an arc from random variable V_i to V_j as indicating that V_i "causes" V_j. Throughout this dissertation, we will follow a standard practice of notations, i.e. upper case letters represent *variables* and low case letters represent *values*. For example, v_i represents $V_i = v_i$, and in the case of binary values, $+v_i$ means $V_i = +1$ and $-v_i$ means $V_i = -1$, and $\mathbf{v}$ represents $(v_1, v_2, \ldots, v_L)$ where L is the number of vertices in graph, .

Since the conditional dependence between random variables is defined by the graph, the joint probability distribution, also called the probability mass function in the discrete case and probability density function in the continuous case, can be



(a) An example of directed graph. (b) An example of undirected graph.

Figure 3.1: Two examples of graphical models.

calculated as a product of the conditional probability of each variable conditioned on its parents, i.e.

$$P(\mathbf{v}) = \prod_{V_i \in V} P(v_i | Pa(v_i)) \tag{3.1}$$

where $Pa(v_i)$ denotes an instantiation of the parents of V_i in the graph, for example, $Pa(v_3) = \{v_1, v_2\}$ in Figure 3.1(a). For a variable with no parents, the conditional probability is just its unconditional prior probability. For instance, given Figure 3.1(a), the joint probability $P(+v_1, -v_2, +v_3, -v_4)$ can be computed as

$$\begin{aligned} & P(+v_1, -v_2, +v_3, -v_4) \\ = \; & P(+v_1)P(-v_2)P(+v_3| + v_1, -v_2)P(-v_4| + v_3). \end{aligned}$$

Given a BN as shown in Figure 1.6(a), one can define the joint probability of variables in the graph. Formally, given a set of observation variables $X = (X_1, X_2, \ldots, X_L)$, a Bayesian network models the joint assignment of all hidden variables $Y = (Y_1, Y_2, \ldots, Y_L)$, where $L = 5$ in this specific example. To find the joint probability $P(Y, X)$, this BN makes two independence assumptions. First, it assumes that each state Y_i is independent of all non-descendents given its direct parent(s) $Pa(Y_i)$, i.e. node Y_3 is independent from Y_1, Y_4 and Y_5, given its parent

Y_2, and Y_4 is independent from Y_1 and Y_3, given its parents Y_2 and Y_5. Second, it also assumes that the observation X_i is independent of other variables given the current state Y_i. With these two assumptions, the joint probability of an observation sequence **x** and a state sequence **y** can be easily calculated as

$$P(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{L} [P(y_i|Pa(y_i))P(x_i|y_i)]. \tag{3.2}$$

3.2 Undirected Graphical Model

An undirected graphical model, also known as a *Markov Net*, can also be represented by a graph $G = (V, E)$ over a set of random variables, differing from BN in that its edges are undirected. A clique in graph G is a set of fully connected vertices and is denoted by c, and the set of all cliques in graph G is denoted by C. For example, in Figure 3.1(b) nodes $\{V_1, V_2, V_3\}$ form a clique, and $\{V_3, V_4\}$ form another clique. Let $\psi_c(V_c)$ denote a nonnegative potential function associated with possible configurations of variable V_c in clique c. For example, the potential function $\psi_{123}(v_{123}) = \psi_{123}(v_1, v_2, v_3)$. The joint probability of the random variables can be defined as the normalized product of the potential functions over all cliques, C, in graph G, i.e.

$$P(\mathbf{v}) = \frac{1}{Z} \prod_{c \in C} \psi_c(v_c), \tag{3.3}$$

where the normalization factor $Z = \sum_{\mathbf{v}} \prod_{c \in C} \psi_c(V_c)$ in the discrete case or $Z = \int_v \prod_{c \in C} \psi_c(V_c)$ in the continuous case. This factor Z is also called the *partition function*. For example, given the undirected graph in Figure 3.1(b), the joint probability $P(+v_1, -v_2, +v_3, -v_4)$ can be computed as

$$\begin{aligned} & P(+v_1, -v_2, +v_3, -v_4) \\ = \; & \frac{1}{Z} \psi_{123}(+v_1, -v_2, +v_3) \psi_{34}(+v_3, -v_4), \end{aligned}$$

where

$$\begin{aligned} Z \; = \; & \psi_{123}(-v_1, -v_2, -v_3) \psi_{34}(-v_3, -v_4) \\ + \; & \psi_{123}(-v_1, -v_2, -v_3) \psi_{34}(-v_3, +v_4) \\ + \; & \psi_{123}(-v_1, -v_2, -v_3) \psi_{34}(+v_3, -v_4) \end{aligned}$$

$$\ldots$$

$$+ \quad \psi_{123}(+v_1, +v_2, +v_3)\psi_{34}(+v_3, +v_4)$$

A *Conditional Random Field* (CRF) may be viewed as an undirected graphical model conditioned upon a set of global observations. In a CRF, given a graph over a set of observations x and labels y, we can define a set of cliques $C = \{(X_c, Y_c)\}$. By the Hammersley-Clifford Theorem [34], the conditional probability of the labels $\mathbf{y}$ given the observations $\mathbf{x}$ can be modeled by a conditional exponential family over cliques in the graph, i.e.

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \exp \sum_{c \in C} \psi_c(x_c, y_c), \tag{3.4}$$

where $Z = \sum_{\mathbf{y}} \exp \sum_{c \in C} \psi_c(x_c, y_c)$.

We use the undirected graph shown in Figure 1.6(b) to illustrate CRFs. Different cliques can be defined for the CRF model in this example, as long as every vertex in the clique is connected to each other. For instance, a set of maximal cliques, C, consists of cliques (Y_1, Y_2), (Y_2, Y_3, Y_4) and (Y_2, Y_4, Y_5), and a set of pairwise cliques is the set of all edges, i.e. (Y_1, Y_2), (Y_2, Y_3), (Y_2, Y_4), (Y_2, Y_5), (Y_3, Y_4), and (Y_4, Y_5). Since each observation X_i is attached to its state node Y_i, cliques can be defined over these pairs of state-observation nodes as (X_i, Y_i). Given the observation sequence $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ where $L = 5$, the conditional distribution has the form

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z} \exp[(\psi_{12}(y_1, y_2) + \psi_{234}(y_2, y_3, y_4) + \psi_{245}(y_2, y_4, y_5)) \\ &+ \sum_{i=1}^{L} \psi_i(x_i, y_i)], \end{aligned} \tag{3.5}$$

if the maximal cliques are chosen for computing. These potential functions will be defined later in Section 5.2.2.

3.3 Inference Algorithms

Given a specific graphical model, the main goal of inference is to estimate the values of hidden nodes Y, given the values of the observed nodes X, i.e. $P(\mathbf{y}|\mathbf{x})$. To

compute this posterior probability, we can use Bayes' rule:

$$P(\mathbf{y}|\mathbf{x}) = \frac{P(\mathbf{y},\mathbf{x})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})}. \tag{3.6}$$

In general, computing the posterior using Bayes' rule is computationally intractable, because computing the likelihood term $P(\mathbf{x})$ involves a marginalization computation, i.e. $\sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y})$ which is a sum over an exponential number of terms.

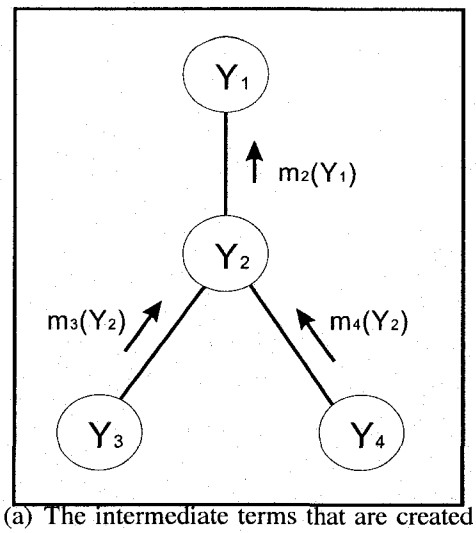
By using the conditional assumptions encoded in the graph, one can speed up the computation of the posterior probability. Since we use an exact inference algorithm in our application, here we only review some of the popular exact inference algorithms. However, there is another large group of algorithms for approximate inference, which each produces approximate target probabilities using more efficient computations. Most commonly-used approximate inference algorithms include loopy belief propagation, sampling methods and variational methods; we refer readers to [7, 33].

Here, we present three of the many exact inference algorithms, i.e. the variable elimination algorithm, the belief propagation (BP) algorithm, and the Junction Tree (JT) algorithm. To keep our illustration simple, we use a four-node graph shown in Figure 3.2(a) for introducing the variable elimination algorithm and BP algorithm. Although the example given here is an undirected graph, the inference algorithms also apply to directed graphs. It is easy to convert a directed graph to the counterpart undirected graph by connecting parent nodes that share a common child and dropping all arrows in the graph, as will be discussed in Section 3.3.3. Also, note all observation nodes X for SVM outputs are removed for now.

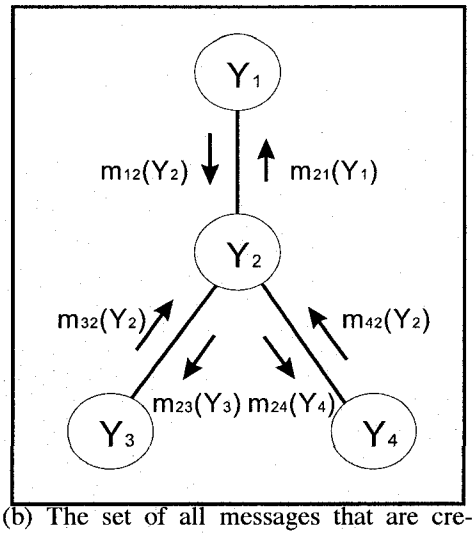
3.3.1 Variable Elimination Algorithm

The algorithm is called variable elimination because it eliminates all the irrelevant variables using factored representation of the joint probability. Consider an example in Figure 3.2(a). Suppose that potential functions are fixed or can be extracted from the conditional probability table of the corresponding directed graph, then the marginal probability $p(y_1)$ is

$$p(y_1) = \frac{1}{Z} \sum_{y_2} \sum_{y_3} \sum_{y_4} \psi(y_1, y_2)\psi(y_2, y_3)\psi(y_2, y_4)$$



(a) The intermediate terms that are created by the elimination algorithm when nodes 2, 3 and 4 are eliminated in a four-node tree.



(b) The set of all messages that are created by the belief propagation algorithm in a four-node tree.

Figure 3.2: Message passing in a four-node tree.

$$
\begin{aligned}
&= \frac{1}{Z}\sum_{y_2}\psi(y_1,y_2)\sum_{y_3}\psi(y_2,y_3)\sum_{y_4}\psi(y_2,y_4) \\
&= \frac{1}{Z}\sum_{y_2}\psi(y_1,y_2)m_3(y_2)m_4(y_2) \\
&= \frac{1}{Z}m_2(y_1) \qquad (3.7)
\end{aligned}
$$

where intermediate factors m_2, m_3 and m_4 are considered as "messages" passing from the marginalized variables. The limitation of the elimination algorithm is that it only computes a single marginal probability. In real-world applications, more than one marginal probability is often required, and multiple runs of an elimination algorithm become very inefficient, and therefore a dynamic-programming-like inference algorithm is desirable.

3.3.2 Belief Propagation Algorithm

Belief Propagation [49], also known as the sum-product algorithm, is a dynamic programming form of variable elimination for calculating the marginals in a tree. It updates the marginal (or belief) of each node iteratively by passing messages from the neighbors until they converge.

We will illustrate the BP algorithm by an example shown in Figure 3.2(b). The

marginal probability $p(y_1)$ can be calculated by

$$\begin{aligned} p(y_1) &= \frac{1}{Z}\psi(y_1)\prod_{i\in N(Y_1)} m_{i1}(y_1) \\ &= \frac{1}{Z}\psi(y_1)m_{21}(y_1) \end{aligned} \tag{3.8}$$

where $N(Y_1)$ denotes the neighborhood set of node Y_1, and $m_{i1}(y_1)$ is the message passed from the neighboring node y_i to node y_1 which is given by

$$\begin{aligned} m_{21}(y_1) &= \frac{1}{Z}\sum_{y_2}\psi(y_1,y_2)\psi(y_2)\prod_{i\in N(Y_2)\backslash Y_1} m_{i2}(y_2) \\ &= \frac{1}{Z}\sum_{y_2}\psi(y_1,y_2)\psi(y_2)m_{32}(y_2)m_{42}(y_2) \end{aligned} \tag{3.9}$$

where $N(Y_2)\backslash Y_1$ refers to all nodes neighboring Y_2 except Y_1. $m_{32}(y_2)$ and $m_{42}(y_2)$ can be computed following the same procedure. If marginals of the other nodes y_2, y_3 and y_4 are also desired, the messages that were computed for getting $p(y_1)$ can be reused as in dynamic programming, and only messages passed in the opposite direction need to be computed, i.e. $m_{12}(y_2)$, $m_{23}(y_3)$ and $m_{24}(y_4)$. It can be shown that the number of possible messages that BP computes is twice the number of edges in the tree.

As a summary, the BP algorithm sends messages from all leaf nodes to the neighboring nodes and continues sending messages in this manner until all possible messages in the tree have been sent exactly once. Once all messages are obtained, the marginal of a variable in the tree is simply the product of the incoming messages of all its adjacent nodes. The BP algorithm is capable of handling the inference problem in any *acyclic* graph such as a chain or a tree, but does not function well in loopy graphs, i.e. graphs with cycles, and a more complex method is needed to tackle this cycle issue.

3.3.3 Junction Tree Algorithm

The Junction Tree algorithm is one of the most widely used algorithms for exact marginalization in loopy graphs. Two versions of the JT algorithm were developed in the late 1980s. One version by Shafer and Shenoy [53], and the other was initially

developed by Lauritzen and Spiegelhalter [45]. The latter version was soon refined to a message passing scheme, which is described in this section.

In essence, the JT algorithm performs belief propagation on a modified graph called a junction tree, which is obtained by clustering cycles into single nodes. The main steps involved in performing the JT algorithm can be summarized as follows [7]:

1. **Moralisation** Given a directed graph, one can moralize it by adding a link between any pair of nodes with a common child and dropping edge orientations. If given an undirected graph, then this step can be skipped.

2. **Triangulation** An undirected graph is triangulated if every cycle of length 4 or more contains an edge to connect two nonadjacent nodes.

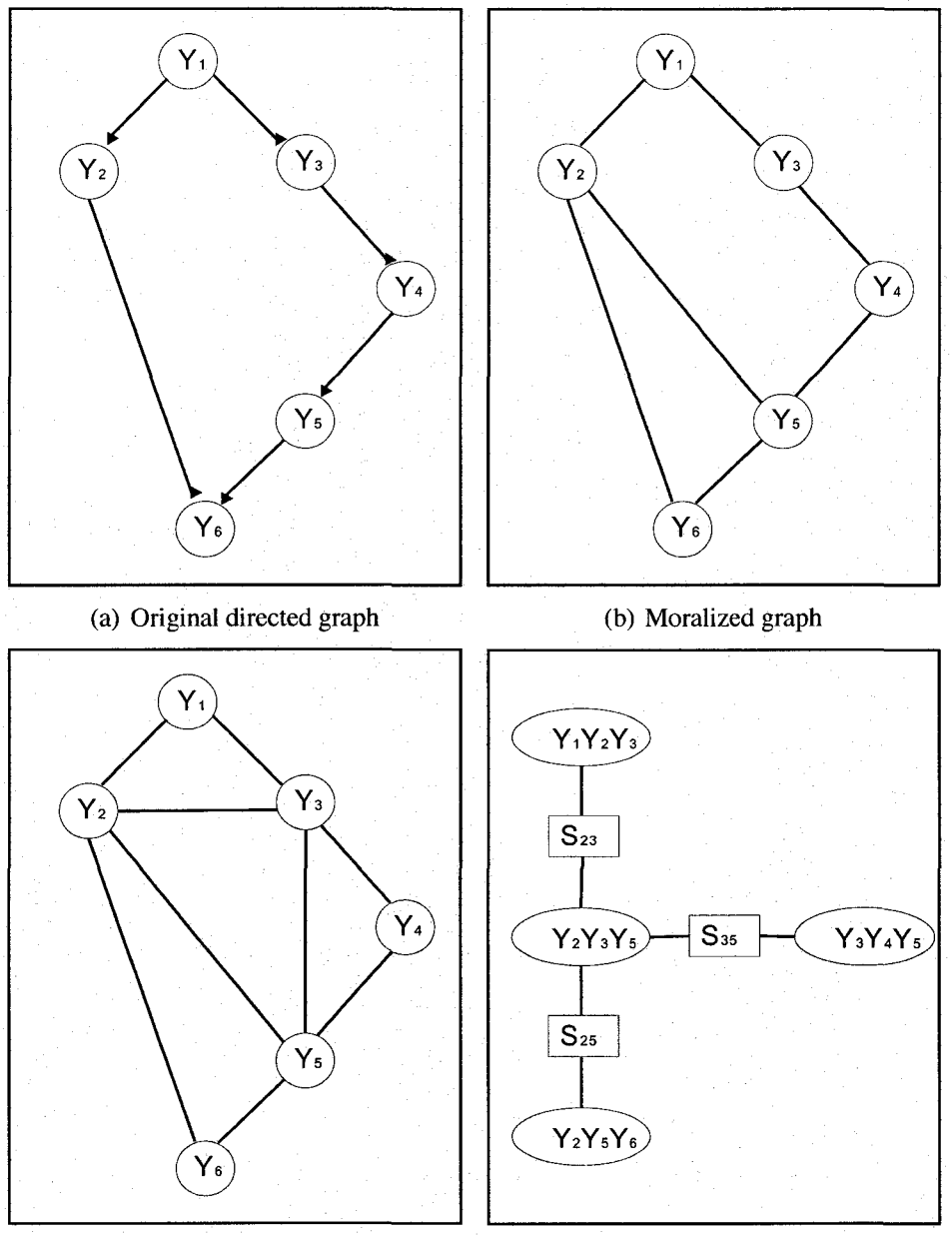
3. **Form the Junction Tree** Form a JT by forming a cluster representation from cliques of the triangulated graphs.

4. **Potential Assignment** Assign the potentials to the cliques on the JT and assign the separator potentials on the JT to unity.

5. **Message Propagation** Pass messages until updates have been passed along both directions of every link on the JT.

All steps except the second are deterministic. That is, there is only one moral graph and a unique set of cliques of the triangulated graph. There may be several junction trees due to different ways of triangulating an undirected graph, and it is an NP-hard problem to find the optimal triangulated graph (i.e., one which minimizes the sum of the clique potentials) [5]. There exists a number of triangulation techniques in the literature [39, 51, 60, 67], and good heuristics are often used in a real-world application.

Next, we will illustrate how the JT algorithm works on a directed graph by the example shown in Figure 3.3. Figure 3.3(a) shows a cyclic directed graph. One can moralize the graph by connecting nodes Y_2 and Y_5, which share the same child node Y_6, and then droping all arrows. The resulted undirected graph is shown in Figure



(a) Original directed graph

(b) Moralized graph

(c) Triangulated graph

(d) Cliques and separators in the junction tree

Figure 3.3: The Junction Tree algorithm

3.3(b). One way to triangulate the moralized graph is to add edges between nodes Y_2 and Y_3 and nodes Y_3 and Y_5, as shown in Figure 3.3(c). Then, the cliques in the triangulated graph are $c_{123} = \{Y_1, Y_2, Y_3\}$, $c_{235} = \{Y_2, Y_3, Y_5\}$, $c_{256} = \{Y_2, Y_5, Y_6\}$ and $c_{345} = \{Y_3, Y_4, Y_5\}$. Given the set of cliques, we can form a junction tree by joining cliques with edges labeled by a set of *separators*, i.e. s_{23}, s_{25}, and s_{35}, labeled by the intersection of the indices of the adjacent two cliques. The original cyclic directed graph now is transformed to a tree-like structure, and the generated junction tree is shown in Figure 3.3(d). In a tree, marginals of nodes can be computed in a way similar to that in a regular belief propagation algorithm. We first set the separator potentials on the JT to unity, and assign the potentials of the cliques as the product of $P(Y_c|Pa(Y_c))$ in the directed graph where Y_c are variables in one clique. For example, we initialize the potential of clique c_{123} as

$$\psi_{c_{123}} = P(y_1)P(y_2|y_1)P(y_3|y_1), \tag{3.10}$$

and potentials of all separators as 1. Then, messages start being passed between cliques via separators. When a message is passed from clique c_{123} to clique c_{235} via separator s_{23}, a new separator potential is obtained by marginalizing out the variables in clique c_{123} that are not in s_{23}, i.e.

$$\psi^*_{s_{23}} = \sum_{c_{123}\setminus s_{23}} \psi_{c_{123}} = \sum_{y_1} P(y_1)P(y_2|y_1)P(y_3|y_1), \tag{3.11}$$

and a new potential for clique c_{235} is obtained by

$$\psi^*_{c_{235}} = \psi_{c_{235}} \frac{\psi^*_{s_{23}}}{\psi_{s_{23}}}. \tag{3.12}$$

Note that in this message-passing scheme, a clique passes a message to a neighboring cluster only after it has received messages from all the other neighbors. This message propagation procedure continues until updates have been sent along both directions of every edge on the tree. Now, the clique potentials can be read from the JT, and the marginal probability of each variable of interest Y_i can be computed by

$$P(y_i) = \sum_{c\setminus\{Y_i\}} \psi_c, \tag{3.13}$$

where c is the cluster containing Y_i.

It is believed that there cannot be a much more efficient exact inference algorithms than the Junction Tree algorithm in a general loopy graph, since every other approach must contain a hidden triangulation [39].

3.4 GO Hierarchy Vs. Dependencies in Function Labels

In principle, a correct DAG is considered as a model for describing the data generating process. Specifically, given the GO hierarchy, one can assume that protein function annotations are induced accordingly because of the rule of consistent labeling. For example, given the DAG in Figure 1.5, one would expect a set of function annotations $D = \{(V_1), (V_5), (V_1, V_2), (V_1, V_2, V_3), (V_1, V_2, V_5), (V_1, V_2, V_4, V_5), (V_1, V_2, V_3, V_4, V_5)\}$. The BN model has the advantage that it has a simple causal interpretation to model the dependencies in this DAG.

However, a DAG that only partially describes the data generation process may mislead the causality analysis. In particular, there may be spurious dependencies that are entailed by the hidden variables. Given the above set of function labels D, one may find that functions V_1 and V_5 always appear together. This relationship in generating function labels is not represented in the DAG, as the GO hierarchy is organized for biological parent-child relationships. Therefore, constructing a dependency graph by learning from the actual data is the ultimate goal. However, learning an optimal graph structure that best explains the data is an NP-Hard problem [20, 21], since the number of DAGs on L variables is super-exponential in L. Alternatively, artificially adding some arcs into the DAG may help recover the true data-generating relationships that are not shown in the original DAG. Under such a circumstance, the undirected model is advantageous over the directed model, because the undirected model does not assume directions when adding edges into the graph. This DAG issue will be taken into account when we build our CRF model in Section 5.2.

Chapter 4

Function Prediction Using Local SVM Predictors

As reviewed in Section 1.1, for each protein, the CHUGO system produces a set of independent binary SVM predictions, and then uses positive up-propagation to make the final predictions. However, we show that a probabilistic SVM prediction is more effective as an input to a graphical model. This chapter starts with a general introduction to Support Vector Machines, illustrates how to fit the real-valued SVM outputs using a Laplace mixture distribution and a single Laplace, and introduces an Expectation Maximization algorithm for discovering model parameters.

4.1 Support Vector Machines

The Support Vector Machine is a learning algorithm designed to maximize the margin of confidence of a classifier on the training data set. It was first introduced by Vapnik [63] and others [8, 15], and was inspired from theoretical concepts of statistical learning theory. An SVM works by mapping a set of labeled data to a feature space, and finding the "optimal" separating hyperplane in the feature space. This optimal plane maximizes the distances from the hyperplane to the nearest data points i.e. support vectors, and those distances are called *margins*. In practice, even an optimal hyperplane can not separate an arbitrary data set perfectly due to noisy data or insufficient features. Each data point that appears in the region of a different label is known as a *misclassification*. The number of misclassification can be minimized by carefully selecting SVM parameters.

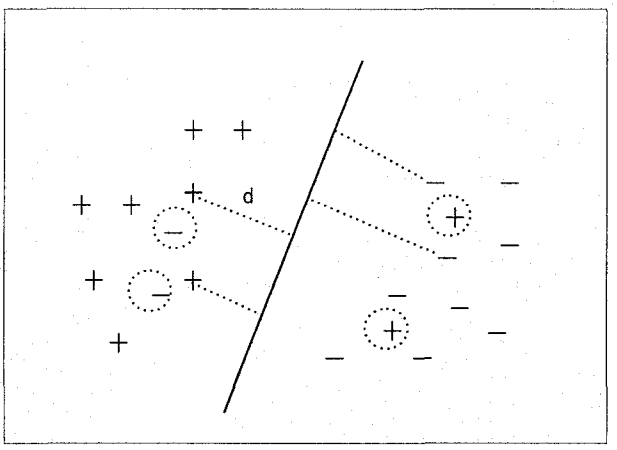


Figure 4.1: A linear Support Vector Machine. Each +/- point represents a training instance. Points (+) are labeled with one class, and points (-) are labeled with the other. d_i is the distance from a data point i to the hyperplane. Circled +/- points are misclassified.

Figure 4.1 shows a linear SVM, in which two classes of data are separated by a straight line. Formally, given a set of training data where each can be represented by a feature vector $\mathbf{x}$, we want to classify each instance as one of two classes in $y \in \{-1, +1\}$. The SVM outputs are signed distances from data points to the hyperplane that can be calculated as:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{4.1}$$

where $\mathbf{w}$ is a vector of weights and b is the offset of the hyperplane. This hyperplane is optimal because the margin between the separable training instances and the hyperplane is made as large as possible. The same hyperplane can typically well separate unseen test instances as well, if they are consistent with the training data.

SVMs can be extended to deal with non-linear boundary in the feature space by introducing different *kernels*, which find separating hyperplanes of higher dimensions. The most popular non-linear kernels include polynomial kernels, radial basis functions (RBFs) and sigmoid functions. These kernel-based SVMs perform better on data that is not linearly separable. However, they tend to over-fit data as they adjust to fit outliers, and the computational cost is also significantly higher than linear SVMs due to its higher dimensionality in the feature space. In all our experiments, we use linear SVMs since they were shown to be efficient and successful in the

CHUGO system.

To make a prediction y on an unseen instance, $\mathbf{x}$, using the weight vector $\mathbf{w}$ and offset b obtained from training, one uses the sign of $f(\mathbf{x})$ in Equation 4.1. y is labeled positive (+1) if $(\mathbf{w}^T\,\mathbf{x} + b) \geq 0$, and is negative (-1) if $(\mathbf{w}^T\,\mathbf{x} + b) < 0$. However, the problem with a binary SVM prediction is that every data point on the same side of the hyperplane is treated equally. However, the original SVM outputs $f(\mathbf{x})$ indicates not only the label (+1 or -1) but also the confidence of this prediction, based on the value of $f(\mathbf{x})$. Intuitively, the larger the absolute value of $f(\mathbf{x})$, the more confident our prediction is. The next section defines a way to learn a new distribution from the real-valued SVM outputs obtained from the training data, and shows how to use the distribution on unseen data to estimate predictive accuracy with a specified probability.

4.2 Probabilistic Support Vector Machines

In many cases, the posterior probability $P(y = \pm 1|f)$ is difficult to compute directly, but instead the class-conditional density $P(f|y = \pm 1)$ is more useful for making a probabilistic prediction based on the SVM output. To fit probabilities to the output of an SVM, Hastie and Tibshirani [35] proposed fitting Gaussians to the class-conditional densities, which was adopted by Barutcuoglu et al. in their hierarchical prediction of GO functions [9]. In a separate study, Lin and Weng [46] investigated modeling the distribution of SVM outputs by a Gaussian and a Laplace, i.e.

$$P(f(\mathbf{x})|\mathcal{N}(\mu,\sigma^2)) = \frac{1}{\sqrt{2\pi}\sigma}\exp(\frac{-(f(\mathbf{x})-\mu)^2}{2\sigma^2}) \tag{4.2}$$

and

$$P(f(\mathbf{x})|\mathcal{L}(\mu,\sigma)) = \frac{1}{2\sigma}\exp(-\frac{|f(\mathbf{x})-\mu|}{\sigma}) \tag{4.3}$$

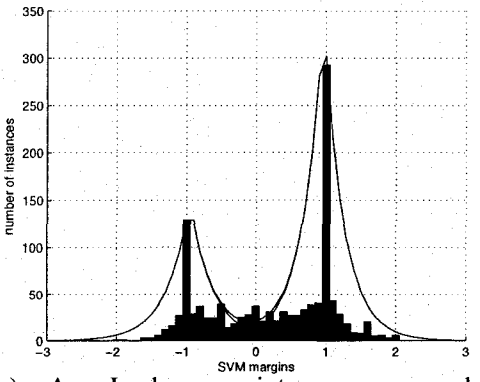
where $\mathcal{N}(\mu,\sigma^2)$ and $\mathcal{L}(\mu,\sigma)$ denotes a Gaussian and Laplace respectively, both characterized by a location parameter μ and a scale parameter σ which can be learned from the training data. Lin and Weng also showed that, in all their experiments, Laplace estimation outperformed Gaussian. It is also worth noting that Platt [50] introduced a direct approach to model the distribution of SVM outputs

$P(y = \pm 1|f)$ using a sigmoid model.

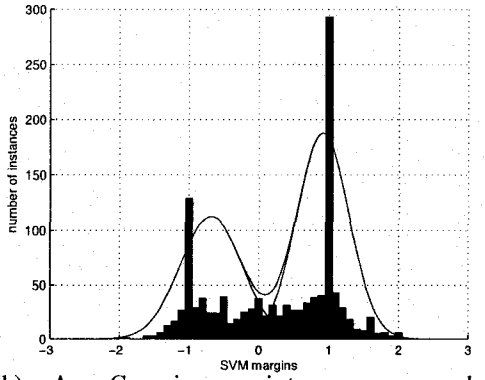
Although those studies have chosen different models to fit SVM outputs, they share some common assumptions and methodologies. First, they all assume that the distribution of the target value y depends on its input $\mathbf{x}$ only through the predicted value $f(\mathbf{x})$. In theory, the distribution of SVM outputs may depend on the input $\mathbf{x}$, and the length of the predictive interval may vary with different input values. However, the assumption often works well in practice and provides a good estimate for initial analysis. Second, they all assume that the SVM outputs, $f(\mathbf{x})$, are generated independently, and thus the class-conditional density $P(f|y = \pm 1)$ can be modeled by simple parametric functions. These assumptions for fitting SVM outputs also apply for this thesis work. In terms of initial estimation, they all use histograms which give a visual representation of the SVM output distribution for more sophisticated analysis.

4.3 Fitting Local SVM Outputs Using a Laplace Mixture Model

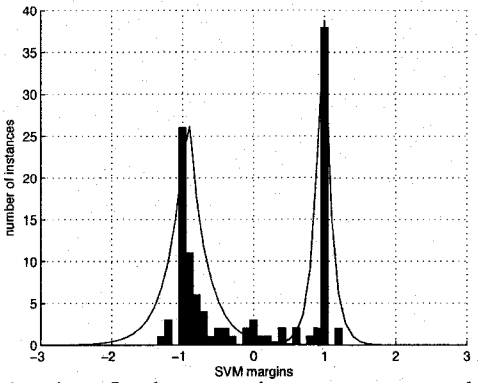
Given the SVM outputs for each GO node, we want to estimate the class-conditioned density $P(f|y = \pm 1)$. Parameters of $P(f|y = +1)$ can be estimated from the positive examples of that class, and parameters of $P(f|y = -1)$ can be estimated from the negative examples of that class. Figure 4.2 shows histogram plots of SVM outputs obtained from the positive training instances on two nodes GO0030528 and GO0030246, and Figure 4.3 shows histogram plots of SVM outputs obtained from the negative instances on these nodes. Since there are significantly more negative training instances than positive instances in almost all GO classes, SVM performance on positive examples is consistently worse than on negatives. As the histograms show, SVM outputs from positive examples tend to spread more from the two centers, +1 and -1, than SVM outputs from negative examples. This indicates that the SVM is more accurate for negative examples than for positive ones. The histograms also show that a Laplace appears to model the SVM output distribution more closely than a Gaussian distribution. However, we show that we can model



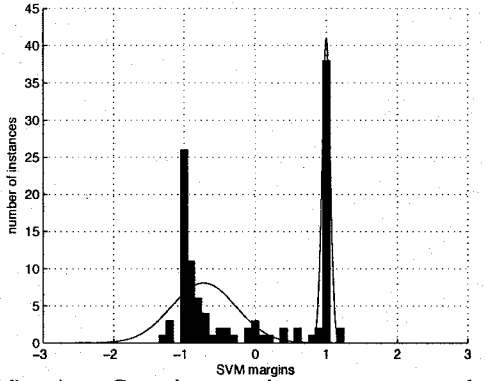
(a) A Laplace mixture on node GO0030528. Parameters of two Laplaces: $\theta_1 = (0.32, -0.87, 0.33)$ and $\theta_2 = (0.68, 0.93, 0.33)$.



(b) A Gaussian mixture on node GO0030528. Parameters of two Gaussians: $\theta_1 = (0.41, -0.68, 0.44)$ and $\theta_2 = (0.59, 0.91, 0.37)$.

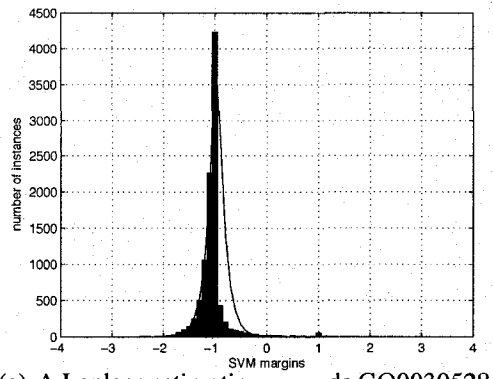


(c) A Laplace mixture on node GO0030246. Parameters of two Laplaces: $\theta_1 = (0.59, -0.93, 0.25)$ and $\theta_2 = (0.41, 0.98, 0.11)$.

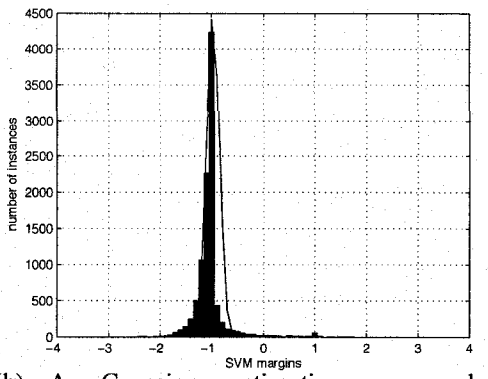


(d) A Gaussian mixture on node GO0030246. Parameters of two Gaussians: $\theta_1 = (0.61, -0.72, 0.46)$ and $\theta_2 = (0.39, 1.0, 0.057)$.

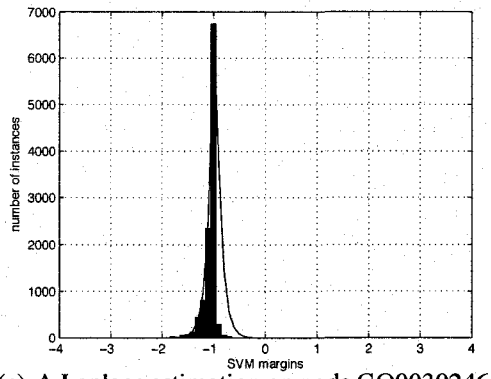
Figure 4.2: The histograms of SVM outputs obtained from *positive* training instances. SVM outputs obtained from positive examples concentrate at +1 and -1. Both the Laplace and Gaussian mixtures are parameterized by $\theta_1 = (\pi_1, \mu_1, \sigma_1)$ and $\theta_2 = (\pi_2, \mu_2, \sigma_2)$ where π denotes the weight of a particular component, μ denotes the location parameter, σ denotes the scale parameter. Note the y-axis has different scales based on the number of instances in different classes.



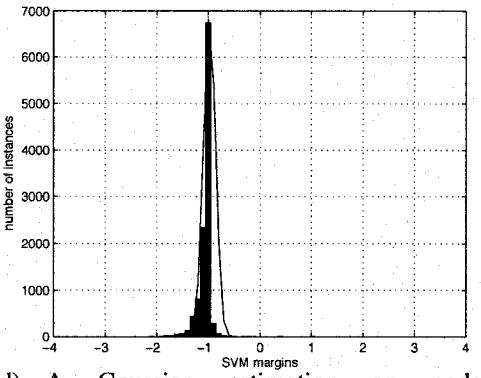
(a) A Laplace estimation on node GO0030528 with $\mu = -0.96$ and $\sigma = 0.145$.



(b) A Gaussian estimation on node GO0030528 with $\mu = -0.98$ and $\sigma = 0.125$.



(c) A Laplace estimation on node GO0030246 with $\mu = -0.97$ and $\sigma = 0.105$.



(d) A Gaussian estimation on node GO0030246 with $\mu = -0.98$ and $\sigma = 0.115$.

Figure 4.3: The histograms of SVM outputs obtained from *negative* training instances. SVM outputs obtained from negative examples are centered at -1. Both the Laplace and Gaussian are parameterized by $\theta = (\mu, \sigma)$. Note the y-axis has different scales based on the number of instances in different classes.

the output more accurately by using a mixture of two Laplace distributions for SVM positive outputs and a single Laplace distribution for negative outputs. In our experiment, we did implement a system with a Gaussian mixture but the result was worse than a Laplace mixture.

It is simple to model a single Laplace, as defined by Equation 4.3 from the negative samples. Given N SVM outputs $(x_1, x_2, \ldots, x_N)$ from negative training examples for the same class, location parameter μ is the median of these samples [42] and the estimator of the scale parameter σ is

$$\sigma = \frac{1}{N} \sum_{i=1}^{N} |x_i - \mu|. \tag{4.4}$$

It becomes somewhat complicated to model a Laplace mixture distribution. Given a mixture of K Laplaces, the goal is to estimate a set of unknown parameters $\theta = \{(\pi_1, \mu_1, \sigma_1), \ldots, (\pi_K, \mu_K, \sigma_K)\}$ where π_k denote the proportion or weight of the kth Laplacian of the mixture. We use the Expectation Maximization (EM) algorithm for the maximum likelihood estimate of the parameters.

The EM algorithm was well described by Dempster et al. in 1977 [27], and it has been frequently used for data clustering in machine learning. EM alternates between performing an expectation (E) step, which computes an expectation of the likelihood, and a maximization (M) step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood discovered on the E step. Then the parameters found on the M step are used to start another E step. EM iterates until the likelihood converges to a local maximal.

As we are using a mixture of two Laplaces to fit the distribution of the SVM outputs obtained from the positive training examples, the probability density of a set of SVM outputs $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ can be modeled by

$$P(\mathbf{x}|\theta) = \prod_{j=1}^{N} \sum_{k=1,2} P(k, x_j|\theta), \tag{4.5}$$

where $P(k, x_j|\theta)$ denotes the joint probability of sampling the kth Laplace model and sampling a particular SVM output x_j from this Laplace component, i.e.

$$P(k, x_j|\theta) = P(k|\theta)P(x_j|k, \theta) = \pi_k P(x_j|k, \theta). \tag{4.6}$$

Given an initial $\theta^0 = \{(\pi_1, \mu_1, \sigma_1), (\pi_2, \mu_2, \sigma_2)\}$ [1], EM tries to find θ^{i+1} that maximizes the expected value of the log-likelihood of Equation 4.5 given the data $\mathbf{x}$ and the parameter θ^i from a previous iteration, i.e.

$$\theta^{i+1} = \arg\max_{\theta} Q(\theta|\mathbf{x}, \theta^i), \tag{4.7}$$

where

$$\begin{aligned}
Q(\theta|\mathbf{x}, \theta^i) &= E\left[\log P(\mathbf{x}|\theta) \,|\, \mathbf{x}, \theta^i\right] \\
&= E\left[\log \prod_{j=1}^{N} \sum_{k=1,2} P(k, x_j|\theta) \middle| \mathbf{x}, \theta^i\right] \\
&= E\left[\sum_{j=1}^{N} \log \sum_{k=1,2} P(k, x_j|\theta) \middle| \mathbf{x}, \theta^i\right] \\
&= \sum_{j=1}^{N} E\left[\log \sum_{k=1,2} P(k, x_j|\theta) \middle| x_j, \theta^i\right] \\
&= \sum_{j=1}^{N} \sum_{k=1,2} P(k|x_j, \theta^i) \log P(k, x_j|\theta) \\
&= \sum_{j=1}^{N} \sum_{k=1,2} P(k|x_j, \theta^i) \log[\pi_k P(x_j|k, \theta))].
\end{aligned} \tag{4.8}$$

The likelihood $P(x_j|k, \theta)$ can be calculated by the Laplace density function as shown in Equation 4.3. The probability that x_j comes from the kth Laplace of the mixture $P(k|x_j, \theta^i)$ can be estimated by

$$P(k|x_j, \theta^i) = \frac{\pi_k P(x_j|k, \theta^i)}{\sum_{k=1,2} \pi_k P(x_j|k, \theta^i)}. \tag{4.9}$$

Using Equation 4.9 to replace $P(k|x_j, \theta^i))$ in Equation 4.8, we can optimize $Q(\theta)$ by taking partial derivatives of Equation 4.8 subject to $\pi_1 + \pi_2 = 1$.

As the sample median is the maximum likelihood estimator of location parameter μ in a single Laplace, the maximum likelihood estimator of μ for a mixture of two Laplaces is a weighted median [23, 68]. The weighted median is the value β that minimizes the following expression

$$J(\beta) = \sum_{i=1}^{N} w_i |x_i - \beta|, \tag{4.10}$$

[1] Parameters estimated at the ith iteration are denoted by a superscript i.

where weight $w_i = P(k|x_i, \theta)$. Then, the weighted median is selected based on the following two situations:

$$\beta = \left\{ \begin{array}{ll} x_M, & \text{if } \sum_{i=1}^{M} w_i > \frac{1}{2} \sum_{i=1}^{N} w_i \\ \frac{1}{2}(x_M + x_{M+1}), & \text{if } \sum_{i=1}^{M} w_i = \frac{1}{2} \sum_{i=1}^{N} w_i \end{array} \right\} \tag{4.11}$$

For example, consider a set of SVM outputs $\mathbf{x} = [0.7, 0.9, 0.24, 1.1, 0.8]$ associated with weights $\mathbf{w} = [0.1, 0.2, 0.3, 0.2, 0.1]$. After sorting the $\mathbf{x}$ vector, we obtain the sorted SVM outputs with the corresponding weights.

x	1.1	0.9	0.8	0.7	0.24
w	0.2	0.2	0.1	0.1	0.3

Starting from the left, add the weights until the sum is greater than or equal to $\frac{1}{2} \sum_{i=1}^{5} w_i = 0.45$. By adding the weights of the first three outputs (i.e. $M = 3$), the sum is 0.5 exceeding 0.45. The weighted median is therefore 0.8. Figure 4.4 summarizes the EM algorithm for a mixure of two Laplaces.

(Note: Parameters estimated at the ith iteration are denoted by a superscript i.)

Input: data $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ where $\mathbf{x} \in R^N$.

Output: parameters for two Laplaces $\theta_1^{i+1} = (\pi_1^{i+1}, \mu_1^{i+1}, \sigma_1^{i+1})$ and $\theta_2^{i+1} = (\pi_2^{i+1}, \mu_2^{i+1}, \sigma_2^{i+1})$.

Initialization: θ_1^0 and θ_2^0.

While $\theta_1^{i+1} - \theta_1^i > \alpha$ or $\theta_2^{i+1} - \theta_2^i > \alpha$ where α is a pre-defined threshold, do

- **E-step:** Compute the posterior probabilities for all $k = 1, 2$ and $j = 1, \ldots, N$:

$$P^i(k|x_j, \theta^i) = \frac{\pi_k^i P(x_j|k, \theta^i)}{\sum_k \pi_k^i P(x_j|k, \theta^i)} \tag{4.12}$$

  where $P(x_j|k, \theta^i) = \frac{1}{2\sigma_k^i} \exp(\frac{|x-\mu_k^i|}{\sigma_k^i})$.

- **M-step**

$$\mu_k^{i+1} = \arg\min_{\beta} \sum_{j=1}^{N} P^i(k|x_j, \theta^i)|x_i - \beta| \tag{4.13}$$

$$\sigma_k^{i+1} = \frac{1}{2} \sum_{j=1}^{N} P^i(k|x_j, \theta^i)|x_j - \mu_k^{i+1}|, \tag{4.14}$$

$$\pi_k^{i+1} = \frac{1}{N} \sum_{j=1}^{N} P^i(k|x_j, \theta^i). \tag{4.15}$$

Figure 4.4: An EM algorithm for mixing of two Laplaces.

Chapter 5

Hierarchical Prediction of GO Function Using Graphical Models

The hierarchical relationship in the GO provides valuable knowledge for constructing a model to make consistent function prediction. We build two graphical models, Bayesian network and Conditional Random Fields, each of which augment structural information of the hierarchy to make local SVM predictions with more globally consistent function prediction. This chapter illustrates how each model is constructed from the hierarchy, and describes approaches for parameter estimation and inference methods.

5.1 Hierarchical Prediction Using Bayesian Networks

5.1.1 Model: Bayesian Networks

Since the GO hierarchy is organized as a DAG, it is natural and convenient to induce a dependency graph based on the parent-child relationships in the hierarchy. We use the hierarchical structure of the GO illustrated in Figure 1.5 to construct a Bayesian network as illustrated in Figure 1.6(a). The network spans two sets of variables, i.e. $Y = (Y_1, Y_2, \ldots, Y_L)$ and $X = (X_1, X_2, \ldots, X_L)$. Each node Y_i represents a GO function and has two possible states, +1 for positive and -1 for negative. Each node Y_i represents an output from the local SVM predictor which is a real number. Node Y_i is conditioned on its parent classes, and local observed node X_i is conditioned on the directly connected node Y_i. The edges between Y nodes encourage hierarchical consistency in the graph. By arranging entries in the CPT, we can ensure that a node

is guaranteed to be positive if any of its children is positive (+1) and to be negative if any of its parents is negative (-1). The edge from each Y_i to the corresponding local observation X_i represents the predictive accuracy of the local SVM classifier. The constructed Bayesian network is able to make hierarchical prediction compatible with the consistent-labeling rule by considering the parent-child relationship in the hierarchy, and to provide more accurate predictions by integrating the probabilistic outputs from the local predictors.

Two assumptions have been made by this particular Bayesian network structure. First, each local prediction X_i is independent from the other local predictions and the other GO nodes, given its direct parent Y_i. Second, each GO node Y_i is conditionally independent from all non-descendants, given its immediate parents. As discussed in Section 3.4, the second assumption is not valid because the GO hierarchy may not be the true dependency model that is used to generate the function labels. However, it is reasonable to believe that the GO hierarchy is part of the true dependency graph, and more likely forms the fundamental structure of the graph. Therefore, the GO hierarchy is used to build the Bayesian network in this research.

For this network structure, our goal is to find the maximal joint probability distribution, also known as the most probable explanation (MPE), given a set of local SVM outputs x for a query protein. Mathematically, given a set of observations $\mathbf{x}$, we want to find a set of labels $\mathbf{y}^*$ so that

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}). \tag{5.1}$$

By Bayes' rule, the joint conditional probability can be written as

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} P(\mathbf{y}) P(\mathbf{x}|\mathbf{y}), \tag{5.2}$$

where the normalization factor is $Z = \sum_{\mathbf{y}} P(\mathbf{y})P(\mathbf{x}|\mathbf{y})$. Due to the assumptions on our Bayesian network, the two terms in Equation 5.2 can be simplified as

$$P(\mathbf{y}) = \prod_{i=1}^{L} P(y_i|Pa(y_i)) \tag{5.3}$$

where $Pa(y_i)$ denotes all parent nodes of y_i, and

$$P(\mathbf{x}|\mathbf{y}) = \prod_{i=1}^{L} P(x_i|y_i). \tag{5.4}$$

Next, let us denote $P(y_i|Pa(y_i))$ and $P(x_i|y_i)$ as parameters $\theta_{y_i|Pa(y_i)}$ and $\theta_{x_i|y_i}$ respectively, and we will discuss how to estimate these parameters from the training data in the next section.

5.1.2 Parameter Estimation and Inference

The parameter $\theta_{y_i|Pa(y_i)}$ represents hierarchical structure imposed by the graph, and indicates how likely a node y_i will be labeled as positive or negative given the assignment of all its parent nodes. We construct a conditional probability table (CPT) at each node to enforce the inheritance properties of the GO hierarchy. As described in Section 1.2, the rule of consistent labeling in the hierarchy simplifies the CPTs in the Bayesian network, and only one parent-child configuration, in which y_i is positive given all its parents being positive, needs to be computed from the true labels of training data. The CPT entries for all the other parent-child configurations, i.e. y_i is positive given that *any* of its parents is negative, are merely 0's. For example, given the BN in Figure 1.6(a), the CPT at node Y_4 would be:

y_2	y_5	$+y_4$	$-y_4$
$-$	$-$	0	1
$-$	$+$	0	1
$+$	$-$	0	1
$+$	$+$	α	$1-\alpha$

where Y_2 and Y_5 are parents of Y_4, and α is a probability computed from the training data. This CPT encodes the fact that a negative label for either Y_2 or Y_5 implies a negative label for Y_4.

The other parameter $\theta_{x_i|y_i}$ represents the predictive accuracy of the local SVM classifier, and represents reliability of an SVM prediction given the knowledge of the true label y_i. Given the assumption that local SVMs have similar performance on the training data and the test data, $P(x_i|y_i)$ can be estimated by aggregating SVM cross-validation results on the training data. For binary SVM outputs, $P(x_i|y_i)$ can be estimated using the confusion matrices from cross validation. For example, $P(+x_i| + y_i)$ represents the ratio of positive instances predicted correctly, i.e. TP/(TP+FN) where TP, true positives, refers to positive instances classified correctly and FN, false negatives, refers to positive instances classified incorrectly.

Since our local SVM prediction was modified to output the raw SVM margins, x_i is a real number rather than a binary bit, and therefore the estimation technique described in Section 4.3 is used for fitting the likelihood term $P(x_i| \pm y_i)$.

The Junction Tree algorithm described in Section 3.3.3 is utilized for inference in the BN.

5.2 Hierarchical Prediction Using Conditional Random Fields

The Bayesian network models dependencies among GO terms using the arrows in the original GO graph. To generalize the second assumption made by the BN, i.e. function annotations are only generated according to the GO hierarchy, we would like to take the GO hierarchy and add edges between spouse nodes who share a common child term and between sibling nodes who share a common parent node.

We cannot use a BN for this generalization, so to model the parent-child, spouse-spouse and sibling-sibling dependencies, we use an undirected graphical model, the Conditional Random Field (CRF). This model has been widely used for studies of complex graph dependencies.

5.2.1 Model: Conditional Random Fields

The undirected graph is constructed by taking the Bayesian network, adding edges to all pairs of spouse nodes and sibling nodes, and dropping arrows in the directed graph. An undirected graph is shown in Figure 1.6(b), and it is converted from the corresponding BN in Figure 1.6(a). Given the undirected graph $G = (V, E)$, two types of cliques are defined: edge cliques C_E and vertex cliques C_V. Edge cliques C_E, the local-consistency factor, includes all edges, i.e. parent-child, spouse-spouse and sibling-sibling edges in the proposed undirected graph. Node cliques C_V is defined to capture dependencies between the local predictions and the class labels. For simplicity, a pairwise neighborhood system is adopted for modeling the label consistency structure. For example, for Figure 1.6(b), edge cliques $C_E = \{(Y_1, Y_2), (Y_2, Y_3), (Y_2, Y_4), (Y_2, Y_5), (Y_3, Y_4), (Y_4, Y_5)\}$, and vertex cliques $C_V = \{(X_1, Y_1),$

(X_2, Y_2), (X_3, Y_3), (X_4, Y_4), $(X_5, Y_5)\}$.

Given local SVM outputs, the conditional distribution over labels Y is defined as:

$$P(y|x) = \frac{1}{Z} \exp\{ \sum_{(v_i,v_j)\in E} \psi_E(y_i, y_j) + \sum_{v_i \in V} \psi_V(y_i, x_i)\} \tag{5.5}$$

where $\psi_E(y_i, y_j)$, the transition function, denotes the potential over a pair of neighbor nodes, $\psi_V(y_i, x_i)$, the state-observation function, denotes the potential over node v_i, and Z is the partition function. Each $\psi_E(y_i, y_j)$ can be expressed as a linear function i.e.

$$\psi_E(y_i, y_j) = \sum_{k=1}^{4} \theta_k f_k(y_i, y_j), \tag{5.6}$$

where $f_k(y_i, y_j)$ is the indicator function of state assignments over a pair of nodes (y_i, y_j) and consists of four possible features for each pairwise clique, i.e. $f(-y_i, -y_j)$, $f(-y_i, +y_j)$, $f(+y_i, -y_j)$ and $f(+y_i, +y_j)$. The formulation of function $f_k(y_i, y_j)$ captures co-occurrences between labels in the hierarchy.

Since $\psi_V(y_i, x_i)$ captures relationships between the class label and local SVM prediction, we define the feature function as the class-conditioned density function $P(x_i| \pm y_i)$ as described in Section 4.3.

5.2.2 Parameter Estimation and Inference

The proposed CRF model needs to estimate the parameters of the transition function and state-observation function. Since parameters of the state-observation function $\psi_V(y_i, x_i)$ are defined by the likelihood function $P(x_i|y_i)$ of the SVM outputs, they are computed as described in Section 4.3, i.e. parameters for a Laplace mixture and a single Laplace.

Parameters of the transition function, i.e. θ_k in Equation 5.6 for each edge in the graph, can be estimated by counting the occurrences of all possible assignments of two nodes associated with the edge in the training data. For a spouse-spouse and sibling-sibling edge, all four assignments of (y_i, y_j) are possible, while only three assignments are possible for a parent-child edge since $f(-y_i, +y_j)$ never appears due to the rule of consistent labeling.

Inference in CRFs is implemented using the same algorithm as in BNs (described in Section 5.1.2), the Junction Tree algorithm, except there is no moralisation step since it is already an undirected graph.

Chapter 6

Experiments for Hierarchical Protein Function Prediction

We conducted our experiments on two different protein data sets using SVM, BN, and CRF classifiers. This chapter introduces the performance evaluation technique, describes the two data sets used in the experiments, presents the experimental results, and concludes with a discussion of the results.

6.1 Evaluation

For a fair evaluation of a predictive system, part of the data, called the test set, must be withheld from training. We use a modified 5-fold cross validation technique to split our data and evaluate the prediction results. Prediction performance is evaluated based on the standard F-measure score.

Cross Validation

To evaluate the performance of our proposed models on unseen data, we perform all our experiments using 5-fold cross validation. In a standard 5-fold cross validation, the data is divided into five folds, and each fold of data contains a similar number of instances for each class. Classifiers are trained on any four folds of data and tested on the withheld fold. This procedure continues until every fold has been tested using the classifier trained on the other four folds. In our task, there are two phases. In phase 1, there is a need for estimating the local SVM output distribution from the training data. In phase 2, we estimate the parameters of the graphical model. If

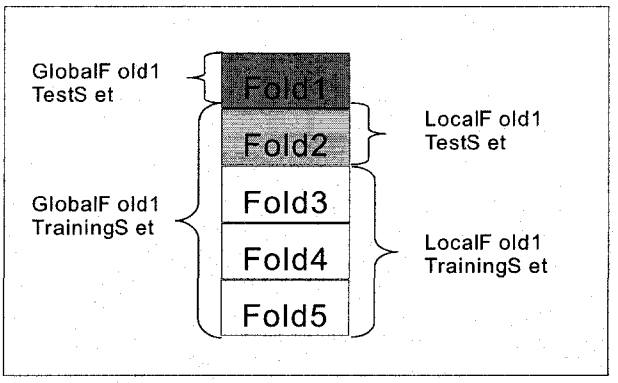


Figure 6.1: Two-phase 5-fold cross validation. Fold1 is held out as a test set for predicting functions globally using graphical models. The other four folds are combined as a training set, in which Fold2 is held out for testing the SVMs trained on the other three folds. The local cross validation continues until every fold in training has been tested.

we perform 5-fold corss validation in all the data to construct the local SVMs then all the data will be used in estimating the parameters of the graphical model and no unused test data will be available. Therefore, we first divide the data into 5 folds for use in phase 2. To estimate the parameters of the local SVMs, we do 4-fold cross validation in the training data by using 3 folds for training and 1 fold for testing. This leaves a fold for testing the parameters of the graphical model.

Specifically, in the four-fold training data, one fold is held out for testing the SVM classifiers trained on the other three, called a local cross validation, and then it is put back to the training data and another fold is pulled out for test. This process continues until every fold in the training set has been tested. The SVM outputs of all four SVM training folds are combined to make a distribution estimation using the technique described in Section 4.3. The same operations are applied to the other global folds. In total, we have trained SVMs $C_3^5 = 10$ times on any three folds. One iteration of this two-phase cross validation technique is shown in Figure 6.1.

Performance Measures

The standard F-measure is adopted for performance comparison between different models. F-measure is defined as

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6.1}$$

where the precision measures the performance of a classifier's positive predictions and is defined as

$$Precision = \frac{TP}{TP + FP}, \tag{6.2}$$

and the recall measures the percentage of the positive instances that are predicted as positive and is defined as

$$Recall = \frac{TP}{TP + FN}, \tag{6.3}$$

where TP is the number of true positives, FP tallies false positives and FN denotes the number of false negatives.

6.2 Data Set

We create two protein data sets for hierarchical function prediction using different models. As discussed in Section 2.1, each data sets consists of three components: protein sequences, GO hierarchy, and known protein labels - i.e. functions. To create more confident labellings, only function labels that were derived from a biological experiment are considered in this dissertation. That is, we only include labels identified by evidence code IDA, IEP, IGC, IGI, IMP, and TAS, and exclude IC, IEA, ISS, NAS, ND, RCA, and NR. Following the consistent labeling approach, each experimentally-annotated label is propagated up to the root node. Since the root term GO0003674 *molecular function* is true for all proteins, it is removed from the label set for prediction.

To directly compare with the prediction performance obtained by Eisner et al., the same data set is used. The data set (data set 1) consists of the Uniprot release 2.0 (TrEMBL release 27 and Swiss-Prot release 44), August 28, 2004 version of the GO molecular function ontology, and August 11, 2004 version of the GOA mapping file. A sufficient number of positive training instances is required to create accurate local function predictors. To be consistent with Eisner et al [29], we also set the minimum number of positive instances at a node after the label propagation to 20. This leaves us 399 nodes in the pruned GO hierarchy[1], and 14,018 proteins.

[1]There were 406 GO nodes in Eisner's experiments. However, having confirmed with the author, there is a bug in the code used to extract the GO terms, so 7 nodes were removed.

To evaluate the performance of the proposed models on a larger scale and more recent data, we create another data set (data set 2), which includes the Uniprot release 10.0 (TrEMBL release 35 and Swiss-Prot release 52), June 5, 2007 version of the GO molecular function ontology, and August 17, 2007 version of the GOA. A cutoff of 20 positive instances at each node is preserved, and the final version of the data set contains 792 GO terms and 45,956 unique proteins.

Table 6.1 shows the complexity of the constructed GO hierarchies in terms of hierarchy depth[2], number of parents and number of children. Overall, both hierarchies are complex and deep.

6.3 Experimental Results

Protein similarity search and feature extraction are implemented by following the steps in Eisner et al. Proteins in data set 1 and 2 are BLASTed against Swiss-Prot database 44 and 49, respectively, and the cutoff of E-value is set to be 10^{-3}. PA features for each similar protein obtained from BLASTing are extracted from the main entries of the Swiss-Prot database, including the *Keywords*, *SUBCELLULAR LOCATION*, and *InterPro* fields.

The result from local SVM predictors, without propagating positive predictions up to the root, is used as the baseline. A linear kernel is chosen for all experiments, and the penalty parameter in SVM is set as $C = 1$. LibSVM [19] is adopted for the implementation of SVM.

A Java tool for modeling Bayesian networks, SamIam [26] is adopted to implement the Bayesian networks, and a CRF tool, GRMM [58] is modified to implement the CRF model.

Table 6.2(a) and 6.2(b) shows prediction results using SVM without up-propagation (SVM LOCAL), SVM with up-propagation (SVM UP), BN and CRF on two data sets, respectively. SVM UP may improve or deteriorate the overall prediction performance, as the F-score has increased by 0.28% in data set 1 and decreased by 0.17% in data set 2 by simply propagating up positive local SVM predictions. The

[2]In case of multiple parents, the hierarchy depth is defined by the longest path from the root to the node.

(a) Number of nodes in terms of depth.

Depth in Hierarchy	Number of Nodes	
	Data set 1	Data set 2
0	10	11
1	54	76
2	98	138
3	97	178
4	72	160
5	31	106
6	27	68
7	9	49
8	1	5
9	0	1
Total number of nodes	399	792

(b) Number of nodes in terms of number of parents.

Number of Parents	Number of Nodes	
	Data set 1	Data set 2
0	10	11
1	330	625
2	53	141
3	4	15
4	1	0
5	1	0
Total number of nodes	399	792

(c) Number of nodes in terms of number of children.

Number of Children	Number of Nodes	
	Data set 1	Data set 2
0	173	384
1	132	212
2	46	88
3	21	43
4	10	23
5	8	18
6	2	7
7	2	6
> 7	5	11
Total number of nodes	399	792

Table 6.1: Complexity of GO hierarchies in terms of the hierarchy depth, number of parents and number of children.

(a) Experimental results for data set 1.

	TP	FP	FN	Precision	Recall	F-score	Std. Dev.
SVM LOCAL	44,285	12,537	25,155	0.7794	0.6377	0.7015	0.00872
SVM UP	45,008	13,357	24,432	0.7711	0.6482	0.7043	0.00665
BN	48,023	14,538	21,417	0.7676	0.6916	0.7276	0.00530
CRF	48,465	14,725	20,975	0.7670	0.6980	0.7309	0.00693

(b) Experimental results for data set 2.

	TP	FP	FN	Precision	Recall	F-score	Std. Dev.
SVM LOCAL	171,486	62,313	72,754	0.7335	0.7021	0.7175	0.01366
SVM UP	173,438	66,953	70,802	0.7215	0.7101	0.7158	0.01463
BN	176,884	65,735	67,356	0.7291	0.7242	0.7266	0.01491
CRF	177,105	64,272	67,135	0.7337	0.7251	0.7294	0.01447

Table 6.2: Experimental results by using SVM LOCAL, SVM UP, BN, and CRF.

F-measure has increased by 2.61% and 0.91% for the two data sets, respectively, by using the BN directed model. The increases of F-measures are due primarily to the improvement on the poor recalls, which raise from 63.77% and 70.21% to 69.16% and 72.42%, with a small sacrifice of the precisions (down by 1.18% and 0.44%). By applying the proposed CRF method, the F-score has increased by 2.94% and 1.19%, respectively, and the recall has increased by 6.03% and 2.3% with a little (down by 1.24%) or no loss of precision. The increases of F-measures on data set 1, due to BN and CRF, are both statistically significant.

Table 6.3 shows the number of GO nodes whose F-measures have been changed due to the use of SVM UP, BN, and CRF. Clearly, the straightforward up-propagation method has a limited influence on prediction of a small number of nodes, while BN and CRF can improve prediction performance on over half of the GO nodes. Figure 6.2 and 6.3 shows the F-score of BN and CRF compared to the F-score of SVM local in two data sets, respectively.

6.4 Discussion

SVM UP is a simple and quick operation that forces the final predictions to be consistent, but it is experimentally proved to be unstable and may hurt overall performance, such as on data set 2. Given sufficient training instances, confident statistics

F-measure	Data set 1			Data set 2		
	SVM UP	BN	CRF	SVM UP	BN	CRF
Increased	45	219	261	53	328	493
Decreased	24	71	114	106	187	259
No change	330	109	24	633	277	40
Total	399			792		

Table 6.3: Number of nodes whose F-measures have been increased, decreased or unchanged by using SVM UP, BN and CRF than using SVM LOCAL.

can be learned from the dependencies between nodes and SVM outputs, and probabilistic inferences are provided by the graphical models, i.e. BN and CRF, to make "selective" up-propagations and down-extensions.

By using graphical models, the recalls have significant increases which contributes most to the improvement of F-measures. The change of recall is determined by the number of true positives since the total number of positives is fixed for a given data set. There are significant increases of true positives in data set 1 and 2, which is a result of two factors from the graphical models:

1. Using the estimated distribution of local SVM outputs, instead of binary values.

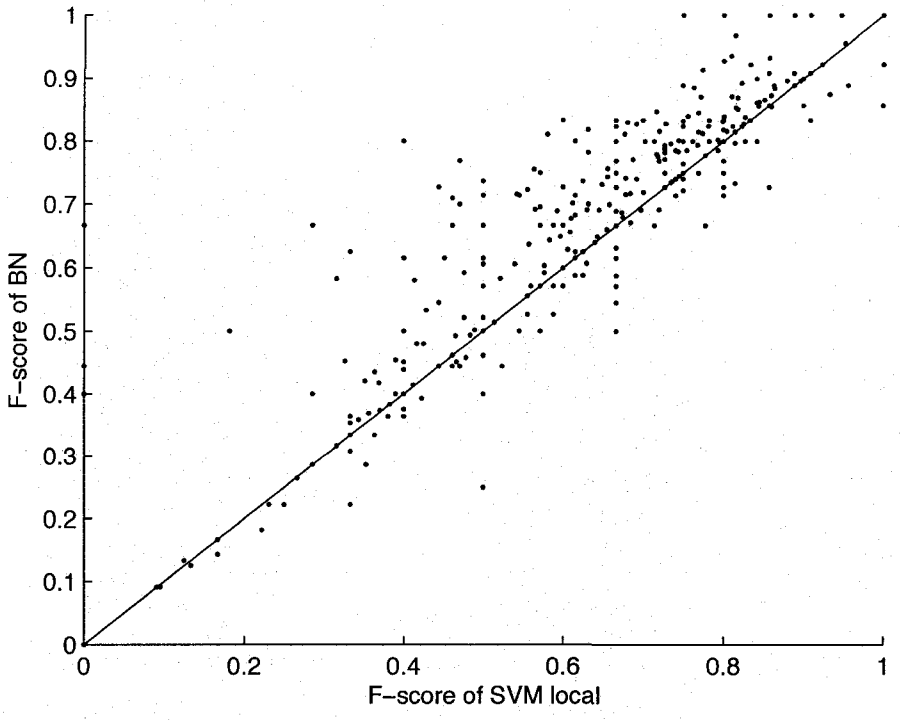
2. Using hierarchical information in the GO to extend a positive prediction to its children.

As discussed in Section 4, a binary SVM predictor treats SVM outputs on the same side of the hyperplane equally, while an estimated SVM output distribution integrates the confidence of making such a local prediction. The Laplace Mixture estimation model we built, based on the positive instances, could turn some former FN predictions to be "less" negative for having some probability of being positive. It works even better when there is a larger number of FNs, i.e. poorer recall, such as in data set 1. The second factor makes additional TPs possible because the probability $P(+y_i| + Pa(y_i))$ in BN or $P(+y_i, +Pa(y_i))$ in CRF learned from training data on a query node may indicate it is very likely that the node is positive given that all its parents are positive.

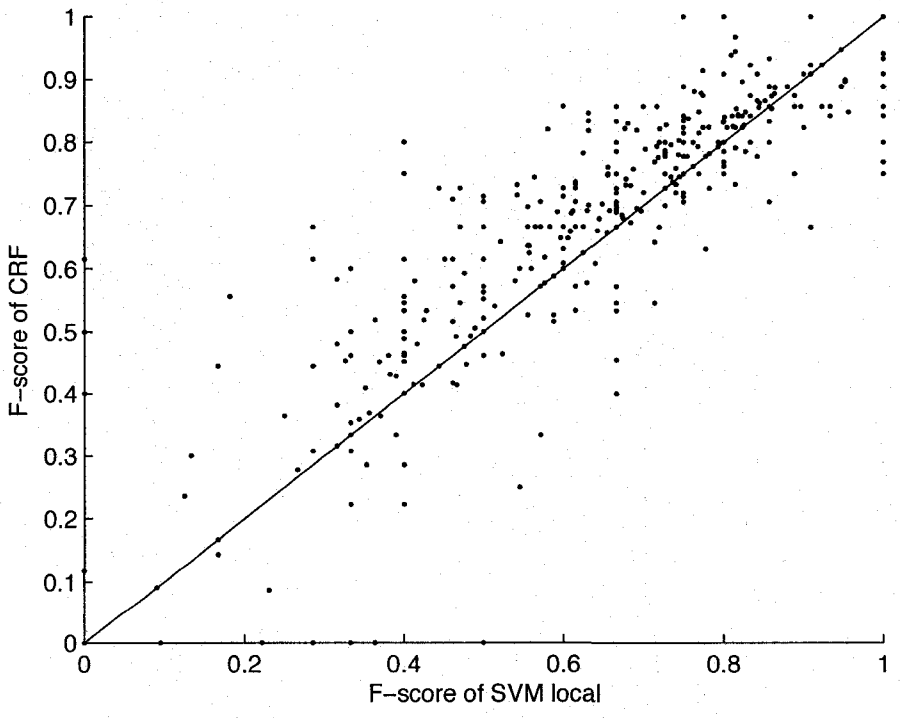
Figures 6.4 and 6.5 shows the F-score difference between BN/CRF and SVM local in terms of the number proteins belonging to each class in the two data sets, respectively. Note, the x-axis is scaled logarithmically on the number of proteins. From those figures, one can tell that most changes (increase or decrease) of the F-measures occur at node classes that have fewer training instances, and so presumably are (typically) at a lower level of the graph. There are several reasons for this observation. First, the fact that lower level nodes have a smaller number of proteins leads to two consequences: (1) adding or removing a few TP predictions for those classes can change the F-measure significantly; (2) statistical information, including both the dependency relationship and SVM output estimation, collected from a small number of training instances, may not truly represent unseen data. Also note, the performance of our graphical models at higher level nodes is mostly better or not worse than the local SVM. This observation indicates that, if given adequate training instances, the hierarchical information and SVM output estimation captured by our graphical models can almost certainly improve the result.

As we attempt to characterize GO nodes whose F-measures have been increased or decreased by the graphical models, we do not find any clear evidence that can be used to identify those nodes apriori. Figures 6.6 and 6.7 shows error bars, with respect to the hierarchy depth and number of parents, of F-score changes between using SVM LOCAL, and BN and CRF, respectively. There is no discernable pattern in any of the graphs due to the large variances at each level.

Although the undirected graphical model CRF only has marginal improvement of the F-measure over the BN model, it shows that co-occurrences between GO terms other than parent-child pairs do exist and they are helpful in constructing a more accurate dependency graph. This result sheds light on the future research of learning the dependency graph from the actual data.

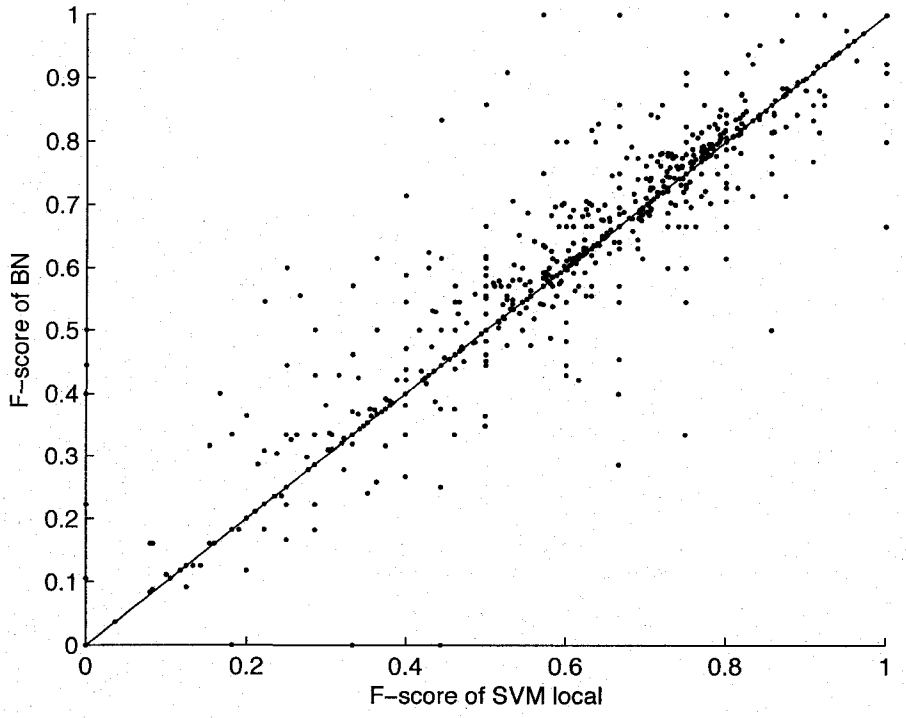


(a) F-score of BN Vs. F-score of SVM local

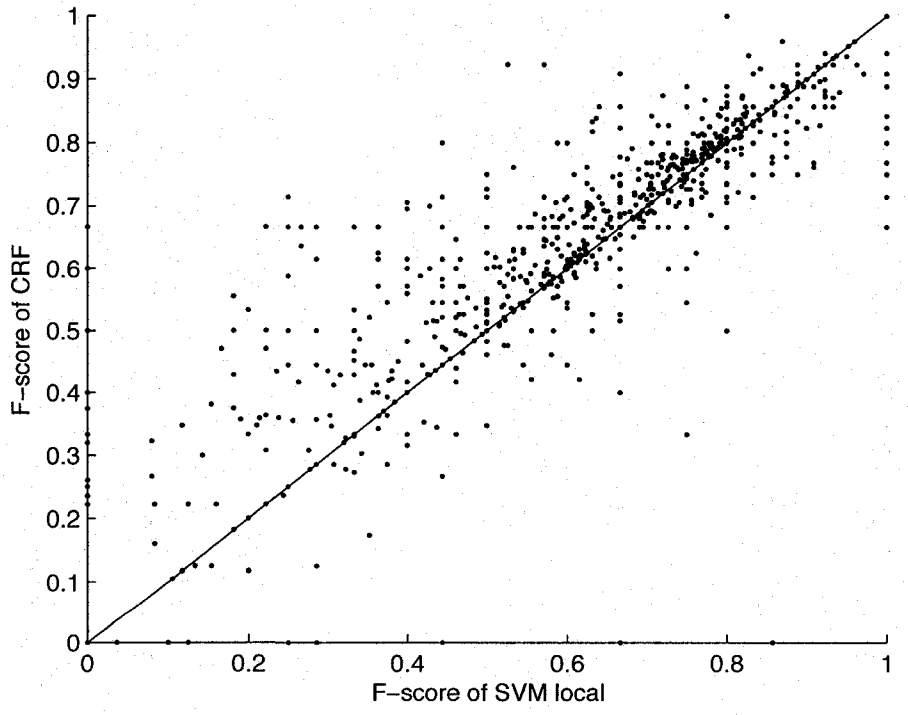


(b) F-score of CRF Vs. F-score of SVM local

Figure 6.2: Data set 1: F-score of BN and CRF Vs. F-score of SVM local

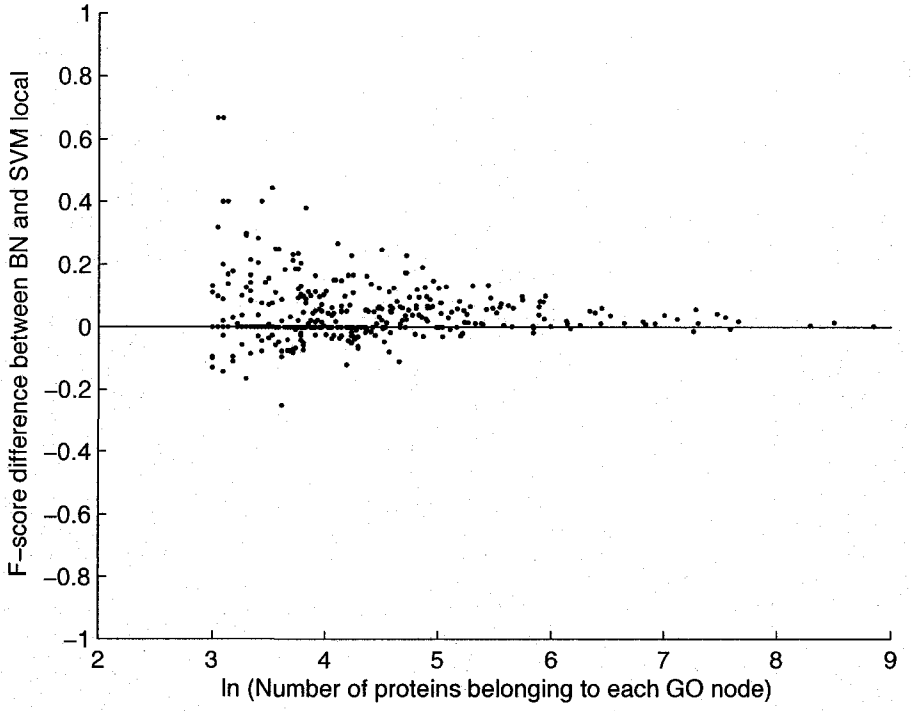


(a) F-score of BN Vs. F-score of SVM local

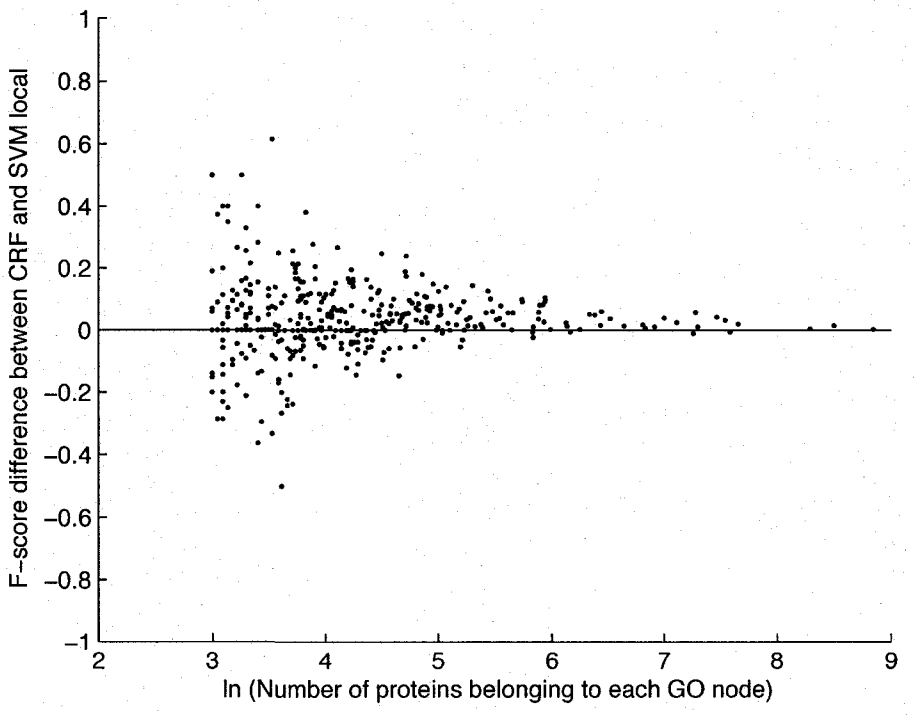


(b) F-score of CRF Vs. F-score of SVM local

Figure 6.3: Data set 2: F-score of BN and CRF Vs. F-score of SVM local

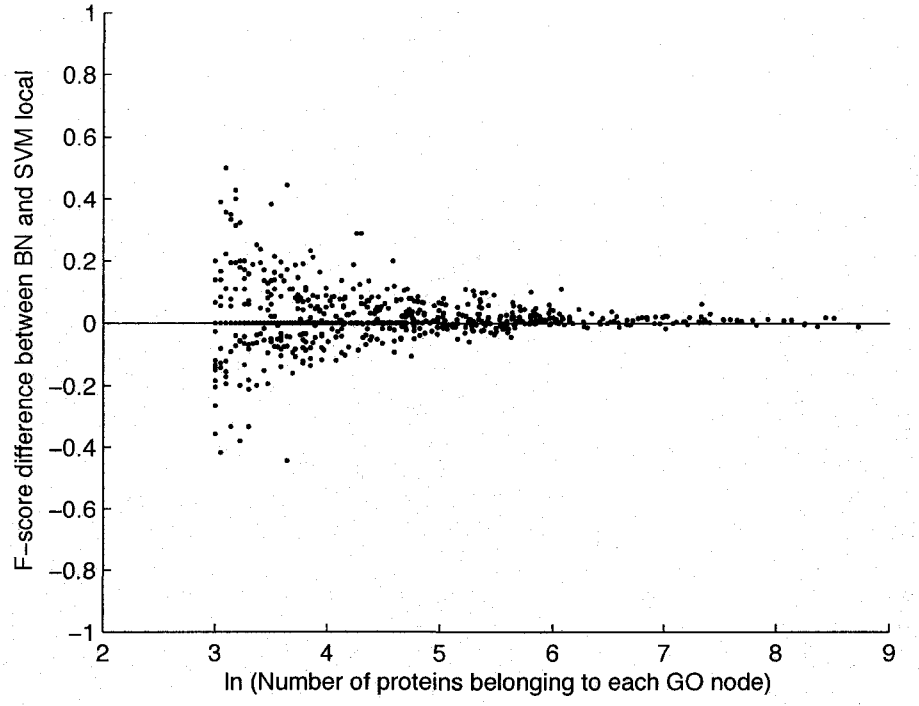


(a) F-score differences between BN and SVM local

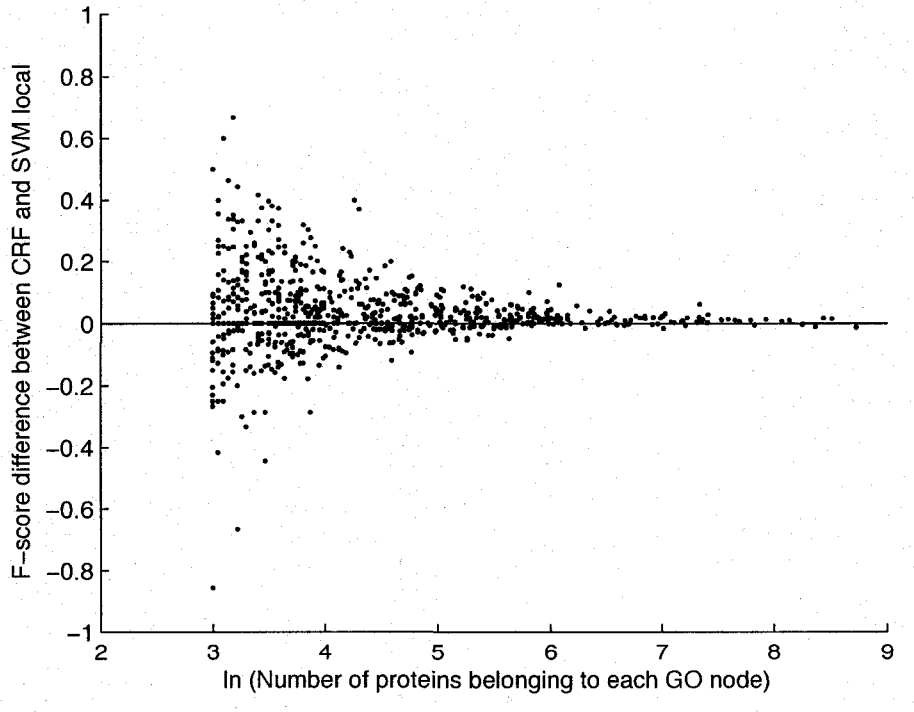


(b) F-score difference between CRF and SVM local

Figure 6.4: Data set 1: F-score difference between BN/CRF and SVM local with respect to the number of proteins belonging to each GO class. The x-axis is scaled based on a natural logarithm.

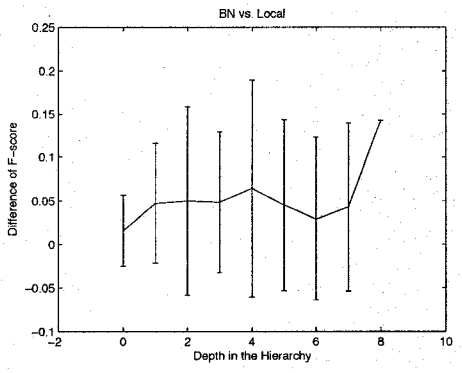


(a) F-score difference between BN and SVM local

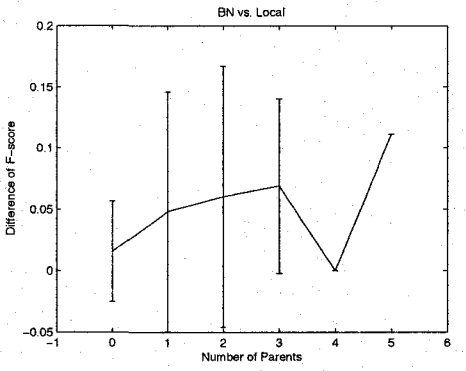


(b) F-score difference between CRF and SVM local

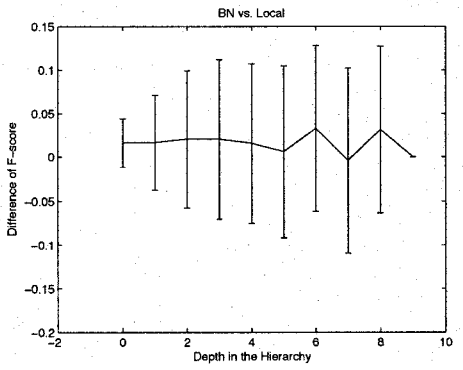
Figure 6.5: Data set 2: F-score differences between BN/CRF and SVM local with respect to the number of proteins belonging to each GO class. The x-axis is scaled based on a natural logarithm.



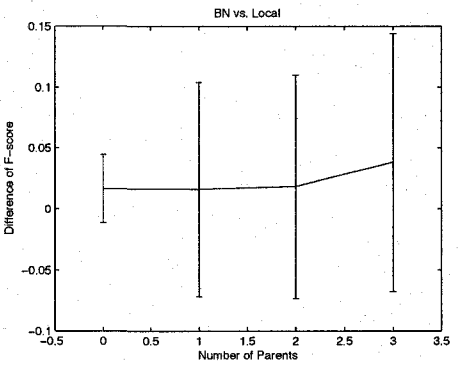
(a) Data set 1: F-measure difference with respect to the depth.



(b) Data set 1: F-measure difference with respect to the number of parents.

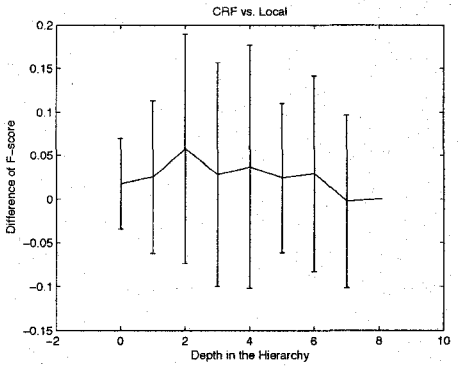


(c) Data set 2: F-measure difference with respect to the depth.

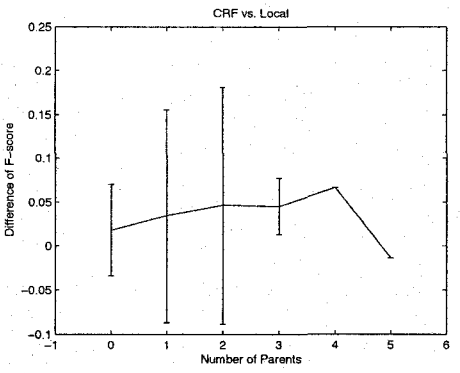


(d) Data set 2: F-measure difference with respect to the number of parents.

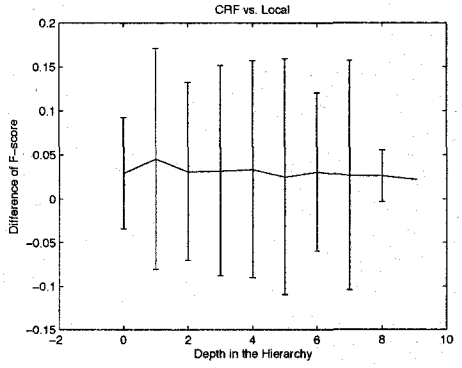
Figure 6.6: F-measure difference between BN and local SVM with respect to the hierarchy depth and the number of parents. Error bars: mean $\pm$ 1 standard deviation.



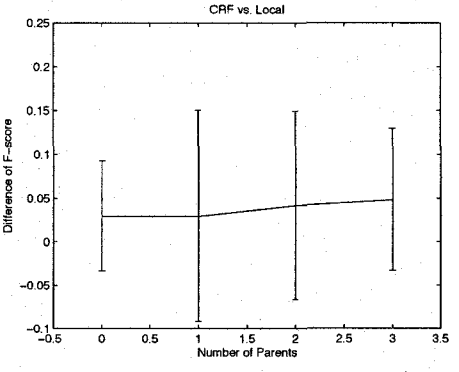
(a) Data set 1: F-measure difference with respect to the depth.



(b) Data set 1: F-measure difference with respect to the number of parents.



(c) Data set 2: F-measure difference with respect to the depth.



(d) Data set 2: F-measure difference with respect to the number of parents.

Figure 6.7: F-measure difference between CRF and local SVM with respect to the hierarchy depth and the number of parents. Error bars: mean $\pm$ 1 standard deviation.

Chapter 7

Future Work and Conclusion

7.1 Future Work

A possible extension of this work is to model function classes simultaneously using the big-bang approach. Several studies [18, 70, 71] have shown that learning a set of related classes at the same time will improve the overall prediction performance. Thus, constructing a system that trains SVM classifiers at once and uses them as an input to our graphical models may be beneficial for making consistent and more accurate hierarchical predictions.

As the results of our experiments demonstrated, connecting sibling nodes has marginally improved the prediction performance over using the directed graphical model based on the parent-child relationship. The GO is a biological hierarchy, and does not explicitly indicate the co-occurrences between all pairs of classes in the graph, except connected nodes. It may be worth learning the dependency graph purely from the training data and ignoring the GO hierarchical structure or augmenting it. The resulting graph should not only contain the majority of the original GO edges but also new arcs between unrelated nodes if co-occurrences between those nodes do exist in the training data. However, the computational cost is very expensive for performing such a structure learning task. For an exact solution, one would need to examine 2^L possible states, where L is the number of nodes in the hierarchy. In practice, heuristics and the consistency rule in the hierarchy can be used to speed up the computation.

Other future areas of investigation include: using features from other domains,

such as protein structure, protein interaction networks, and gene expression, to improve the overall accuracy; applying our approach on the other two GO categories, i.e. biological process, and cellular component, although the higher occurrence of part-of relationships may cause problems, and exploring more useful information from the GO hierarchy, for example the path distances between terms, to construct more sophisticated training data set.

7.2 Summary

In this dissertation, we investigated the use of hierarchical information and SVM output distribution to construct two graphical models, i.e. BN and CRF, in making consistent function predictions. Since the GO hierarchy provides useful information regarding the structure of protein function, better predictions should be achievable by incorporating this additional information into a prediction system. To examine this conjecture, we built our graphical models, based on a set of local SVM predictors, by converting the GO hierarchy into a dependency graph. The parameters of dependencies were learned from the true annotations of proteins in the training data, and the parameters for SVM output distributions were estimated by some Laplace models. Our approach provided better functional predictions in two Uniprot data sets compared to the methods of local SVM and local SVM with up-propagation.

Although SVMs have been used as the local predictor throughout all our experiments, our approach is a generic ensemble system that allows us to integrate the local predictions from any other type of classifier into the graphical model. If a probabilistic classifier like Naive Bayes is adopted for local prediction, there is no need to estimate the distribution of local predictions.

The methods that have been presented here can be applied to many other areas where a standardized hierarchy in the form of a directed acyclic graph exists, such as web content, document classification and object categorization. Independent classifiers for a hierarchy can violate hierarchical consistency between labels, while our approach using graphical models may correct such inconsistencies and improve the overall accuracy.

Bibliography

[1] A. Al-Shahib, R. Breitling, and D. Gilbert. FrankSum: New feature selection method for protein function prediction. *International Journal of Neural Systems*, 15(4):259–275, 2005.

[2] B. Alberts, K. Roberts, J. Lewis, M. Raff, P. Walter, and A. Johnson. *Molecular Biology of the Cell*. Garland Pub, 2002.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[4] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[5] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in k-trees. *SIAM Journal of Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[6] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, 25:25 – 29, 2000.

[7] D. Barber. Machine learning: a probabilistic approach. 2001 - 2006.

[8] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

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

[10] H. Blockeel, H. Bruynooghe, S. Dzeroski, J. Ramon, and J. Struyf. Hierarchical multi-classification. In *Proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining (MRDM-2002)*, 2002.

[11] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. J Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, 2003.

[12] P. Bork, T. Dandekar, Y. Diaz-Lazcoz, F. Eisenhaber, M.A. Huynen, and Y.P. Yuan. Predicting function: from genes to genomes and back. *Journal of Molecular Biology*, 283(4):707–725, 1998.

[13] P. Bork and E. V. Koonin. Proteome Analyst: Custom predictions with explanations in a web-based tool for high-throughput proteome annotations. *Current opinion in structural biology*, 6(3):366–76, 1996.

[14] S. Brunak, A. Danchin, M. Hattori, H. Nakamura, K. Shinozaki, T. Matise, and D. Preuss. Nucleotide sequence database policies. *Science*, 298, 2002.

[15] C. Burges. Geometry and invariance in kernel based methods,. In *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[16] C. Z. Cai, L. Y. Han, Z. L. Ji, X. Chen, and Y. Z. Chen. SVM-Prot: web-based support vector machine software for functional classification of a protein from its primary sequence. *Nucleic Acids Research*, 31(13):3692–3697, 2003.

[17] E. Camon, M. Magrane, D. Barrell, D. Binns, W. Fleischmann, P. Kersey, N. Mulder, T. Oinn, J. Maslen, A. Cox, and R. Apweiler. The Gene Ontology Annotation (GOA) Project: Implementation of GO in SWISS-PROT, TrEMBL, and InterPro. *Genome Research*, 13(4):662–672, 2003.

[18] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[19] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[20] D. Chickering. *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer-Verlag, 1996.

[21] D. Chickering, C. Meek, and D. Heckerman. Large-sample learning of Bayesian Networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[22] A. Clare and R. D. King. Predicting gene function in saccharomyces cerevisiae. In *Proceedings of the European Conference on Computational Biology*, pages 42–49, 2003.

[23] A. Cord, C. Ambroise, and J. P. Cocquerez. Feature selection in robust clustering based on Laplace mixture. *Pattern Recognition Letters*, 27(6):627–635, 2006.

[24] F. Crick. On protein synthesis. *Symposium Society Experimental Biology*, 12:138–163, 1958.

[25] S. D'Alessio, K. Murray, R. Schiaffino, and A. Kershenbaum. The effect of using hierarchical classifiers in text categorization. In *Proc. of the 6th Int. Conf. "Recherche d'Information Assistee par Ordinateur"*, pages 302–313, 2000.

[26] A. Darwiche. SamIam - Sensitivity Analysis, Modeling, Inference and More, 2004. `http://reasoning.cs.ucla.edu/samiam/`.

[27] A. Dempster, N. Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society*, volume 39 of *B*, pages 1–38. 1977.

[28] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proc. of the 23rd ACM Int. Conf. on Research and Development in Information Retrieval*, pages 256–263, 2000.

[29] R. Eisner, B. Poulin, D. Szafron, P. Lu, and R. Greiner. Improving protein function prediction using the hierarchical structure of the gene ontology. In *2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, November 2005.

[30] B. E. Engelhardt, M. I. Jordan, K. E. Muratore, and S. E. Brenner. Protein molecular function prediction by bayesian phylogenomics. *PLoS Comput Biol*, 1(5):e45, 2005.

[31] B. E. Engelhardt, M. I. Jordan, K. E. Muratore, and S. E. Brenner. A graphical model for predicting protein molecular function. In *Proceedings of the 23rd international conference on Machine learning*, volume 148, pages 297 – 304, 2006.

[32] J. A. Gerlt and P. C. Babbitt. Can sequence determine function? *Genome Biology*, 1(5):1–10, 2000.

[33] H. Guo and W. H. Hsu. A survey of algorithms for real-time bayesian network inference. *Working Notes of the Joint Workshop (WS-18) on Real-Time Decision Support and Diagnosis, AAAI/UAI/KDD-2002*, 2002.

[34] J. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. 1971.

[35] T. Hastie and R. Tibshirani. Classification by pairwise coupling. Technical report, Standford University and University of Toronto, 1996.

[36] T. Hawkins and D. Kihara. Function prediction of uncharacterized proteins. *Journal of Bioinformatics and Computational Biology*, 5(1):1–30, 2007.

[37] J. Y. Huang and D. L. Brutlag. The E-Motif Database. *Nucleic Acids Research*, 29(1):202–04, 2001.

[38] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. D. Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. A. Sigrist. The PROSITE database. *Nucleic Acids Research*, 34 (Database issue), 2006.

[39] F. V. Jensen and F. Jensen. Optimal junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 1994.

[40] L. J. Jensen, R. Gupta, N. Blom, D. Devos, J. Tamames, C. Kesmir, H. Nielsen, H. H. Staerfeldt, K. Rapacki, C. A. F. Workman, S. Knudsen, A. Krogh, A. Valencia, and S. Brunak. Prediction of human protein function from post-translational modifications and localization features. *Journal of Molecular Biology*, 319:1257–1265, 2002.

[41] L. J. Jensen, R. Gupta, H. H. Staerfeldt, and S. Brunak. Prediction of human protein function according to gene ontology categories. *Bioinformatics*, 19(5), 2003.

[42] O. J. Karst and H. Polowy. Sampling properties of the median of a laplace distribution. *The American Mathematical Monthly*, 70(6):628–636, 1963.

[43] R. D. King, A. Karwath, A. Clare, and L. Dehaspe. The utility of different representations of protein sequence for predicting functional class. *Bioinformatics*, 17(5):445–454, 2001.

[44] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proc. of the 14th Int. Conf. on Machine Learning*, 1997.

[45] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of Royal Statistical Society*, 50(2):157–224, 1988.

[46] C.-J. Lin and R. C. Weng. Simple probabilistic predictions for support vector regression. Technical report, Department of Computer Science, National Taiwan University, 2004.

[47] A. H. Liu and A. Califano. Functional classification of proteins by pattern discovery and top-down clustering of primary sequences. *IBM Systems Journal*, 40(2):379 – 393, 2001.

[48] G. Pandey, V. Kumar, and M. Steinbach. Computational approaches for protein function prediction: A survey. Technical report, Department of Computer Science and Engineering, University of Minnesota, 2006.

[49] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[50] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Scholkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 2000.

[51] D. Rose, G. Lueker, and R. E . Tarjan. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Computing*, 5:266–283, 1976.

[52] M. Sasaki and K. Kita. Rule-based text categorization using hierarchical categories. In *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 2827–2830, 1998.

[53] P. P. Shenoy and G. Shafer. Propagating belief functions using local computation. *IEEE Expert*, 1(3):43–52, 1986.

[54] R.B. Russell S.S. Hannenhalli. Analysis and prediction of protein subtypes from multiple sequence alignments. *Journal of Molecular Biology*, 303(1):61–76, 2000.

[55] J. Struyf, S. Dzeroski, H. Blockeel, and A. Clare. Hierarchical multi-classification with predictive clustering trees. In *Proceedings Lecture Notes in Computer Science*, volume 3808, pages 272–283, 2005.

[56] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *roceedings of the 1st IEEE International Conference on Data Miningg*, 2001.

[57] A. Sun, E.-P. Lim, and W.-K. Ng. Performance measurement framework for hierarchical text classification. *Journal of the American Society for Information Science and Technology*, 54(11):1014–1028, 2003.

[58] C. Sutton. GRMM: A graphical models toolkit, 2006. `http://mallet.cs.umass.edu`.

[59] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, Fyshe A, and Meeuwis D. Proteome Analyst: Custom predictions with explanations in a web-based tool for high-throughput proteome annotations. *Nucleic Acids Research*, 32:W365–W371, 2004.

[60] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Computing*, 13:566–579, 1984.

[61] K. Toutanova, F. Chen, K. Popat, and T. Hofmann. Text classification in a hierarchical mixture model for small training sets. In *Proc. of the 10th Int. Conf. on Information and Knowledge Management*, pages 105–112, 2001.

[62] K. Tu, H. Yu, Z. Guo, and X. Li. Learnability-based further prediction of gene functions in gene ontology. *Genomics*, 84:922 – 928, 2004.

[63] V Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[64] K. Verspoor, J. Cohn, and S. Mniszewski. A categorization approach to automated ontological function annotation. *Protein Science*, 15:1544–1549, 2006.

[65] K. Wang, S. Zhou, and Y. He. Hierarchical classification of real life documents. In *Proc. of the 1st SIAM Int. Conf. on Data Mining*, 2001.

[66] J. C. Whisstock and A. M. Lesk. Prediction of protein function from protein sequence and structure. *Quarterly reviews of biophysics*, 36(3):307–340, 2003.

[67] M. Yannakakis. Computing the minimal fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, 2:77–79, 1981.

[68] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo. Weighted median filters: A tutorial. *IEEE Transactions on Circuits and Systems*, 43:157–192, 1996.

[69] E. M. Zdobnov and R. Apweiler. InterProScan–an integration platform for the signature-recognition methods in InterPro. *Bioinformatics*, 17(9):847–848, 2001.

[70] J. Zhang, Z. Ghahramani, and Y. Yang. Learning multiple related tasks using latent independent component analysis. In *Proceedings of NIPS 2005*, 2005.

[71] S. Zhu, X. Ji, W. Xu, and Y. Gong. Multi-labelled classification using maximum entropy method. In *SIGIR 2005: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval*, pages 274 – 281, 2005.