Trying to determine the structure of a protein by UV spectroscopy was like trying to determine the structure of a piano by listening to the sound it made while being dropped down a flight of stairs.

– Francis Crick, British Molecular Biologist

University of Alberta

Predicting Protein Function Using Machine-Learned Hierarchical Classifiers

by

Roman Eisner ©

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

Department of Computing Science

Edmonton, Alberta
Fall 2005

To Linda, who gave me a purpose.

Abstract

High performance and accurate protein function prediction is a challenging problem in Bioinformatics. Many contemporary ontologies, such as Gene Ontology, have a hierarchical structure that can be exploited to improve the prediction accuracy, and lower the computational cost of protein function prediction. The structure of the hierarchy is leveraged in two ways: First, a novel method of creating hierarchy-aware training sets for machine-learned classifiers is introduced and shown to be the most accurate method. Second, the hierarchy is used to reduce the computational cost of classification. A sound methodology for evaluating hierarchical classifiers using global cross-validation is introduced. Biologists often use BLAST to identify potential functions of new proteins. Therefore, hierarchical methods are compared to BLAST as a baseline, and show improvements in predictive performance, and coverage. This dissertation focuses on the prediction of protein function within the Gene Ontology, but the techniques are applicable to hierarchical classification in general.

Acknowledgements

I would like to thank Linda for being there for support during the more stressful times. I also appreciate the fact that my family has supported all of my decisions. I would like to thank Duane Szafron and Paul Lu for being great Supervisors, who gave me plenty of creative freedom, and indispensable guidance. I also thank the entire Proteome Analyst research group for constructive discussions, and for many good times.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the wake of the Human Genome Project, there has been increased attention to research at the *proteome* level. While the term *genome* refers to all of the genes in an organism, the term proteome refers to all of the proteins expressed by all of the genes in an organism. Proteins are fundamental to life as we know it, performing a variety of essential functions. Protein functions include catalyzing reactions, structural and mechanical roles, storage and transport of other molecules. As proteins are studied, knowledge is gained about what these proteins do in an organism, where in the cell they perform their functions, and which higher level processes they are involved in. This knowledge is important because knowledge about proteins gives insights into drug discovery, gene therapy, and the understanding of how all life functions. However, the process of analyzing proteins is very time consuming.

The term *wet lab* is often used to refer to a biological laboratory, where experiments are performed on the biological entities themselves. In contrast, *dry lab* refers to working with computational tools, or working with theory. Automated computational tools can help researchers analyze proteins by giving good leads about what proteins do in the cell. Some lengthy experiments can be bypassed or shortened when some knowledge about proteins is known *a priori* to work in the wet lab. Computational protein function prediction is therefore a companion to laboratory methods.

Through the study of thousands of proteins, various documents have been published in biological literature describing various aspects of proteins. However, many times, different researchers will use different terminology to describe similar traits of proteins. Standard vocabularies provide a method of communicating ideas in a consistent way. These standard vocabularies are called *ontologies*.

Standardizing the way research is described supports the use of automated methods. If the knowledge contained in all publications describing proteins in various organisms were to be stored in a database, ideally the knowledge should be stored using a common vocabulary, independent of the specific wording the authors used to describe their research. Knowledge representation is important to computational tools such as databases, and tools used by molecular biologists such as predictive systems. Furthermore, standardized vocabularies help scientists in any domain by

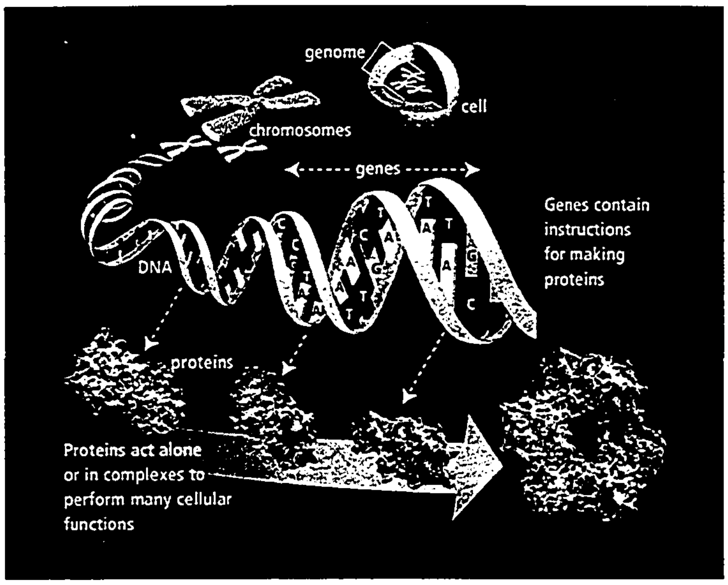

genome
cell
chromosomes
genes
DNA
Genes contain instructions for making proteins
proteins
Proteins act alone or in complexes to perform many cellular functions

Figure 1.1: The Central Dogma of Molecular Biology

The process of protein synthesis. DNA is used to create RNA (transcription), which is then used to create proteins (translation). The image is courtesy of the U.S. Department of Energy Human Genome Project.

providing a canonical way of describing their research results.

This dissertation focuses on leveraging an ontology that represents protein functions in a consistent manner. The large amount of protein data that is available is used to create a system that can predict the function of unknown proteins. However, before delving into the computational aspect of the research, a basic understanding of the biology involved in this research domain is presented.

1.1 The Central Dogma of Molecular Biology

The Central Dogma of Molecular Biology [17] states that information in a cell is transferred from DNA to RNA to protein[1]. This process is shown in Figure 1.1, and is often referred to as protein synthesis.

A protein is a linear chain of amino acids. Since there are only 20 amino acids commonly found in organisms, each amino acid can be represented by a letter of the alphabet (for example the amino acid Glutamine is represented with the letter Q). Therefore, any protein can be represented as a string of these letters (Figure 1.2). When a protein is synthesized in the ribosome, the amino acids are attached

[1]Originally the central dogma referred to the theory that no information is ever transferred from protein to DNA. Contemporary use of the term describes this and also the process of protein synthesis [52].

```
>P18077 - R35A_HUMAN
MSGRLWSKAIFAGYKRGLRNQREHTALLKIEGVYARDETEFYLGKR
CAYVYKAKNNTVTPGGKPNKTRVIWGKVTRAHGNSGMVRAKFRSNL
PAKAIGHRIRVMLYPSRI
>Q9WVI9 - JIP1_MOUSE
MAERESGLGGGAASPPAASPFLGLHIASPPNFRLTHDISLEEFEDE
DLSEITDECGISLQCKDTLSLRPPRAGLLSAGSSGSAGSRLQAEML
QMDLIDAAGDTPGAEDDEEEEDDELAAQRPGVGPPKAESNQDPAPR
SQGQGPGTGSGDTYRPKR...
```

Figure 1.2: Two proteins in FASTA format

The human protein R35A, which has UniProt accession number P18077 is 110 amino acids in length. Experimental results show that the protein is a structural constituent of the ribosome, and is involved in RNA binding [3]. The mouse protein JIP1 has UniProt accession number Q9WVI9, and is 706 amino acids in length. Experimental results show that JIP1_MOUSE is involved in protein kinase binding and kinesin binding [3]. Only the first 156 amino acids of the sequence for JIP1_MOUSE is shown.

together in a chain. This protein *sequence*, shown in Figure 1.2 is called the *primary structure* of a protein. As a protein is synthesized, the chain begins to curl, and fold into helices and beta sheets. This is known as the *secondary structure* of a protein. This secondary structure of the protein will then fold and orient itself in 3-dimensional space. This is called the *tertiary structure* (or 3D structure). The function of a protein is dependent on the arrangement of at least one of these structures. For example a protein which has a structural role will be able to perform this function because of its tertiary structure. On the other hand, a protein which binds to other proteins in the cell may only perform this function because of a functional *domain* in its primary structure. A domain refers to a section of a protein which is responsible for that protein being involved in a reaction.

However, a protein's primary structure, or sequence, is the basis of any structural orientations it may assume. Furthermore, evolutionary mechanisms affect proteins in such a way that biologically important regions of the sequence are conserved. Additionally, a protein's secondary and tertiary structures are fully dependant on the primary structure. Therefore, a protein's function must be related in some way to its primary structure. This dissertation attempts to utilize a large amount of protein sequence data to predict the molecular function of novel proteins.

The *molecular function* of a protein describes the protein's activities at the molecular level [4], and specifies the tasks it performs. When proteins are characterized, many aspects of their role are studied, such as their structure, their interactions on a chemical level, their location in the cell, and their function. Ideally this knowledge is published, and freely accessible in one of many protein databases. The term *annotation* refers to a single piece of knowledge known about a protein,

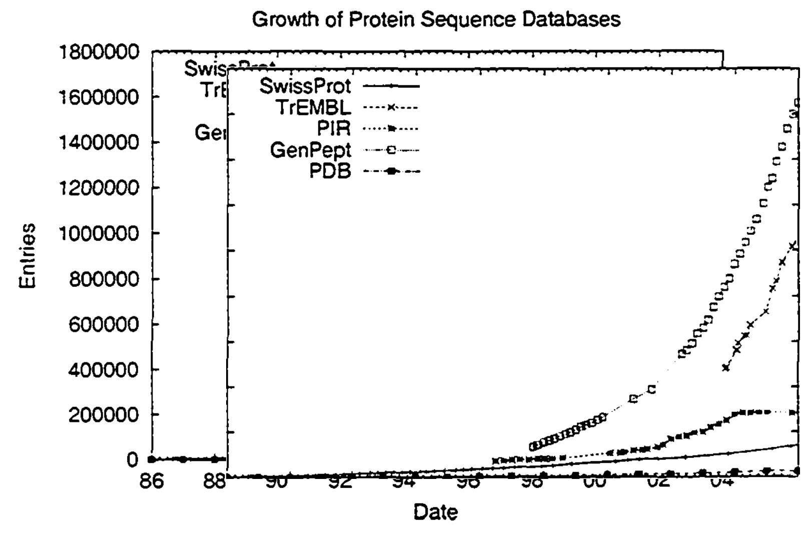


Figure 1.3: Database Growth

The Swiss-Prot and TrEMBL databases have quickly increased in size since their inception in 1986. The Swiss-Prot database is a human curated and annotated protein database, while the TrEMBL database is a computationally annotated supplement to Swiss-Prot. It is clear that the human annotators cannot keep up with the huge amount of protein sequences that have been added to the TrEMBL database. Automated annotation is required to make better use of the sequence information that is available. Data courtesy of the NIAS DNA Bank (National Institute of Agrobiological Sciences, Japan). Image and caption courtesy of Poulin [39].

such as its function.

1.2 Bioinformatics

The rate at which sequencing methods are producing genomic and proteomic data is far outpacing the rate at which these sequences are being experimentally annotated and understood. This trend is depicted in Figure 1.3. The number of human annotated proteins (Swiss-Prot, PDB) is small compared to the number of proteins for which only the sequence is known (TrEMBL, GenPept). In response, there has been a growing focus on ways to speed up the process of determining protein function through the use of computer systems that predict protein function.

1.2.1 Protein Function Prediction and Determination

In response to the overwhelming increase in protein sequence data, there has been much research in automated computational protein function prediction as demonstrated by the literature (Chapter 2.4.1) and the Automated Function Prediction Special Interest Group meeting at the 2005 Intelligent Systems for Molecular Biology

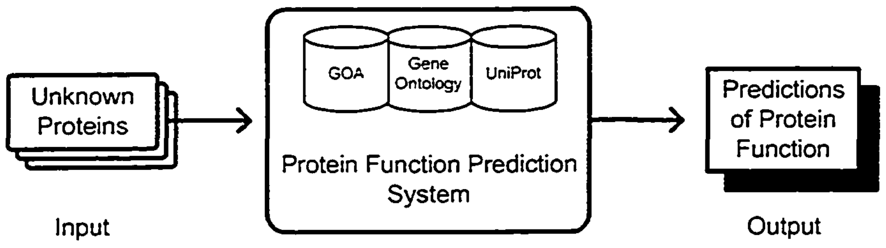

Unknown Proteins
Input
GOA
Gene Ontology
UniProt
Protein Function Prediction System
Predictions of Protein Function
Output

Figure 1.4: Protein Function Prediction

When unannotated proteins are input into the prediction system, the result is a prediction of what function(s) each of these proteins perform.

conference [2].

Protein function determination refers to the process of performing wet lab experiments to discover what function a protein serves. These methods can involve studying the protein's structure through Nuclear Magnetic Resonance or X-ray crystallography. Also, information about when proteins react or bind such as assays, and 2-hybrid interactions are useful to understand the functions that a protein performs. Many approaches exist to understand what individual proteins do, however all of them are costly in terms of equipment and manpower.

Protein function prediction provides biologists with predictions of the most likely functions that proteins perform (Figure 1.4). This can help in the process of protein function determination by providing likely functions proteins perform, and thus which experiments should be carried out. These methods should be highly accurate to be useful, and they should be high-throughput so they can be used for a large amount of data.

Another desirable feature of a prediction system is *transparency* [46]. Transparency refers to how well a user can understand *why* certain predictions were made. This can build a user's trust in the prediction system and thus can give clues as to the best experiments that should be performed in the wet lab. Alternatively, a user may decide that a prediction is incorrect by looking at the data used to make the prediction. Either way value is added when transparency is a feature of the prediction system.

Prediction methods often use machine learning approaches to model the problem domain. Machine learning leverages large datasets to extend knowledge about existing data, and supports the study of new, data. Protein function prediction can speed up, and increase the quality of, protein function determination. Protein function prediction is a confluence of research in the biological sciences, mathematical and statistical sciences, computing science, and philosophy.

1.2.2 Ontologies

In general, prediction is a mapping from instances to class. Before creating a prediction system, the type of predictions that it can make must be predefined. For example, in protein function prediction, we need to know what the possible protein functions are. An ontology is a set of terms describing the problem domain in a standardized way, and defines the possible predictions that can be made. This addresses the issue of different researchers using different terminology to describe the same functions. For example the terms "peroxiredoxin activity" and "thioredoxin peroxidase activity" both refer to the catalysis of the reaction shown in Equation 1.1. Through the use of an ontology, this reaction is described through a standardized term, so that there is no future confusion about what is being described.

$$Reduced\ Thioredoxin + H_2O_2 = Oxidized\ Thioredoxin + H_2O \tag{1.1}$$

Figure 1.5 shows a possible ontology for protein function. A variety of functions that proteins could perform are shown, and various wet lab experiments could imply that a protein performs each of them. Upon closer inspection it is evident that some functions are more similar to each other than others. For example, the functions "nucleotide binding" and "protein binding" are more similar to each other than either function is to "hydrolase activity". Furthermore, some functions are more general descriptions of the same function. For example, "peptidase activity" is a specific type of "hydrolase activity", in that every protein that performs the function "peptidase activity" necessarily performs the function "hydrolase activity". To represent these relationships between functions, the ontology can be structured in a hierarchy as shown in Figure 1.6. An unstructured ontology such as the one shown in Figure 1.5 is often called a *flat ontology*, whereas a hierarchically structured ontology such as the one shown in Figure 1.6 is called a *hierarchical ontology*.

Although flat ontologies such as GeneQuiz [9] and others [42] are suitable for describing the general function of proteins, a more sophisticated approach is essential in describing more specific functions of proteins. Furthermore, different experiments to verify the function of proteins provide different levels of detail about that protein's functions, which leaves many proteins with incomplete or general annotations. Hierarchical ontologies are an effective way of addressing these issues.

In hierarchical ontologies such as EC [30], SCOP [37] and Gene Ontology [22], both general and specific knowledge is represented in a hierarchical structure, where general terms are represented by nodes near the root of the graph and specific terms are represented by nodes near the leaves of the graph. The ontology shown in Figure 1.6 is actually a part of the Gene Ontology molecular function hierarchy, represented more completely (if illegibly) in Figure 1.7.

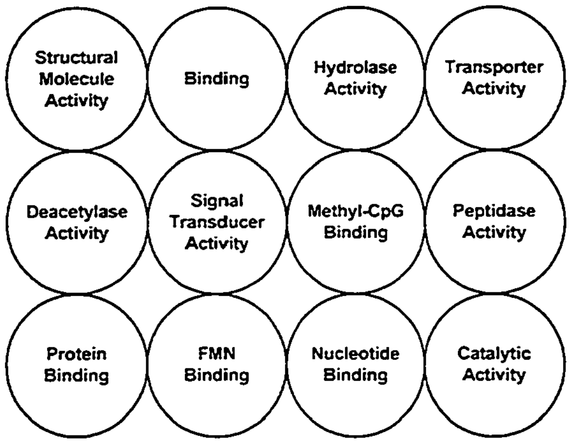


Figure 1.5: A Flat Ontology

An ontology describing some possible protein functions that may be deduced from experiments.

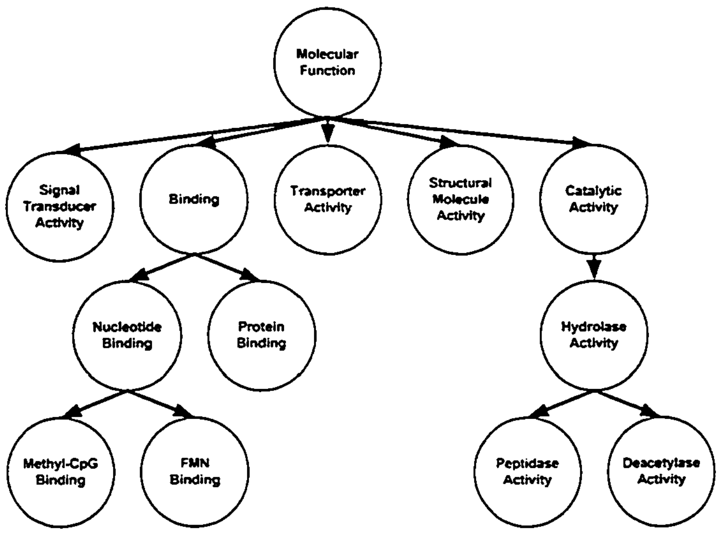


Figure 1.6: A Structured Ontology

The same protein functions shown in Figure 1.5, however the intuitive relationships between the functions are shown using a hierarchical structure. This ontology is a subset of the Gene Ontology hierarchy.

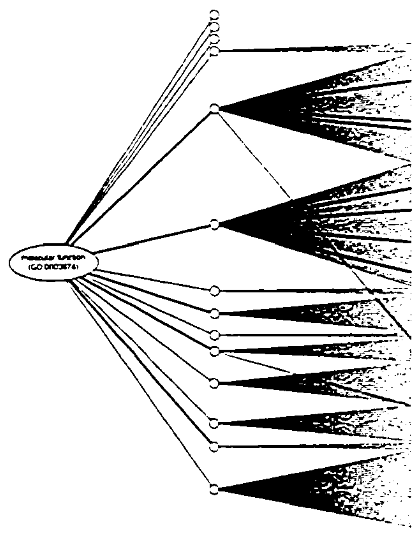


Figure 1.7: Gene Ontology

The Gene Ontology molecular function hierarchy of terms is a standardized way of describing the functions proteins perform in the cell. The hierarchical structure represents general to specific functions from left to right. Only the first three levels of the ontology are shown (Used courtesy of Poulin [39]).

1.3 Research Goal

The general goal of the research described in this dissertation is to produce a technique for accurate and efficient protein function predictions from protein sequences [21]. A high level diagram demonstrating the approach taken in this dissertation is shown in Figure 1.4. The thesis of this dissertation is that the structure of a hierarchical ontology encodes important information about the problem domain that is important when creating an accurate and efficient prediction system. Related work in hierarchical classification has confirmed this (Chapter 2.4.2). This dissertation presents novel ways that the hierarchy can be exploited in the context of protein function prediction. However, the techniques presented should be generalizable to general hierarchical classification.

Ultimately, the result of this research will be incorporated into the Proteome Analyst [46] suite of web tools, which will make it publicly available, and easily accessible.

1.4 Contributions

This dissertation presents a system called Classification in a Hierarchy Under Gene Ontology (CHUGO), that exploits the hierarchical structure of the GO to make faster and more accurate predictions of protein function. The issue of evaluating predictions within a hierarchical ontology is examined. Also, a novel method of exploiting the structure of a hierarchical ontology to create predictors is presented and evaluated. This dissertation shows how to exploit the hierarchical nature of GO to lower the computational cost of predicting within the ontology without compromising accuracy. Finally, it shows that the precision and recall of a classification system can be improved when the hierarchical knowledge is utilized.

This dissertation makes contributions in three main areas:

1. **Evaluation Methodology** - In hierarchical ontologies, precision, recall, and cross-validation are concepts that must be re-visited. This dissertation defines and illustrates a hierarchy-aware evaluation methodology.

2. **Training Set Design** - Structured ontologies are encoded with important information about relationships between terms, and are a way of representing incomplete data. This dissertation presents a novel and effective approach to training set design that exploits the inherent structure of a hierarchical ontology. By considering the structure of the ontology, our algorithms increase the F-measure of hierarchical classification from 46% to 70%.

3. **Accurate and efficient protein function prediction** - This dissertation exploits the structure of a hierarchical ontology at prediction time to improve predictive performance, and lower computational costs. CHUGO can increase recall for those proteins that are similar to experimental proteins by 2%, and in the case of proteins that are dissimilar to the set of experimental proteins, precision can be increased by 37%, and recall by 12%. The computational cost of local predictors can be lowered to as low as 2% of the cost of running all local predictors when the hierarchy is considered.

1.5 Outline

Chapter 2 first introduces necessary terminology and concepts in machine learning. Next related work in protein function prediction and general hierarchical classification are discussed. Finally each of the tools used in this dissertation are introduced.

Chapter 3 describes the data set used for all experiments described in this dissertation. The issues that are raised when predicting within a hierarchical ontology are described and addressed. Chapter 3 concludes by presenting a first attempt at protein function prediction within the Gene Ontology.

In Chapter 4 protein function prediction is revisited. First, the way training examples are selected for each local predictor is explored while keeping the structure of the ontology in mind. Second, it is demonstrated that the structure of the ontology can be used to lower the cost of prediction without a penalty to accuracy. The chapter concludes by revisiting the issue of cross-validation in a hierarchy.

Each predictor method is described in detail in Chapter 5, and then optimized. These predictors are then combined into ensemble classifiers which are used to predict each molecular function in the ontology.

Chapter 6 applies the prediction technologies presented in the previous chapters. Other approaches of lowering the computational cost of prediction are also presented. The issue of coverage is also addressed, and CHUGO is shown to have a higher coverage than BLAST and Protfun.

Chapter 2

Background and Related Work

Since the inception of automated procedures there has been a large increase in the amount of data that needs to be processed and understood. Machine learning is a way of addressing these issues, by automating and facilitating the process of understanding relationships and patterns in data.

Proteomics research is an area where this issue is relevant. The amount of protein sequence data available far surpasses our ability to determine and catalog the function of each protein sequence. This chapter will introduce machine learning in general, and make these concepts concrete by connecting them to a real world example – protein function prediction.

2.1 Machine Learning

Machine learning [7][23] is an area of Artificial Intelligence that attempts to "learn" patterns and behaviors from real world data. There are two major areas of machine learning: supervised and unsupervised learning.

In unsupervised learning, raw unlabeled data is given as input, and the goal is to find patterns in this data. These patterns give information about similarities in the instances in the data set, but ultimately must be interpreted by users knowledgeable in the problem domain since no *a priori* knowledge about the data is given as input.

In supervised learning, the data given as input also includes associated *labels* with each instance in the data set. The labels are descriptions of the problem domain. The goal of supervised learning is to learn a function representing the data set, which can then be used to predict labels for future instances where the labels are unknown.

This dissertation only deals with the latter case of supervised learning.

2.2 Classification

In supervised learning we are given instances and corresponding labels for each of these instances. The case when these labels come from a finite, discrete set, is called *classification*. The more general case, when the labels can be any real value is called regression. Since the topic of this dissertation is protein function prediction and it is an example of the former, regression will not be described further.

In the classification of proteins by their function, each data instance is a protein, and each instance's label is that protein's function. One common approach in machine learning is to represent each instance as a *feature vector*, $\overline{x}$. Each component of the vector is a feature that describes some aspect of that instance. In protein function prediction, this vector can contain various biological properties of a protein [49], annotations describing similar proteins [46], or other attributes (see Chapter 2.4.1). Feature vectors are not the only way to represent instances. For instance, the protein sequence can be modeled directly [39].

Each function is represented by a *label* (also called a *class*). Therefore, along with each instance, a corresponding label, y, is given. Each labeled instance is represented by an attribute, label pair: $(\overline{x}, y)$, and each label must be one of a standard set of terms (the ontology). Gene Ontology contains the possible functions a protein may perform.

An important observation is that a protein may perform more than one function. For example, the protein JIP1_MOUSE performs the functions "Kinesin Binding" and "Protein Kinase Binding". These two functions are the labels for the protein JIP1_MOUSE. This has important consequences in the construction of a prediction system, and in evaluating such a system. This will be addressed in Chapter 3.2.

The goal of classification is to use the labeled data (also called the *training set*) to create a classifier (Equation 2.1). A classifier is a model or function that, when applied to an instance, $\overline{x}$, returns a prediction of its class, $\hat{y}$. During evaluation, the prediction for an instance $\hat{y}$ is compared with the instance's true label y and scored in some way.

$$f(\overline{x}) = \hat{y} \tag{2.1}$$

Supervised learning occurs in two stages (Figure 2.1). First, the process of creating a classifier is called learning or training. Here, one of a variety of algorithms is applied to the labeled data set to create the prediction function $f(\overline{x})$. Second, the process of running a *query instance* (a protein instance whose label should be predicted) through a classification function and returning a *predicted label*, is called prediction or classification. Some terminology that will be used throughout this dissertation is summarized in Table 2.1.

2.3 Hierarchical Classification

In traditional classification problems, the set of candidate labels, Y, are independent of each other, meaning that they are not related. This arrangement of labels is commonly called a flat ontology, shown in Figure 1.5. In hierarchical classification

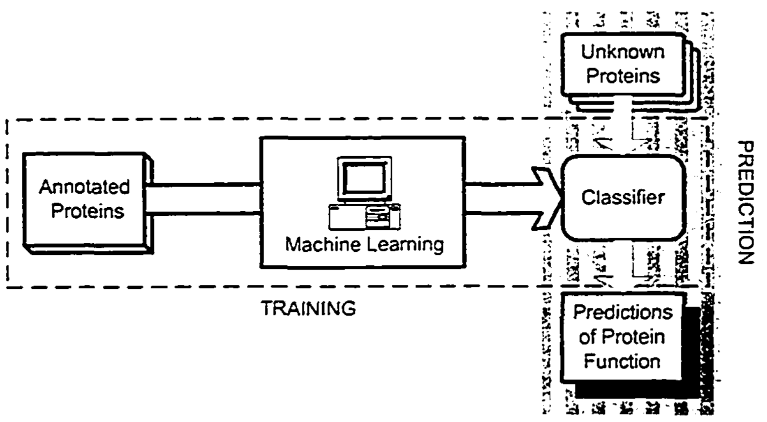


Figure 2.1: The two stages of supervised learning

Supervised learning occurs in two stages. The first, shown horizontally is called training, and describes the process of creating a classifier function from data. The second, shown vertically is called prediction, and describes the use of a classifier function to make predictions on new instances.

Table 2.1: Terminology

Summary of terms used throughout this dissertation. Examples are given in the domain of protein function prediction. Synonyms are also given in the term column.

Term	Definition	Example
Instance, $\overline{x}$	A single data element.	Protein, JIP1_MOUSE
Ontology, Label Set, Valid Classes	A standard set of terms describing the problem domain.	Gene Ontology
Label, Class, Node	An element in the ontology that describes an instance	"Hydrolase Activity"
Predicted Label, $\hat{y}$, Classification, Prediction	A label predicted by a classifier.	"Kinesin Binding", "Protein Kinase Binding"
Annotated Label, y	A label assigned by an oracle, considered as truth	"Kinesin Binding", "Protein Kinase Binding"
Feature, $\overline{x}_i$	An attribute of an instance. These can be obtained through a variety of methods. Features are the components of the attribute vector $\overline{x}$.	Molecular Weight

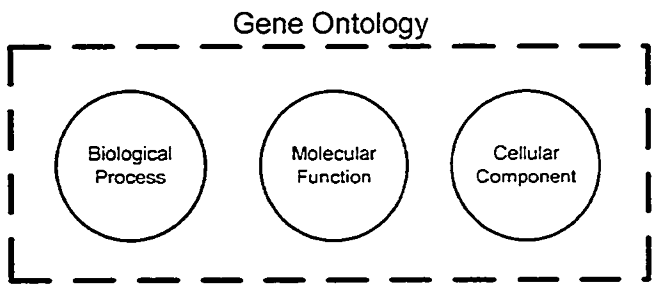


Figure 2.2: The three aspects of Gene Ontology annotations.

the labels are arranged in a hierarchy, where the nodes in the hierarchy represent the candidate labels, and the edges represent the relationship between the labels. The fact that the labels are not independent raises several issues during training, prediction, and evaluation.

A structured ontology encodes important information about how the labels relate to each other. This dissertation shows that it is unwise to ignore these relationships. This information describes which functional annotations are similar to each other, and thus aids in the creation of training data for classifiers. This structure can also be exploited to increase the predictive accuracy and lower the computational cost of a classification system. Previous work has also shown that this structure can be exploited in other ways (Chapter 2.4.2). t̂ab:terminology summarizes some machine learning terminology.

2.3.1 Semantics of a Hierarchy

The Gene Ontology (GO) controlled vocabulary is an ontology of terms describing three aspects of protein annotations. Each aspect is organized into a hierarchy (Figure 1.6). The three aspects of the GO hierarchy are *molecular function*, *biological process*, and *cellular component* [22] (Figure 2.2). Each of these aspects are an independent hierarchy in the Gene Ontology. That is, no node within a single aspect is connected to any of the nodes within the other two aspects. This fact allows us to focus on any single aspect at a time without considering the others. This dissertation focuses on protein function prediction, so only the molecular function aspect is used for all experiments in this dissertation.

In hierarchical ontologies, the edges represent relationships between labels. These edges generally represent two types of relationships. These are: the *is-a* relationship, and the *part-of* relationship. The is-a relationship denotes a child being a more specific description than its parent. In the molecular function aspect of GO, the term "metal ion binding" is a child of the term "ion binding", since it is a more specific description of molecular function (Figure 2.3). Any protein that is a "metal ion binding" protein is also, by definition, an "ion binding" protein.

The part-of relationship describes a sub-component relationship. That is, a child

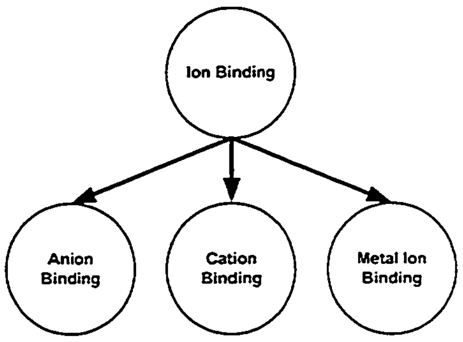


Figure 2.3: The ion binding node and its children

The relationship between the nodes is the is-a relationship. Children nodes are conceptually specializations of parent nodes. These nodes are part of the molecular function aspect of the Gene Ontology.

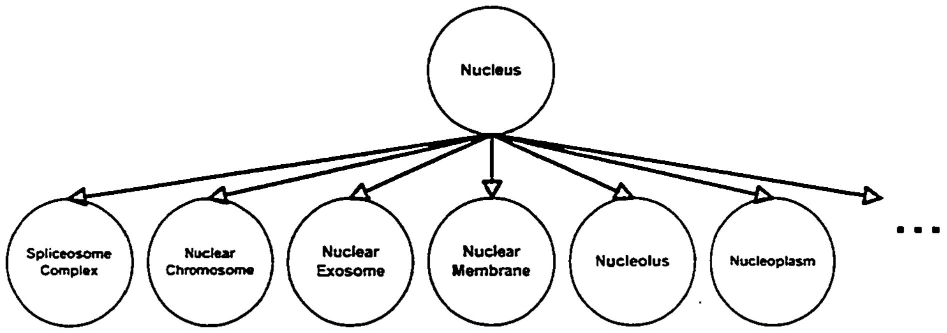


Figure 2.4: The nucleus node and some of its children

The relationship between the nodes is the part-of relationship. Children nodes are conceptually part-of parent nodes. These nodes are part of the cellular component aspect of the Gene Ontology.

class is a component of the parent class. In the cellular component aspect, the component "nuclear chromosome" is part-of the component "nucleus" (Figure 2.4).

Within the molecular function aspect of GO, the relationship between the majority of the nodes is the is-a relationship, with the part-of relationship only occurring for two terms (the children of "telomerase activity" - not in figures). We limit ourselves to only those nodes under an is-a relationship. This allows us to treat the hierarchy's edges uniformly.

When structuring an ontology in a graph, each label is referred to as a node. Some nodes in the graph are called *leaf nodes* if they have no children (such as "metal ion binding", "anion binding" and "cation binding" in Figure 2.3). All other nodes are referred to as *non-leaf nodes*.

If a protein's function is "ion binding", then implicitly, it is also a "binding" protein. This intuition is called the *True Path Rule* by the Gene Ontology Consortium [4]. If a protein is annotated with a certain node, then it is implicitly annotated with all of this node's parent nodes all the way up to the root node as well. This propagation of annotations allows for the labels of a protein to be consistent. Intuitively, it would be inconsistent to consider a protein a "ion binding" protein, but not a "binding" protein.

A difficulty with hierarchical ontologies is scoring. The hierarchy encodes intuition about which nodes are closer to each other than others, however it is difficult to measure wrong predictions in the hierarchy. This issue is addressed further in Chapter 3.2.2.

Another difficulty is that knowledge about an area is constantly changing, and this may affect how this knowledge is organized. To address this issue the GO consortium meets regularly to update, and sometimes reorganize the ontology. There is also a user meeting that allows for users of the ontology to voice their concerns [6].

Ontologies have also been an area of research in Philosophy since the days of the ancient Greeks [5]. Understanding how to represent knowledge into ontologies gives us a better understanding of our own thought processes. Having a predefined ontology also promotes the standardization of terms in future use. Furthermore, having an understanding of the semantics of a hierarchy, and how to apply them correctly helps to alleviate inconsistencies in scientific knowledge. Philosophical principles help to make the structure and wording of an ontology more consistent and formal, and have been applied to the Gene Ontology [45].

2.3.2 Hierarchical Classification of Protein Function

Our data set consists of proteins that have been annotated with their Gene Ontology molecular functions. These annotations are derived from experiments performed on these proteins, electronic predictions of these protein's functions, or *putative* functions based on homology and other methods. Putative functions refer to those that are commonly accepted as true. Our dataset does not use all of these annotations, since they are not equally reliable. However, it is important to note that each of these annotations represent the most specific experiment performed to assess a particular protein's function, but may not be the absolute correct answer.

This issue can cause problems during the evaluation of our predictors. For example, a protein that is annotated as only an "ion binding" protein, but in actuality is a "anion binding" protein (Figure 2.3), would non-intuitively give us a better score when we predict it as "*not* anion binding". This is because a future experiment may show that this protein is indeed "anion binding", however "not anion binding" matches the annotations, which we consider as the correct answer during evaluation. Our predictor may answer "anion binding" because of legitimate, machine-learned similarities between the protein and other proteins in the "anion binding" set. Furthermore, a future experiment may show that the protein is indeed "anion binding".

When annotating proteins with Gene Ontology terms, each protein can be assigned multiple terms. A protein may be assigned multiple Gene Ontology function terms for two reasons. First, a protein may have multiple functional domains or react to more than one molecule. For example the protein JIP1_MOUSE is annotated with "Kinesin Binding" and "Protein Kinase Binding" (shown in Figure 2.5). Neither of these terms is a direct specialization of the other, so there is no ancestor/descendant relationship between these terms in GO.

Second, due to the hierarchical nature of Gene Ontology, a protein may be explicitly annotated with a GO term and also the parent of this GO term. This a result of what each Gene Ontology annotation represents. For example, one experiment may show that a protein is an "ion binding" protein. Another experiment may be more specific and show that a protein is an "anion binding" protein. These annotations may represent the same function that a protein performs. We later show that it is useful to explicitly represent all GO terms describing a protein's functions according to the true path rule.

There are two unusual labels in the GO hierarchy that are treated differently during experiments. These are the "unknown" (0005554), and "obsolete" (0008369) nodes. A protein mapped to the "unknown" node has been experimented on and no positive results have been found, in contrast with those proteins that have not been studied at all and so have no GO annotations. The "obsolete" node describes annotations to GO terms that were removed in a newer release of GO. These obsolete nodes are moved so they become children of the term "obsolete". If a protein maps to either one of these nodes or one of their child nodes (in the case of "obsolete"), these annotations are removed from the data set, since they do not provide any useful information.

For each protein, experimental annotations are provided. The set of those nodes that are explicitly annotated for a protein will be called the *mapped nodes*. According to the semantics of the hierarchy, all of the parent nodes of the mapped nodes apply to this protein as well. If all of these annotations are propagated upward in the hierarchy according to the true path rule, we arrive at the set of labeled nodes. In Figure 2.5 "protein kinase binding" and "kinesin binding" are the mapped nodes, and "protein kinase binding", "kinase binding", "enzyme binding", "protein binding", "binding", "kinesin binding", and "cytoskeletal protein binding" are the labeled nodes. The set of mapped nodes are those explicitly annotated to a protein, and the set of labeled nodes are those inferred from the ontology and the set of mapped nodes. This terminology will become useful later when evaluation is discussed.

Classification of protein function within the Gene Ontology vocabulary is a general form of hierarchical classification in several ways. First, the hierarchy allows for multiple parents. In other words, the structure of the hierarchy is a directed-acyclic graph (DAG), where a node can descend from two or more parent nodes, such as in Figure 2.5 "Receptor Binding" descends from "Binding" and "Signal Transducer Activity". This is a more general form of hierarchical ontologies that

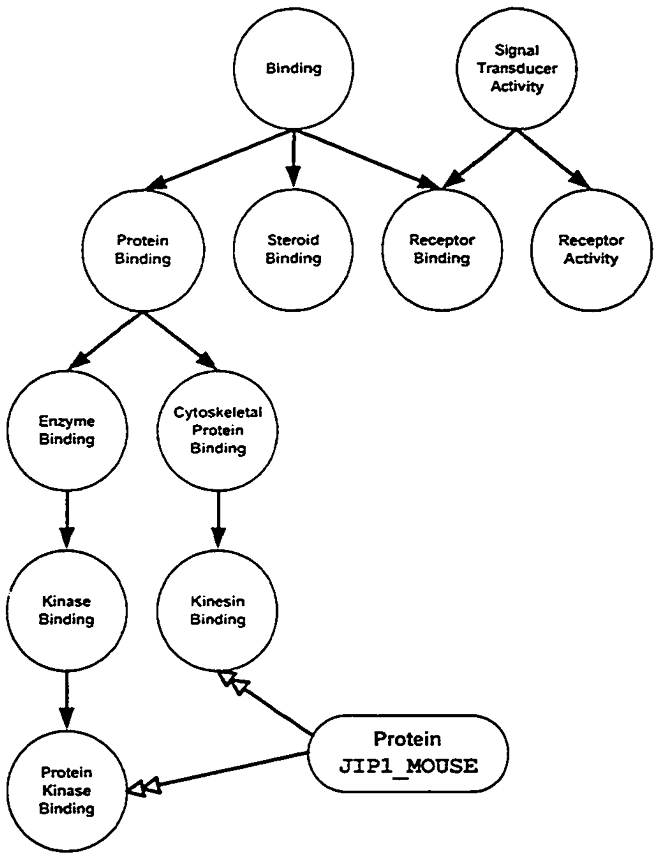


Figure 2.5: A protein (JIP1_MOUSE) and its GO annotations

Each protein can be assigned multiple terms. Also note that Receptor Binding has two parent terms making the ontology a Directed Acyclic Graph (DAG). Only part of the ontology is shown.

are tree-shaped, where a node can only have a single parent. Second, each protein can be assigned more than one mapped node in the hierarchy. This restricts the types of classification technology that can be used since the prediction system must be able to predict more than one label for each instance.

2.4 Related Work

2.4.1 Protein Function Prediction

The prediction of protein function is important to supplement the labour-intensive process of protein function determination. Predictions for the functions of proteins can help select likely candidates for further study, such as in pharmaceutical research. Alternatively, when studying a single protein, function predictions can give good leads as to which experiments should be performed to further elucidate the protein's functions. A variety of approaches to protein function prediction exist.

One approach is to use the protein's structure (i.e. tertiary structure) to predict a protein's functions [38]. Since this approach requires that the protein's structure be solved (or at least predicted accurately) it provides limited coverage across all of the proteins in any particular organism.

Another approach uses documents describing proteins to predict the functions of these proteins [27]. This technique also has extremely limited coverage on a per-organism basis. Not only does information have to be published about a protein for this technique to work, but also the documents must be correctly associated with the protein in question (which documents discuss which protein may not always be easily discernible information) and correctly parsed, which may be difficult.

Coverage can be improved by using sequence-based approaches. By far the most commonly used method is BLAST [8]. BLAST is an efficient way to search a database of protein sequences for sequences similar to a query protein. Other techniques attempt to leverage the results of a BLAST search to make more accurate predictions [49]. However, merely validating BLAST results will not improve the coverage of a predictor. Furthermore, we demonstrate (Chapter 6) that BLAST does not work well for predicting functions when the most similar sequences found are below the 10^{-3} *E-value* threshold. We found that this often limits the coverage of these predictors to approximately 60% of the proteins in a proteome.

Proteome Analyst (PA) is another tool that utilizes BLAST. PA uses the annotations associated with similar proteins to predict the functions of query proteins [46]. Proteome Analyst does not consider the hierarchy when creating its predictors. However the hierarchy is used to create the pruned ontology shown in Figure 2.7. Those nodes with a large number of annotated proteins are kept in the ontology, in a way that is consistent with the structure of Gene Ontology. That is, a node is never included in the ontology without its parent node being included as well.

Protfun [24] is another example of a sequenced-based predictor. It uses local sequence properties, such as predicted post-translational modifications, sorting signals and properties computed from amino acid composition as input for predictions. No *a priori* knowledge of the protein is required, other than its sequence. However, Protfun does not exploit knowledge of the hierarchy during the training of its classifiers, or during the selection of its ontology (Figure 2.6). Furthermore, the set of 14 GO terms that it predicts is relatively small (and only 9 of the 14 nodes are part of the molecular function aspect of GO). In the future, we would like to incorporate biological features such as those used by protfun in local predictors using our training set design schemes.

Other methods attempt to represent sequences in more complex ways. PFAM uses Hidden Markov Models to represent protein families [12]. These Hidden Markov Models can then be used to predict whether unknown proteins fit into each of these protein families with varying degrees of confidence. We use these predictors as features for some of our classifiers.

InterProScan [10] combines a variety of prediction and database tools into a single prediction system. When a query protein is run through InterProScan it is

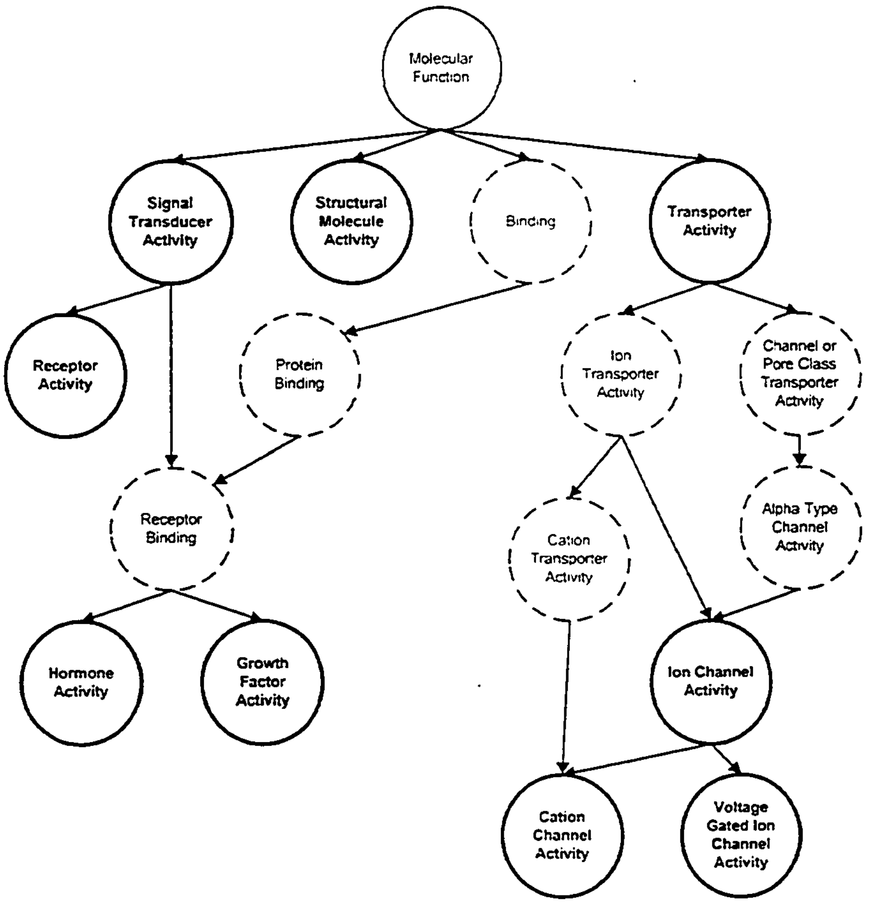


Figure 2.6: The ProtFun ontology

A general molecular function ontology, used by the ProtFun 2.2 prediction system. Solid nodes are those that ProtFun uses in its prediction ontology. Dashed nodes are those that are not in the ProtFun ontology, but are intermediate nodes of those that are included. Protfun also includes 5 nodes from the biological process aspect of Gene Ontology, which are not shown.

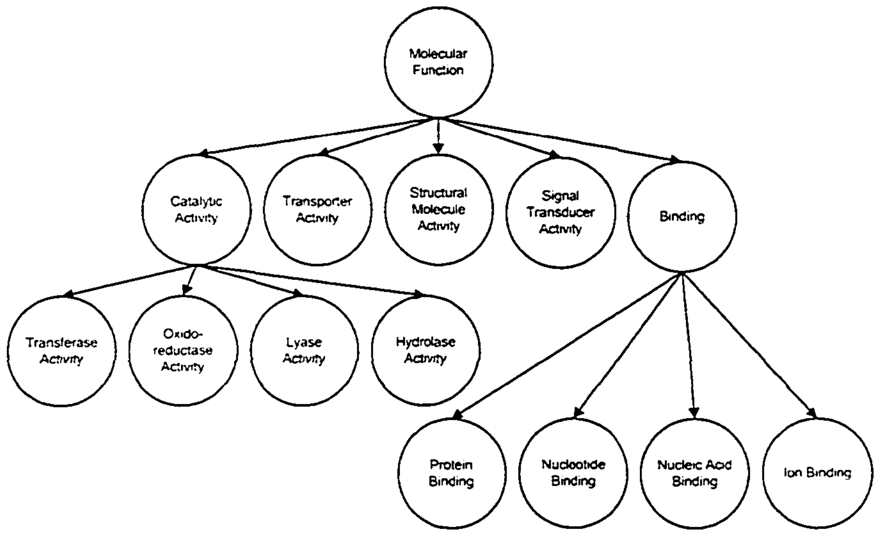


Figure 2.7: The Proteome Analyst ontology

A general molecular function ontology, created by pruning the original Gene Ontology vocabulary, and used by the Proteome Analyst online system.

assigned a variety of InterPro codes. Some of these codes can then be mapped to the Gene Ontology, if they represent functional classes. These InterPro codes are included within Proteome Analyst features, which are then incorporated into our prediction system.

King *et al* [26] present another approach that is only based on the existing annotations of proteins. The system examines existing annotations, and predicts annotations that often correlate with the existing predictions. This is done because the authors correctly observe that protein annotations are often *incomplete*. They note that protein annotations are incomplete because "...there are genes whose attributes are not yet all known, and because there is literature that has not yet been digested by the database curators" [26]. However, in Chapter 4.1.4 we argue that even existing annotations are incomplete, because they may become more specific in the future. The methodology presented by King *et al* also suffers from a fact that functions that are often correlated may not *always* occur together. In CHUGO, each function is predicted individually, where correlations would occur as a natural result of the prediction process.

2.4.2 Hierarchical Classification

The machine learning literature has described attempts to utilize the structure of a hierarchical ontology to improve classification. Kiritchenko *et al* [28] used the hierarchy to increase the number of training instances at each node, by first making the training data for local predictors more consistent with the ontology. We extend this work by investigating different degrees of consistency with the hierarchy when creating our predictors.

Koller and Sahami consider the hierarchy during the creation of training sets, and compare these results to considering the ontology without any structure [29]. Their results show a close predictive performance with the two methods where our experiments show a much wider gap. We believe this is because their hierarchy describes web documents, which can actually be classified to non-leaf nodes in the ontology, whereas in Gene Ontology a protein is assigned to a non-leaf node because of incomplete information. That is, we may know that a protein binds to a metal, but we may not know which metal. Therefore, with complete knowledge, we believe that all proteins should be annotated with leaf nodes in Gene Ontology, whereas in the classification of web documents, assigning an instance to a non-leaf node is valid.

Chakrabarti *et al* [15] and King *et al* [26] showed that the structure of the ontology can be exploited to define the structure of a Bayesian network. Similarly, the structure of the ontology can also be used to define the underlying structure of a hierarchical mixture of experts model [43]. Although our system does not use more complex models such as these, they could be combined with our training set design schemes to potentially increase predictive performance.

Sharma and Poole showed that when the semantics of the hierarchy are considered in a Bayesian Network, the computation of probabilities can be converted to an equivalent flat model for some given evidence [44]. Although their research is not directly related to ours, the fact that they consider the semantics of the hierarchy during prediction is related to the methods of training set construction presented in this dissertation.

Other research [35] has shown that a statistical technique known as shrinkage can be used to set the parameters in a hierarchy of predictors. Here, the ontology is exploited as prior knowledge to understand which classes are closely related, and thus, which parameters should have similar settings.

Dekel *et al* [18] use a hierarchy to change the formulation of Support Vector Machines with the hierarchy in mind. Wang *et al* [51] used a similar approach to modify the way Association Rules are created, keeping the hierarchy in mind. In principle, these approaches could be combined with our own.

It has also been observed [1] that a top-down decision model in hierarchical classification could have poor results since all predictors along the path to the true label must agree. Also, other methods of training individual term predictors were mentioned but not explored. Our methods of training set design allow for a more inclusive classifier in which the top-down model is more feasible, and thus we can reduce computational complexity without a resulting loss of precision and recall.

2.5 Introduction to Tools

In general, predictors over a hierarchy fall into two categories: global predictors, and local predictors. Global predictors are executed once per protein, and predict labels over the entire ontology. For example, the commonly used sequence similarity search tool, BLAST [8], can be used as a Nearest-Neighbor (NN) tool in the domain of protein function prediction. A typical use case of BLAST would be searching a trusted database such as Swiss-Prot for proteins similar to a protein of interest, such as JIP1_MOUSE. The results of a BLAST search would be proteins similar to JIP1_MOUSE (the nearest neighbors). The user would then examine the annotations of these similar proteins and then assume that JIP1_MOUSE performs the same, or similar functions. Since this process does not depend on the size of the ontology, its computational cost does not increase as the size of the ontology increases[1]. In general, all NN predictors can be used as global predictors.

In contrast, local predictors only predict a single label at a time. For example, a single local predictor would only predict whether a protein does or does not have the molecular function "transporter activity". In machine learning, this is called a binary predictor because the output is one of two possibilities ("transporter activity" or "*not* transporter activity"). Local predictors attempt to model a specific molecular function, and when given an unknown protein decide whether this protein belongs to this functional class of proteins or not. Since local predictors have the potential to model the subtle differences between molecular functions more accurately, they have the potential to help improve the accuracy of protein function prediction. However, the overall computational cost of using local predictors is much higher than using a global predictor, and is dependent on the size of the ontology.

We use a variety of machine learning approaches and feature extraction methods. Support Vector Machines (SVMs) and Probabilistic Suffix Trees (PSTs) are used to create local predictions at each GO node. BLAST (a global predictor) is exploited for its accuracy and computational efficiency. The remainder of this Chapter will describe each in detail. Evaluations of each tool, and how the technologies are combined are discussed in Chapter 5.

2.5.1 BLAST

The results of a BLAST search against a database is a list of proteins that are similar to the query protein. This list of proteins is ordered according to how similar they are to the query protein. The user can decide which proteins are similar enough to their query protein and examine their annotations. However, proteins that are not similar to well-studied proteins will not return a good BLAST result, so the biologist must decide to either examine proteins that are not very similar to their protein of interest, to look for other sequence information, or to proceed with "wet

[1]BLAST's complexity does, however, increase with the size of the database being searched.

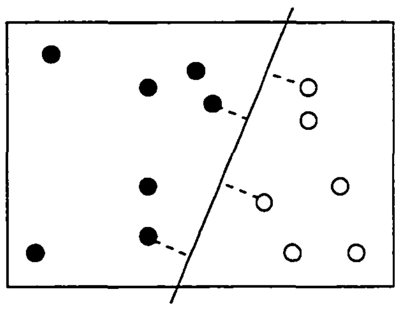


Figure 2.8: A Support Vector Machine

A Support Vector Machine works by finding a hyperplane which splits the data according to its labels. Each dot represents a training instance. Black dots are labeled with one class, and white dots are labeled with the other. The dimensionality of the space in which the hyperplane is formulated is defined by the number of features that can be assigned to each instance. In practical applications the data is not as easily divided as shown in the figure. For a more detailed description of SVMs see Hastie *et al* [23]. Image courtesy of Poulin [39].

lab" experiments (possibly without any initial idea of the protein's function).

The results of a BLAST search are a set of protein sequences in the database, ranked by their similarity. Whether a BLAST result is good enough to accept as potentially homologous is decided by the user. The predictions are ranked and it is up to the user of the system to set a score for which the results are acceptable. We therefore use the same approach of setting this threshold such that precision and recall are maximized during evaluation. This is discussed further in Chapter 5.4.

BLAST is so commonly used that it has become a verb (e.g. "BLASTing a sequence"). Due to the ubiquitous use of BLAST, our system will be compared to BLAST in terms of predictive accuracy, coverage, and computational cost. Also, BLAST provides important information that can be used to increase the accuracy and decrease the computational cost of local predictors.

2.5.2 Support Vector Machines

Support Vector Machines [48] (SVMs) are a way of learning a classifier function (Equation 2.1) from labeled data, which have proven to be accurate in a wide range of machine learning applications. Other advantages of SVMs are that they have good theoretical justifications, and provide the ability to model data in higher dimensional spaces. SVMs work by splitting the feature space of instances, according to their labels (Figure 2.8).

For the input to SVM, each protein must be represented by a feature vector. We use two methods to represent a protein as a feature vector, which be discussed in detail seperately. SVMs are used to train local predictors for each molecular function term in the ontology.

In Figure 2.8, the hyperplane is a linear discriminant. Support Vector Machines can use a dual formation that allows for the use of *kernels*, which allow hyperplanes to be non-linear in the feature space. Although these more complex formulations of SVMs sometimes prove to be more accurate, we limit ourselves to linear SVMs. We use linear SVMs because they prove to be very accurate on a wide range of problem domains, and their results are easily explainable to users. The linear SVM representation of the classifier function is:

$$f(\overline{x}) = sign(\overline{x} \cdot \overline{w} - b) \tag{2.2}$$

The prediction for an instance, $\overline{x}$ is predicted as positive if the sign of $\overline{x} \cdot \overline{w} - b$ is positive, and vice versa. Remember that each SVM is a local predictor for a single molecular function. So, for the SVM trained at the node "electron transfer activity", the predictor tells us whether or not a protein enables the movement of electrons throughout or in between cells.

In Equation 2.2, the weight vector, $\overline{w}$, and the bias term b are calculated during training. An advantage to using linear SVMs is that the weight vector and bias terms are meaningful. Figure 2.9 shows an example where the feature vector contains 4 terms, and is run through a classifier function. The classifier function has been trained, and is represented by the weight vector $\overline{w} =< 0.95, -0.44, 0, 0.5 >$, and the bias term $b = 0.6$.

The instance being predicted contains features $< \overline{x}_1, \overline{x}_2, \overline{x}_3 >$, whereas the feature $\overline{x}_4$ is absent. These feature could represent any property of the instance (such as the protein's tertiary structure, or biochemical properties) and in general could be real values. For simplicity, consider the case when the features are either 0 or 1 depending on whether the associated token is present or absent from the instance, respectively.

In Equation 2.2, the two vectors $\overline{x}$ and $\overline{w}$ are combined using the dot product, which is the sum of the product of each vector's components (Equation 2.4). Since the dot product is intuitive, each term in the sum of Equation 2.4 can be thought of as a contribution to the prediction of the SVM. Therefore, $\overline{x}_1$ will contribute 0.95 to the prediction (a positive contribution) since it is present, $\overline{x}_2$ will contribute -0.44 to the prediction (a negative contribution) since it is present, and $\overline{x}_3$ will not contribute anything to the prediction of the SVM classifier (since it is probably uncorrelated with the predicted label). Finally, $\overline{x}_4$ contributes nothing to the prediction since its associated token is not present in the instance. The final prediction of the classifier is negative for this particular function. In real-world applications the feature vector is often thousands of terms long, which makes the training of an SVM much more difficult than this simple example suggests. Therefore, a standard SVM library is usually used to implement the SVM model. We use the LIBSVM [16] implementation of SVMs.

This example shows that linear SVMs show great potential for explainability [41]. To users of the system, the predictions can be made intuitive by viewing

i	$\overline{x}_i$	$\overline{w}_i$	$\overline{x}_i\,\overline{w}_i$
1	1	0.95	0.95
2	1	-0.44	-0.44
3	1	0	0
4	0	0.5	0

$$\begin{aligned} b &= 0.6 && (2.3) \\ f(\overline{x}) &= sign(\overline{x} \cdot \overline{w} - b) \\ &= sign(\sum_{i=1}^{n}(\overline{x}_i\overline{w}_i) - b) && (2.4) \\ &= sign((0.95 - 0.44 + 0 + 0) - 0.6) \\ &= sign(-0.09) \\ &= negative \end{aligned}$$

Figure 2.9: SVM Transparency

An example showing how the results of a linear SVM prediction are interpretable by users. Each feature's contribution to the prediction is that feature's value ($\overline{x}_i$) multiplied by the corresponding weight in the classifier function ($\overline{w}_i$).

each feature's contribution to the final prediction. Because we want to build user confidence in the prediction system, we believe this transparency of predictions is vital to an automated system, and thus linear SVMs are used for all experiments.

PFAM SVM

The PFAM database [12] is a collection of domains shared by functionally similar proteins. Each PFAM domain is created by first performing a multiple sequence alignment on a set of similar proteins (similar in terms of function and/or structure). These domains are then used to construct Hidden Markov Models, which can then be used to detect these domains in others protein sequences. A single protein may contain more than one PFAM domain.

For our predictors, each protein is run through HMMer [20], which detects PFAM domains in sequences. If a protein has a PFAM domain, this domain is used as a feature describing the protein. Currently, there are over 7000 domains in the PFAM database.

Proteome Analyst SVM

Proteome Analyst (PA) is a tool used for predicting the general function [46][47] and subcellular localization [34][33] of proteins. The PA tool works by taking an input protein sequence, and finding similar sequences in Swiss-Prot using BLAST.

PA then extracts information from the annotations of these similar proteins from Swiss-Prot. This information is then used as features for a naive Bayes classifier.

For our experiments, we use Proteome Analyst to extract features for proteins in the data set. Although the web-based Proteome Analyst site uses naive Bayes classifiers, for our experiments, PA is used to extract features, which are then used to train Support Vector Machine classifiers. An SVM classifier is trained using PA features for each node in the ontology.

2.5.3 Probabilistic Suffix Trees

Probabilistic Suffix Trees [13] (PSTs) are a way of representing strings as variable length Markov chains. Since proteins can be represented as a string of amino acids, PSTs readily apply. PSTs have been shown to be good predictors of molecular function in the past [39].

Given a protein sequence (such as the one shown in Figure 1.2), $s = s_1 \dots s_m$, where s_i represents a single amino acid, we can create a probability model of a set of sequences using a Markov chain [39]:

$$\begin{aligned} P(s) &= P(s_1 s_2 \dots s_{m-1} s_m) && (2.5)\\ &= P(s_1)P(s_2|s_1)P(s_3|s_1 s_2)\dots \\ &\quad P(s_{m-1}|s_1 \dots s_{m-2})P(s_m|s_1 \dots s_{m-1}) && (2.6)\\ &= P(s_1)P(s_2|s_1)P(s_3|s_2)\dots \\ &\quad P(s_{m-1}|s_{m-2})P(s_m|s_{m-1}) && (2.7)\\ &= P(s_1)\prod_{i=2}^{m} P(s_i|s_{i-1}) && (2.8) \end{aligned}$$

Going from Equation 2.6 to Equation 2.7 is done through the first-order markov assumption. We investigate several n-order markov models in Chapter 5.3. Although it may initially seem non-sensical to look at individual amino acid distributions in a protein to predict function, it does prove to be accurate for some functions [40]. Other functions depend on larger functional domains, and these are modelled by varying the value of n.

To make a local predictor for molecular function using PSTs, we create a model for the proteins annotated with a molecular function (the positive model, $P_+(s)$), and also a model for those that are not (the negative model $P_-(s)$). The probabilities of these two models are then combined into a single score using a log-odds ratio (Equation 2.9). We then find the log-odds ratio threshold that maximizes precision and recall during cross-validation for each PST local predictor. This score is then used for future predictions.

$$\textit{log-odds ratio}\,(s) = \log\left(\frac{P_+(s)}{P_-(s)}\right) \qquad (2.9)$$

2.6 Summary

Protein function prediction is an important problem in Bioinformatics. Machine learning has proven to be an effective way of leveraging the vast amount of data that has come out of many proteomic sequencing projects. Hierarchical ontologies such as Gene Ontology provide a standardized set of terms describing protein functions.

Recent work has shown that the hierarchical structure of these ontologies demonstrate great potential for improving the quality and efficiency of machine learning algorithms. We use a wide variety of bioinformatics approaches to predict protein function, and will attempt to leverage the hierarchy to improve the quality of these predictors.

Chapter 3

Hierarchical Classification using Local Predictors

There are several considerations for evaluating a prediction system over a hierarchical ontology such as GO, as opposed to a flat ontology. These considerations stem from the fact that the predictions must be consistent with the semantics of the hierarchy. This Chapter first discusses data set construction for the experiments described in this dissertation. Then, the issues that are imposed by working within the context of a hierarchical ontology are addressed. Finally, the use of traditional machine learning predictors in the context of hierarchical classification is examined using a specific local approach.

3.1 Data Set

Our data set consists of protein sequences, and their respective molecular functions. This data set is created using a combination of three sources: the Gene Ontology controlled vocabulary, the UniProt protein database, and the Gene Ontology Annotation project's annotations for proteins in the UniProt database.

3.1.1 The Gene Ontology controlled vocabulary

The molecular function ontology is taken from the Gene Ontology website [4]. The Gene Ontology contains three aspects of protein annotation (Figure 2.2), but all experiments in this dissertation focus on the molecular function aspect. The August 28, 2004 version of the GO molecular function ontology is used.

The ontology defines the possible molecular function annotations. These are standardized terms used to describe potential functions that proteins perform, and the structure of the ontology defines the logical relationship between these terms. These functions range from general (near the root of the ontology), to specific (near the leaves of the ontology).

Since the knowledge of biological systems is constantly changing, the Gene Ontology Consortium regularly releases new versions of the ontology. As knowledge about the possible functions of proteins is gained, terms are added to the ontology. As our understanding of this knowledge changes, the structure of the ontology can change. Terms may be removed if they are felt to be redundant, the wording of functions may be changed if their intent is unclear, and parts of the ontology may be restructured if they do not adhere to the intended semantics of the hierarchy. Although the ontology is not perfect, there is a great deal of information about biological knowledge encoded within its structure that will be useful in later experiments.

3.1.2 The UniProt Database

Our protein sequence data is obtained from the UniProt database [11]. The UniProt database consists of protein annotations, along with protein sequences. Release 27 of TrEMBL, and release 44 of Swiss-Prot (which together comprise UniProt release 2.0) are used.

The UniProt database is a joint database, containing Swiss-Prot and TrEMBL databases [14]. The Swiss-Prot database contains high-quality, human-curated protein annotations. The TrEMBL database contains electronically annotated proteins that have not yet been added to Swiss-Prot. Although the annotations in TrEMBL are of a lower quality, we only use the sequence data, which is reliable. Initially, our data set was created using only proteins from the Swiss-Prot database, since it is often considered to be of higher quality. However, our experiments have shown that each predictor has performed similarly on the entire UniProt database, which shows that the GOA annotations for proteins in the UniProt database, and the sequence data itself is of consistent quality with the Swiss-Prot database. Using the UniProt database also results in a larger ontology after pruning. The Gene Ontology annotations for the proteins are derived from the Gene Ontology Annotation project.

3.1.3 The Gene Ontology Annotation Project

The Gene Ontology Annotation (GOA) project [3] at the European Bioinformatics Institute assigns GO terms to proteins in the UniProt database. Each of these annotations is accompanied by an evidence code, which states how each was derived. To create a reliable data set, only those annotations that were not assigned using computational methods are used. As shown by evidence codes, all experiments in this dissertation were derived from a biological experiment, rather than a computationally predicted annotation. This allow for more confidence in the labellings, and ensures that as little bias as possible is introduced into the data set, while keeping the data set large. Bias is impossible to avoid completely, since the biologists annotating proteins will have an inherent bias as to which will be studied, and due to inherent flaws in experimental methods. The August 11, 2004 version of the GOA mapping file is used. Table A.1 (in Appendix A) shows how many annotations exist for proteins in Swiss-Prot and UniProt for each of these annotation codes.

Annotation evidence codes that are included in the data set:

1. **IDA (inferred from direct assay)** - Refers to a biological experiment using direct assays.

2. **IEP (inferred from expression pattern)** - The annotation is inferred from the timing or location of expression of a gene, as measured by an experiment.

3. **IGI (inferred from genetic interaction)** - Experimental data about interactions between genes.

4. **IMP (inferred from mutant phenotype)** - Annotations derived from mutations or abnormal levels of products.

5. **IPI (inferred from physical interaction)** - Interaction data such as yeast-2-hybrid interactions.

6. **TAS (traceable author statement)** - Knowledge with a traceable experiment, or "common knowledge", as in text books, etc.

Annotation evidence codes that are **not** included in the data set:

1. **IC (inferred by curator)** - Annotations that are reasonably inferred from existing GO annotations. Since we do not know how the annotations that these are inferred from were obtained, we exclude these annotations.

2. **IEA (inferred from electronic annotation)** - These annotations are obtained through some form of computational method such as BLAST, or from another database. In the former case, we can not assume BLAST will elucidate the true function of a protein, and in the latter case we do not know how the entry was annotated.

3. **ISS (inferred from sequence or structural similarity)** - This is a computational method.

4. **NAS (non-traceable author statement)** - Since this knowledge is not traceable, it is ambiguous.

5. **ND (no biological data available)** - Ambiguous.

6. **NR (no record)** - Ambiguous.

The final dataset consists of 14,362 proteins, each labeled with their experimentally-verified functions. UniProt provides the proteins, Gene Ontology provides the ontology, and the GOA project provides the molecular function annotations for many of the proteins in UniProt. The combination of these sources maps proteins to the Gene Ontology hierarchy of terms, as depicted in Figure 2.5. Only those proteins in the UniProt database that have valid annotations in the GOA project are kept in our dataset.

There are 7,399 nodes in the August 11, 2004 version of the GO molecular function hierarchy. However, to create accurate local GO term predictors, a sufficient number of positive training instances is required. Therefore, only those GO terms that have at least 20 proteins annotated at or below them in the hierarchy are considered. This decreased the size of the ontology to 406 nodes. More statistics about the hierarchy are summarized in Table A.2 (in Appendix A), and the entire ontology is shown in Appendix C.

3.2 Evaluation Issues within a Hierarchical Ontology

To be able to objectively evaluate a prediction system, two requirements must be met. First, a quantifiable measure of the quality of predictions is required to compare various methods. There exist standard measures of precision, recall, and accuracy for traditional classification problems [33], but these measures do not apply to the case when instances can have one or more labels. Also, when the ontology has a structure, different types of errors should be scored differently, whereas in traditional evaluation schemes all prediction errors are treated uniformly. With some modifications, the traditional precision and recall metrics can be modified to extend to hierarchical classification and multiply labeled instances. Several other measures have also been proposed in literature, and they will be examined as well.

Second, predicting the function of *unknown* proteins is the ultimate goal of this thesis. Thus, the prediction process should simulate the classification of unknown proteins when evaluating performance. In machine-learning terminology, test error (prediction quality on previously unseen instances) is considered more important than training error (prediction quality on the data used for training). Traditionally this is accomplished using hold-out sets, or cross-validation [7]. This causes complexities in the context of hierarchical classification, and must be dealt with carefully. The following two sections of this Chapter will address these requirements in detail.

3.2.1 Scoring Predictions

Proteins can be assigned multiple labels (each protein is assigned an average of 1.35 experimentally verified functions in GOA). However, some classifier technologies can only predict one label per instance. Two options are available. The first is to try to change the formulation of a classification algorithm so that it can predict multiple functions. Another approach in such cases is to build a series of "local" *binary predictors* that predict "yes" or "no" for each term in the ontology. These

Table 3.1: Binary Confusion Matrix

A confusion matrix is a visual representation of predicted labels vs. known labels for each instance that has been run through a binary classifier. Each of these entries are mutually exclusive.

	Predicted Positive	Predicted Negative
Known Positive	True Positive (TP)	False Negative (FN)
Known Negative	False Positive (FP)	True Negative (TN)

are called binary since they predict one of two values, and local since they predict for a single node in the ontology.

A binary classifier is trained for each term in the ontology, and during the prediction process, each classifier returns a positive or negative prediction for their corresponding term in the ontology. For example, the local predictor for the GO node "enzyme inhibitor activity" only predicts whether an instance does or does not perform the function of inhibiting the activity of an enzyme.

When evaluating the predictions of a single binary classifier, there are four categories that each prediction can fall into: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN), which are summarized in Table 3.1. This arrangement of predictions for a classifier is called a *confusion matrix*. For example, if an instance is known to be a positive, but the classifier predicts negative, then this instance is a False Negative, and so on.

The *precision* measure, shown in Equation 3.1, shows how many of the positive predictions from a binary classifier were actually labeled with that term. *Recall*, shown in Equation 3.2, is the percentage of the positive instances that are predicted as positive by the predictor. An ideal classifier has high precision *and* high recall.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

If a molecular biologist is using a predictive system to obtain an idea of a protein's function, precision and recall would have different levels of importance depending on the situation. If the biologist is concerned with having those functions that are predicted as positive as being very likely to be correct, then precision should be high. In this case, this is usually accompanied by an increase in correct functions being predicted as negative (false negatives). If the biologist is concerned with making sure all of the true functions are predicted as positive, then recall should be high.

Similarly, there will often be an increased amount of incorrect positive predictions (false positives).

To optimize a predictive system, a single measure of its performance is required. A first approach may be to use accuracy (Equation 3.3). However, when working within a large ontology where most of the labels are negative, accuracy will always be high when a small number of labels are predicted, even if our predictions are not correct at all. In the data set used for experiments, approximately 2% of the terms are assigned to each protein on average, which leaves 98% as negative. Accuracy tends to be high in this case because True Negatives (TN) would dominate the accuracy score, keeping it high even when prediction quality is intuitively poor.

An alternative is to use an average of precision and recall. This is a standard machine learning approach, and is called the F-measure, shown in Equation 3.4. F-measure uses a harmonic mean as a weighted average of precision and recall. When β is set to 1, precision and recall are given equal weighting. As the value of β increases from 1, recall is given more weight. As the value decreases from 1, precision is given more weight. β can be adjusted in accordance with which measure is considered more important. For all experiments, we use $\beta = 1$.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

$$F - measure = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall}, \beta \in [0, \infty) \tag{3.4}$$

These measures work for a single classifier, whether it be binary or multi-class, as long as a maximum of one label can be predicted per instance. However, since proteins can be assigned multiple positive labels, these measures do not apply in their current form.

3.2.2 Scoring Predictions in a Hierarchy

In hierarchical classification, all resulting predictions must obey the true path rule. Therefore, as a post-processing step, all positive predictions are propagated upward in the ontology. This means that even if a protein could only perform a single function, a single annotation would represent multiple terms in the hierarchy. Therefore, the issue of multiply labeled instances implicitly applies to all prediction within a hierarchy, not just when multiple terms are explicitly assigned to instances.

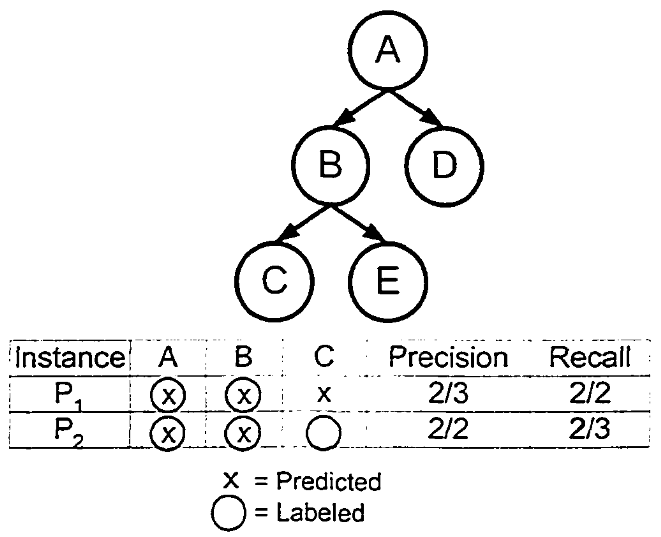
One solution is to compute precision and recall for each local predictor as shown above, and then average the results. This is also called *hierarchical macro averaging* [36] by Moskovitch *et al.* This approach however gives each class in the hierarchy equal weighting. A class that may describe half of the instances is given the same weighting as a class that describes only 1% of instances. Another approach is to compute scores for each instance and average these scores. This approach is called *hierarchical micro averaging* [36]. Here, equal weight is given to proteins regardless of how many functions they serve, or how many functions are predicted for each. Both approaches suffer from bias. Our approach however is based on predicted and annotated labels, and therefore class distributions are intrinsically taken into account.

Intuitively, predictions that are "close" to the correct label should score better than predictions that are in an unrelated part of the hierarchy. An evaluation methodology should be simple, intuitive, and consistent with the true path rule. For the evaluation methodology we first take the predicted labels for each protein, and add all of these labels' ancestor nodes in the hierarchy to the set of predicted labels. This makes the prediction consistent with the true path rule. If the protein has another predicted label in an unrelated part of the hierarchy, those labels and their propagated labels are also added to the set of predicted labels. This propagation is applied to the correct labels for proteins as well. Now that both the labeled and predicted sets are computed for a protein, the True Positive, True Negatives, False Positives, and False Negatives are tabulated between these two sets. This is done for each protein in the data set, and TP, TN, FP, and FN are calculated over the entire set. Then, the formulas for precision and recall are applied as usual.

Failure to follow the true path rule, leads to distorted evaluation metrics. Incorporating propagation into the evaluation of predictions allows for a graduated scoring system where distance in the ontology is intrinsically taken into account. Hierarchical precision and recall reflect how close, conceptually, predictions are to the correct labels in the ontology.

For example, consider a term hierarchy where A is the parent of B which is the parent of C (Figure 3.1). Assume that protein P_1 is labeled $\{B\}$ by an oracle and protein P_2 is labeled $\{C\}$ by an oracle. By the true path rule, the labeling really should be $\{A, B\}$ for P_1, and $\{A, B, C\}$ for P_2 after propagation (shown by circles in Figure 3.1). Assume that for protein P_1 we predict the label to be $\{C\}$, which is different than the oracle. By the true path rule, we then predict the labels $\{A, B, C\}$ (shown by x's in Figure 3.1) for P_1 as well. Similarly, for P_2 we predict the label to be $\{B\}$ and propagate to get $\{A, B\}$. Both of the initial labels for P_1 and P_2 are different than the oracle, which misleadingly suggests a poor prediction. But, hierarchical precision and recall allow for an evaluation scheme which is more in tune with intuition.

Despite the differences with the oracle, our prediction for P_1 should have perfect recall, since it correctly recalled that P_1 has terms $\{A, B\}$. But the precision is $2/3$ since only 2 out of the 3 predicted labels were correct, which is an intuitively sound penalty for the imperfect prediction. Similarly, our prediction for P_2 should have perfect precision, since every predicted term is correct, but recall is $2/3$ since only 2 out of 3 correct labels were recalled. This example shows that predicting too high in the hierarchy (i.e. P_2) reduces recall, but does not affect precision and that predicting too low in the hierarchy (i.e. P_1) reduces precision, but does not affect recall. Lastly, a prediction that is in the wrong part of a hierarchy altogether (not



Instance	A	B	C	Precision	Recall
P_1	ⓧ	ⓧ	x	2/3	2/2
P_2	ⓧ	ⓧ	○	2/2	2/3

Figure 3.1: An Evaluation Example

Predicting too deep in the hierarchy (e.g. P_1) results in high recall but at a cost of precision. Predicting too shallow in the hierarchy (e.g. P_2) results in high precision but at a cost of recall.

shown in the example) will have neither high precision nor high recall. The ability to handle close predictions and altogether wrong predictions are important aspects of this hierarchical evaluation methodology.

The original, formal presentation of this approach to *multiclass precision* and *multiclass recall* was made by Poulin [39] and in this dissertation, it is extended to hierarchies by propagating labels according to the true path rule. Independently, the same approach was used by Kiritchenko *et al* [27], also in the domain of GO, and a formal definition was published later [28]. These measures are known as *hierarchical precision* and *hierarchical recall*. A similar scoring metric was also presented by Wu *et al* [53], but it is a single measure, and thus lacks the intuitive value of precision and recall. Since all experiments are within a hierarchy, precision and recall will be used to refer to hierarchical precision and hierarchical recall respectively for the remainder of this dissertation.

Other approaches, such as the one presented by Lin [31], have attempted to model the distances between classes in the hierarchy by using information content of the classes. Lord *et al* have shown that these measures do correlate with sequence similarity [32], however there are two problems with this approach. First the assumption in the paper is that sequence similarity from BLAST is the ground truth for representing Gene Ontology classes. Second, all of the measures presented lack the ability to represent both precision and recall in an intuitive manner within a hierarchy that our measures of hierarchical precision and hierarchical recall have.

The major downside to our approach is that it assumes that each edge in the hierarchy represents the same distance. For example, it is unclear how the distance between the nodes "nucleic acid binding" and "DNA binding" compares to the distance between the nodes "kinase regulator activity" and "kinase inhibitor activity". Several attempts have been made to quantify these differences [31] [51] [50]. However, in this dissertation we do not attempt to address this issue, opting instead for our evaluation measures which provide the intuitive measures of precision and recall.

Another important aspect of evaluating predictions is *coverage*. Coverage is the percentage of proteins for which we are able to make predictions (Equation 3.5). If no predictions are given for a protein whatsoever, the prediction system has not contributed anything to the knowledge of the protein and coverage would be lowered. Therefore, a predictive system should have a high coverage to push the bounds of protein annotation as much as possible, while retaining high precision and recall so that the predictions are useful.

$$Coverage = \frac{NumberOfInstancesWithPredictions}{TotalNumberOfInstances} \quad (3.5)$$

The measures of Precision, Recall, and F-measure will be given the most importance when evaluating experiments throughout this dissertation. Although recall indirectly measures coverage, the issue of coverage will be addressed when it is relevant.

3.2.3 Cross-Validation in a Hierarchical Ontology

A classifier may perform well on the training data (the data given to create the classifier), but users of a classification system are generally more concerned with how well it will perform on new instances. That is, when we are faced with an unknown protein (that was not in the training set), we wish to know how well we can predict its molecular function. Although this is impossible to know exactly, there are several ways of approximating the prediction of unknown proteins for evaluation purposes.

The simplest method is to divide the entire data set into two parts. The first part is used as the training set, and the second is used as a testing set. This method is called hold-out validation, and ensures that during the creation of the classifier, there is no knowledge of the testing set. When the evaluation is performed, we simply measure the performance of the classifier on the test set. The problem is that the set chosen for evaluation may have a disproportional amount of proteins that the classifier performs well or poorly on. Therefore, the result of the evaluation in hold-out validation may not be indicative of the performance of the final classification system applied to unknown protein sequences.

To address this problem, a technique called *cross-validation* is often used (Figure 3.2). For all experiments, 5-fold cross-validation is used. In 5-fold cross-validation, the data set is first split up into 5 parts of equal size. Then, for 5 iterations, one fold is withheld as the test set, and the remaining 4 folds are used as

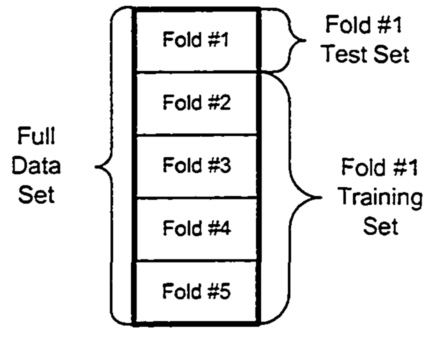


Figure 3.2: 5-Fold Cross-Validation

In iteration 1, fold 1 is used as test data, and all other folds are used as training data (shown in figure). For 5 iterations, each fold is used as testing data, and all others are used as training data.

the training set to create classifiers. Precision, recall and F-measure are computed for the predictions made for each fold. The statistics for the 5 folds are averaged to give an accurate representation of the predictive performance of the classifier on future instances.

An advantage of cross-validation is that it helps guard against the problem of *overfitting*. Overfitting refers to a classifier performing very well on training data, but poorly on new, test data. Since cross-validation simulates the process of predicting on unknown data, overfitting can be recognized by poor predictive performance during cross-validation.

When evaluating a system of local predictors in a hierarchical classification system, this limits the way in which cross-validation can be performed. One approach is to perform cross-validation on each of the local predictors individually. This way, when splitting the data set into 5 folds at a node, we can ensure that a sufficient number of positive training instances are kept in each of the folds. The classifiers trained for each fold will then perform similarly to the classifier in the final system. This method will be referred to as *local cross-validation*.

In the absence of a hierarchical ontology, local cross-validation is sound. The problem with local cross-validation is best illustrated with an example. Suppose protein P_1 is annotated with labels A and B (as in Figure 3.1), and that the local cross-validation folds at A and B are not identical. Now, consider the case when the predictor for node A predicts negative and the predictor for node B predicts positive for P_1. Since our predictions must obey the true path rule, we may choose to propagate the positive prediction at B upward in the ontology[1]. During this propagation the negative prediction at A must be overridden with a positive prediction. Thus, there is interaction between the predictions for A and B. During cross-validation, a

[1]The argument can be reversed if we choose to propagate negative predictions downward instead of predicting positive predictions upward. Either way the predictions must be consistent with the true path rule.

predictor can only use those instances within the current fold's training set to make predictions. Since the training sets were not the same (due to different splits of the data into folds) at A and B, the prediction at B used data which was in A's training set, and in B's testing set, indirectly. This is a violation of the purpose of cross-validation, since in testing data has an influence on prediction.

Therefore, in hierarchical classification, the protein should either be in the test set (1 fold) or the training set (the remaining 4 folds) in a consistent, global manner across all GO terms and nodes. This strategy will be referred to as *global cross-validation.* With this approach, protein P_1 is always in the same fold (e.g. fold 2 of 5) for all nodes. Thus, when globally evaluating the fold containing P_1, no predictor for any node should use knowledge of P_1 directly, or indirectly. However, a global split of the data into folds that both maintains the local node distributions (the number of positive and negative training instances are constant across the folds at each node) and is globally consistent in assigning instances to folds may be difficult to obtain. For example, if we try to preserve the distribution of a specific label, then by splitting the instances into folds within this label, we may have forced another label's training data distribution to become vastly different from the original data set (since a global partitioning of the data is being used). Failure to maintain local node distributions results in fewer training instances for some folds, which results in poor classifier performance. As discussed above, failure to do global fold assignment leads to an inconsistent use of training versus test instances. To obtain good accuracy, the number of positive and negative training instances in each fold should be approximately the same.

The first approach was to ignore local node distributions. The training set was randomly split into 5 folds, and these folds were used to evaluate the predictors during cross-validation in a consistent manner using the scoring method presented. Other approaches will be discussed in Chapter 4 of this dissertation to address the local node distribution problem.

3.3 Predicting Protein Function with Local Predictors

Table 3.2 shows the results of evaluating PA-SVM predictors (Proteome Analyst features for SVMs) for each node in the Gene Ontology hierarchy. Precision is quite high, but the method suffers from a low recall.

In this case, the PA-SVM predictors for each node in the ontology must be computed for each query sequence. This can be costly, especially when compared to the low cost of running a global predictor such as BLAST. The cost in Table 3.2 is the number of predictors that had to be computed for each protein. In this case, since there is one predictor for every node in the ontology, and there are 406 nodes in our pruned ontology this cost is the execution of 406 local predictors.

One approach to lowering the computational cost of predicting in a hierarchy

Table 3.2: Training local predictors using Proteome Analyst features

An SVM classifier is trained for each node in the GO hierarchy. Statistics shown are during global cross-validation. Precision, Recall, and F-measure are all the hierarchical variants presented in Chapter 3.2.1.

Measure	Value
Precision	0.758
Recall	0.328
F-measure	0.458
Cost per Protein (Number of Local Predictors Computed)	406

Table 3.3: Lowering the cost of using Local Predictors.

A top-down approach is used to lower the computational cost of predicting within the hierarchy. The computational cost cannot be lowered with this method without a significant penalty to recall.

Method	Precision	Recall	F-measure	Average Cost per Instance
All PA-SVM	0.758	0.328	0.458	406
Top-Down PA-SVM	0.572	0.002	0.004	10
TD-1 PA-SVM	0.719	0.099	0.173	72
TD-2 PA-SVM	0.728	0.163	0.266	181
TD-3 PA-SVM	0.749	0.232	0.354	291

is to use a top-down decision model. In a top-down model, we start at the root node, and check all children nodes. If any of the children node's local predictors predict positive, then those terms are added to the list of predicted labels and we test the child nodes of the positive predicted labels. Then we recursively apply the decision algorithm until we reach the leaves of the ontology, or until all current local predictors predict negative.

The results of using a top-down model (also called a pachinko machine [29]) are shown in the row marked "Top-Down" PA-SVM in Table 3.3. This method has lowered the cost of prediction from running 406 predictors to an average of 10 per protein. However, the recall of the predictor has been significantly reduced.

The loss of precision by using the top-down approach may be counter-intuitive initially. One reason for this is that general classes may be harder to represent than more specific classes. These general classes actually contain more variation in functional classes that more specific ones. This point was also previously made by Wang *et al* [51]. A top-down approach requires that predictors near the root be accurate. Otherwise the prediction system will not make predict beyond high level nodes.

One way to address this problem is to not record the decision made at a node permanently. For instance we could allow a negative prediction at a node, and still compute the children nodes. The approach called "TD-1 PA-SVM" uses this technique, and computes children nodes of a negative prediction once. If a node, and all of its children nodes predict negative, then the search stops. The other approaches, "TD-N PA-SVM" use the same approach, but they accept N negative predictions in a row during the top-down search. In this approach, if a node predicts negative, and one of its descendant nodes predicts positive, the true path rule is used to override the negative prediction.

However, no matter how much the cost is lowered, the prediction system will still produce low recall. Also, when the recall is almost as high as computing all local predictors (TD-3 PA-SVM), then the cost is not much lower than computing all nodes. Leaving out some nodes from the prediction can only increase precision, and will often reduce recall. In Table 3.3 we can see that recall is in fact the area that this prediction technique suffers the most. Therefore, before considering lowering the computational cost of using local predictors, we must first increase the predictors' recall.

3.4 Summary

The data set used for experiments is a combination of three reliable sources: the Gene Ontology Annotation project at EBI, the molecular function aspect of the Gene Ontology, and sequence data from the UniProt database. We only use those protein annotations that correspond to reliable experimental results when creating and evaluating predictors. This way we introduce the least amount of bias as possible into our predictors.

When predicting within a hierarchical ontology such as GO, the issues of evaluation functions, and cross-validation must be revisited. Even when not predicting within a hierarchical ontology, but when multiple labels per instance are valid, the evaluation functions must be readdressed. We present a way of addressing these issues in the context of protein function prediction, but they apply to hierarchical classification in general.

Finally, we showed that building a local predictor at each GO node using Proteome Analyst features for SVMs produces a classification system that has high precision but low recall. Furthermore, each local predictor must be computed for every query protein. We can lower the cost of this predictive system by using a top-down decision model, however we notice a large negative impact to precision and recall with this approach.

Chapter 4

Hierarchy-Aware Local Predictors

In Chapter 3, local predictors were created without fully exploiting the structure of the Gene Ontology. The ontology's structure was considered during evaluation, since doing otherwise would be unsound (it would violate the true path rule), but the structure was not considered during the construction of local predictors. Furthermore, during prediction time it was difficult to lower the computational cost of prediction without penalizing predictive performance. In this Chapter, the creation of local predictors will be readdressed with the ontology's structure in mind to attempt to create better predictors.

The issues that were addressed in Chapter 3 – global cross-validation and a scoring scheme – still apply here, since these issues are independent of how predictors are created and used. However, the issue of global cross-validation is revisited, to attempt to maintain local node distributions between folds. No matter which prediction method is used, the same scoring methodology is applied. Therefore, the issue of scoring predictions will not be readdressed.

First, the issue of training set construction of local predictors is explored. It is shown that local predictors can perform better when the hierarchy is considered during their construction. A spectrum of methods are explored for training set design, and each is evaluated.

Second, the idea of lowering the computational cost of prediction using local predictors is revisited using a new training methodology. When using an *inclusive training* strategy (defined later in this chapter), the top down approach can reduce computational cost without incurring a large penalty to precision and recall.

The issue of global cross-validation is then readdressed. Better splits of the data are found so that a sufficient number of positive training examples are maintained across each fold during cross-validation. The effect of this fold design is evaluated.

4.1 Training Set Design

For local predictors, we must train classifiers before any prediction is performed. To obey the spirit of cross-validation, only those sequences in the current fold's training set can be used to create the local predictors for each GO node. However, it is valid to use the sequences in the training set in any way to create each node predictor, therefore a spectrum of methods which we denote exclusive to inclusive will be compared.

4.1.1 Exclusive vs. Inclusive Classifiers

During the construction of a local predictor, the proteins that will be used to represent the positive model (representing this particular molecular function), and the negative model (those outside the particular molecular function) are selected. In Chapter 3, we chose all those proteins which were explicitly assigned a GO term N as that term's positive training data, and all other proteins as the term's negative training data. This approach is the traditional method used to create classifiers when the ontology is flat.

However, this approach ignores the relationships between the terms in the ontology when creating the training set. Although this approach may seem naive, this training set design could, in theory, produce a classification system that predicts perfectly in terms of hierarchical precision and hierarchical recall. In fact, previous research in hierarchical classification has shown that this method can perform quite well on some data sets [29].

Intuitively, when using this approach for training set design, we are creating a local predictor that will only predict positively for those proteins that belong exactly at this node, and not to any more general or specific node, as shown in Figure 4.1. Those proteins which belong at a particular node are predicted as negative by all local predictors[1] except for the one where it belongs (according to the GOA annotations). We call these classifiers *exclusive classifiers*, since proteins are excluded from all nodes except for the exact location where it is annotated. Of course, a post-processing step is needed if an exclusive classifier is used, since all parent nodes of the predicted node would be added to the prediction to satisfy the true path rule. This should not be regarded as an error in prediction since the exclusive classifier is behaving exactly the way it has been designed – to pick the most specific node in the hierarchy that applies to the protein.

However, as Chapter 3 showed, this approach did not show promising results in terms of recall. Also, when attempting to lower the cost of the local predictors, there was a large decrease in recall since the classifiers are exclusive. Figure 4.1 shows intuitively why a system of exclusive local predictors will perform poorly when using a top-down approach. A top down predictor would stop before reaching node N since there are negative predictions at parent nodes.

There are a series of observations that can be made which will make the training sets for each local predictor more consistent with the ontology:

1. The descendant nodes of N are not good negative instances for the predictor at N since they are positive according to the true path rule. Including these

[1]Assuming the protein does not have another, unrelated function, for simplicity.

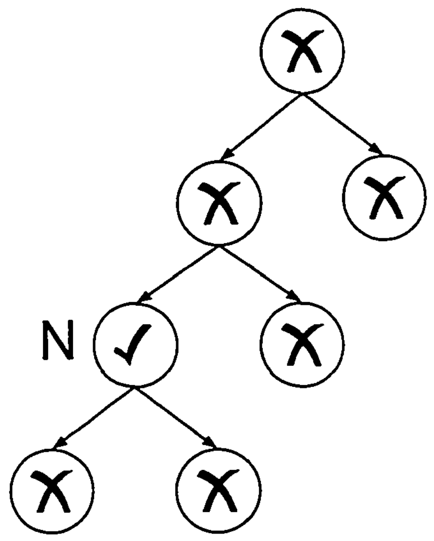


Figure 4.1: An Exclusive Classifier System

A perfect exclusive classifier only predicts positive exactly where an instance should lie. The correct label for this instance is N. The predictions for each local exclusive classifier are shown by check marks for positive, and X's for negative.

instances as negative in the training set could confuse the classifier training algorithm, since they are actually positive. Therefore these instances are excluded from the set of negative examples. This strategy is labeled as *less exclusive* in Table 4.1.

2. All descendants of N are not only poor negative examples, but they could in fact be used as positive training examples for N, due to the nature of the is-a relationship in the GO hierarchy. Ignoring this observation will limit the number of positive training examples that are presented to the classifier training algorithm. This approach has previously been presented in [28]. This method is called *less inclusive* in Table 4.1, and is consistent with the true path rule, and the nature of an is-a hierarchy in general.

3. To be most consistent with the hierarchy, observe that those proteins that are annotated as ancestors of N could in fact be instances of N. As was discussed in Chapter 3, proteins are annotated with the most specific function terms for which experiments have been performed. Since it is common for future experiments to supply more specialized terms, it could be dangerous to include proteins annotated with ancestor terms in constructing a negative training set for a term. On the other hand, we do not know that these proteins will be specialized to N in the future (they may be specialized to a sibling or not specialized at all), so they should not be included in the positive training set either. Therefore, they are not used in training at all. This most consistent

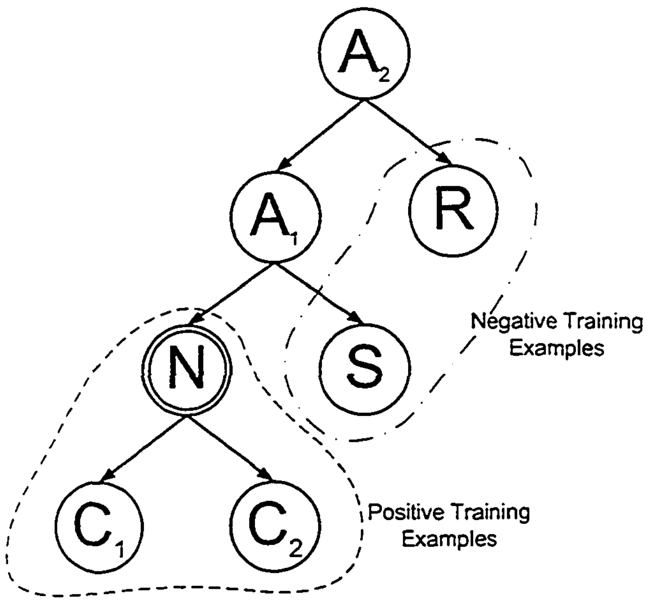


Figure 4.2: Inclusive Training Set Construction Scheme for Node N

The training set design for an inclusive local predictor at node N. Proteins mapped to parents of the current node (A_1, A_2) are not included in positive or negative training sets due to ambiguity.

approach is called *inclusive* in Table 4.1, and is depicted in Figure 4.2.

Intuitively, an inclusive classifier predicts positive for any proteins belonging within that node, and any proteins below that node. The result is that when a node belongs at node N, positive predictions are returned for the node predictor at node N, and all of the ancestor nodes of N (Figure 4.3). This is because all proteins below the current node were included in the positive training set.

Training set rule 3 disallows any proteins labeled with a node in the negative training sets of predictors for this node's child terms. One could also argue that future experiments could add any arbitrary new term to a protein, so that no negative training instances can be used with confidence. This is a good point, but it applies anytime a classification task can have multiple positive answers and negative (experimental) evidence is not available, not recorded, or incomplete. However, negative training instances are required and at least the more common case of more specific annotations following less specific ones is guarded against.

Assuming that we have a perfect classification system, both exclusive and inclusive classifiers in a hierarchy would perform perfectly on test data, due to the way hierarchical precision and hierarchical recall are calculated. When using perfect exclusive predictors, if a protein should be assigned GO node N, every node in the hierarchy will return a negative prediction, except for node N that returns a positive prediction. During evaluation, this prediction is propagated upward, and evaluates at 100% for precision and recall.

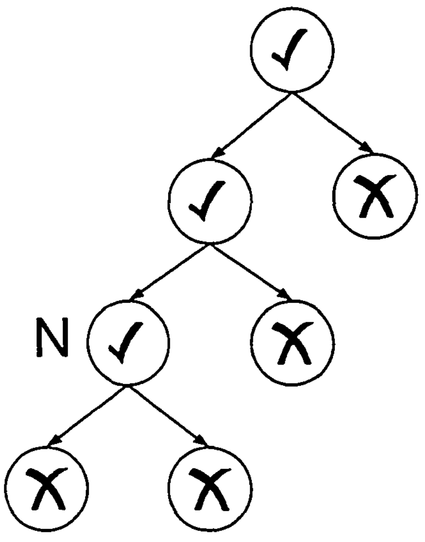


Figure 4.3: An Inclusive Classifier System

A perfect inclusive classifier predicts positive at each node it belongs to, according to the true path rule. The correct label for this instance is N. The predictions for each local exclusive classifier are shown by check marks for positive, and X's for negative.

In the inclusive case, given a protein that belongs at node N, a perfect term predictor for N predicts positive, and so do all of N's ancestors in the hierarchy. Here, the classification would evaluate at 100% precision and recall as well.

The less inclusive and less exclusive classifiers represent the intermediate points in the spectrum between the two training set designs. Since it is possible for both exclusive and inclusive designs to have perfect precision and recall, there is no *a priori* reason to choose inclusive or exclusive classifiers. However, real data and therefore classifiers trained from that data are often far from perfect. Thus, local predictors will often perform differently in practice than they do in theory. Thus, all four schemes to construct training data, which range from exclusive to inclusive in their nature, have been evaluated.

4.1.2 Comparison of Training Set Design Schemes

The four training methods (summarized in Table 4.1) were evaluated. For simplicity, the training methods were evaluated with a single technology. In this case, Proteome Analyst features in conjunction with Support Vector Machines were used. The PA-SVM classifier was chosen for this experiment since it has proven to be reliable during the lifetime of the Proteome Analyst project. The issues presented, however, apply to all local predictor methods, although the numerical scores could vary.

Table 4.2 summarizes the results of cross-validation for each of the four training

Table 4.1: Various Training Set Construction Schemes

This table describes four methods for choosing instances for a local predictor at node N. N denotes all proteins mapped directly to N. Descendants(N) are all proteins mapped to descendant nodes of N (C_1 and C_2 in Figure 4.2), and Ancestors(N) are all proteins mapped to ancestor nodes of N (A_1 and A_2 in Figure 4.2).

Method	Positive Examples	Negative Examples	Not Used
Exclusive	N	Not N	-
Less Exclusive	N	Not[N ∪ Descendants(N)]	Descendants(N)
Less Inclusive	N ∪ Descendants(N)	Not[N ∪ Descendants(N)]	-
Inclusive	N ∪ Descendants(N)	Not[N ∪ Descendants(N) ∪ Ancestors(N)]	Ancestors(N)

set construction methodologies. The precision of all four techniques is comparable, but there are significant differences in recall and F-measure.

The column "exceptions per protein" in Table 4.3 describes how often a local predictors predict positive, and an ancestor node's local predictor predicts negative. This is equivalent to the number of negative predictions that must be overridden with positives when we propagate to make predictions consistent with the true path rule. The fewer the exceptions, the more consistent the technique is with respect to the true path rule. By their nature, exclusive classifiers are more likely to have many exceptions, while inclusive classifiers are likely to have few exceptions (as depicted in Figure 4.1 and Figure 4.3). The data in Table 4.3 matches this intuition. As previously discussed, the evaluation methodology requires that we first propagate positive predictions upward in the ontology before computing precision and recall, which ameliorates the effect of exceptions, so our test is fair to all four strategies. The differences between techniques can be explained via differences in the size of the positive training set, and noise in the data used for training.

The column "exception precision" is the number of propagated predictions that are actually correct. That is, when a local predictor predicts positive, and this prediction is inconsistent, we must propagate the positive prediction upward in the hierarchy. When overriding ancestor terms with positive predictions, exception precision measures how often these overridden predictions were correct. The exclusive

Table 4.2: Comparison of Training Set Schemes

Each evaluation used the same split of the data for global cross-validation. The best values in each column are marked with bold text. The 95% Confidence Interval for the F-measure is also shown.

Method	Precision	Recall	F-measure	95% CI (F-measure)
Exclusive	0.758	0.328	0.458	±0.007
Less Exclusive	**0.777**	0.404	0.531	±0.008
Less Inclusive	0.773	0.638	**0.699**	±0.009
Inclusive	0.753	**0.652**	**0.699**	±0.009

Table 4.3: Exceptions for Training Set Schemes

Exceptions per Protein is the number of times that a positive prediction at a node has negative prediction at ancestor nodes. Exception precision is the precision on propagating positive predictions upward in the ontology.

Method	Exceptions per Protein	Exception Precision
Exclusive	1.524	0.794
Less Exclusive	1.739	0.805
Less Inclusive	0.052	0.481
Inclusive	0.092	0.467

and less exclusive schemes show the highest exception precision, which may be unintuitive. However, these schemes by their nature will have a lot of exceptions, many of which are the correct answer. Inclusive schemes, however, have very few exceptions. When an inclusive classifier system does have exceptions, it is straying from the theory of a perfect inclusive classifier, indicating that something may be wrong with the predictions being made. Therefore, a low exception precision for inclusive classifiers is not only expected, but also matches the intuition of a perfect inclusive classifier. In any case, the effects of the exception precision do not have a large impact on the performance of inclusive classifiers, since the exceptions happen so infrequently.

One may argue that the results in Table 4.2 are not fair to the exclusive and less exclusive training schemes because the ontology was selected in a way that is biased toward more inclusive schemes. Recall from Chapter 3.1.3 that the ontology was selected by keeping all nodes that had 20 or more proteins mapped at or below them in the ontology. This is biased toward less inclusive and inclusive classifiers, since the criteria for selecting nodes in the ontology is the same as selecting positive training examples for inclusive and less inclusive classifiers. To evaluate each classifier approach in a manner that is more fair to the exclusive methods, we pruned the ontology to the nodes that have 20 or more proteins mapped directly to them. This resulted in an ontology of 137 nodes. The results in Table 4.4 show how well each local predictor scheme performs on these 137 nodes. Similarly, this data shows that the more inclusive schemes are superior.

It is important to note however that exclusive classifiers do predict some functions very well. For example, the exclusive local predictor for the GO term "olypeptide N-acetylgalactosaminyltransferase activity" scored 100% precision and 100% recall during cross validation. This node had 27 proteins assigned to it directly, and 27 through propagation (that is, all known proteins of this function were mapped directly to this node).

As the classifiers become more and more inclusive, recall and F-measure are increased. It is also important to note that the less exclusive scheme has the highest precision in both Table 4.4, and Table 4.2. However, this small increase in precision over inclusive schemes (about 2.5%) comes at a large cost of recall (about 50%). Furthermore, if a molecular biologist was using the prediction system to predict functions of proteins, and therefore find possible experiments that could be performed on these proteins, recall is very valuable. A low recall will result in a prediction system that could miss many important functions of proteins, and could therefore cause the biologist to miss an important discovery. As long as precision is kept reasonably high, it is desirable to give as much of a boost to recall as possible.

The key reason for the improved performance on the inclusive side of the spectrum is that the number of positive training examples is increased, so the predictors become better at recognizing those proteins that should belong at each node. The largest jump in recall happens between less exclusive and less inclusive, so this explanation matches the data.

Table 4.4: Comparison of Training Set Schemes on Smaller Ontology

Each evaluation used the same split of the data for global cross-validation. The best values in each column are marked with bold text. Predictors are only evaluated on nodes which have more than 20 proteins mapped directly to them, which results in an ontology of 137 nodes. The 95% Confidence Interval for the F-measure is also shown

Method	Precision	Recall	F-measure	95% CI (F-measure)
Exclusive	0.744	0.361	0.486	±0.014
Less Exclusive	**0.759**	0.430	0.549	±0.010
Less Inclusive	0.756	0.648	**0.698**	±0.008
Inclusive	0.734	**0.660**	0.695	±0.008

Another factor affecting the better performance of inclusive classifiers is the improvement in quality of the training data. First, as classifiers become more inclusive by no longer using intuitively positive instances in the negative training set (going from exclusive to less exclusive) there is a rise in recall. Second, by excluding ambiguously labeled instances from the negative training set (going from less inclusive to inclusive) the noise in the negative training data is further reduced.

This approach prevents intuitively negative instances from being put into the positive training set, and vice versa. For example, during the construction of a local predictor for "ion binding", the proteins that are labeled as "metal ion binding" should not be used as negative training instances, since they are actually a type of "ion binding". Furthermore, the proteins labeled as "metal ion binding" proteins can be used in the positive training set of the "ion binding" protein, since according to the true path rule, they are specific examples of this function.

As the classifiers become more inclusive, there is a higher chance that a false negative at a node will be offset by a true positive prediction at a descendant node, which is desired. In a sense, inclusive local predictors reinforce each other along the path in a hierarchy, whereas in a system of exclusive local predictors, one predictor must make the correct call for each assigned label.

Although the training data for each node is selected differently, the test sets cannot change. The classifiers cannot choose to be evaluated only on some sequences, since this would be contradictory to what we are attempting to infer from cross-validation. By keeping the test sets constant, predictions based on various design strategies are comparable, since they are evaluated on the same test proteins.

Support Vector Machines are used in these experiments, but the issue of training set construction must be addressed regardless of which machine learning technology is used. Therefore the methods presented are applicable to all local predictors, regardless of prediction technique specifics.

Table 4.5: Lowering the cost of using Local Inclusive Predictors.

When using an inclusive training set design, the cost of using local predictors can be significantly lowered using a top-down approach, without a significant impact on precision and recall. All results shown in this table use the inclusive training set design scheme. The best values in each column are marked with bold text.

Method	Precision	Recall	F-measure	Average Cost per Instance
PA-SVM	0.753	**0.652**	**0.699**	406
Top-Down PA-SVM	**0.760**	0.644	0.697	**32**
TD-1 PA-SVM	0.755	0.649	0.698	112
TD-2 PA-SVM	0.754	0.651	0.698	220

Table 4.6: Lowering the cost of using Local Less Inclusive Predictors.

When using a less inclusive training set design, the cost of using local predictors can be significantly lowered using a top-down approach, without a significant impact on precision and recall. All results shown in this table use the less inclusive training set design scheme. The best values in each column are marked with bold text.

Method	Precision	Recall	F-measure	Average Cost per Instance
PA-SVM	0.773	**0.638**	0.699	406
Top-Down PA-SVM	**0.782**	0.630	0.698	**32**
TD-1 PA-SVM	0.776	0.635	0.698	112
TD-2 PA-SVM	0.774	0.637	0.699	220

4.1.3 Top-Down Search Revisited

The results in Chapter 3.3 showed that using an exclusive training set design strategy does not support the use of a top-down decision model to lower the cost of prediction. This matches the intuition for an exclusive classifier, since only the nodes containing the true annotation of proteins will predict positive, and all parents will predict negative.

The inclusive training set design intuitively matches the top-down decision model, as shown in Figure 4.3. The results of training a top-down decision model on our inclusive and less inclusive training set design schemes are shown in the first two rows of Table 4.5 and Table 4.6.

The results match the intuition behind training set designs that are on the inclusive side of the spectrum. Both inclusive and less inclusive classifiers are more amenable to the top-down decision model. In both cases the simple Top-Down approach has very similar recall to the recall obtained when running all node predictors, and precision is in fact raised. The cost of this approach is only 8% of running all predictors for the simple top-down approach, which is a significant savings. This result shows that the more complex and computationally intensive methods of TD-N PA-SVM classifier are not necessary when using more inclusive classifier schemes.

The results in this section have shown that inclusive and less inclusive training schemes are superior to exclusive and less exclusive schemes in two ways:

1. The inclusive and less inclusive classifiers have a significantly higher F-measure during cross-validation than the exclusive and less exclusive schemes.
2. The inclusive and less inclusive classifiers' cost can be lowered using a top-down search scheme to selectively compute only some of the local predictors without a significant penalty to F-measure. The exclusive and less exclusive classifiers produced significantly lower F-measures when a top-down approach was used.

Although the inclusive scheme intuitively seems more sound than the less inclusive scheme, since ambiguous proteins are not used in its negative training set, the difference between inclusive and less inclusive has thus far been only justified by intuition. The next section will attempt to quantify this difference.

4.1.4 Robustness to Incomplete Annotations

The cross-validation experiment showed that excluding ancestors from the negative training set (the inclusive strategy) only has a small advantage over the less inclusive strategy. However, the actual advantages of an inclusive design may be greater than shown by this experiment. The nature of cross-validation tests, and the fact that an absence of a label in the GO hierarchy does not necessarily mean a label is wrong, may lead to lower quantitative results for what is, arguably, the correct design decision.

Specifically, proteins may not be annotated with all the labels that are appropriate. As discussed earlier, a missing experiment results in an incomplete label. A desirable goal is to have predictors that can predict labels more specific that are not currently known. This systemic side-effect of taking the annotation as complete truth (even when it is not complete) is a difficult issue to measure and address.

The results of cross-validation and using a top-down search are still inconclusive as to which training scheme is better: inclusive or less inclusive. As discussed in Chapter 4.1.2 the inclusive training scheme is intuitively superior due to the fact that the Gene Ontology annotation data is incomplete. That is, including proteins annotated with an ancestor node as negative training examples is dangerous, because a future experiment could show that they are in fact positive training examples.

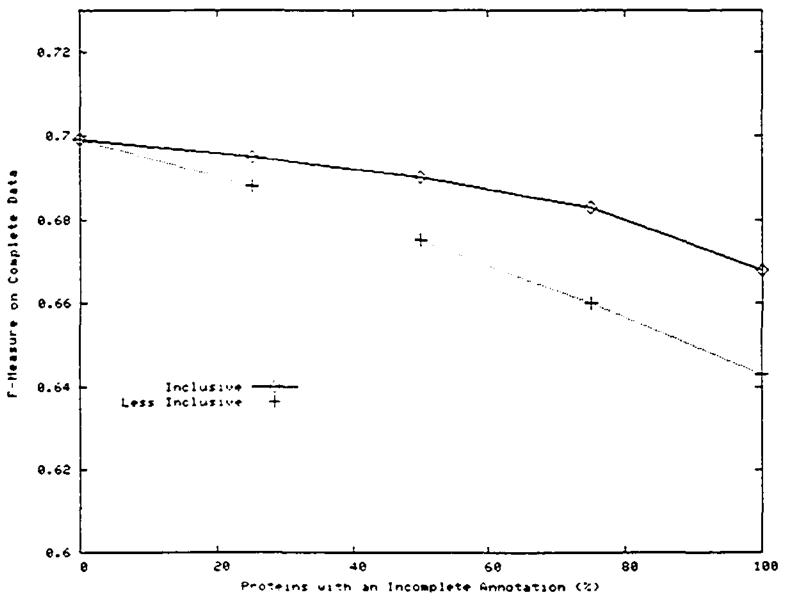


Figure 4.4: Inclusive classifiers are more Robust to Incomplete Data

Inclusive and Less Inclusive training schemes are trained for various degrees of incomplete training data. Both are then evaluated on the most complete data available. Inclusive classifiers are more robust to incomplete annotations. Note that the y-axis does not start at 0.

To test this theory the data was modified so that the annotations are less complete. This modified data was then used to train local predictors, but when evaluating the predictions during cross-validation, the original unmodified annotations were used to score the system's predictions. This experiment simulates incomplete annotations.

For example, if a protein was annotated with node N in Figure 4.2, we would make the annotation more incomplete by moving the annotation to node A_1. This is consistent with the previous labeling, but it is incomplete in that the knowledge about the protein is now less specific.

In Figure 4.4, we make the data more incomplete by randomly choosing $X\%$ of the proteins in the data set, and moving one of their annotations up a level in the hierarchy. X is varied from 0 to 100%. The predictions are then evaluated using the complete data, and results are compared using F-measure.

When the data was not altered (at 0% incomplete) the inclusive and less inclusive schemes produce the same F-measure. However, as the training set became more incomplete, the inclusive strategy maintains its high F-measure better than the less-inclusive strategy. This is because we removed the ambiguous proteins from the negative training set of the local predictors.

Although the graph shows the two training schemes to be equally good at 0% incomplete, even the full data set is incomplete. That is, the complete annotations for all proteins is unknown. If the data were complete, each protein would be mapped to a leaf node in the ontology. Therefore even though the two lines appear to be together at the 0% incomplete point, if the complete data was known, the inclusive training scheme would likely have a higher F-measure than the less inclusive training scheme.

These results show that the inclusive scheme shows more robustness to incomplete data, and thus is better for predicting the functions of unknown proteins. Therefore, for the remainder of this dissertation, all local predictors are trained using an inclusive strategy.

4.2 Global Cross-Validation Revisited

In Chapter 3.2.3, the method of global cross-validation was explained. For all experiments described so far, the data set was split once, and then evaluated. This split however did not take into account the individual node distributions when splitting the data set into 5 partitions. Since the goal of cross-validation is to understand how well the prediction system will perform on future instances, we must ensure that each classifier has enough positive training instances to be representative of the final system. Since some nodes have as little as 20 positive instances, it is possible that some have very few training instances in some of the 5 folds of cross-validation.

In practice, it may be difficult (or impossible) to perfectly preserve local node distributions[2] and have a consistent, global split of the data. To address this issue, 5,000 candidate global splits of the data set were randomly generated, and the split that matched the original data set's node distribution most closely was chosen. This was measured by the average squared deviation of each node in each of the 5 folds.

For example, if a node had 100 positive training instances and 10,000 negative training instances in the original data set then its original distribution is 1% positive. If, after splitting into 5 partitions, the node has 13 positive training instances and 1,800 negative training instances in fold 1, then the fold 1 distribution is 0.72%. Therefore this node's distribution has a change of -28% ((0.72 − 1.0)/1.0). The squared deviation for each node in each fold was then averaged.

The best and worst splits (according to average squared deviation) are shown in Figure 4.5 and Figure 4.6. Comparing the two figures, it is evident that there is less variation from the original distributions in Figure 4.5. This suggests that a global partition of the data is feasible, and that a sufficient amount of training data for all nodes can be retained using a simple randomized approach.

To examine the effect of how the data is split on local predictors, the best and worst splits of the data are used to train PA-SVM classifiers using the inclusive training scheme (Table 4.7). The results show that there is not a significant impact on the overall cross-validation accuracy from the choice of the global partition.

[2]Only the distributions for the training sets need to be preserved. If the distributions for the testing sets vary this is not a concern since they are not used to train the classifier.

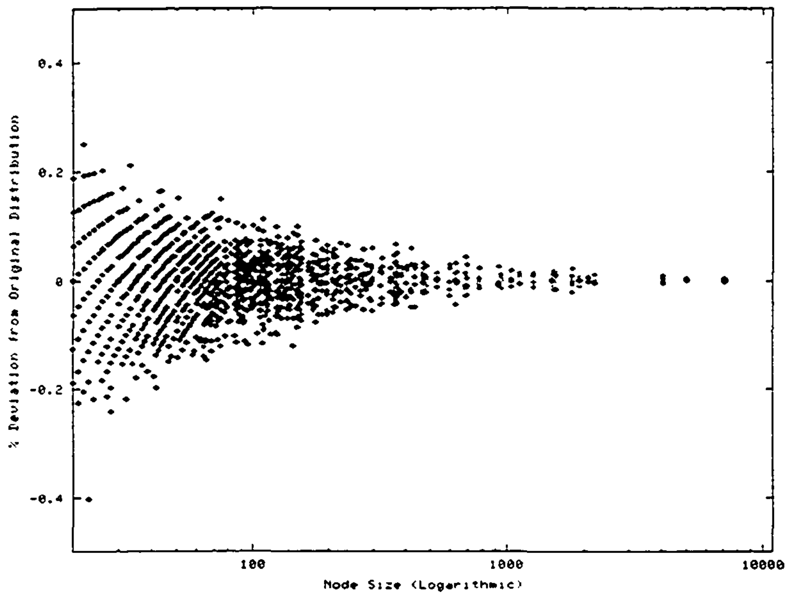


Figure 4.5: The Best Global Split of the Data Set

Each dot represents a single node in a fold's training set. The y-axis shows how far this node in this fold's training set deviates from the original node's distribution. The 5 largest changes are -40%, 25%, -24%, -22%, and -22%.

The goal of finding an optimal split is not to maximize precision and recall during cross-validation. It is rather to have results that are indicative of how well the final classification system will perform. Therefore, for the remaining experiments the best split of the data was used for cross-validation.

4.3 Summary

We extended the work in Chapter 3 by considering the semantics of the hierarchy when creating training sets for local predictors. We presented a spectrum of training set design schemes and evaluate each using global cross-validation.

When more inclusive training set designs were used, it was possible to lower the computational cost of prediction using a top-down approach, without a significant penalty to precision and recall.

The differences in cross-validation between inclusive and less inclusive designs was minor. However, the hypothesis is that inclusive classifiers will perform better on new proteins due to incomplete annotations in the data set. An experiment simulating incomplete data has confirmed this.

Finally the issue of global cross-validation was readdressed. The presented method of picking the best global split of the data did not result in a significant change in cross-validation precision and recall. However, we will continue to use

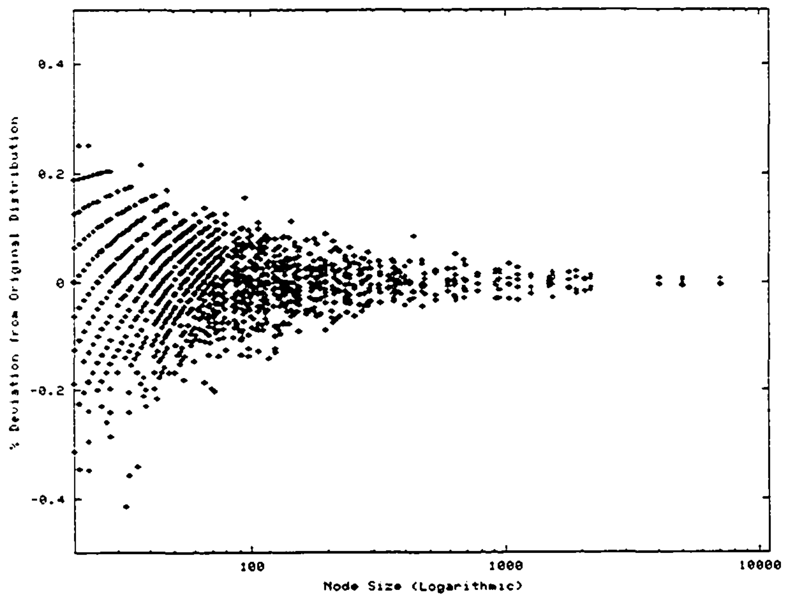


Figure 4.6: The Worst Global Split of the Data Set

Each dot represents a single node in a fold's training set. The y-axis shows how far this node in this fold's training set deviates from the original node's distribution. The 5 largest changes are -41%, -35%, -35%, -35%, and -34%.

Table 4.7: Comparing the worst and best global splits of the data for training inclusive classifiers using PA-SVM

Picking a global split of the data according to the average deviation of each node's distribution in each of the folds training sets does not appear to have a large impact on cross-validation performance of local predictors.

Method	Precision	Recall	F-measure	95% CI (F-measure)	Average Deviation
Best Split	0.754	0.648	0.697	±0.010	0.048
Worst Split	0.752	0.655	0.700	±0.005	0.058

the best global split of the data for all remaining cross-validation experiments.

Chapter 5

Optimizing Predictors of Protein Function

Now that a general training strategy has been derived for local predictors, each local predictor technology will be optimized individually. Due to the pervasive use of BLAST in the community, it will be used as a comparison method for predicting protein function. BLAST will also be optimized so that the comparison is fair between methods.

5.1 PFAM-SVM

PFAM (Protein FAMilies) uses Hidden Markov Models (HMMs) to model functional domains of proteins. Each of these protein families describes a functional class of proteins. When creating these classes, proteins with similar functions are collected, and then HMMs are trained on the part of the sequence that is conserved between the proteins in the set. The result is an HMM that can be used to predict whether a query protein belongs to this particular class of proteins.

These PFAM HMMs are run on all proteins in the data set, and when there is a match between a PFAM family and a query protein, that family is used as a feature for the protein. The confidence of each prediction is measured by *E-value*, which is the expected number of proteins that would have matched the PFAM family by random chance (which is intuitively the same as the *E-value* for BLAST). The lower the *E-value* of a hit, the more similar it is to the query protein. The first row in Table 5.1 shows the performance of using all PFAM matches as features for an SVM classifier.

If the use of PFAM matches as features is more stringent (i.e. a lower *E-value* threshold), the performance of the classifier can be increased. Table 5.1 shows that if we only accept matches with *E-value* $\leq 10^{-2}$, F-measure is maximized. As the *E-value* cutoff becomes more stringent (i.e. lower) the number of families that are used is decreased. This is because the number of PFAM families used as features is defined as the number of families that are assigned to at least one protein in the

Table 5.1: Optimizing PFAM Predictors

Only those PFAM matches below a certain *E-value* threshold are used as features. The best values in each column are marked with bold font.

E-value Threshold	Precision	Recall	F-measure	95% CI (F-measure)	Number of PFAM Families
Any	0.703	0.395	0.506	±0.012	7,483
2	0.717	0.513	0.598	±0.006	5,267
10^{-1}	0.736	**0.577**	**0.647**	±0.009	2,640
10^{-2}	**0.740**	0.575	**0.647**	±0.009	2,438
10^{-3}	0.737	0.571	0.643	±0.008	2,388
10^{-7}	0.737	0.563	0.638	±0.009	2,329

data set. When the *E-value* cutoff is low, there is less chance of each PFAM family matching any protein.

Since the F-measure was maximized at a cutoff of *E-value* $\leq 10^{-2}$ with the lowest number of PFAM features used, these PFAM predictors will be used for all experiments for PFAM-SVM for the remainder of this dissertation.

5.2 PA-SVM

When presented with a query protein, Proteome Analyst (PA) uses BLAST to find similar proteins in the Swiss-Prot database. Next, the PA system looks at the existing annotations for these similar proteins, and these annotations are used as features for a classifier that predicts the function of the query protein. Currently, the PA system parses words from the *Keywords*, *SUBCELLULAR LOCALIZATION*, and *InterPro* fields of Swiss-Prot entries, which are then used as features for a classifier. The Interpro and Keywords fields are parsed directly, and used as features, since they use standard vocabularies. The SUBCELLULAR LOCALIZATION field is parsed using a controlled vocabulary, because it is free-form. Parsing this field directly would produce inaccurate results since there would be a lot of unimportant features in free form text [34]. The results of using PA features for an SVM classifier at each node are shown in the first row of Table 5.2.

There are many other fields in Swiss-Prot entries that PA currently does not utilize. Although previous experiments [33] have shown that the three fields PA currently uses produce the best results for the prediction of subcellular localization, there may be potential to using other fields when predicting protein function across a large ontology.

One approach is to use Gene Ontology names as a vocabulary for parsing phrases from other fields in each Swiss-Prot entry. Using the names of the 406 nodes in our

Table 5.2: Parsing other Swiss-Prot fields for PA Classifiers

Using traditional PA Features is compared to using PA features in conjunction with parsing other fields with a controlled vocabulary (PA + Valid GO), and PA features in conjunction with parsing the GO field (PA + GO Field).

Method	Precision	Recall	F-measure	95% CI (F-measure)
PA Features	0.754	0.648	0.697	±0.010
PA + Valid GO	0.766	0.591	0.667	±0.010
PA + GO Field	0.759	0.626	0.686	±0.012

pruned ontology as a vocabulary, the "SIMILARITY", "FUNCTION", and "SUBCELLULAR LOCATION" fields of Swiss-Prot entries were parsed. If any of the terms in the vocabulary were found in any of those fields, that term was used as a feature for the classifier, in addition to traditional PA features. This approach is labeled "PA + Valid GO" in Table 5.2. This approach did not show an improvement over the standard PA features.

Another approach is to look directly into the "GO" field of each Swiss-Prot entry. This is called "PA + GO Field" in Table 5.2. Surprisingly this lowers recall, and slightly raises precision.

None of the approaches to improving PA predictors showed improvements in precision *and* recall. Although there was some improvement in precision, it was not significant enough to make changes to the PA-SVM predictor. The sheer number of combinatorial possibilities for parsing Swiss-Prot entries made pursuing this topic further beyond the scope of this dissertation. Therefore, the PA method of choice is the classic PA feature parsing algorithm.

5.3 Probabilistic Suffix Trees

Rather than using features to describe proteins (as the PA-SVM and PFAM-SVM predictors did), Probabilistic Suffix Trees (PSTs) model the protein sequence directly. PSTs are an efficient implementation of variable length Markov models (VMM). As was described in Chapter 2.5.3, PSTs model the sequence by assuming that the probability of each amino acid is conditioned on the previous N amino acids. Then, the probability of the protein is the product of the probabilities of each of the amino acids. A VMM that uses the above model is called an N-order Markov model.

Several parameters can be tuned for PSTs. These are:

1. **Smoothing Term** - During the calculation of the probability of a sequence, a

Table 5.3: PST Parameter Search Space

The possible parameter settings for PSTs when performing the brute force search of the parameter space.

Parameter	Possible Values
Smoothing Term	10, 1, 10^{-2}, 10^{-6}
Window Length	10, 20, 40, 100, 200, 400, *Global*
Order	2, 3, 4, 5, 6

sequence may be encountered that was never seen in the training data. During the calculation of the probability of this protein, the term for this unseen sequence would be 0, which would make the probability of the sequence 0 (Equation 2.7). To safeguard against this, a process called smoothing [25] is often used. Instead of assigning previously unseen examples a probability of 0, they are given a minimum value, which is the smoothing term.

2. **Window Length** - The window length is the number of amino acids in the protein sequence that are considered when calculating the probability of the protein. Sometimes the entire sequence is not relevant, because some biological functions are served by small regions of the sequence called *functional domains*. This setting can also be set to "Global" to consider the entire sequence of the protein when calculating its probability.

3. **Order** - An N-order Markov model represents the probability of an amino acid conditioned on the previous N amino acids. Keeping the order small prevents overfitting, but keeping it large allows for more representative power.

For baseline performance, a PST was trained for each node using an inclusive training strategy, and using the parameters *Smoothing Term = 0.01*, *Order = 5*, and *Window Length = Global*. The results are shown in the first row of Table 5.4. To optimize these predictors, a brute force search through the parameter space was performed. A finite set of parameter values was chosen which have shown relevance in previous function prediction experiments [40]. The possible values for each parameter are shown in Table 5.3.

For each node in the GO hierarchy, a local predictor was trained on one setting of the parameters. Cross-validation was performed (using the global folds), and the performance was recorded. The process was repeated for each setting of the parameters. The parameter settings that result in the highest cross-validation F-measure were kept for each node. At the end of the process, each local predictor was trained using the parameters found to be the best for that particular node. The performance of these optimized local predictors is shown in the second row of Table

Table 5.4: Results of PST Parameter Search

After doing a brute force search through the predefined parameter space (Table 5.3). PSTs show an increase in precision and recall. Results shown are overall statistics across all local predictors.

Method	Precision	Recall	F-measure	95% CI (F-measure)
Baseline PST	0.549	0.588	0.568	±0.008
PST Parameter Search	0.575	0.636	0.604	±0.010

5.4. Histograms showing how often each parameter setting was used in optimal settings are shown in Tables B.3, B.2, and B.1 in Appendix B.

The average improvement of each node's F-measure was +5% from the baseline parameter settings. None of the parameters showed a significant correlation with each other, the largest correlation coefficient being -0.32 between *Order* and *Smoothing Term*. All other parameter combinations had a smaller correlation coefficient. If some parameters did have a high correlation the parameter space could be reduced for the future training of PST predictors.

The improvements for each of the nodes are shown in Figure 5.1. The graph shows that the predictors that benefit from the parameter search the most are those that lie near the middle of the F-measure score. Those nodes that perform very well in the baseline are difficult to improve further, and those that perform very poorly in the baseline are often difficult to represent and may not be amenable to PSTs no matter what the parameter settings are.

5.4 BLAST

During cross validation, BLAST was run for each of the test proteins against the current fold's training set of proteins. A *BLAST hit* was considered a match for this test protein against the current training set. Each BLAST hit was scored with an *E-value*, which (as for PFAM) is the expected number of proteins that would have matched the query protein in the database by random chance. So, as the *E-value* increases, the less similar the match is to the query protein. If an *E-value* cutoff was set to 10^{-1} no BLAST hits with *E-value* $> 10^{-1}$ were accepted.

The BLAST predictor performs best when the threshold for accepting BLAST hits is set at an *E-value* of 10^{-3} (Table 5.5). This is because proteins that have no BLAST hit with *E-value* $\leq 10^{-3}$ are proteins that are quite different from the set of well-studied proteins. These proteins will not find a highly similar sequence during cross-validation. Figure B.1 in Appendix B shows similar results in graphical form.

In this approach, BLAST has been used as a Nearest Neighbor method where

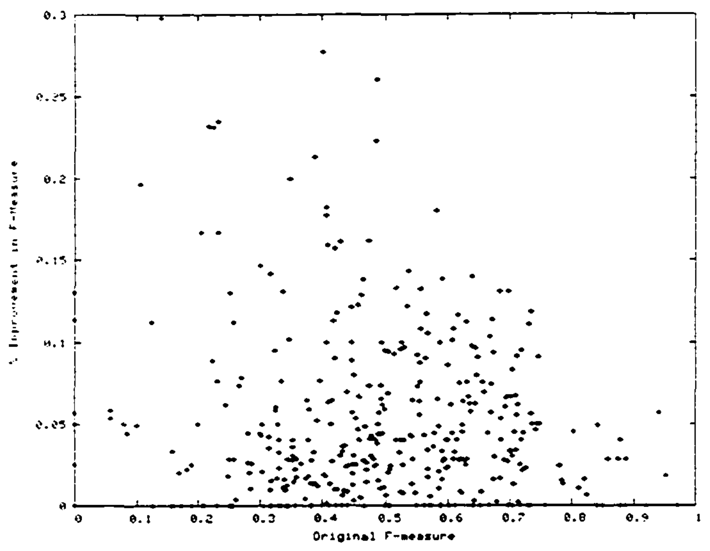


Figure 5.1: Improvements for each Local PST Predictor

Each point on the graph shows a single node. The x-axis shows the cross-validation performance of this node using the baseline parameter settings. The y-axis shows the improvement in F-measure after using a brute-force search of the parameter space.

Table 5.5: Varying E-value for BLAST

An *E-value* cutoff of 10^{-3} results in the best BLAST predictor in terms of F-measure. "No predictions" is the number of proteins that had no valid BLAST hits found at this *E-value* threshold. The best values in each column are marked with bold font.

E-value	Precision	Recall	F-measure	95% CI (F-measure)	No Predictions
10	0.708	**0.716**	0.712	±0.009	**126**
1	0.729	0.712	0.720	±0.009	613
10^{-1}	0.752	0.705	0.727	±0.008	1,181
10^{-3}	0.767	0.696	**0.730**	±0.007	1,637
10^{-5}	0.774	0.688	0.729	±0.008	1,927
10^{-9}	0.782	0.674	0.724	±0.009	2,344
0	**0.831**	0.300	0.441	±0.017	9,550

Table 5.6: Using more than one BLAST Hit

Performing the union on the annotations of more than one BLAST hit increases recall but decreases precision. Intersecting the results of more than one BLAST hit decreases recall but increases precision. All results use an *E-value* cutoff of 10^{-3}. The best values in each column are maked with bold font.

Method	Precision	Recall
BLAST-NN	0.767	0.696
BLAST-2-Union	0.672	0.751
BLAST-3-Union	0.608	0.775
BLAST-5-Union	0.531	0.796
BLAST-7-Union	0.482	0.806
BLAST-All-Union	0.252	**0.828**
BLAST-2-Intersect	0.849	0.597
BLAST-3-Intersect	0.870	0.526
BLAST-5-Intersect	0.884	0.437
BLAST-7-Intersect	**0.887**	0.385
BLAST-50-Intersect	0.862	0.238

only the top BLAST hit's functions are assigned to the query protein. Using just the single most similar protein ignores other matches that could also be of good quality in terms of *E-value*. Table 5.6 summarizes two ways of incorporating other BLAST hits' into predictions.

The BLAST-N-Union approach takes the top N BLAST hits, and performs the set union operator on the labels associated with these hits. The BLAST-N-Intersect approach takes the top N BLAST hits, and performs the set intersect operator on the labels associated with these matches. That is, only labels that appear in all N hits label sets are predicted for the query protein. The BLAST-N-Union method will in general increase recall, but may suffer a penalty to precision, and the BLAST-N-Intersect method will generally increase precision, but may suffer a penalty to recall.

These methods can be used if a user is more concerned with recall (BLAST-N-Union) or precision (BLAST-N-Intersect). BLAST NN will be used as the baseline predictor. However, BLAST-N-Union will prove to be useful in Chapter 6 to lower the computational cost of using local predictors.

5.5 Combining Predictors

This chapter has presented 4 different predictors for protein function: PA-SVM, PFAM-SVM, PST, and BLAST, where all of these methods work in fundamentally

Table 5.7: Intersections of Good Performing Node Sets

The number in square brackets under each heading is the total number of nodes on which the local predictor performed well (*F-measure* $\geq 60\%$) on. Each entry in the table is the intersection of the nodes that the two predictors did well on. For example 177 nodes performed well (*F-measure* $\geq 60\%$) on both the PA-SVM and PFAM-SVM predictors.

	PFAM-SVM [184]	PA-SVM [256]	PST [162]	BLAST [283]
PFAM-SVM	-	177	111	182
PA-SVM	-	-	150	245
PST	-	-	-	159

different ways. The next section of this Chapter will explore combining these predictors at each node to improve the quality of local predictors.

5.5.1 Characterizing Predictors

When combining local predictors, the combined prediction system works best when each predictor works well on a different subset of the data. This way, when the predictors are combined, the weaknesses of one predictor are offset by the strengths of another.

To see whether this is the case, we determined the set of nodes for which each local predictor approach worked well (the nodes that have $\geq 60\%$ *F-measure*) – see Table Table 5.7. The cutoff of 60% is an *ad hoc* choice, for which the local predictors performed reasonably well. The intersection of each of these sets was then computed. Ideally these intersections should be as small as possible if the predictors are to have different strengths.

For example, BLAST had over 60% F-measure on 283 nodes. PFAM-SVM performed above the 60% F-measure mark on 182 of these 283 nodes. Out of the 184 nodes that PFAM-SVM performed well on, 182 of them BLAST also performed well on. One interesting note, not shown in the table, is that the only node that all four approaches had 0% precision and 0% recall on the node "protein C-terminus binding".

This result shows that there is a potential to combine the classifiers into an *ensemble*. An ensemble is a way of combining classifiers using a weighted voting, or some other learning function.

Table 5.8: Combining Predictors

Voting proves to be an accurate method of combining prediction technologies. Although using SVMs in this way to learn weights is invalid, it does show the potential of using a weighted linear function of the classifiers.

Method	Precision	Recall	F-measure	95% CI (F-measure)
Voting - 1	0.535	**0.819**	0.647	±0.005
Voting - 2	0.763	0.733	**0.748**	±0.008
Voting - 3	0.846	0.609	0.708	±0.009
Voting - 4	**0.907**	0.387	0.542	±0.010
Linear SVM	0.776	0.708	0.741	±0.005

5.5.2 Ensemble Methods

An ensemble is created at each node using a simple voting scheme. This ensemble uses each local predictor discussed so far as input:

1. **SVM with PFAM as features**
2. **SVM with PA features**
3. **PSTs**
4. **BLAST**

For a Voting-N classifier, N or more positive votes must be given before the function is predicted as positive for that classifier. For example, if at a node PFAM-SVM and PA-SVM predict positive, and we are using a Voting-2 scheme, the function is predicted as positive by the voting system. The results of varying N are shown in Table 5.8.

Although using a simple voting technique for the predictors is not very sophisticated, it works quite well in practice. As a comparison, an SVM was used to learn the weights for each predictor, given all of the prediction data. This is the best that can be hoped for since the SVM is given the correct answer for training on all of the data. Table 5.8 shows that there is not much potential in learning more complicated linear weighting functions for the predictors, since the SVM did not perform better than voting. Therefore, weighting functions were not pursued further.

5.6 Summary

Each prediction tool has been optimized individually. Local predictors were trained using an inclusive strategy, and optimized based on their individual approaches.

PFAM-SVM was optimized by only accepting good PFAM matches. For PA-SVM, different feature parsing methods were evaluated, but the traditional PA feature extraction algorithm proved to be the best out of the methods that were attempted. For PSTs, a brute force search through the parameter space was performed. BLAST was optimized using an *E-value* cutoff to avoid using poor BLAST hits for prediction.

These methods were then combined using a voting scheme that, although simple, proved to have good results. These voting classifiers are used as predictors in the next Chapter.

Chapter 6

Experiments for Hierarchical Classification of Protein Function

As shown in Chapter 5, the results of a BLAST search are quite accurate, when highly similar sequences are found. Therefore, the information gained from running a BLAST search is important since it is both accurate, and computationally efficient.

As was also demonstrated in Chapter 5, recall can be increased from using BLAST alone when using a voting ensemble of PFAM, Proteome Analyst, Probabilistic Suffix Trees, and BLAST.

As Table 6.1 shows, there are a significant number of proteins where a good BLAST hit is not found during cross-validation. This also occurs often during real use of BLAST because a protein being studied may be far (in terms of sequence similarity and thus homology) from the set of well studied proteins. Thus, these two cases – when a protein has high sequence similarity to a protein in the training set, and when it does not – will be examined separately.

In the case when a good BLAST hit is found for proteins, the BLAST result can be used to lower the cost of running local predictors. The case when no good BLAST hit is found is the more challenging, and arguably the more important scenario, because these are often proteins that are very different from the set of experimentally annotated proteins. In this case, CHUGO performs much better than BLAST NN. Finally, the local predictors presented in Chapter 5 are compared to BLAST in terms of coverage.

6.1 Proteins with Good BLAST Hits

During cross-validation, 89% (12,725 out of 14,362) of the proteins in the data set had at least one good BLAST hit. The results of using the voting scheme, presented in Chapter 5, are shown again in Table 6.2, and are compared to each individual predictor method for these proteins. To compare each method in t erms of computational cost, each node predictor is assigned a cost of 1 for each time it is run, and BLAST is assigned a constant cost of 1 since it is a global predictor and thus only

Table 6.1: Histogram of How Often X Hits are found by BLAST

Histogram of how often X number of hits are found per protein by BLAST during cross-validation on the set of experimentally annotated proteins. The results use BLAST set at an *E-value* cutoff of 10^{-3}.

Number of Hits	# of Proteins
0	1,637
1	1,259
2	944
3	784
4	607
≥ 5	9,131

run once per protein. Thus, the cost of 1219 for voting over the ontology is obtained by the formula $(NumberOfNodes) \times (NumberOfPredictorsAtEachNode) + (CostOfBLAST)$. Since three local predictors are used at each node (PA-SVM, PFAM-SVM, and PSTs) this value is $406 \times 3 + 1 = 1219$. The cost of calculating the result of voting is not added since this is a trivial computation. Although the costs of the various predictors in the ensemble would in fact vary, this measure of cost gives an initial idea of how computationally intensive each approach is.

When applying a top-down approach, similar to that in Chapter 3.3, the cost of prediction is lowered significantly. This is because each classifier in the voting ensemble is trained using an inclusive training strategy. However, the results of a BLAST search can be exploited to lower the cost of running local predictors even below the cost of a top-down approach. Two methods of exploiting BLAST to lower the cost of local predictors are presented. Both methods rely on using BLAST results to find GO nodes that are likely to be annotated to a protein, and then running local predictors only for these *candidate nodes.* Since the local predictors are not being run for every node in the ontology, their overall cost is lowered.

A good method for generating candidate nodes would have two properties:

1. **High Recall** - A high recall for the candidate node set will ensure that the final prediction recall will be high. After running predictors on the candidate node set, the recall cannot increase, therefore the candidate set must have as high of a recall as possible.

2. **Minimal Size of Candidate Node Set** - The size of the candidate node set should be as small as possible. The smaller the size of the candidate node set, the smaller the computational cost of verifying these nodes using local predictors. As the size of the candidate node set approaches the size of the entire ontology, the benefit of using candidate nodes is diminished.

Table 6.2: Performance on proteins with Good BLAST results

Performance of various prediction approaches on proteins that produce a good BLAST hit during cross-validation. Here, voting refers to the voting ensemble of BLAST NN, PA-SVM, PFAM-SVM, and PST described in Chapter 5.5.

Method	Overall Precision	Overall Recall	Average Cost Per Protein
BLAST NN	77%	78%	1
PA-SVM	76%	69%	406
PFAM-SVM	75%	62%	406
PST	61%	64%	406
Voting	77%	80%	1219
Voting Top-Down	77%	79%	111

These two factors are competing forces. One easy method of raising recall is to add all nodes to the candidate set, but the second criteria keeps this from happening. The presented candidate generating methods meet the two above criteria, and are therefore useful for lowering the cost of using local predictors.

6.1.1 BLAST-N-Union

In Chapter 5.4 it was shown that using more than one BLAST hit, and combining the annotations of these hits using a union operation can significantly raise recall when this set of annotations is used as a prediction. However, precision suffers as a result. Since the recall is high, this set of nodes from the union operation can be used as candidate nodes for local predictors.

This first candidate generating method, called BLAST-N-Union, uses multiple BLAST hits' annotations, where N denotes the number of hits used. Intuitively, if the union of more than one BLAST hit's annotations is used, more of the likely functions will be covered by this candidate set. Performing the union of multiple BLAST hits' annotations increase recall, but decrease precision. However, since this set is smaller than the entire ontology, and the recall is high, we can run our predictors on this set of labels and improve precision while keeping computational cost low.

6.1.2 BLAST-Search-N

As was demonstrated, BLAST NN does not always return correct predictions for protein function, since precision and recall are not 100% (Chapter 5.4). Figure 6.1 demonstrates that when BLAST NN is incorrect, it tends to be *close* to the correct

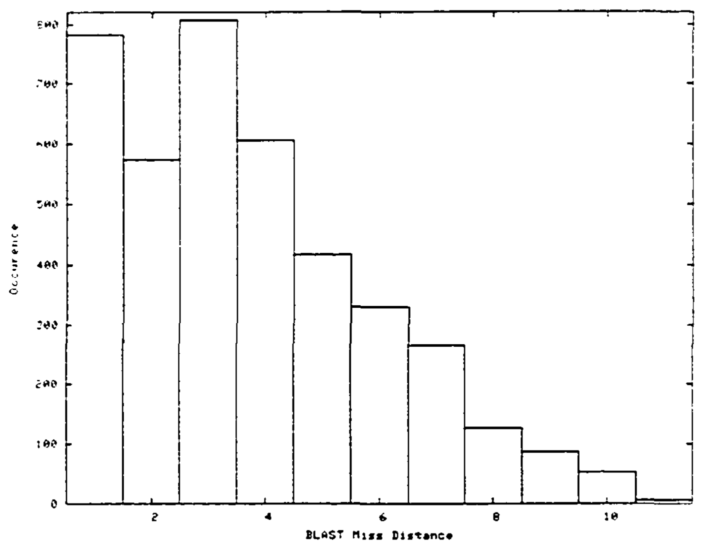


Figure 6.1: BLAST NN Miss Distance

When BLAST NN does not return the correct nodes from good hits, the nodes that are returned tend to be close to the correct answer.

answer. The BLAST Miss Distance in the figure is the graph distance between the predicted label for a protein, and the nearest annotated label for that protein. Graph distance is the shortest path between these two nodes in the ontology.

This fact can be exploited to find candidate nodes for the computation of local predictors. The results of a BLAST search can be used as a *seed* to begin searching the ontology outward for candidate nodes. Since the structure of the ontology is known during prediction, the nodes that are nearby the nodes found by BLAST NN can be used as candidate nodes.

This second candidate generating method, called BLAST-Search-N, exploits the annotations returned by BLAST and searches in the neighborhood of the top BLAST hit's annotations (Figure 6.2). Here, N is the graph distance from the seed annotations in which we add nodes to our set of candidate nodes. The set of GO terms that BLAST NN returns is added to the candidate set, and all of the terms in the N-neighborhood are added to the candidate node set as well. As in the first method, recall will be increased by searching in the neighborhood, and the node predictors are used to compensate for the drop in precision by removing false positives. Also, similar to the first proposed method, computational runtime is lowered from running the validating predictors for all nodes.

6.1.3 Evaluation of Candidate Generating Methods

The advantage of the BLAST-Search-N and BLAST-N-Union approaches is that they decrease the computational cost by not running all of the local node predic-

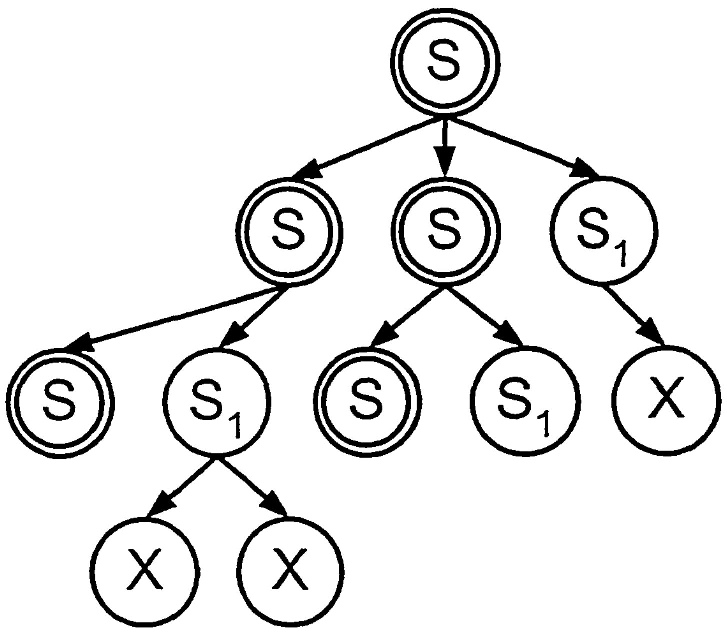


Figure 6.2: Using BLAST-Search-N to Generate Candidate Nodes

Those nodes which are returned from BLAST NN (marked S for seed) are added to the set of candidate nodes. Those nodes within a graph distance of N (in this case $N = 1$) from these seed annotations are also added to the set of candidate nodes (marked S_1). Those nodes that are beyond this distance do not have their local predictors computed (marked X).

Table 6.3: Exploiting BLAST Results

Comparing methods of using BLAST to find candidate nodes, and then validating these nodes using local predictor ensembles. The best values in each category are marked with bold text.

Candidate Generating Method	Precision	Recall	Average Cost per Protein
BLAST-1-Union	**81%**	75%	**16**
BLAST-2-Union	79%	78%	20
BLAST-3-Union	78%	79%	22
BLAST-10-Union	77%	**80%**	32
BLAST-Search-1	**80%**	77%	**82**
BLAST-Search-2	78%	78%	221
BLAST-Search-3	78%	**79%**	430

tors. This is done by only evaluating local predictors for the set of candidate nodes. The results of using BLAST-Search-N and BLAST-N-Union as candidate node generating methods for various values of N is presented in Table 6.3. An interesting side effect is that constraining our candidate nodes for our predictors can also raise precision since those GO terms that are unlikely to be assigned to an instance are never considered. An important note is that regardless of which method is computationally cheaper for finding the set of candidate GO term predictors to run, the BLAST-Search-N method must be used when only a single good BLAST hit is found (since there are no other good hits to union).

Although the BLAST-2-Union and BLAST-Search-3 methods produce similar precision and recall, the BLAST-Search-3 method is more costly. Therefore, using BLAST-2-Union whenever possible (when there are at least 2 good BLAST hits) would be preferred, and BLAST-Search-3 should be used only when a single BLAST hit is found.

The results of combining BLAST-2-Union and BLAST-Search-3 method are shown in Table 6.4. This approach results in a cost that is approximately half of using a top-down approach. An added advantage of this approach is that precision is raised since the candidate node set is smaller. The combination of BLAST-2-Union and BLAST-Search-3 is used by CHUGO whenever at least 1 good BLAST hit is found against experimental data.

6.2 Proteins with No Good BLAST Hits

During cross-validation, 11% (1,637 out of 14,362) of the proteins in the data set had no good BLAST hits. These are sequences that are disparate from those which

Table 6.4: Comparing Methods for Lowering Prediction Cost

Using BLAST-2-Union when possible, and otherwise using the BLAST-Search-3 method results in a lower cost than the top-down approach. The BLAST-Search-3 and BLAST-2-Union approach is incorporated into the CHUGO system.

Method	Precision	Recall	Average Cost per Protein
Voting Top-Down	77%	79%	111
BLAST-Search-3 and BLAST-2-Union (CHUGO)	79%	79%	59

have been studied, and thus BLAST will not be able to find a similar sequence in the database of studied proteins. One option is to simply accept the top BLAST result, regardless of its *E-value*. However, this method proves to be quite inaccurate, as shown in Table 6.5. Since our predictors of protein function model the sequences in a variety of ways, they can make predictions on a wider range of protein sequences.

For the prediction of these proteins, the PFAM and PA predictors prove to be the best combination of predictors to use in a voting ensemble (BLAST and PST are too inaccurate). Since BLAST performs so poorly on these proteins, it cannot produce good candidate nodes for lowering the cost of prediction. Therefore, a top-down approach is used to lower the cost of Voting predictors. This approach is used in CHUGO when no good BLAST hits are found against experimentally verified data.

Some molecular functions may not be as amenable to certain machine learning techniques as others. For example, some functional classes may depend on a small portion of the protein sequence (such as a functional domain), whereas others may be determined by the overall tertiary structure of the protein. Therefore, some prediction technologies may be better suited to some functional classes than other approaches.

To see if the reason that BLAST performs significantly worse than CHUGO on these dissimilar proteins is due to this phenomenon, each predictor was evaluated on a subset of the ontology (Figure 6.3). This subset was created for CHUGO by starting with no nodes in the ontology. Next, the node that increases the performance of CHUGO by the largest margin (in terms of hierarchical F-measure) was added to the ontology. Only nodes that made a consistent ontology were added. In other words, a node was not added to the ontology unless all of its parents nodes had been added. The process was repeated until all 406 nodes were added to the ontology. The entire process was then repeated for BLAST. Although this greedy approach could get caught in local maxima, it did give each prediction method a

Table 6.5: Performance on proteins with no Good BLAST result

Performance on proteins that do not produce a good BLAST hit during cross-validation. For BLAST, any BLAST hit is accepted as a nearest neighbor, regardless of its *E-value*. In this case, the voting ensemble is made up of PA-SVM and PFAM-SVM, with 1 vote required between the two to make a positive prediction. The Voting Top-Down approach is incorporated into the CHUGO system for proteins with no good BLAST hits.

Method	Precision	Recall	F-measure	95% CI (F-measure)	Cost
BLAST	19%	20%	19%	±1.9%	1
PA-SVM	59%	25%	35%	±3.0%	406
PFAM-SVM	54%	21%	30%	±2.3%	406
PST	16%	10%	12%	±1.5%	406
Voting	55%	32%	41%	±2.5%	812
Voting Top-Down (CHUGO)	56%	32%	41%	±2.4%	58

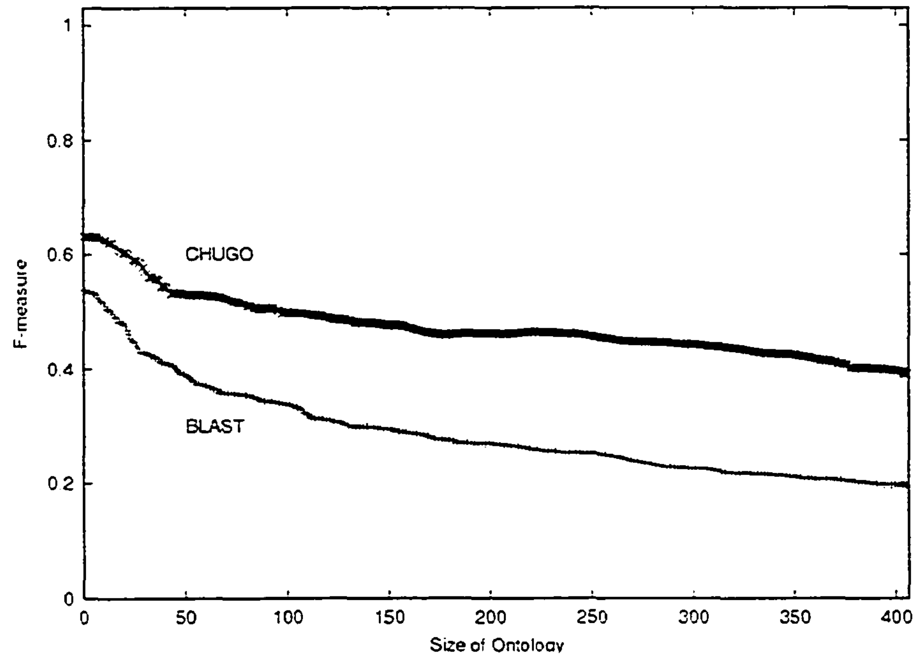


Figure 6.3: Comparing BLAST and CHUGO on Pruned Ontologies

CHUGO produces a better classification system even when the ontology is created in each predictors favor.

Table 6.6: Coverage against model organisms

Not finding good BLAST result against experimental data accounts for a large percentage of sequenced proteomes.

Organism	Good BLAST Hit	No Good BLAST Hit
D. melanogaster	60%	40%
S. cerevisae	62%	38%

fair chance to perform well on a consistent subset of the ontology.

Since the creation of the ontology is done separately for CHUGO and BLAST, the two lines are not directly comparable. That is, when the size of the ontology is 50, the point for CHUGO does not represent the performance on the same ontology as the point for BLAST. Although the two ontologies may not be comparable, the graph does show that even when BLAST was given a fair chance at being evaluated on a smaller ontology, CHUGO consistently performed better.

6.3 Coverage

Knowing how often a good BLAST hit is found during cross-validation is useful, but ultimately a predictive system will be used on unknown proteins, possibly in newly sequenced organisms. BLAST is used as a baseline because of its ubiquitous use. It is therefore important to know how often each of the cases – when a protein has a good BLAST hit against experimental data, or when it does not – would occur in reality. To approximate how often future unknown sequences would not result in a good BLAST hit, a BLAST query was run for each protein in two model organisms against the entire data set of experimentally annotated proteins (Table 6.6).

Within an entire proteome, the number of proteins that do not find a good BLAST hit against the experimental data set is much higher than found during cross-validation. This shows an increased importance for the case of no good BLAST hits found, since this happens more often in these proteomes. This difference is most likely because there are many more proteins in these proteomes whose function is unknown, and have no well-studied homologs. The effects would be magnified when examining an organism that is not well studied (relative to those shown in Table 6.6).

Table 6.7: Comparison to ProtFun

CHUGO outperforms ProtFun for the prediction of the 1,637 proteins which do not find a good BLAST hit during the cross-validation of CHUGO.

Predictor	Precision	Recall
PST	0.153	0.067
BLAST NN	0.188	0.198
PA-SVM	0.727	0.247
PFAM-SVM	0.674	0.112
CHUGO	0.689	0.288
ProtFun	0.143	0.128

6.4 Comparison to ProtFun

As a final experiment, CHUGO predictions were compared to those made by another protein function prediction system, Protfun [24] version 2.2 (Table 6.7). Protfun uses an ontology that contains 14 Gene Ontology nodes, where 9 of these nodes are from the molecular function aspect (Figure 2.6). Both systems were evaluated on their predictions for the 1,637 proteins which did not have a good BLAST hit during cross-validation, and only on those 9 molecular function nodes which are in the ProtFun ontology.

Even though the comparison is not completely fair to CHUGO (we do not have control over the training set for ProtFun, and thus some of the submitted proteins may have been used in the training of the system), Table 6.7 shows that CHUGO outperforms Protfun by a large margin on these proteins. Since these proteins had no good BLAST hit, the CHUGO system consisted of ensemble classifiers made up of PA-SVM and PFAM-SVM at each of the 9 nodes. The results of using other predictors presented in this dissertation are shown as well.

6.5 Summary

Since each of the local prediction methods presented in this dissertation are in fact sequence-based, their performance on disparate proteins dwindles similarly to BLAST-based predictors. However, since each tool models the protein sequence in a different manner, combining the predictors provides a system that can make predictions on a wider range of proteins. When dealing with these proteins, which are very dissimilar to well-studied proteins, biologists would appreciate any leads they can get before beginning lengthy experiments on them. Although CHUGO's predictors may not be extremely reliable on these disparate sequences, they do allow for a large increase in predictive accuracy over simply using BLAST NN.

The results exploiting BLAST NN to lower the computational cost of prediction within a hierarchy are applicable to the general problem of hierarchical classification. Any global prediction method can be used to generate candidate nodes for which local predictors are computed, as long as the global predictor adheres to the principles for candidate generating methods.

Chapter 7

Conclusion

7.1 Discussion of Results

This dissertation has shown that it is beneficial to include knowledge about the hierarchical structure of an ontology during both training and prediction. Furthermore, the increased cost of using local predictors can be lowered when the hierarchy's structure is considered by two different approaches. The first is a top-down approach predictor to lower the cost. This can be done without a significant impact on precision and recall when an inclusive training strategy is used. The second approach applies when an accurate global predictor is available (in this case BLAST). The predictions it produces can be used to find candidate nodes for local predictors, lowering the overall cost of prediction. A hierarchy-aware evaluation methodology was also outlined in this dissertation. This includes an intuitive scoring scheme within a hierarchy, and a sound methodology for cross-validation.

The Gene Ontology hierarchy provides a mechanism to represent incomplete information. Biological experiments vary in how specifically they determine protein functions. Organizing these functions in a hierarchy allows these differences to be made explicit.

The fact that the labels in GOA are incomplete in that most annotations are not fully specified, makes the problem of protein function prediction different from some other hierarchical classification problems. One such example is the hierarchical classification of text documents [29]. In contrast to the work in this dissertation, Koller *et al* showed that exclusive and inclusive classifiers perform similarly. However, the structure of their ontology was created artificially, and thus each instance was assigned to a leaf node (and therefore complete).

Furthermore, in domains such as web document classification [19], instances may be validly assigned to non-leaf nodes. For example, a web page may be about "Health and Fitness", but not any more specific. This is contrasted with GO, where proteins that are mapped to non-leaf nodes are only annotated at that node because their annotations are incomplete.

When dealing with proteins which are far (in terms of sequence similarity, and thus homology) from well-studied proteins, the biologists considering them would likely appreciate any leads they can get before beginning lengthy experiments on them. CHUGO is therefore a way of pushing the boundaries of sequence analysis, and ultimately a way of speeding up the process of protein function determination in general.

An important aspect to protein function prediction is that there may be a fundamental limit to how well functions of proteins can be predicted from a protein's structures (primary, secondary, or tertiary). This is because proteins are very sensitive to their environment. Proteins "...may only be active in their native state, over a small pH range, and under solution conditions with a minimum quantity of electrolytes" [5]. It is highly likely that a protein's environment is essential knowledge to determine what functions that protein can perform. Therefore, the lack of this knowledge in a prediction system may set an upper bound on how well protein functions can be predicted.

7.2 Future Work

In the future, it would be beneficial to include other types of local predictors of protein function. One option is to use biological properties of proteins as features for classifiers. This was shown to be effective by Jensen *et al* [24]. The addition of other features could be added to CHUGO by using the hierarchy aware training strategies presented in this dissertation.

Another aspect that could improve the performance of CHUGO is to make the PA-SVM predictors more accurate. The results in Chapter 5.2 did not show promising results for using other fields for prediction, however there is a large amount of knowledge in the Swiss-Prot entries of proteins that shows great potential for improvement. Other combinations of these annotations or more complex parsing techniques may prove to be useful in leveraging this data.

In Chapter 4, four training schemes were evaluated. These schemes were based on different degrees of consistency with the semantics of the hierarchy. However, these are by no means the only possibilities for creating training sets. There are many possibilities for choosing instances for positive and negative training sets. For instance, the distance in the hierarchy could be used to decide whether an instance is far enough from the current node to be used as a negative training instance. Other approaches could be attempted to increase the quality of all local prediction technologies used in this dissertation.

Chapter 4 also attempted to use a randomized approach to find better global splits of the data for cross-validation. An algorithmic approach could be used to split the data in a way that would preserve local node distributions optimally. However, the complexity of such an algorithm should be evaluated since it may prove to be intractable.

Another topic that could use further attention is finding an even more efficient way of using the local predictors when there are no good BLAST hits. Accepting a lower quality hit does not appear to be a good way of finding candidate terms to compute, but perhaps there are other global predictors that could be used to seed the search for candidate nodes in the hierarchy. For now, CHUGO uses the top-down approach on proteins that do not find a good BLAST result.

It would also be desirable to have a better approximation of each predictor's cost during the comparison of prediction schemes, rather than assigning a constant value for each. Although using the simple approach of assigning a constant cost to each predictor is a good way of getting an idea of the total cost of different prediction methods, a more accurate measure would be beneficial to deciding how expensive CHUGO predictions really are.

The fact that there are so many variables to tune for each predictor (parameters for each machine learning technology, feature selection, kernels, combinations of predictors) shows that there is a large room for potential improvement by focusing on each local predictor.

Finally, the experiments in this dissertation could be repeated for the other two aspects of Gene Ontology – Cellular Component and Biological Process. The approaches presented in this dissertation may not be as amenable to these other two aspects as they are to Molecular Function, but it would be interesting to know if they are. Knowing the Cellular Component and Biological Process for an unknown protein would be useful information for molecular biologists.

7.3 Summary

High-throughput and accurate protein function prediction is important to closing the gap between sequenced proteins and experimentally determined protein functions. Ontologies such as GO help to alleviate this problem by providing standardized, hierarchical vocabularies with which to describe the problem domain. This dissertation has presented three novel methods to exploit the hierarchical nature of GO to increase predictive performance, and to lower computational cost of using local predictors. First, the hierarchy is utilized to increase the accuracy of local predictors by considering its structure during the construction of training sets. Second, the hierarchy was used to lower the computational cost of running local term predictors by using a top-down approach or by exploiting predictions seeded by BLAST. Third, an evaluation methodology that produces hierarchical precision and hierarchical recall was examined, and a sound method of global cross-validation was presented.

The methods that have been presented may be applicable to many other domains where there is a standardized, hierarchical ontology, such as document classification, medical diagnosis, web documents, and many others. By leveraging the knowledge encoded in the hierarchical structure of these ontologies, prediction methods can become more accurate and more efficient in domains where there are a large number of classes.

Bibliography

[1] On learning hierarchical classification. http://citeseer.ist.psu.edu/38202.html, 1997.

[2] The automated function prediction special interest group meeting, held at ISMB 2005. http://ffas.burnham.org/AFP, 2005.

[3] Gene Ontology Annotation @ EBI. http://www.ebi.ac.uk/GOA/, 2005.

[4] the Gene Ontology website. http://www.geneontology.org/, 2005.

[5] Wikipedia, the free encyclopedia. http://www.wikipedia.org, 2005.

[6] The GO consortium user meeting. Bergen, Norway, September 14 - 15 2005.

[7] E Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[8] S Altschul, W Gish, W Miller, E W Myers, and D Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[9] M A Andrade, N P Brown, C Leroy, S Hoersch, A de Daruvar, C Reich, A Franchini, J Tamames, A Valencia, C Ouzounis, and C Sander. Automated genome sequence analysis and annotation. *Bioinformatics*, 15:391–412, 1999.

[10] EM Zdobnov R Apweiler. InterProScan - an integration platform for the signature-recognition methods in InterPro. *Bioinformatics*, 17(9):847–8, 2001.

[11] R Apweiler, A Bairoch, CH Wu, WC Barker, B Boeckmann, S Ferro, E Gasteiger, H Huang, R Lopez, M Magrane, MJ Martin, DA Natale C O'Donovan, N Redaschi, and LS Yeh. UniProt: the Universal Protein knowledgebase. *Nucleic Acids Research*, 32:D115–D119, 2004.

[12] A Bateman, E Birney, L Cerruti, R Durbin, L Etwiller, S R Eddy, S Griffiths-Jones, K L Howe, M Marshall, and E L Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 30(1):276–280, 2002.

[13] G Bejerano and G Yona. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics*, 17(1):23–43, 2001.

[14] B Boeckmann, A Bairoch, R Apweiler, MC Blatter, A Estreicher, E Gasteiger, MJ 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*, 31:365–370, 2003.

[15] S Chakrabarti, B E Dom, R Agrawal, and P Raghaven. Using taxonomy, discriminants and signatures for navigating in text databases. In *Proc. of the 23rd International Conference on Very Large Databases*, pages 446–455, 1997.

[16] C C Chang and C J Lin. LIBSVM: A library for support vector machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm, 2001.

[17] F Crick. On protein synthesis. In *Symposium Society Experimental Biology*, number 12, pages 138–163, 1958.

[18] O Dekel, J Keshet, and Y Singer. Large margin hierarchical classification. In *Proc. of the 21th International Conference on Machine Learning*, pages 209–216, 2004.

[19] S T Dumais and H Chen. Hierarchical classification of Web content. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR, 2000.

[20] S R Eddy. HMMER: Profile hidden markov models for biological sequence analysis. http://hmmer.wustl.edu, 2001.

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

[22] M Ashburner et al. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[23] T Hastie, R Tibshirani, and J Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2001.

[24] L J Jensen, H Staerfeldt, and Sren Brunak. Prediction of human protein function according to Gene Ontology categories. *Bioinformatics*, 19:635–642, 2003.

[25] D Jurafsky and JH Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[26] O D King, R E Foulger, S S Dwight, J V White, and F P Roth. Predicting gene function from patterns of annotation. *Genome Research*, 13:896–904, 2003.

[27] S Kiritchenko, S Matwin, and A Fazel Famili. Hierarchical text categorization as a tool of associating genes with Gene Ontology codes. In *Proc. of the Second European Workshop on Data Mining and Text Mining for Bioinformatics*, pages 26–30, Pisa, Italy, 2004.

[28] S Kiritchenko, S Matwin, and A Fazel Famili. Functional annotation of genes using hierarchical text categorization. In *Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05)*, Detroit, USA, 2005.

[29] D Koller and M Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning*, pages 170–178, 1997.

[30] C Li, editor. *Biochemical Nomenclature and Related Documents.* Portland Press, second edition edition, 1992.

[31] D Lin. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, pages 296–304, 1998.

[32] P W Lord, R D Stevens, A Brass, and C A Goble. Semantic similarity measures as tools for exploring the Gene Ontology. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 601–612, Lihue, Hawaii, 2003.

[33] Z Lu. Predicting protein sub-cellular localization from homologs using machine learning algorithms. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2003.

[34] Z Lu, D Szafron, R Greiner, P Lu, D Wishart, B Poulin, J Anvik, C Macdonell, and R Eisner. Predicting sub-cellular localization using machine-learned classifiers in proteome analyst. *Bioinformatics*, 20:547–556, 2004.

[35] A K McCallum, R Rosenfeld, T M Mitchell, and A Y Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of the 15th International Conference on Machine Learning*, pages 359–367, 1998.

[36] R Moskovitch, S Cohen-Kashi, U Dror, I Levy, A Maimon, and Y Shahar. Multiple hierarchical classification of free-text clinical guidelines. In *Workshop on intelligent data analysis in medicine and pharmacology*, 2004.

[37] A G Murzin, S E Brenner, T Hubbard, and C Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

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

[39] B Poulin. Sequence-based protein function prediction. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2004.

[40] B Poulin. Personal communication. 2005.

[41] B Poulin, R Eisner, D Szafron, P Lu, R Greiner, and D Wishart. Visual explanation and auditing of evidence with additive classifiers. In Preparation, 2005.

[42] M Riley. Functions of the gene products of escherichia coli. In *Microbiological Reviews*, volume 57, pages 862–52, 1993.

[43] M E Ruiz and P Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[44] R Sharma and D Poole. Probabilistic reasoning with hierarchically structured variables. In *Proc. of the 19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[45] B Smith, J Williams, and S Schulze-Kremer. The ontology of the Gene Ontology. In *Proc. of the 2003 American Medical Informatics Association Symposium*, 2003.

[46] D Szafron, P Lu, R Greiner, D Wishart, Z Lu, B Poulin, R Eisner, J Anvik, and C Macdonell. Proteome Analyst - transparent high-throughput protein annotation: Function, localization and custom predictors. In *International Conference on Machine Learning Workshop on Machine Learning in Bioinformatics (ICML Workshop - Bioinformatics)*, pages 2–10, Washington, U.S.A., August 2003.

[47] D Szafron, P Lu, R Greiner, DS Wishart, B Poulin, R Eisner, Z Lu, J Anvik, C Macdonell, A Fyshe, and D Meeuwis. Proteome Analyst: custom predictions with explanations in a web-based tool for high-throughput proteome annotations. *Nucleic Acids Research*, 32(2):W365–371, 2004.

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

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

[50] K Wang, S Zhou, and Y He. Hierarchical classification of real life documents. In *Proceedings of the 1st SIAM International Conference on Data Mining*, Chicago, US, 2001.

[51] K Wang, S Zhou, and S C Liew. Building hierarchical classifiers using class proximity. In *Proc. of the 25th International Conference on Very Large Databases*, pages 363–374, 1999.

[52] E Werner. Genome semantics, in silico multicellular systems and the central dogma. *FEBS Letters*, 579:1779–1782, 2005.

[53] H Wu, Z Su, F Mao, V Olman, and Y Xu. Prediction of functional modules based on comparative genome analysis and Gene Ontology application. *Nucleic Acids Research*, 33(9):2822–2837, 2005.

Appendix A

Data Set Information

Table A.1: Evidence Code Histogram

The number of Gene Ontology Annotations provided by GOA for proteins in the Uniprot and Swissprot databases.

Evidence Code	Number of Annotations for UniProt Proteins	Number of Annotations for SwissProt Proteins
IEA	2,219,999	483,200
IDA	7,773	4,089
TAS	6,468	5,581
ISS	4,326	2,499
NAS	3,985	2,711
IPI	3,162	1,445
ND	1,312	827
IMP	1,056	729
NR	676	629
IGI	272	204
IEP	96	83
IC	42	24

Table A.2: General Data Set Statistics

General Statistics about the ontology, and the experimentally annotated data set.

Statistic	Value
Original Number of Nodes	7,399
Number of Nodes in Pruned Ontology	406
Average Number of Parents Per Node (In Original Ontology)	1.154
Number of Nodes in Pruned Ontology with > 1 Parent	60
Number of Nodes in Original Ontology with > 1 Parent	1,036
Number of Proteins Used in Data Set	14,362
Mappings Per Protein	1.355
Labels Per Protein (Propagating from Mapped)	4.935
Maximum Depth to a Node in Pruned Ontology	10

Appendix B

Supplementary Results

Table B.1: Parameter Search for the PST order

Performing a brute force parameter search for each node in the GO hierarchy. Histogram shows how many of the 406 nodes had each setting of the order.

Value	Occurence
5	156
4	117
6	116
3	13
2	4

Table B.2: Parameter Search for Smoothing term

Performing a brute force parameter search for each node in the GO hierarchy. Histogram shows how many of the 406 nodes had each setting of the smoothing term.

Value	Occurence
10^{-2}	264
10^{-6}	97
1	42
10	3

Table B.3: Parameter Search for Window Length

Performing a brute force parameter search for each node in the GO hierarchy. Histogram shows how many of the 406 nodes had each setting of window length.

Value	Occurence
1,000	91
200	84
100	77
10	56
40	38
400	31
20	29

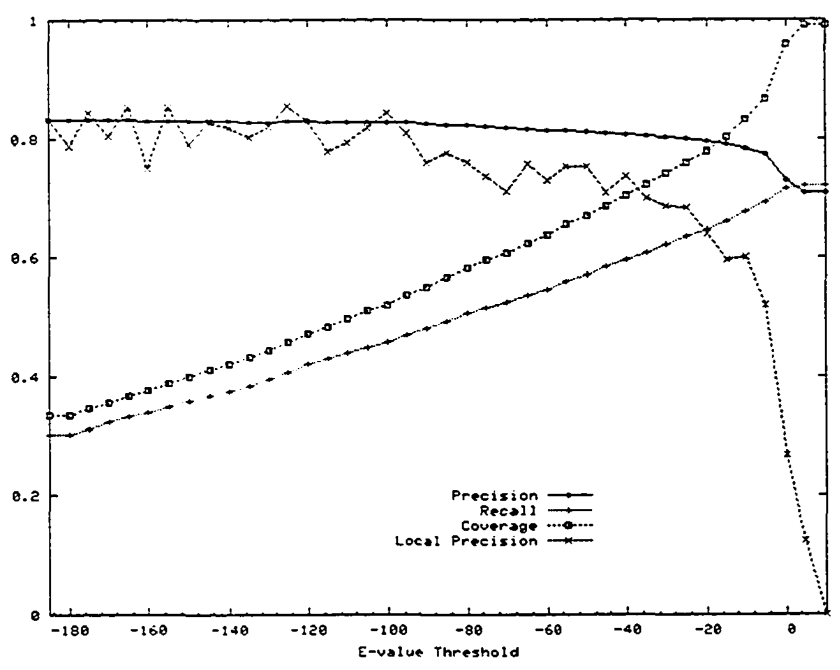


Figure B.1: Evaluating BLAST with Varying E-Value

E-value is varied for accepting a BLAST hit for BLAST NN. Local precision refers to how precise the matches added from the previous E-value are. This reinforces the results shown in [39], although the curves are not as high since a more conservative data set is used here..

Appendix C

The Pruned Ontology

This Appendix lists the pruned Gene Ontology used for the experiments in this dissertation. For readability, long node names are shortened. Not all 14,396 proteins were used in experiments. Only those which mapped to at least one node in the pruned ontology were kept. This lowered the size of the dataset to 14,362 proteins. 406 out of 7,399 original molecular function terms are in the pruned ontology. Some terms are shown more than once due to multiple inheritance.

```
{Depth} GO_ID>>GO_Name - [InheritedProteins, MappedProteins] - <#Children, #Parents>

{00}0003673>>"Gene_Ontology" - [14396, 0] - <1, 0>
{01}  0003674>>"molecular_function" - [14396, 0] - <15, 1>
{02}    0016209>>"antioxidant activity" - [74, 5] - <4, 1>
{03}      0004601>>"peroxidase activity" - [55, 18] - <16, 2>
{04}        0004602>>"glutathione peroxidase activity" - [21, 20] - <1, 1>
{02}    0005488>>"binding" - [7105, 25] - <38, 1>
{03}      0003823>>"antigen binding" - [53, 25] - <3, 1>
{04}        0042605>>"peptide antigen binding" - [26, 24] - <2, 2>
{03}      0030246>>"carbohydrate binding" - [137, 10] - <2, 1>
{04}        0030247>>"polysaccharide binding" - [63, 5] - <6, 2>
{05}          0005539>>"glycosaminoglycan binding" - [54, 5] - <2, 1>
{06}            0008201>>"heparin binding" - [36, 36] - <0, 1>
{04}        0005529>>"sugar binding" - [64, 39] - <3, 1>
{05}          0048029>>"monosaccharide binding" - [24, 0] - <7, 1>
{03}      0008144>>"drug binding" - [28, 6] - <8, 1>
{03}      0043167>>"ion binding" - [293, 0] - <3, 1>
{04}        0043169>>"cation binding" - [271, 0] - <3, 1>
{05}          0005509>>"calcium ion binding" - [124, 123] - <1, 2>
{05}          0046914>>"transition metal ion binding" - [152, 0] - <9, 2>
{06}            0005506>>"iron ion binding" - [23, 10] - <2, 1>
{06}            0008270>>"zinc ion binding" - [97, 97] - <0, 1>
{04}        0046872>>"metal ion binding" - [293, 10] - <5, 1>
{05}          0005509>>"calcium ion binding" - [124, 123] - <1, 2>
{05}          0000287>>"magnesium ion binding" - [21, 21] - <0, 1>
{05}          0046914>>"transition metal ion binding" - [152, 0] - <9, 2>
{06}            0005506>>"iron ion binding" - [23, 10] - <2, 1>
{06}            0008270>>"zinc ion binding" - [97, 97] - <0, 1>
{03}      0008289>>"lipid binding" - [104, 19] - <6, 1>
{04}        0005543>>"phospholipid binding" - [73, 38] - <4, 1>
{05}          0035091>>"phosphoinositide binding" - [23, 3] - <4, 1>
{03}      0042165>>"neurotransmitter binding" - [113, 0] - <5, 1>
{04}        0042166>>"acetylcholine binding" - [28, 0] - <1, 1>
{05}          0015464>>"acetylcholine receptor activity" - [28, 15] - <2, 3>
{04}        0042923>>"neuropeptide binding" - [46, 1] - <2, 2>
{05}          0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
{04}        0030594>>"neurotransmitter receptor activity" - [112, 2] - <4, 2>
{05}          0015464>>"acetylcholine receptor activity" - [28, 15] - <2, 3>
{05}          0016917>>"GABA receptor activity" - [31, 1] - <2, 2>
{06}            0004890>>"GABA-A receptor activity" - [27, 27] - <0, 1>
{05}          0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
{03}      0003676>>"nucleic acid binding" - [2057, 12] - <3, 1>
{04}        0003677>>"DNA binding" - [1494, 780] - <23, 1>
{05}          0003682>>"chromatin binding" - [78, 78] - <1, 1>
{05}          0003684>>"damaged DNA binding" - [51, 51] - <0, 1>
{05}          0003690>>"double-stranded DNA binding" - [42, 39] - <2, 1>
{05}          0003697>>"single-stranded DNA binding" - [45, 42] - <2, 1>
{05}          0003700>>"transcription factor activity" - [694, 680] - <1, 2>
{04}        0003723>>"RNA binding" - [530, 267] - <16, 1>
{05}          0003725>>"double-stranded RNA binding" - [23, 23] - <0, 1>
{05}          0003729>>"mRNA binding" - [186, 50] - <6, 1>
{06}            0008248>>"pre-mRNA splicing factor activity" - [97, 97] - <3, 1>
{05}          0003727>>''single-stranded RNA binding'' - [26, 13] - <3, 1>
{05}          0030515>>''snoRNA binding'' - [22, 22] - <0, 1>
{04}        0008135>>''translation factor activity, nucleic a...'' - [61, 25] - <5, 2>
{05}          0003743>>''translation initiation factor activity'' - [30, 30] - <0, 1>
{03}      0000166>>''nucleotide binding'' - [208, 12] - <5, 1>
{04}        0017076>>"purine nucleotide binding" - [195, 0] - <2, 1>
{05}          0030554>>"adenyl nucleotide binding" - [125, 0] - <5, 1>
{06}            0005524>>"ATP binding" - [118, 118] - <0, 1>
{05}          0019001>>"guanyl nucleotide binding" - [74, 7] - <3, 1>
{06}            0005525>>"GTP binding" - [63, 63] - <0, 1>
```

```
{03}      0019825>>"oxygen binding" - [26, 25] - <1, 1>
{03}      0001871>>"pattern binding" - [73, 0] - <4, 1>
{04}        0030247>>"polysaccharide binding" - [63, 5] - <6, 2>
{05}          0005539>>"glycosaminoglycan binding" - [54, 5] - <2, 1>
{06}            0008201>>"heparin binding" - [36, 36] - <0, 1>
{03}      0042277>>"peptide binding" - [242, 4] - <5, 1>
{04}        0042923>>"neuropeptide binding" - [46, 1] - <2, 2>
{05}          0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
{04}        0042605>>"peptide antigen binding" - [26, 24] - <2, 2>
{04}        0001653>>"peptide receptor activity" - [177, 12] - <2, 2>
{05}          0008528>>"peptide receptor activity, G-protein..." - [166, 3] - <21, 2>
{06}            0001637>>"G-protein chemoattractant receptor ac..." - [41, 0] - <2, 1>
{07}              0004950>>"chemokine receptor activity" - [41, 21] - <3, 2>
{06}            0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
{06}            0004985>>"opioid receptor activity" - [26, 3] - <4, 1>
{04}        0005048>>"signal sequence binding" - [28, 10] - <7, 1>
{03}      0005515>>"protein binding" - [4035, 2931] - <70, 1>
{04}        0005516>>"calmodulin binding" - [45, 45] - <1, 1>
{04}        0019955>>"cytokine binding" - [94, 1] - <6, 1>
{05}          0019956>>"chemokine binding" - [41, 0] - <4, 1>
{06}            0004950>>"chemokine receptor activity" - [41, 21] - <3, 2>
{05}          0019965>>"interleukin binding" - [34, 1] - <28, 1>
{06}            0004907>>"interleukin receptor activity" - [32, 7] - <27, 2>
{04}        0008092>>"cytoskeletal protein binding" - [276, 19] - <12, 1>
{05}          0003779>>"actin binding" - [131, 118] - <2, 1>
{05}          0015631>>"tubulin binding" - [114, 11] - <4, 1>
{06}            0008017>>"microtubule binding" - [99, 99] - <2, 1>
{04}        0019899>>"enzyme binding" - [107, 18] - <14, 1>
{05}          0019900>>"kinase binding" - [79, 5] - <1, 1>
{06}            0019901>>"protein kinase binding" - [74, 40] - <5, 1>
{04}        0019838>>"growth factor binding" - [33, 0] - <16, 1>
{04}        0008034>>"lipoprotein binding" - [23, 5] - <4, 1>
{04}        0008022>>"protein C-terminus binding" - [26, 26] - <0, 1>
{04}        0046983>>"protein dimerization activity" - [106, 3] - <2, 1>
{05}          0046982>>"protein heterodimerization activity" - [45, 45] - <0, 1>
{05}          0042803>>"protein homodimerization activity" - [68, 68] - <0, 2>
{04}        0019904>>"protein domain specific binding" - [77, 23] - <15, 1>
{05}          0017124>>"SH3 domain binding" - [27, 27] - <0, 1>
{04}        0042802>>"protein self binding" - [97, 17] - <2, 1>
{05}          0042803>>"protein homodimerization activity" - [68, 68] - <0, 2>
{04}        0000149>>"SNARE binding" - [26, 10] - <1, 1>
{04}        0008134>>"transcription factor binding" - [418, 63] - <6, 1>
{05}          0003712>>"transcription cofactor activity" - [359, 46] - <2, 2>
{06}            0003713>>"transcription coactivator activity" - [180, 172] - <2, 1>
{06}            0003714>>"transcription corepressor activity" - [149, 149] - <0, 1>
{04}        0051082>>"unfolded protein binding" - [77, 77] - <0, 1>
{03}      0005102>>"receptor binding" - [591, 147] - <53, 2>
{04}        0005125>>"cytokine activity" - [173, 55] - <27, 1>
{05}          0008009>>"chemokine activity" - [54, 54] - <0, 2>
{05}          0005126>>"hematopoietin/interferon-class (D200 ..." - [37, 7] - <25, 1>
{04}        0001664>>"G-protein-coupled receptor binding" - [65, 9] - <5, 1>
{05}          0042379>>"chemokine receptor binding" - [54, 0] - <3, 1>
{06}            0008009>>"chemokine activity" - [54, 54] - <0, 2>
{04}        0008083>>"growth factor activity" - [86, 65] - <16, 1>
{04}        0005179>>"hormone activity" - [71, 44] - <13, 1>
{05}          0005184>>"neuropeptide hormone activity" - [21, 21] - <5, 1>
{04}        0005178>>"integrin binding" - [45, 45] - <0, 1>
{02}    0003824>>"catalytic activity" - [5005, 47] - <46, 1>
{03}      0009975>>"cyclase activity" - [32, 0] - <6, 1>
{04}        0004383>>"guanylate cyclase activity" - [23, 23] - <0, 2>
{03}      0004386>>"helicase activity" - [91, 13] - <4, 1>
{04}        0008026>>"ATP-dependent helicase activity" - [42, 0] - <2, 2>
{05}          0004003>>"ATP-dependent DNA helicase activity" - [28, 21] - <3, 3>
{04}        0003678>>"DNA helicase activity" - [54, 27] - <4, 1>
{05}          0004003>>"ATP-dependent DNA helicase activity" - [28, 21] - <3, 3>
{04}        0003724>>"RNA helicase activity" - [32, 17] - <1, 1>
{03}      0016787>>"hydrolase activity" - [1786, 9] - <15, 1>
{04}        0016817>>"hydrolase activity, acting on acid anh..." - [400, 0] - <3, 1>
{05}          0016820>>"hydrolase activity, acting on acid anh..." - [86, 0] - <1, 1>
{06}            0042626>>"ATPase activity, coupled to transme..." - [86, 20] - <35, 2>
{07}              0042625>>"ATPase activity, coupled to transmem..." - [62, 1] - <3, 1>
{08}                0015662>>"ATPase activity, coupled to transm..." - [55, 3] - <13, 3>
{05}          0016818>>"hydrolase activity, acting on acid a..." - [400, 1] - <33, 1>
{06}            0016887>>"ATPase activity" - [212, 44] - <2, 1>
{07}              0042623>>"ATPase activity, coupled" - [170, 21] - <7, 1>
{08}                0008026>>"ATP-dependent helicase activity" - [42, 0] - <2, 2>
{09}                  0004003>>"ATP-dependent DNA helicase activity" - [28, 21] - <3, 3>
{08}                0042626>>"ATPase activity, coupled to trans..." - [86, 20] - <35, 2>
{09}                  0042625>>"ATPase activity, coupled to transm..." - [62, 1] - <3, 1>
{10}                    0015662>>"ATPase activity, coupled to tran..." - [55, 3] - <13, 3>
{08}                0008094>>"DNA-dependent ATPase activity" - [47, 19] - <4, 1>
{09}                  0004003>>"ATP-dependent DNA helicase activity" - [28, 21] - <3, 3>
{06}            0003924>>"GTPase activity" - [145, 145] - <0, 1>
{04}        0016810>>"hydrolase activity, acting on carbon-..." - [152, 0] - <12, 1>
{05}          0016814>>"hydrolase activity, acting on carbon-n..." - [45, 0] - <3, 1>
{06}            0019239>>"deaminase activity" - [35, 0] - <24, 1>
{05}          0016811>>"hydrolase activity, acting on carbon-..." - [88, 1] - <77, 1>
{06}            0019213>>"deacetylase activity" - [59, 0] - <15, 1>
{07}              0004407>>"histone deacetylase activity" - [56, 48] - <2, 1>
{04}        0016788>>"hydrolase activity, acting on ester bonds" - [634, 4] - <9, 1>
{05}          0016789>>"carboxylic ester hydrolase activity" - [116, 3] - <68, 1>
{06}            0016298>>"lipase activity" - [88, 5] - <5, 1>
{07}              0004620>>"phospholipase activity" - [73, 1] - <5, 1>
{08}                0004623>>"phospholipase A2 activity" - [22, 9] - <2, 1>
{08}                0004629>>"phospholipase C activity" - [25, 11] - <2, 2>
{05}          0004518>>"nuclease activity" - [144, 3] - <4, 1>
{06}            0004536>>"deoxyribonuclease activity" - [44, 5] - <3, 1>
{07}              0004520>>"endodeoxyribonuclease activity" - [31, 11] - <10, 2>
{06}            0004519>>"endonuclease activity" - [92, 17] - <4, 1>
{07}              0004520>>"endodeoxyribonuclease activity" - [31, 11] - <10, 2>
{07}              0016893>>"endonuclease activity, active with e..." - [42, 0] - <2, 1>
{08}                0016891>>"endoribonuclease activity, producin..." - [38, 1] - <5, 2>
{07}              0004521>>"endoribonuclease activity" - [45, 5] - <2, 2>
{08}                0016891>>"endoribonuclease activity, producin..." - [38, 1] - <5, 2>
{06}            0004527>>"exonuclease activity" - [44, 7] - <8, 1>
{07}              0008408>>"3'-5' exonuclease activity" - [28, 15] - <3, 1>
{06}            0004540>>"ribonuclease activity" - [70, 11] - <9, 1>
```

```
(07)            0004521>>"endoribonuclease activity" - [45, 5] - <2, 2>
(08)              0016891>>"endoribonuclease activity, producin..." - [38, 1] - <5, 2>
(05)        0042578>>"phosphoric ester hydrolase activity" - [363, 1] - <3, 1>
(06)          0008081>>"phosphoric diester hydrolase activity" - [85, 0] - <22, 1>
(07)            0047394>>"glycerophosphoinositol inositolphosp..." - [36, 0] - <1, 1>
(08)              0004112>>"cyclic-nucleotide phosphodiesterase..." - [36, 2] - <3, 1>
(09)                0004114>>"3',5'-cyclic-nucleotide phosphodie..." - [32, 6] - <6, 1>
(07)            0004629>>"phospholipase C activity" - [25, 11] - <2, 2>
(06)          0016791>>"phosphoric monoester hydrolase acti..." - [291, 9] - <62, 1>
(07)            0004437>>"inositol or phosphatidylinositol ph..." - [28, 6] - <10, 1>
(07)            0004721>>"phosphoprotein phosphatase activity" - [195, 23] - <5, 1>
(08)              0004722>>"protein serine/threonine phosphata..." - [57, 22] - <9, 1>
(08)              0004725>>"protein tyrosine phosphatase activity" - [94, 67] - <6, 1>
(08)              0008138>>"protein tyrosine/serine/threonine ..." - [28, 18] - <2, 1>
(05)        0016790>>"thiolester hydrolase activity" - [37, 0]   <17, 1>
(04)      0016798>>"hydrolase activity, acting on glycosyl..." - [129, 5] - <3, 1>
(05)        0016799>>"hydrolase activity, hydrolyzing N-gly..."   [39, 0] - <19, 1>
(06)          0019104>>"DNA N-glycosylase activity" - [30, 6] - <5, 1>
(05)        0004553>>"hydrolase activity, hydrolyzing O-gly..." - [85, 0] - <61, 1>
(04)      0008233>>"peptidase activity" - [467, 30] - <10, 1>
(05)        0008234>>"cysteine-type peptidase activity" - [113, 19] - <6, 1>
(06)          0004197>>"cysteine-type endopeptidase activity" - [71, 24] - <11, 2>
(07)            0030693>>"caspase activity" - [20, 20] - <0, 1>
(06)          0004843>>"ubiquitin-specific protease activity" - [44, 33] - <1, 1>
(05)        0004175>>"endopeptidase activity" - [252, 40] - <8, 1>
(06)          0004197>>"cysteine-type endopeptidase activity" - [71, 24] - <11, 2>
(07)            0030693>>"caspase activity" - [20, 20] - <0, 1>
(06)          0004222>>"metalloendopeptidase activity" - [64, 43] - <28, 2>
(06)          0004252>>"serine-type endopeptidase activity" - [59, 23] - <45, 2>
(05)        0008238>>"exopeptidase activity" - [80, 3] - <7, 1>
(06)          0004177>>"aminopeptidase activity" - [28, 18] - <11, 1>
(06)          0004180>>"carboxypeptidase activity" - [30, 14] - <5, 1>
(06)          0008235>>"metalloexopeptidase activity" - [28, 1] - <11, 2>
(05)        0008237>>"metallopeptidase activity" - [140, 47] - <5, 1>
(06)          0004222>>"metalloendopeptidase activity" - [64, 43] - <28, 2>
(06)          0008235>>"metalloexopeptidase activity" - [28, 1] - <11, 2>
(05)        0008236>>"serine-type peptidase activity" - [95, 43] - <5, 1>
(06)          0004252>>"serine-type endopeptidase activity" - [59, 23] - <45, 2>
(03)    0016853>>"isomerase activity" - [111, 0] - <12, 1>
(04)      0016860>>"intramolecular oxidoreductase activity" - [46, 3] - <14, 1>
(04)      0016866>>"intramolecular transferase activity" - [23, 0] - <23, 1>
(03)    0016301>>"kinase activity" - [1024, 51] - <55, 1>
(04)      0019200>>"carbohydrate kinase activity" - [43, 0] - <19, 1>
(04)      0004428>>"inositol or phosphatidylinositol kinase..." - [46, 8] - <5, 2>
(04)      0001727>>"lipid kinase activity" - [35, 0] - <5, 1>
(04)      0019205>>"nucleobase, nucleoside, nucleotide kina..." - [45, 0] - <5, 2>
(05)        0019206>>"nucleoside kinase activity" - [20, 0] - <4, 1>
(05)        0019201>>"nucleotide kinase activity" - [20, 2] - <5, 1>
(04)      0004672>>"protein kinase activity" - [781, 232] - <6, 2>
(05)        0004674>>"protein serine/threonine kinase ac..." - [470, 213] - <19, 1>
(06)          0004683>>"calmodulin regulated protein kinase a..." - [68, 0] - <7, 1>
(07)            0004685>>"calcium- and calmodulin-dependent p..." - [53, 53] - <0, 1>
(07)            0004687>>"myosin-light-chain kinase activity" - [39, 39] - <0, 1>
(06)          0004680>>"casein kinase activity" - [26, 10] - <2, 1>
(06)          0004693>>"cyclin-dependent protein kinase acti..." - [28, 28] - <0, 1>
(06)          0004697>>"protein kinase C activity" - [22, 16] - <4, 2>
(06)          0004702>>"receptor signaling protein serine/thr..." - [69, 4] - <7, 2>
(07)            0016909>>"SAP kinase activity" - [20, 0] - <5, 1>
(05)        0004713>>"protein-tyrosine kinase activity" - [134, 53] - <3, 1>
(06)          0004714>>"transmembrane receptor protein tyro..." - [69, 38] - <11, 2>
(05)        0019199>>"transmembrane receptor protein kinase ..." - [86, 0] - <3, 2>
(06)          0004714>>"transmembrane receptor protein tyro..." - [69, 38] - <11, 2>
(03)    0016874>>"ligase activity" - [234, 2] - <9, 1>
(04)      0016879>>"ligase activity, forming carbon-nitro..." - [130, 0] - <21, 1>
(05)        0016881>>"acid-D-amino acid ligase activity" - [91, 1] - <29, 1>
(06)          0004842>>"ubiquitin-protein ligase activity" - [70, 70] - <0, 1>
(04)      0016875>>"ligase activity, forming carbon-oxygen ..." - [43, 0] - <1, 1>
(05)        0016876>>"ligase activity, forming aminoacyl-tRN..." - [43, 0] - <5, 1>
(06)          0004812>>"tRNA ligase activity" - [43, 0] - <21, 2>
(04)      0016877>>"ligase activity, forming carbon-sulfur ..." - [37, 0] - <3, 1>
(05)        0015645>>"fatty-acid ligase activity" - [22, 5] - <2, 1>
(04)      0016886>>"ligase activity, forming phosphoric est..." - [51, 0] - <3, 1>
(05)        0008452>>"RNA ligase activity" - [44, 0] - <3, 1>
(06)          0004812>>"tRNA ligase activity" - [43, 0] - <21, 2>
(03)    0016829>>"lyase activity" - [181, 7] - <13, 1>
(04)      0016830>>"carbon-carbon lyase activity" - [48, 0] - <8, 1>
(05)        0016831>>"carboxy-lyase activity" - [31, 3] - <86, 1>
(04)      0016835>>"carbon-oxygen lyase activity" - [64, 0] - <8, 1>
(05)        0016836>>"hydro-lyase activity" - [51, 0] - <107, 1>
(04)      0016849>>"phosphorus-oxygen lyase activity" - [35, 3] - <7, 1>
(05)        0004383>>"guanylate cyclase activity" - [23, 23] - <0, 2>
(03)    0016491>>"oxidoreductase activity" - [688, 19] - <71, 1>
(04)      0015036>>"disulfide oxidoreductase activity" - [20, 0] - <3, 1>
(04)      0015002>>"heme-copper terminal oxidase activity" - [25, 0] - <2, 1>
(05)        0004129>>"cytochrome-c oxidase activity" - [25, 25] - <0, 4>
(04)      0004497>>"monooxygenase activity" - [71, 25] - <141, 1>
(05)        0008395>>"steroid hydroxylase activity" - [24, 8] - <17, 1>
(04)      0016614>>"oxidoreductase activity, acting on CH..." - [123, 1] - <39, 1>
(05)        0016616>>"oxidoreductase activity, acting on ..." - [109, 0] - <242, 1>
(04)      0016675>>"oxidoreductase activity, acting on heme..." - [25, 0] - <3, 1>
(05)        0016676>>"oxidoreductase activity, acting on hem..." - [25, 0] - <1, 1>
(06)          0004129>>"cytochrome-c oxidase activity" - [25, 25] - <0, 4>
(04)      0016651>>"oxidoreductase activity, acting on NAD..." - [66, 2] - <12, 1>
(05)        0003954>>"NADH dehydrogenase activity" - [38, 1] - <1, 2>
(06)          0050136>>"NADH dehydrogenase (quinone) activity" - [37, 0] - <1, 1>
(07)            0008137>>"NADH dehydrogenase (ubiquinone) act..." - [37, 37] - <0, 5>
(05)        0016655>>"oxidoreductase activity, acting on NAD..." - [39, 1] - <7, 1>
(06)          0008137>>"NADH dehydrogenase (ubiquinone) acti..."   [37, 37] - <0, 5>
(04)      0016705>>"oxidoreductase activity, acting on pai..." - [74, 0] - <74, 1>
(05)        0016709>>"oxidoreductase activity, acting on p..." - [21, 0] - <100, 1>
(04)      0016684>>"oxidoreductase activity, acting on pero..." - [55, 3] - <1, 1>
(05)        0004601>>"peroxidase activity" - [55, 18] - <16, 2>
(06)          0004602>>"glutathione peroxidase activity" - [21, 20] - <1, 1>
(04)      0016667>>"oxidoreductase activity, acting on sul..." - [33, 1] - <11, 1>
(04)      0016903>>"oxidoreductase activity, acting on the..." - [65, 0] - <13, 1>
(05)        0016620>>"oxidoreductase activity, acting on th..." - [48, 4] - <61, 1>
(06)          0004028>>"aldehyde dehydrogenase activity" - [23, 17]   <2, 1>
```

```
(04)          0016627>>"oxidoreductase activity, acting on the..." - [45, 0] - <29, 1>
(04)          0016645>>"oxidoreductase activity, acting on the..." - [28, 0] - <19, 1>
(04)          0016638>>"oxidoreductase activity, acting on the..." - [26, 0] - <13, 1>
(03)        0008639>>"small protein conjugating enzyme activity" - [48, 1] - <2, 1>
(04)          0004840>>"ubiquitin conjugating enzyme activity" - [43, 43] - <0, 1>
(03)        0016740>>"transferase activity" - [1911, 4] - <19, 1>
(04)          0016746>>"transferase activity, transferring acy..." - [176, 0] - <3, 1>
(05)            0016747>>"transferase activity, transferring g..." - [155, 3] - <39, 1>
(06)              0008415>>"acyltransferase activity" - [149, 8] - <18, 1>
(07)                0016407>>"acetyltransferase activity" - [91, 9] - <6, 1>
(08)                  0008080>>"N-acetyltransferase activity" - [67, 9] - <27, 2>
(09)                    0004468>>"lysine N-acetyltransferase activity" - [32, 0] - <1, 1>
(10)                      0004402>>"histone acetyltransferase activity" - [32, 21] - <4, 1>
(07)                0016410>>"N-acyltransferase activity" - [73, 3] - <12, 1>
(08)                  0008080>>"N-acetyltransferase activity" - [67, 9] - <27, 2>
(09)                    0004468>>"lysine N-acetyltransferase activity" - [32, 0] - <1, 1>
(10)                      0004402>>"histone acetyltransferase activity" - [32, 21]   <4, 1>
(07)                0008374>>"O-acyltransferase activity" - [36, 1] - <33, 1>
(04)          0016765>>"transferase activity, transferring alk..." - [88, 1] - <46, 1>
(05)            0004364>>"glutathione transferase activity" - [36, 36] - <0, 1>
(05)            0004659>>"prenyltransferase activity" - [27, 3] - <9, 1>
(04)          0016757>>"transferase activity, transferring gly..." - [287, 3] - <7, 1>
(05)            0008373>>"sialyltransferase activity" - [36, 15] - <11, 1>
(05)            0016758>>"transferase activity, transferring h..." - [220, 2] - <39, 1>
(06)              0008376>>"acetylgalactosaminyltransferase activity" - [37, 3] - <9, 1>
(07)                0004653>>"polypeptide N-acetylgalactosaminylt..." - [27, 27] - <0, 1>
(06)              0008375>>"acetylglucosaminyltransferase activity" - [26, 12] - <29, 1>
(06)              0008417>>"fucosyltransferase activity" - [22, 15] - <5, 1>
(06)              0008378>>"galactosyltransferase activity" - [41, 12] - <33, 1>
(06)              0000030>>"mannosyltransferase activity" - [39, 9] - <14, 1>
(05)            0016763>>"transferase activity, transferring pe..." - [30, 2] - <38, 1>
(04)          0016769>>"transferase activity, transferring nitr..." - [24, 0] - <4, 1>
(05)            0008483>>"transaminase activity" - [23, 2] - <72, 1>
(04)          0016741>>"transferase activity, transferring one..." - [141, 0] - <3, 1>
(05)            0008168>>"methyltransferase activity" - [135, 13] - <20, 1>
(06)              0008170>>"N-methyltransferase activity" - [47, 3] - <16, 1>
(07)                0016278>>"lysine N-methyltransferase activity" - [28, 0] - <1, 2>
(08)                  0016279>>"protein-lysine N-methyltransferase ..." - [28, 1] - <3, 2>
(09)                    0018024>>"histone-lysine N-methyltransferase..." - [26, 4] - <5, 2>
(06)              0008276>>"protein methyltransferase activity" - [60, 13] - <6, 1>
(07)                0042054>>"histone methyltransferase activity" - [38, 13] - <1, 1>
(08)                  0018024>>"histone-lysine N-methyltransferase ..." - [26, 4] - <5, 2>
(07)                0016279>>"protein-lysine N-methyltransferase a..." - [28, 1] - <3, 2>
(08)                  0018024>>"histone-lysine N-methyltransferase ..." - [26, 4] - <5, 2>
(06)              0008173>>"RNA methyltransferase activity" - [21, 1] - <3, 1>
(06)              0008757>>"S-adenosylmethionine-dependent meth..." - [79, 2] - <111, 1>
(07)                0016278>>"lysine N-methyltransferase activity" - [28, 0] - <1, 2>
(08)                  0016279>>"protein-lysine N-methyltransferase ..." - [28, 1] - <3, 2>
(09)                    0018024>>"histone-lysine N-methyltransferase..." - [26, 4] - <5, 2>
(04)          0016772>>"transferase activity, transferring p..." - [1115, 2] - <11, 1>
(05)            0019205>>"nucleobase, nucleoside, nucleotide kin..." - [45, 0] - <5, 2>
(06)              0019206>>"nucleoside kinase activity" - [20, 0] - <4, 1>
(06)              0019201>>"nucleotide kinase activity" - [20, 2] - <5, 1>
(05)            0016779>>"nucleotidyltransferase activity" - [114, 1] - <58, 1>
(06)              0003887>>"DNA-directed DNA polymerase activity" - [34, 11] - <16, 1>
(06)              0003899>>"DNA-directed RNA polymerase activity" - [31, 28] - <1, 1>
(05)            0016773>>"phosphotransferase activity, alcoho..." - [924, 0] - <117, 1>
(06)              0004428>>"inositol or phosphatidylinositol kina..." - [46, 8] - <5, 2>
(06)              0004672>>"protein kinase activity" - [781, 232] - <6, 2>
(07)                0004674>>"protein serine/threonine kinase ..." - [470, 213] - <19, 1>
(08)                  0004683>>"calmodulin regulated protein kinase..." - [68, 0] - <7, 1>
(09)                    0004685>>"calcium- and calmodulin-dependent..." - [53, 53] - <0, 1>
(09)                    0004687>>"myosin-light-chain kinase activity" - [39, 39] - <0, 1>
(08)                  0004680>>"casein kinase activity" - [26, 10] - <2, 1>
(08)                  0004693>>"cyclin-dependent protein kinase ac..." - [28, 28] - <0, 1>
(08)                  0004697>>"protein kinase C activity" - [22, 16] - <4, 2>
(08)                  0004702>>"receptor signaling protein serine/t..." - [69, 4] - <7, 2>
(09)                    0016909>>"SAP kinase activity" - [20, 0] - <5, 1>
(07)                0004713>>"protein-tyrosine kinase activity" - [134, 53] - <3, 1>
(08)                  0004714>>"transmembrane receptor protein ty..." - [69, 38] - <11, 2>
(07)                0019199>>"transmembrane receptor protein kinas..." - [86, 0] - <3, 2>
(08)                  0004714>>"transmembrane receptor protein ty..." - [69, 38] - <11, 2>
(05)            0016776>>"phosphotransferase activity, phosphat..." - [25, 0] - <20, 1>
(04)          0016782>>"transferase activity, transferring sulf..." - [69, 0] - <4, 1>
(05)            0008146>>"sulfotransferase activity" - [66, 33] - <37, 1>
(02)      0030234>>"enzyme regulator activity" - [625, 14] - <14, 1>
(03)        0008047>>"enzyme activator activity" - [195, 42] - <17, 1>
(04)          0005096>>"GTPase activator activity" - [100, 46] - <9, 2>
(05)            0005100>>"Rho GTPase activator activity" - [26, 26] - <0, 2>
(03)        0004857>>"enzyme inhibitor activity" - [214, 28] - <17, 1>
(04)          0019210>>"kinase inhibitor activity" - [49, 3] - <1, 2>
(05)            0004860>>"protein kinase inhibitor activity" - [46, 14] - <6, 2>
(06)              0004861>>"cyclin-dependent protein kinase inhi..." - [28, 28] - <0, 1>
(04)          0030414>>"protease inhibitor activity" - [92, 1] - <3, 1>
(05)            0004866>>"endopeptidase inhibitor activity" - [91, 32] - <5, 1>
(06)              0004867>>"serine-type endopeptidase inhibitor ..." - [45, 41] - <3, 1>
(03)        0030695>>"GTPase regulator activity" - [234, 0] - <3, 1>
(04)          0005096>>"GTPase activator activity" - [100, 46] - <9, 2>
(05)            0005100>>"Rho GTPase activator activity" - [26, 26] - <0, 2>
(04)          0005083>>"small GTPase regulatory/interacting ..." - [187, 13] - <18, 1>
(05)            0005085>>"guanyl-nucleotide exchange factor ac..." - [88, 43] - <10, 1>
(06)              0005089>>"Rho guanyl-nucleotide exchange facto..." - [24, 24] - <0, 1>
(05)            0005100>>"Rho GTPase activator activity" - [26, 26] - <0, 2>
(03)        0019207>>"kinase regulator activity" - [110, 2] - <4, 1>
(04)          0019210>>"kinase inhibitor activity" - [49, 3] - <1, 2>
(05)            0004860>>"protein kinase inhibitor activity" - [46, 14] - <6, 2>
(06)              0004861>>"cyclin-dependent protein kinase inhi..." - [28, 28] - <0, 1>
(04)          0019887>>"protein kinase regulator activity" - [101, 2] - <9, 1>
(05)            0016538>>"cyclin-dependent protein kinase regul..." - [21, 21] - <0, 1>
(05)            0004860>>"protein kinase inhibitor activity" - [46, 14] - <6, 2>
(06)              0004861>>"cyclin-dependent protein kinase inhi..." - [28, 28] - <0, 1>
(03)        0019208>>"phosphatase regulator activity" - [47, 1] - <1, 1>
(04)          0019888>>"protein phosphatase regulator activity" - [46, 1] - <9, 1>
(02)      0003774>>"motor activity" - [60, 31] - <2, 1>
(03)        0003777>>"microtubule motor activity" - [24, 24] - <2, 1>
(02)      0004871>>"signal transducer activity" - [2195, 219] - <10, 1>
(03)        0004872>>"receptor activity" - [1262, 208] - <24, 1>
```

```
(04)         0004879>>"ligand-dependent nuclear receptor acti..." - [68, 39] - <5, 1>
(05)           0003707>>"steroid hormone receptor activity" - [20, 13] - <5, 1>
(04)         0030594>>"neurotransmitter receptor activity" - [112, 2] - <4, 2>
(05)           0015464>>"acetylcholine receptor activity" - [28, 15] - <2, 3>
(05)           0016917>>"GABA receptor activity" - [31, 1] - <2, 2>
(06)             0004890>>"GABA-A receptor activity" - [27, 27] - <0, 1>
(05)           0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
(04)         0008329>>"pattern recognition receptor activity" - [21, 11] - <4, 1>
(04)         0001653>>"peptide receptor activity" - [177, 12] - <2, 2>
(05)           0008528>>"peptide receptor activity, G-protein..." - [166, 3] - <21, 2>
(06)             0001637>>"G-protein chemoattractant receptor ac..." - [41, 0] - <2, 1>
(07)               0004950>>"chemokine receptor activity" - [41, 21] - <3, 2>
(06)             0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
(06)             0004985>>"opioid receptor activity" - [26, 3] - <4, 1>
(04)         0001565>>"phorbol ester receptor activity" - [22, 0] - <2, 1>
(05)           0004697>>"protein kinase C activity" - [22, 16] - <4, 2>
(04)         0004888>>"transmembrane receptor activity" - [945, 76] - <34, 1>
(05)           0015464>>"acetylcholine receptor activity" - [28, 15] - <2, 3>
(05)           0004930>>"G-protein coupled receptor activity" - [481, 112] - <8, 1>
(06)             0001584>>"rhodopsin-like receptor activity" - [346, 0] - <18, 1>
(07)               0008227>>"amine receptor activity" - [68, 0] - <9, 1>
(08)                 0004935>>"adrenoceptor activity" - [20, 0] - <2, 1>
(07)               0008528>>"peptide receptor activity, G-prote..." - [166, 3] - <21, 2>
(08)                 0001637>>"G-protein chemoattractant receptor ..." - [41, 0] - <2, 1>
(09)                   0004950>>"chemokine receptor activity" - [41, 21] - <3, 2>
(08)                 0008188>>"neuropeptide receptor activity" - [46, 15] - <9, 3>
(08)                 0004985>>"opioid receptor activity" - [26, 3] - <4, 1>
(07)               0008527>>"taste receptor activity" - [27, 27] - <0, 1>
(06)             0001633>>"secretin-like receptor activity" - [23, 0] - <12, 1>
(05)           0016917>>"GABA receptor activity" - [31, 1] - <2, 2>
(06)             0004890>>"GABA-A receptor activity" - [27, 27] - <0, 1>
(05)           0008066>>"glutamate receptor activity" - [53, 9] - <2, 1>
(06)             0004970>>"ionotropic glutamate receptor activity" - [32, 4] - <3, 1>
(05)           0004896>>"hematopoietin/interferon-class (D200-..." - [58, 1] - <10, 1>
(06)             0004907>>"interleukin receptor activity" - [32, 7] - <27, 2>
(05)           0045012>>"MHC class II receptor activity" - [43, 43] - <0, 1>
(05)           0019199>>"transmembrane receptor protein kinase ..." - [86, 0] - <3, 2>
(06)             0004714>>"transmembrane receptor protein tyro..." - [69, 38] - <11, 2>
(03)       0005102>>"receptor binding" - [591, 147] - <53, 2>
(04)         0005125>>"cytokine activity" - [173, 55] - <27, 1>
(05)           0008009>>"chemokine activity" - [54, 54] - <0, 2>
(05)           0005126>>"hematopoietin/interferon-class (D200-..." - [37, 7] - <25, 1>
(04)         0001664>>"G-protein-coupled receptor binding" - [65, 9] - <5, 1>
(05)           0042379>>"chemokine receptor binding" - [54, 0] - <3, 1>
(06)             0008009>>"chemokine activity" - [54, 54] - <0, 2>
(04)         0008083>>"growth factor activity" - [86, 65] - <16, 1>
(04)         0005179>>"hormone activity" - [71, 44] - <13, 1>
(05)           0005184>>"neuropeptide hormone activity" - [21, 21] - <5, 1>
(04)         0005178>>"integrin binding" - [45, 45] - <0, 1>
(03)       0005057>>"receptor signaling protein activity" - [237, 44] - <15, 1>
(04)         0004702>>"receptor signaling protein serine/threo..." - [69, 4] - <7, 2>
(05)           0016909>>"SAP kinase activity" - [20, 0] - <5, 1>
(04)         0005066>>"transmembrane receptor protein tyrosine..." - [73, 9] - <2, 1>
(05)           0005069>>"transmembrane receptor protein tyrosin..." - [60, 5] - <1, 1>
(06)             0005070>>"SH3/SH2 adaptor protein activity" - [55, 55] - <0, 1>
(02)     0005198>>"structural molecule activity" - [435, 112] - <16, 1>
(03)       0005201>>"extracellular matrix structural constit..." - [42, 40] - <4, 1>
(03)       0005200>>"structural constituent of cytoskeleton" - [110, 110] - <0, 1>
(03)       0008307>>"structural constituent of muscle" - [37, 37] - <0, 1>
(03)       0003735>>"structural constituent of ribosome" - [107, 107] - <0, 1>
(02)     0030528>>"transcription regulator activity" - [1546, 35] - <14, 1>
(03)       0003702>>"RNA polymerase II transcription facto..." - [361, 197] - <4, 1>
(04)         0016251>>"general RNA polymerase II transcriptio..." - [58, 28] - <1, 1>
(05)           0016455>>"RNA polymerase II transcription media..." - [30, 30] - <0, 1>
(04)         0003704>>"specific RNA polymerase II transcripti..." - [93, 85] - <1, 1>
(03)       0003712>>"transcription cofactor activity" - [359, 46] - <2, 2>
(04)         0003713>>"transcription coactivator activity" - [180, 172] - <2, 1>
(04)         0003714>>"transcription corepressor activity" - [149, 149] - <0, 1>
(03)       0003700>>"transcription factor activity" - [694, 680] - <1, 2>
(03)       0016563>>"transcriptional activator activity" - [150, 142] - <1, 1>
(03)       0003711>>"transcriptional elongation regulator act..." - [33, 8] - <5, 1>
(04)         0016944>>"Pol II transcription elongation factor..." - [20, 20] - <0, 1>
(03)       0016564>>"transcriptional repressor activity" - [154, 121] - <2, 1>
(04)         0016566>>"specific transcriptional repressor act..." - [33, 33] - <0, 1>
(02)     0045182>>"translation regulator activity" - [73, 10] - <3, 1>
(03)       0008135>>"translation factor activity, nucleic ac..." - [61, 25] - <5, 2>
(04)         0003743>>"translation initiation factor activity" - [30, 30] - <0, 1>
(02)     0005215>>"transporter activity" - [1512, 99] - <29, 1>
(03)       0005275>>"amine transporter activity" - [124, 3] - <17, 1>
(04)         0015171>>"amino acid transporter activity" - [89, 16] - <10, 2>
(05)           0015179>>"L-amino acid transporter activity" - [45, 0] - <22, 1>
(05)           0015175>>"neutral amino acid transporter activity" - [27, 16] - <11, 1>
(04)         0015203>>"polyamine transporter activity" - [22, 0] - <6, 1>
(03)       0015457>>"auxiliary transport protein activity" - [53, 2] - <5, 1>
(04)         0016247>>"channel regulator activity" - [51, 1] - <5, 1>
(05)           0005246>>"calcium channel regulator activity" - [24, 24] - <1, 1>
(03)       0015144>>"carbohydrate transporter activity" - [51, 0] - <10, 1>
(04)         0015145>>"monosaccharide transporter activity" - [36, 0] - <3, 1>
(05)           0015149>>"hexose transporter activity" - [36, 0] - <15, 1>
(06)             0005355>>"glucose transporter activity" - [34, 31] - <5, 1>
(03)       0005386>>"carrier activity" - [321, 5] - <6, 1>
(04)         0015290>>"electrochemical potential-driven trans..." - [155, 1] - <2, 1>
(05)           0015291>>"porter activity" - [154, 0] - <40, 1>
(06)             0015297>>"antiporter activity" - [43, 0] - <13, 1>
(07)               0015491>>"cation:cation antiporter activity" - [23, 2] - <3, 1>
(06)             0015293>>"symporter activity" - [53, 1] - <7, 1>
(07)               0015294>>"solute:cation symporter activity" - [39, 1] - <8, 1>
(08)                 0015370>>"solute:sodium symporter activity" - [27, 0] - <13, 1>
(04)         0009055>>"electron carrier activity" - [23, 23] - <2, 2>
(04)         0015399>>"primary active transporter activity" - [137, 0] - <9, 1>
(05)           0004129>>"cytochrome-c oxidase activity" - [25, 25] - <0, 4>
(05)           0008137>>"NADH dehydrogenase (ubiquinone) activity" - [37, 37] - <0, 5>
(05)           0015405>>"P-P-bond-hydrolysis-driven transporte..." - [63, 0] - <11, 1>
(06)             0015662>>"ATPase activity, coupled to transmem..." - [55, 3] - <13, 3>
(03)       0015267>>"channel or pore class transporter activity" - [383, 9] - <2, 1>
(04)         0015268>>"alpha-type channel activity" - [372, 0] - <7, 1>
(05)           0005243>>"gap junction forming channel activity" - [29, 10] - <2, 1>
```

```
{05}          0005216>>"ion channel activity" - [318, 15] - <8, 1>
{06}            0005253>>"anion channel activity" - [57, 2] - <4, 1>
{07}              0005254>>"chloride channel activity" - [50, 22] - <8, 1>
{06}            0005261>>"cation channel activity" - [241, 17] - <11, 1>
{07}              0005262>>"calcium channel activity" - [89, 28] - <4, 1>
{08}                0005245>>"voltage-gated calcium channel acti..." - [47, 40] - <5, 2>
{07}              0005267>>"potassium channel activity" - [72, 25] - <3, 1>
{08}                0005249>>"voltage-gated potassium channel ac..." - [46, 17] - <4, 2>
{07}              0005272>>"sodium channel activity" - [28, 4] - <2, 1>
{06}            0015276>>"ligand-gated ion channel activity" - [102, 7] - <7, 1>
{07}              0005230>>"extracellular ligand-gated ion chann..." - [52, 1] - <2, 1>
{08}                0005231>>"excitatory extracellular ligand-gat..." - [48, 0] - <5, 1>
{06}            0005244>>"voltage-gated ion channel activity" - [115, 3] - <5, 1>
{07}              0005245>>"voltage-gated calcium channel activity" - [47, 40] - <5, 2>
{07}              0005249>>"voltage-gated potassium channel act..." - [46, 17] - <4, 2>
{03}      0005489>>"electron transporter activity" - [206, 115] - <25, 1>
{04}        0009055>>"electron carrier activity" - [23, 23] - <2, 2>
{04}        0003954>>"NADH dehydrogenase activity" - [38, 1] - <1, 2>
{05}          0050136>>"NADH dehydrogenase (quinone) activity" - [37, 0] - <1, 1>
{06}            0008137>>"NADH dehydrogenase (ubiquinone) acti..." - [37, 37] - <0, 5>
{03}      0005478>>"intracellular transporter activity" - [36, 10] - <4, 1>
{03}      0015075>>"ion transporter activity" - [367, 4] - <4, 1>
{04}        0008509>>"anion transporter activity" - [106, 5] - <3, 1>
{05}          0015103>>"inorganic anion transporter activity" - [42, 0] - <18, 1>
{05}          0008514>>"organic anion transporter activity" - [69, 20] - <11, 1>
{06}            0008028>>"monocarboxylic acid transporter acti..." - [37, 7] - <18, 2>
{04}        0008324>>"cation transporter activity" - [264, 1] - <6, 1>
{05}          0015662>>"ATPase activity, coupled to transmemb..." - [55, 3] - <13, 3>
{05}          0015082>>"di-, tri-valent inorganic cation tran..." - [81, 2] - <13, 1>
{06}            0015085>>"calcium ion transporter activity" - [20, 4] - <1, 2>
{06}            0005385>>"zinc ion transporter activity" - [22, 19] - <5, 2>
{05}          0015077>>"monovalent inorganic cation transport..." - [137, 2] - <4, 1>
{06}            0015078>>"hydrogen ion transporter activity" - [125, 3] - <10, 1>
{07}              0004129>>"cytochrome-c oxidase activity" - [25, 25] - <0, 4>
{07}              0008137>>"NADH dehydrogenase (ubiquinone) act..." - [37, 37] - <0, 5>
{06}            0015081>>"sodium ion transporter activity" - [43, 3] - <5, 2>
{07}              0008137>>"NADH dehydrogenase (ubiquinone) act..." - [37, 37] - <0, 5>
{05}          0015101>>"organic cation transporter activity" - [23, 12] - <3, 1>
{04}        0046873>>"metal ion transporter activity" - [128, 0] - <10, 1>
{05}          0015085>>"calcium ion transporter activity" - [20, 4] - <1, 2>
{05}          0015081>>"sodium ion transporter activity" - [43, 3] - <5, 2>
{06}            0008137>>"NADH dehydrogenase (ubiquinone) acti..." - [37, 37] - <0, 5>
{05}          0046915>>"transition metal ion transporter acti..." - [57, 0] - <10, 1>
{06}            0005385>>"zinc ion transporter activity" - [22, 19] - <5, 2>
{03}      0005319>>"lipid transporter activity" - [53, 25] - <6, 1>
{04}        0005548>>"phospholipid transporter activity" - [20, 1] - <4, 1>
{03}      0015932>>"nucleobase, nucleoside, nucleotide and n..." - [27, 0] - <5, 1>
{03}      0005342>>"organic acid transporter activity" - [139, 3] - <4, 1>
{04}        0046943>>"carboxylic acid transporter activity" - [136, 1] - <4, 1>
{05}          0015171>>"amino acid transporter activity" - [89, 16] - <10, 2>
{06}            0015179>>"L-amino acid transporter activity" - [45, 0] - <22, 1>
{06}            0015175>>"neutral amino acid transporter acti..." - [27, 16] - <11, 1>
{05}          0008028>>"monocarboxylic acid transporter activity" - [37, 7] - <18, 2>
{03}      0005344>>"oxygen transporter activity" - [20, 20] - <0, 1>
{03}      0015197>>"peptide transporter activity" - [27, 8] - <7, 1>
{03}      0008565>>"protein transporter activity" - [66, 59] - <13, 1>
{03}      0015223>>"vitamin or cofactor transporter activity" - [27, 1] - <15, 1>
```