A Tutorial on Hierarchical Classification with Applications in Bioinformatics

Alex A. Freitas
Computing Laboratory
University of Kent
Canterbury, CT2 7NF, UK
A.A.Freitas@kent.ac.uk
http://www.cs.kent.ac.uk/~aaf

André C. P. L. F. de Carvalho
Department of Computer Science
University of São Paulo
São Carlos, SP, Brazil
andre@icmc.usp.br
http://www.icmc.usp.br/~andre

ABSTRACT

In Machine Learning and Data Mining, most of the works in classification problems deal with flat classification, where each instance is classified in one of a set of possible classes and there is no hierarchical relationship between the classes. There are, however, more complex classification problems where the classes to be predicted are hierarchically related. This chapter presents a tutorial on the hierarchical classification techniques found in the literature. We also discuss how hierarchical classification techniques have been applied to the area of Bioinformatics (particularly the prediction of protein function), where hierarchical classification problems are often found.

INTRODUCTION

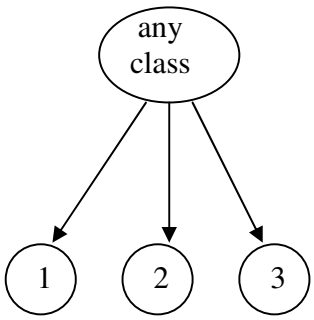
Classification is one of the most important problems in Machine Learning (ML) and Data Mining (DM). In general, a classification problem can be formally defined as:

> *Given a set of training examples composed of pairs $\{x_i, y_i\}$, find a function $f(x)$ that maps each x_i to its associated class y_i, $i = 1, 2, \ldots, n$, where n is the total number of training examples.*

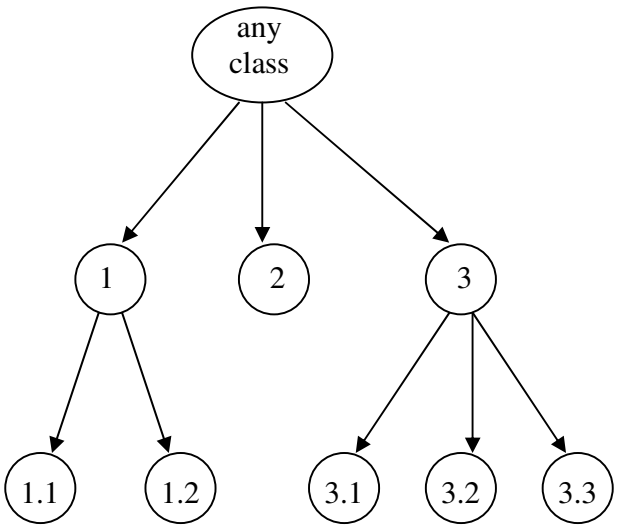
After training, the predictive accuracy of the classification function induced is evaluated by using it to classify a set of unlabeled examples, unseen during training. This

evaluation measures the generalization ability (predictive accuracy) of the classification function induced.

The vast majority of classification problems addressed in the literature involves flat classification, where each example is assigned to a class out of a finite (and usually small) set of classes. By contrast, in hierarchical classification problems, the classes are disposed in a hierarchical structure, such as a tree or a Directed Acyclic Graph (DAG). In these structures, the nodes represent classes. Figure 1 illustrates the difference between flat and hierarchical classification problems. To keep the example simple, Figure 1(b) shows a tree-structured class hierarchy. The more complex case of DAG-structured class hierarchies will be discussed later. In Figure 1, each node – except the root nodes – is labeled with the number of a class. In Figure 1(b), class 1 is divided into two sub-classes, 1.1 and 1.2, and class 3 is divided into three sub-classes. The root nodes are labeled "any class", to denote the case where the class of an example is unknown. Figure 1 clearly shows that flat classification problems are actually a particular case of hierarchical classification problems where there is a single level of classes, i.e., where no class is divided into sub-classes.



(a) Flat classification



(b) Hierarchical, tree-structured classification

Figure 1: An example of flat vs. hierarchical classification

In the flat classification problem of Figure 1(a), there is a single level of classes to be assigned to an example, but the class hierarchy of Figure 1(b) offers us more flexibility to specify at which level of the hierarchy a class will be assigned to an example.

For instance, one could require that an example should be assigned to a leaf, most specific, class. In the case of Figure 1(b), this means that the candidate classes to be assigned to this example are 1.1, 1.2, 2, 3.1, 3.2 and 3.3. At first glance, by defining that only leaf classes can be assigned to an example, we are implicitly transforming the hierarchical classification problem into a flat one, since we could use a flat classification algorithm to solve it. Note, however, that in this case the flat classification algorithm would ignore valuable information in the structure of the class hierarchy. For instance, the fact that class 1.1 is more similar to class 1.2 than to class 3.1. By contrast, a truly hierarchical classification algorithm will take into account the structure of the class hierarchy. Even if we require the hierarchical algorithm to perform class assignments at the leaf level, the algorithm exploits the structure of the class hierarchy to look for a more accurate classification function.

On the other hand, we could be more flexible and allow the hierarchical classification algorithm to classify an example at any appropriate level, depending on the predictive power of the available data. For instance, an example could be reliably classified as having the specific class 1.1, whilst, perhaps, another example could only be reliably classified as having the general class 1.

Note that, in general, class assignments at internal nodes can be carried out in a more reliable manner than class assignments at leaf nodes, because discriminating between the most specific classes at leaf nodes is more difficult than discriminating between the more general classes at internal nodes and, as a related factor, the number of examples per leaf node tends to be considerably smaller than the number of examples per internal node. On the other hand, class assignments at the leaf node tend to be more useful than class assignments at internal nodes, because the former provides more information about the class of an example. This trade-off between the reliability of a class assignment and its usefulness is common in hierarchical classification problems.

This chapter also presents applications of concepts and methods of hierarchical classification to problems in Bioinformatics, at present one of the most important groups

of DM applications. The application of DM techniques to Bioinformatics is a very active research area, and it is not feasible to address the entire area in a single book chapter. Hence, this chapter focuses on a particular kind of Bioinformatics problem, namely the prediction of protein function.

Proteins are large molecules that execute nearly all of the functions of a cell in a living organism (Alberts et al., 2002). They consist essentially of long sequences of amino acids, which fold into structures that usually minimize energy. Proteins can fold into a large number of structures, where different structures are usually associated with different functions. Due to the progress in genome sequencing technology, the number of proteins with known sequence has grown exponentially in the last few years. Unfortunately, the number of proteins with known structure and function has grown at a substantially lower rate. The reason for this is that, in general, determining the structure of a protein is much more difficult, time-consuming and expensive than determining its sequence. The main database with information about protein structures is the Protein Data Bank (PDB), which contains information related to experimentally-determined structures. The doubling time for the number of experimentally-determined protein structures available in the PDB has been recently calculated as 3.31 years, which, although impressive, represents a considerably slower growth than the doubling time for the number of sequences in the GenBank, which is just 1.4 years (Higgs & Attwood, 2005).

There are several motivations to investigate the application of concepts and methods of hierarchical classification to the prediction of protein function. First, proteins have a large diversity of functions, which can be categorized in many different ways. This naturally gives rise to hierarchical classification problems where the classes to be predicted are arranged in a tree-like or a DAG-like structure.

Furthermore, bearing in mind that the ultimate goal of DM is to discover useful knowledge, the prediction of protein function can potentially lead to better treatment and diagnosis of diseases, design of more effective medical drugs, etc., constituting an important and much needed application of DM in the context of Intelligent Systems for human health improvement. An additional, related, motivation is that the contents of all the major protein databases are freely available on the web. Thus, DM researchers do not

have to face the problems of data privacy and restricted data access found in commercially-oriented DM applications.

Another motivation to focus this chapter on hierarchical classification applied to the prediction of protein function is that this is still a relatively new, under-explored research area, where there are many opportunities for further research and improvement of current practice. For instance, the vast majority of works on hierarchical classification for the prediction of protein function are still using a conventional measure of predictive accuracy for flat classification problems. In principle, it would be more effective to use a measure of predictive accuracy tailored for hierarchical classification problems, as will be discussed later.

It should be noted that, recently, there has been an extensive research on hierarchical classification, but the majority of that research has been oriented to Text Mining, rather than Bioinformatics – see e.g. the survey of (Sun et al., 2003b). On the other hand, a large amount of work has been lately published reporting the use of DM techniques in Bioinformatics. However, the majority of this research focuses on the analysis of gene expression data (Slonim et al, 2000), (Jiang et al., 2004) which is very different from the problem of predicting protein function addressed in this chapter. Given the aforementioned exponential growth of protein sequence data available in biological databases and the important role of effective protein function predictions in improving understanding, diagnosis and treatment of diseases, it is timely to focus on the problem of protein function prediction.

To summarize, this chapter has two main contributions. First, it presents a comprehensive tutorial on hierarchical classification, discussing the main approaches and different techniques developed for solving these problems. Second, it discusses how concepts and methods from hierarchical classification have been applied to a very challenging and important DM problem in Bioinformatics, namely the prediction of protein function.

The remainder of this chapter is organized as follows. The second section presents a gentle introduction to the problem of protein function prediction for data miners who are not familiar with molecular biology. The third section describes several approaches for categorizing hierarchical classification problems. The fourth section covers different

hierarchical classification methods, including methods based on transforming a hierarchical classification problem into one or more flat classification problems. The fifth section reviews several works on hierarchical classification applied to the prediction of protein function. Finally, the sixth section presents the conclusions and future research directions.

AN OVERVIEW OF PROTEIN FUNCTION PREDICTION

Proteins are large molecules consisting of long sequences (or chains) of amino acids, also called polypeptide chains, which fold into a number of different structures and perform nearly all of the functions of a cell in a living organism.

We can distinguish four levels of organization in the structure of a protein (Alberts et al., 2002). The primary sequence of a protein consists of its linear sequence of amino acids. The secondary structure consists of α helices (helical structures formed by a subsequence of amino acids) and β sheets (subsequences of amino acids folded to run approximately side by side with one another). The tertiary structure consists of the three-dimensional organization of the protein. Some proteins also have a quaternary structure, a term used to refer to the complete structure of protein molecules formed as a complex of more than one polypeptide chains. These four levels of protein structure are illustrated in Figure 2, adapted from (Lehninger et al., 1998). The leftmost part of the figure shows part of a sequence of amino acids – each one with its name abbreviated by three letters. This part of the sequence forms an α helix – an element of secondary structure, as shown in the second part of the figure. This α helix is part of the larger tertiary structure, as shown in the third part of the figure. Finally, the rightmost part of the figure shows two polypeptide chains assembled into a larger quaternary structure.

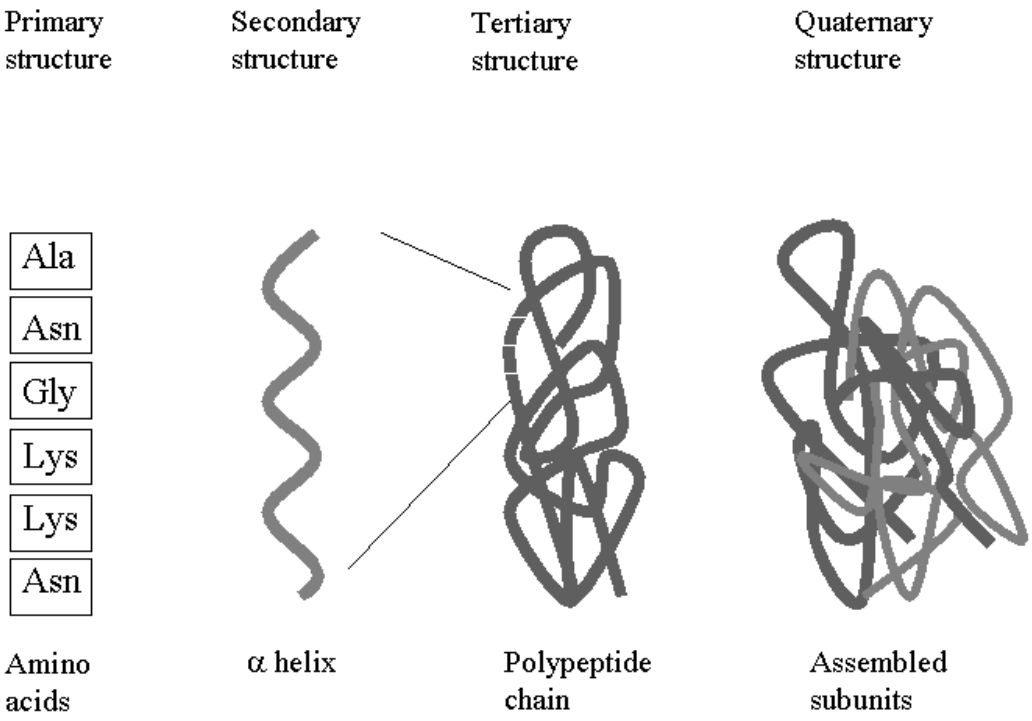


Figure 2: Four levels of structure in a protein

In addition to these four levels of organization, there is a unit of organization named protein domain, which seems particularly important for the prediction of protein functions. A protein domain consists of a substructure produced by some part of a polypeptide chain that can fold, independently from other parts of the chain, into a compact, stable structure. Therefore, protein domains can be regarded as "higher-level building blocks" from which proteins are built. The term "higher-level" has been used to distinguish protein domains from the lower-level, fundamental building blocks of proteins, namely the amino acids composing its primary sequence. To see the importance of protein domains, it has been argued that protein domains, rather than genes, are the fundamental units of evolution (Nagl, 2003). The presence of one or more protein domains in a given polypeptide chain is often a useful clue to predict that protein's function, since different domains are often associated with different functions.

The most used approach to predict the function of a new protein given its primary sequence consists of performing a similarity search in a protein database. Such database contains proteins for which we know both their sequence and function. In essence, the program finds the protein whose sequence is most similar to the sequence of the new protein. If the similarity is higher than a threshold, the function of the selected protein is transferred to the new protein. This method will be hereafter referred to as *similarity-based protein function prediction*, and is illustrated in Figure 3.

The similarity between a new protein and each protein in the database is usually computed by measuring the similarity between the sequences of amino acids of the two proteins, which involves performing some kind of alignment between these two amino acid sequences (Higgs & Attwood, 2005).

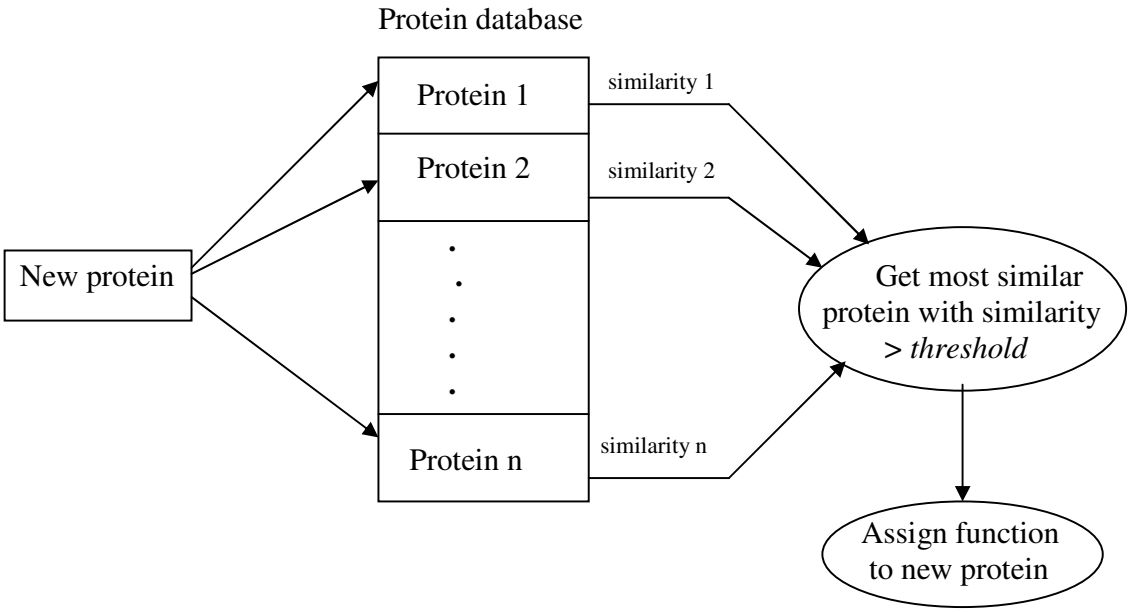


Figure 3: Overview of similarity-based protein function prediction

A very simple example of this kind of alignment is shown in Figure 4. Note that in Figure 4(b) a gap, denoted by the symbol "–", was inserted into the fifth position of the second sequence, which had the effect of sliding the subsequence "VACFW" to the right. This allows the system to detect that the two sequences have the same amino acids at their last five positions. Comparing the sequences in Figure 4(b) in a position-wise

fashion, there are only two differences between them: a difference in the amino acid at the second position and the presence of a gap at the fifth position in the bottom sequence. Intuitively, this can be considered a high similarity, much larger than the similarity of two randomly-chosen sequences of the same size, since each position could contain any of the 20 amino acids. Intuitively, this high similarity would suggest that the two sequences are homologous, in the sense that they evolved from a common ancestor (Higgs & Attwood, 2005), and the difference between them could be explained by just two mutations. First, a mutation in the second position, which either changed an A to a K or a K to an A (there is no way of knowing which was the case by just comparing the two sequences), and, second, a mutation in the fifth position, where either a P was inserted into the top sequence or a P was deleted at the bottom sequence (again, we do not know which was the case).

DACAPVACFW	DACAPVACFW
DKCAVACFW	DKCA–VACFW
(a) before alignment	(b) after alignment

Figure 4: Example of alignment between two amino acid sequences

From a ML and DM perspective, the similarity-based protein function prediction method can be regarded as an instance of the Instance-Based Learning (IBL) or Nearest Neighbor paradigm (Aha et al., 1991), (Aha, 1992), sometimes called “lazy learning” (Aha, 1997). The latter term is sometimes used because in this paradigm learning is delayed to the moment when a new test instance is to be classified, rather than first learning a model from the training data and then use the model to classify a new test instance. The main difference between the similarity-based protein function prediction method and the majority of conventional IBL algorithms is that, in the latter, the measure of distance (the dual of similarity) between two instances is usually simpler, involving a generic distance measure, such as the well-known Euclidean distance or another distance measure suitable for the target problem (Liao et al., 1998). More precisely, in conventional IBL algorithms, the distance measure usually is computed between two instances with the same number of attributes and the distance between the values of a

given attribute in two instances is usually computed in a straightforward manner, say by the subtraction of two numbers in the case of numeric attributes.

By contrast, when comparing two protein sequences, matters are considerably more complicated. The two sequences being compared usually have different lengths, so that they first have to be aligned, as illustrated in Figure 4. Alignments are often performed by using a dynamic programming algorithm that finds alignments with optimal scores. However, the results of the algorithm will, of course, depend on several parameters of the heuristic scoring function. One such parameter, for instance, is how much an alignment score should be penalized for each gap. Another parameter involves how to compute the similarity between two different amino acids. The naïve approach of assigning a similarity score of 1 to two identical amino acids and a similarity score of 0 to any pair of distinct amino acids is not effective, because it ignores the well-established fact that some kinds of amino acids are more similar to each other than others. In order to take this aspect into account, the computation of the similarity between two amino acid values is usually based on a matrix M, where each entry $M_{i,j}$ represents the similarity score between amino acid i and amino acid j. Such matrices are generally constructed from data involving sets of proteins believed to be homologous (because they have sequence similarity higher than a certain threshold). However, it should be noted that the construction of such matrices is a heuristic process, depending on both assumptions associated with the evolutionary model used and the data available at the time the amino acid distance matrix is constructed.

In any case, despite the complexity of measuring the similarity between the amino acid sequences of two proteins, we emphasize that the similarity-based protein function prediction method can still be considered as an instance of the IBL paradigm. In particular, it has the following core IBL properties: (a) the "training phase" of the method is very simple, consisting of storing known-function proteins in a large database; (b) in the "testing phase", the protein database is directly used as a "classification model", by assigning the new protein to the functional class(es) of the most similar protein(s) stored in the protein database, as long as the similarity is above a threshold value.

One advantage of this method – which is inherent to virtually all IBL methods – is that the simplicity of its training phase makes it naturally incremental. That is, as more and

more proteins with known function are added to the database, the training set – and so the classification model – is immediately expanded, which should in principle increase the predictive accuracy of new functional predictions.

Although this method is very useful in several cases, it also has some limitations, as follows. First, it is well-known that two proteins might have very similar sequences and perform different functions, or have very different sequences and perform the same or similar function (Syed & Yona, 2003), (Gerlt & Babbitt, 2000). Second, the proteins being compared may be similar in regions of the sequence that are not determinants of their function (Schug et al., 2002). Third, the prediction of function is based only on sequence similarity, ignoring many relevant biochemical properties of proteins (Karwath & King, 2002), (Syed & Yona, 2003).

In order to mitigate these problems, another approach consists of inducing from protein data a classification model, so that new proteins can be classified by the model. In this approach, instead of computing similarities between pairs of sequences, each protein is represented by a set of attributes. The learning algorithm induces a model that captures the most relevant relationships between the attributes and the functional classes in the training dataset. As long as the examples in the training dataset being mined have the same number of attributes, as usual in most ML and DM classification problems, this approach opens up the opportunity to use a large variety of classification algorithms for the prediction of protein function.

Additionally, if the induced classification model is expressed in a comprehensible representation – say, as a set of rules, a decision tree or perhaps a Bayesian network – it can also be shown to a biologist, to give her/him new insights regarding relationships between the sequence, the biochemical properties and the function of proteins; and possibly suggest new biological experiments. The induced model approach also has the advantage that it can predict the function of a new protein even in the absence of a sequence similarity measure between that protein and other proteins with known function (King et al., 2001).

In the remainder of this chapter, we will assume the use of this framework of classification model induction, unless mentioned otherwise. In any case, it should be emphasized that the induced model-based approach aims mainly at complementing –

rather than replacing – the conventional similarity-based approach for protein function prediction. Ideally, both approaches should be used in practice.

Since the focus of this chapter is on hierarchical classes, the issue of how to create predictor attributes to represent proteins is not directly related to the focus of this chapter. However, the choice of predictor attributes is crucial to any classification task, so it is worth briefly mentioning here some predictor attributes previously used in the literature to predict protein function:

a) *Attributes directly derived from protein sequence*: for instance, composition percentage of each of the 20 amino acids and the 400 possible pairs of possible amino acids (Syed & Yona, 2003), (King et al., 2001), molecular weight, average hydrophobicity, average isoeletric point, etc;
b) *Attributes predicted from the sequence by using some computational method*: this includes predicted attributes based on the secondary structure (Syed & Yona, 2003), (Jensen et al., 2002), (Karwath & King, 2002) and post-translational modifications, e.g. glycosylation (Gupta & Brunak, 2002), (Stawiski et al., 2002);
c) *Attributes obtained from biological databases*: for instance, UniProt/SwissProt contains information about tissue specificity, organism, and domain/motifs associated with proteins (whose details can be obtained by following links to databases like PRODOM and PROSITE). As another example, predictor attributes based on protein-protein interaction data can be derived from the Database of Interacting Proteins (DIP) (Jensen et al., 2002) or other data sources (Hendlich et al., 2003), (Deng et al., 2002).

CATEGORIZING HIERARCHICAL CLASSIFICATION PROBLEMS

In order to understand hierarchical classification problems, the first step is to categorize those problems according to relevant criteria. Intuitively, this categorization should be useful in the design of a new algorithm for hierarchical classification, i.e., the algorithm should be tailored for the specific characteristics of the target hierarchical classification problem. Several relevant criteria for this categorization are described next. Note that, in this Section, we analyze the structure of hierarchical classification *problems*,

regardless of the kind of algorithm used to solve them. An analysis of the structure of hierarchical classification *algorithms* will be presented in the next section.

Categorization Based on the Structure of the Class Hierarchy

In general, there are two main types of structure for a class hierarchy, namely a *tree structure* and *a Direct Acyclic Graph (DAG) structure*. These structures are illustrated in Figures 5 and 6, respectively. In these figures, each node represents a class – identified by the number inside the node – and the edges between the nodes represent the corresponding super-class and sub-class relationships. Figures 5 and 6 show just a two-level class hierarchy, to keep the pictures simple. In both figures, the root node corresponds to "any class", denoting a total absence of knowledge about the class of an object. The main difference between the tree structure and the DAG structure is that in a tree structure each class node has at most one parent, whilst in a DAG structure each class node can have more than one parent.

In the tree structure of Figure 5, each node is labelled with the number (id) of its corresponding class. The root node is considered to be at level 0, and the level of any other node is given by the number of edges linking that node to the root node. Nodes at the first level have just one digit, whereas nodes at the second level have two digits: the first digit identifies the parent class (at the first level) and the second digit identifies the subclass at the second level, as a child of the parent class.

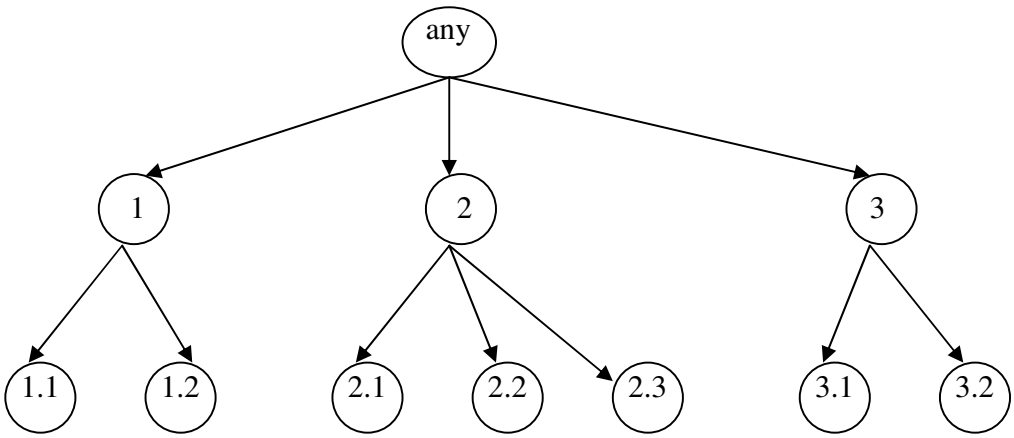


Figure 5: Example of a class hierarchy specified as a tree structure

In the context of protein functional classification, a typical example of a class hierarchy structured as a tree is the functional classification of enzymes. Enzymes are proteins specialized in catalyzing (or accelerating) chemical reactions (Alberts et al., 2002), (Dressler & Potter, 1991). Each enzyme is assigned to a class according to its EC code, which specifies the chemical reaction catalysed by enzymes. The EC code consists of a series of four digits, where the first digit specifies the most general class of the enzyme and the fourth digit the most specific one. For instance, the EC code 1.1.1.1 specifies the class Alcohol Dehydrogenase.

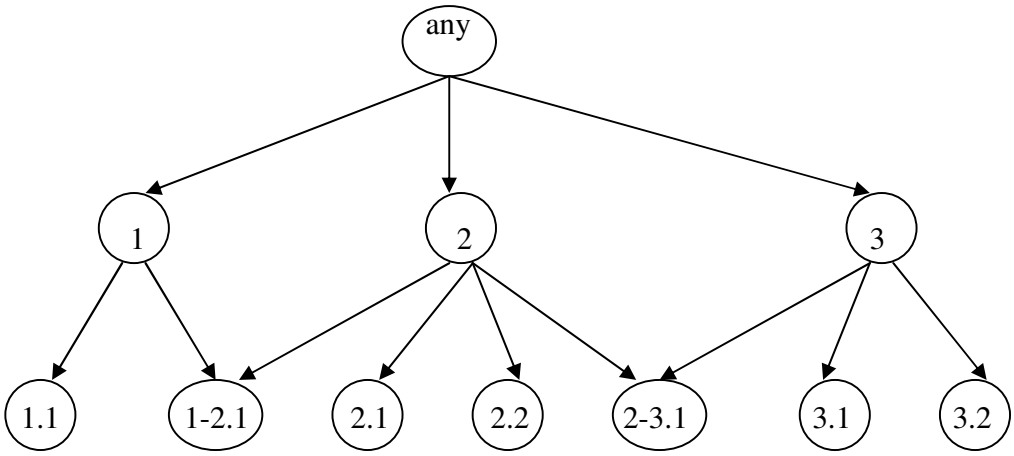


Figure 6: Example of a class hierarchy specified as a DAG (Direct Acyclic Graph)

The notation for the labels of class nodes in Figure 6 is similar to the notation in Figure 5. The main difference occurs for nodes with multiple parents, namely the nodes 1-2.1 and 2-3.1 at the second level of the class hierarchy. In this notation, the parent classes are specified by not just a single digit, but rather two digits before the class level-delimiter ".", i.e., one digit for each of the parent classes. Hence, in the class number 1-2.1, the notation "1-2" indicates that this node has parent classes 1 *and* 2, and the "1" after the "." indicates that this is the first child class of those two parent classes considered as a whole (rather than each parent class individually).

In the context of protein functional classification, a typical example of a class hierarchy structured as a DAG is the Gene Ontology (GO), a relatively recent approach for classifying gene/protein functions (GO Consortium, 2000), (Lewis, 2004), (Camon et al., 2003). GO consists of three categories of functions, namely biological process, molecular function and cellular component, which are implemented as three independent ontologies. GO has important advantages when compared with previous schemes for classifying protein functions. It specifies a well-defined, common, controlled vocabulary for describing protein functions. Hence, it improves the interoperability of genomic databases and provides a generic framework for protein functional classification. In addition, it is a *pan-organism* classification, i.e., it can potentially be applied to all organisms, contributing to the unification of Biology. The use of GO and its associated DAG structure are increasingly popular in the protein function prediction literature (Jensen et al., 2003), (King et al., 2003), (Laegreid et al., 2003), (Thomas et al., 2003). In addition, several other Bioinformatics ontologies have lately been developed based on some basic ideas from GO, particularly the use of a DAG structure to represent the class hierarchy. Two examples are the cell type ontology (Bard et al., 2005) and the mammalian phenotype ontology (Smith et al., 2004).

Categorization Based on the Hierarchical Level of Predicted Classes

Two main groups of problem can be distinguished according to this criterion. First, in some problems, all examples must be assigned to classes that are leaf nodes in the class hierarchy. We will refer to this group of problems as *mandatory leaf-node prediction* problems. Note that, in the case of a tree structure, when the system assigns a leaf class to an example, the system is also implicitly assigning to that example a unique class at each internal (non-leaf) level of the class hierarchy.

For instance, if the system predicts that the EC code of an enzyme is 2.1.3.1, the system is effectively predicting not only the enzyme's class at the fourth level, but also the enzyme's classes at the first three levels. However, this is not necessarily true in the case of a DAG structure, where a leaf node can have more than one parent.

Second, in some problems, examples can be assigned to classes that are either leaf nodes or internal nodes in the class hierarchy. We will refer to these problems as *optional*

leaf-node prediction problems. In this case, the system has autonomy to decide, depending on the predictive power of the available data, how deep in the hierarchy the predicted class should be for each example.

For instance, although some enzymes could be easily classified by the system at the level of the fourth EC code digit (leaf class), for other enzymes the system could be much less confident about a fourth-level (most specific) prediction and decide to make a prediction only at the second, or perhaps first, level of the class hierarchy.

Of course, ideally, this decision should take into account not only the confidence in the predictions at different levels, but also their relative usefulness to the user. In general, the more specific (the deeper in the class hierarchy) a prediction is, the more it tends to be useful to the user, but the more difficult the prediction tends to be.

This holds particularly true for a tree structure, where the prediction of a leaf class subsumes the prediction of its parent classes, as mentioned earlier. The difficulty of prediction usually increases with the depth of the tree because the number of examples per class node in a deep level of the classification tree tends to be much smaller than in a shallow level, which means there is less data to support the discovery of a reliable classification model.

The situation is more complex in the DAG structure, where a class node can have more than one parent. In this case, it is actually possible that a class node has more examples than each of its parent class nodes. For instance, suppose that class C is a child of the classes P_1 and P_2; class P_1 has 100 examples, 70 of which also belong to its child class C; class P_2 has 50 examples, 40 of which also belong to its child class C. In this case, class C has 110 examples, a number larger than the number of examples in each of the parent classes P_1 and P_2. Thus, other things being equal, with respect to the number of examples per class, it might be easier to predict the more specific class C than any of its more generic parent classes P_1 and P_2 individually. In practice, however, even in a DAG, it is usually the case that the predictive accuracy decreases with an increase in the depth (specificity) of the prediction. For instance, (Pal & Eisenberg, 2005) obtained "…about 85% correct assignments at ontology depth 1 and 40% at depth 9".

Before we proceed, we must make a note about the terminology used in this subsection. The proposed terms *mandatory leaf-node prediction* and *optional leaf-node*

prediction correspond to the types of class hierarchy named *virtual category tree (or DAG)* and *real category tree (or DAG)* by (Sun & Lim, 2001), (Sun et al., 2003b). In this chapter, we prefer the new terminology introduced in this subsection because it intuitively has a more direct meaning.

Categorization Based on the Predictive Performance Measure

There are several alternatives for measuring the predictive performance of a classification algorithm. A recent paper (Caruana & Niculescu-Mizil, 2004) discusses 9 different measures of predictive performance in the context of standard, flat classification problems. A discussion of conventional measures of predictive performance for flat classification is out of the scope of this chapter, since this topic is well covered in the mentioned paper and in data mining textbooks (Witten & Frank, 2005), (Tan et al., 2006). Here we are rather interested in the orthogonal problem of how to take the class hierarchy into account when measuring predictive performance. We can identify at least four different ways of measuring predictive performance in hierarchical classification. Each of these is discussed in one of the following four subsections. Related discussions on predictive performance measures for hierarchical classification can also be found in (Blockeel et al., 2002), (Sun & Lim, 2001).

Uniform misclassification costs

In this category of predictive performance measure, all misclassification costs are the same (say, unitary cost) for all possible values of the predicted class and the true class of an example – regardless of the level of the predicted and the true classes in the hierarchy.

Note that uniform misclassification costs can be straightforwardly employed both in *mandatory leaf-node prediction* problems, where all examples must be assigned to classes that are leaf nodes in the class hierarchy, and in *optional leaf-node prediction* problems, where examples can be assigned to internal or leaf class nodes. In the latter, we simply compare the class predicted for an example with the true class of the example at the same level of the predicted class, of course. For instance, if the system has assigned an enzyme to the EC code 3.1, we just have to compare this class (predicted at level 2 of

the class hierarchy) with the true class of this enzyme at the second level, which effectively ignores the enzyme's true classes at the third and fourth level.

However, the use of uniform misclassification costs in optional leaf-node prediction problems is not recommended, because it ignores the fact that the prediction of deeper-level classes tends to be more difficult than the prediction of shallower-level classes (particularly for tree-structured class hierarchies), as discussed earlier. In addition, a misclassification at a deep class level will mislead the user relatively little, by comparison with a misclassification at a shallow class level. Hence, other things being equal, misclassification costs at shallower levels of the classification tree should be larger than misclassification costs at deeper levels of the classification tree.

Note also that, even when all classes are predicted at the same level of the class hierarchy, the uniform misclassification costs approach is usually not the ideal approach for measuring predictive performance in hierarchical classification problems. This occurs because it ignores the fact that classes that are closer to each other in the hierarchy (say, "sibling" classes) are normally more similar to each other than classes that are further away to each other in the hierarchy (say, "cousin" classes) (Koller & Sahami, 1997). For instance, if the true second-level class of an enzyme is 2.1, misclassifying it as its sibling class 2.2 should have a smaller misclassification cost than misclassifying it as its cousin class 3.2. This introduces the motivation for another measure of predictive performance, discussed in the next subsection.

Distance-based misclassification costs

This category of predictive performance measure consists of assigning, to each misclassification, a cost that is proportional to the distance between the example's predicted class and the example's true class in the hierarchy. This category can be further sub-divided into two sub-categories. In the first one, the distance between two classes can be measured in the same way, regardless of the level (depth) of these two classes in the hierarchy. This will be called *depth-independent distance-based misclassification costs*. The typical approach in this sub-category consists of defining the distance between two nodes in the class hierarchy as the number of edges in the shortest path connecting them. We stress that by "path" we mean here a sequence of *undirected* edges – i.e., we do not

take into account the original direction of each of the edges in the graph for the purpose of finding paths between two nodes. For instance, the distance between classes 1.1 and 3.2 in the tree-structured hierarchy of Figure 5 is 4. In a tree, the computation of this distance is very simple. If we consider only paths where each edge is used at most once, there is exactly one path between each pair of distinct nodes in the tree.

The situation is more complex in a DAG, where there can be multiple paths (each of them without edge duplication) between a pair of nodes. For instance, there are several paths between the class nodes 1-2.1 and 2-3.1 in Figure 6. More precisely, using the notation "<n_1, n_2>" to denote an edge linking node n_1 and node n_2, we can observe the following paths between nodes 1-2.1 and 2-3.1 in Figure 6:

(a) <1-2.1, 1>, <1, any>, <any, 3>, <3, 2-3.1>;
(b) <1-2.1, 1>, <1, any>, <any, 2>, <2, 2-3.1>;
(c) <1-2.1, 2>, <2, any>, <any, 3>, <3, 2-3.1>;
(d) <1-2.1, 2>, <2, 2-3.1>.

Using the common definition of distance between two class nodes as the number of edges in the shortest path connecting these nodes, the distance in the previous example would be equal to 2, corresponding to the number of edges in the path (d).

This predictive performance measure has the advantage of simplicity, but it has the disadvantage of not taking into account that fact that, in many problems, misclassification costs at higher levels of the class hierarchy tend to be higher than misclassification costs at lower levels of the tree. For instance, suppose an example has leaf class 1.1.1.1 in a four-level hierarchy and the predicted class for this example is 1.1.1.4. The distance between the true class and the predicted class for this example is 2, since they are sibling classes separated by two edges – their common parent class is 1.1.1. Now suppose that the same example is assigned just to the internal, second-level class 1.2 – assuming an *optional leaf-node prediction* problem. Again, the distance between the true class (1.1) and the predicted class (1.2) is 2. However, intuitively, the latter misclassification should have a larger cost than the former, because an error in the second-level (fairly general) class of an enzyme is much less forgivable than an error in the fourth-level (very specific)

class of an enzyme. To overcome this deficiency, we should use another category of distance-based misclassification costs, where the cost of a misclassification depends on both the distance between the predicted and true classes and their depth in the class hierarchy. This will be called *depth-dependent distance-based misclassification costs*. If this measure is adopted, when computing the distance of a path, we can consider weighted edges, where deeper edges have smaller weights than shallower edges (Blockeel et al., 2002). The distance between two nodes can be computed as the shortest weighted path between these two nodes.

Semantics-based misclassification costs

This category of predictive performance measures is based on a determination of class similarity that is independent of the distance of the classes in the hierarchy. The measure of similarity is supposed to be based on some notion of "semantics" of the classes, which is independent of the structure of the class hierarchy. This approach has been proposed, in the context of text mining – more precisely document classification – by (Sun and Lim, 2001), (Sun et al., 2003a). In this context, each class of documents is represented by a feature vector derived by adding up the features vectors of all documents belonging to that class. Then, the similarity between two classes is computed by some measure of similarity between their corresponding vectors (the cosine measure, often used in information retrieval, was proposed by the authors).

In the context of Bioinformatics – particularly protein functional classification – the idea of a predictive performance measure based on class similarity independent from distance in the class hierarchy might be somewhat controversial. On one hand, presumably, the class hierarchy of several protein functional classifications – such as the EC code for enzymes – was constructed based on extensive biological knowledge, and its structure is already supposed to reflect some notion of the semantics of the classes, so that the distance between classes in the hierarchy should be considered when measuring predictive performance. On the other hand, it is interesting to observe that, once a measure of class similarity (independent of class distance in the hierarchy) has been precisely defined and its value computed for each pair of classes, this could, in principle, be used to construct another class hierarchy by using, say, a hierarchical clustering

algorithm. This new hierarchical classification – rather than the original one – could then be given to the DM algorithm, whose predictive performance could then be evaluated by taking into account the distances between classes in the new hierarchy. This approach seems very under-unexplored in the literature about hierarchical classification. It could potentially shed a new light into some existing biological hierarchical-classification problems.

Hierarchical misclassification cost matrix

This category of predictive performance measures consists of explicitly specifying the cost associated with each possible misclassification, by using a hierarchical misclassification cost matrix. This type of matrix is a generalization of the well-known misclassification cost matrix for standard flat classification (Witten & Frank, 2005). There are different ways of making this generalization. Let us start with a hierarchical misclassification cost matrix for *mandatory leaf-node prediction*, whose matrix structure is illustrated in Figure 7. To keep the figure simple, there are only two classes at the first level of this matrix. Each of these classes has only two subclasses at the second level. However, the matrix in Figure 7 can be straightforwardly extended to represent a larger number of class levels and larger numbers of classes per level. Note that in the simple structure of Figure 7, although the structure is hierarchical, the misclassification costs are specified only at the level of the leaf class nodes. In the main diagonal the misclassification costs are zero, as usual, since the cells along that diagonal represent correct classifications. In the other cells, the misclassification costs are labelled a,…,l, and each of these values is assumed to be larger than zero. It is trivial to look up the matrix in Figure 7 to determine a misclassification cost in the case of a mandatory leaf-node prediction problem. For instance, if the class predicted for an example is 2.1 and the true class of that example is 1.1, this wrong prediction has a misclassification cost of g.

Note that, although the basic structure of the matrix in Figure 7 is quite simple, in general, it is flexible enough to implement the previously-discussed categories of predictive performance measures as particular cases, with a proper specification of the misclassification costs a,…,l for each category. This can be shown as follows:

- To obtain *uniform misclassification costs*, all the costs $a,\ldots,l$ are set to the same value;
- To obtain *distance-based misclassification costs,* where the costs are given by the number of edges in the hierarchical classification tree – so that the costs associated with sibling class nodes are equal to 2 and the costs associated with cousin class nodes are equal to 4 – the costs should be set as follows:
  $a = d = i = l = 2$ and $b = c = e = f = g = h = j = k = 4$.
- To obtain *semantics-based misclassification costs*, the costs $a,\ldots,l$ are simply set to the corresponding values of semantic distance between each pair of classes.

			True Class			
			1		2	
			1.1	1.2	2.1	2.2
Predicted Class	1	1.1	0	*A*	*b*	*c*
		1.2	*d*	0	*e*	*f*
	2	2.1	*g*	*H*	0	*i*
		2.2	*j*	*K*	*l*	0

Figure 7: Structure of a Hierarchical Misclassification Cost Matrix for *Mandatory* Leaf-Node Prediction

Recall that the discussion so far has assumed the context of a mandatory leaf-node prediction problem. If the target problem is *optional leaf-node prediction*, the scenario becomes more complex. In this new scenario, it seems natural to extend the matrix in Figure 7 in order to represent hierarchical misclassification costs involving non-leaf classes. This leads to the more complex hierarchical misclassification cost matrix shown in Figure 8. In that matrix, the notation ".*" refers to a wild card denoting any class at the second level. For instance, the notation "1.*" for a predicted class means that class 1 was predicted at the first, non-leaf, level and no class was predicted at the second, leaf level. The misclassification costs involving at least one first-level, non-leaf class (i.e. a row or column with the wild care ".*"), are denoted by upper case letters, whilst the misclassification costs involving only second-level, leaf, classes, are denoted by lower case letters (using the same notation as the simpler matrix of Figure 7). For instance, the

cell at the intersection of the 1.* row and the 1.* column represents the misclassification cost *A* of predicting class 1 at the first level and making no prediction at the second level, when the true class at the first level is 1 and the true class at the second level is unknown. Note that intuitively this cost should be larger than zero, to take into account the lack of prediction for the second-level class. The cost zero should be reserved only for the cases where both the predicted and the true classes at the second level are known to be equal. As another example of the use of the ".*" notation, the cell at the intersection of the 2.* row and the 1.2 column specifies the misclassification cost *M* of predicting class 2 at the first level and making no prediction at the second level, when the true class is known to be 1.2.

			True Class					
			1			2		
			1.*	1.1	1.2	2.*	2.1	2.2
Predicted Class	1	1.*	*A*	*B*	*C*	*D*	*E*	*F*
		1.1	*G*	0	*a*	*H*	*b*	*c*
		1.2	*I*	*d*	0	*J*	*e*	*f*
	2	2.*	*K*	*L*	*M*	*N*	*O*	*P*
		2.1	*Q*	*g*	*h*	*R*	0	*i*
		2.2	*S*	*j*	*k*	*T*	*l*	0

Figure 8: Structure of a Hierarchical Misclassification Cost Matrix for *Optional* Leaf-Node Prediction

CATEGORIZING HIERARCHICAL CLASSIFICATION APPROACHES

Transforming a Hierarchical Classification Problem into a Flat Classification Problem

A standard flat classification problem can be regarded as a particular case (or a degenerated case) of a hierarchical classification problem where none of the classes to be predicted has super-classes or sub-classes. Hence, a simple, somewhat naïve, way to "solve" a hierarchical classification problem consists of transforming it into a flat

classification problem, and then apply one of the very many flat classification algorithms available to solve the new problem.

Consider, for instance, the tree-structured class hierarchy in Figure 5. We could transform the hierarchical classification problem associated with this figure into the problem of predicting classes only at the first level (i.e., most general classes) of the hierarchy. In this case, however, we would miss the opportunity of making more specific predictions at the second level of the hierarchy, which in principle would be more useful, providing more knowledge to the user. Alternatively, one could transform the original hierarchical classification problem into the problem of predicting only classes at the second level (i.e. the leaf classes) of the hierarchy. As mentioned earlier, by predicting classes at the leaves of the hierarchy, we are implicitly predicting the classes at higher levels (internal nodes). In this case, however, we would miss the opportunity of predicting classes at higher levels of the hierarchy, which presumably can be predicted with more confidence than classes at lower, deeper levels, as also previously discussed.

Hierarchical Class Predictions Using Flat Classification Algorithms

One way of avoiding the missed opportunities discussed in the previous subsection consists of transforming the original hierarchical classification problem into a set of flat classification problems, more precisely one flat classification problem for each level of the class hierarchy, and then use a flat classification algorithm to solve each of these problems *independently*. For instance, in the case of Figure 5, the associated hierarchical classification problem would be transformed into two problems, namely predicting the classes at the first level and predicting the classes at the second level. A flat classification algorithm would then be applied to each of these two problems independently, i.e., each of the two runs of the algorithm would ignore the result of the other run.

Note that, in this case, in principle each of the two runs of the classification algorithm would be associated with its own measure of predictive performance, since the two independent runs effectively correspond to two distinct flat classification problems. In other words, the multiple runs of the classification algorithm are independent both in the training phase and in the test phase (classification of new, previously unknown, examples).

One problem with this approach is that there is no guarantee that the classes predicted by the independent runs at different class levels will be compatible with each other. For instance, still referring to the simple hypothetical example of class hierarchy in Figure 5, it is possible, in principle, to have a situation where the classifier at level 1 assigns a test example to class 1, whilst the classifier at level 2 assigns the example to class 2.1, which is clearly incompatible with the first-level prediction.

A more sophisticated approach to hierarchical class predictions consists of having multiple runs of a classification algorithm in such a way that results from independent training runs are used together during the test phase. Thus, there is a single measure of predictive performance on the test set associated with the results of all the training runs. To summarize, in this approach the multiple runs of the classification algorithm are independent during training, but integrated or dependent during the test phase.

The most common way of implementing this approach consists of training a different classification model for each node of the class hierarchy. Typically, each trained classification model is a binary classifier that decides, for each test example, whether or not the example should be assigned the class associated with the corresponding classification node. Note that this approach can be implemented in a way that naturally allows a test example to be assigned to more than one class at any level of the hierarchy, if the example satisfies the conditions of the corresponding binary classifiers.

In passing, notice that this is a straightforward approach to implement a multi-label classifier – i.e., a classifier that has the autonomy to assign one or more classes to each example, rather than the conventional single-label classifiers that assign just one class to an example. Multi-label classification is out of the scope of this chapter, but a more detailed discussion about how to integrate multi-label classification and hierarchical classification into a single algorithm can be found in (Blockeel et al., 2002).

In any case, if we do not want to allow an example to be assigned to more than one class at each level of the class hierarchy, this constraint can be easily incorporated in the classification procedure during the test phase, by forcing the procedure to assign a test example only to the most likely class at each level of the class hierarchy.

Big-Bang versus Top-Down Hierarchical Classification

Probably the most important watershed to categorize truly hierarchical classification algorithms is the distinction between the *big bang* and the *top down* approaches. The main characteristics of these approaches are as follows (Sun and Lim, 2001), (Sun et al., 2003a).

In the big bang approach, a single (relatively complex) classification model is built from the training set, taking into account the class hierarchy as a whole during a single run of the classification algorithm. When used during the test phase, each test example is classified by the induced model, a process that can assign classes at potentially every level of the hierarchy to the test example. An example of the big-bang approach in the context of text mining can be found in (Sasaki & Kita, 1998), whereas an example of this approach in the context of Bioinformatics – the focus of this paper – will be discussed in the next section.

In the top-down approach, in the training phase, the class hierarchy is processed one level at a time, producing one or more classifiers for each class level. In the test phase, each example is classified in a top-down fashion, as follows. First, the test example is assigned to one or more classes by the first-level classifier(s). Then the second level classifier(s) will assign to this example one or more sub-classes of the class(es) predicted at the first level, and so on, until the example's class(es) is(are) predicted at the deepest possible level.

In order to produce a hierarchical set of classifiers in the top-down approach, we can either train a single classifier per class level or train multiple classifiers per level. In the former case, we use a multi-class classification algorithm. Thus, at each class level, we build a classifier that predicts the class(es) of an example at that level. In the latter case, we typically train a binary classifier at each class node. Therefore, for each test example and for each class level, we present the example to each of the binary classifiers at that level. As a result, the test example will be assigned to one or more classes at each level, and this information will be taken into account in the next level, as previously explained.

The top-down approach has the advantage that each classification model (built either for a class level or a single class node) is induced to solve a more modular, focused classification problem, by comparison with the big-bang approach, where a more complex classification model has to be built by considering the entire class hierarchy at

once. The modular nature of the top-down approach is also exploited in the test phase, where the classification of an example at a given class level guides its classification at the next level. However, the more modular nature of the top-down approach does not guarantee that this approach will have a better predictive performance than the big-bang approach. In particular, the top-down approach has the disadvantage that, if a test example is misclassified at a certain level, it tends to be misclassified at all the deeper levels of the hierarchy. The probability that this kind of error occurs can be reduced by using a procedure that tries to recover from misclassifications in a shallower level of the class tree – see (Dumais & Chen, 2000), (Sun et al., 2004) for a discussion of procedures to address this kind of problem.

A REVIEW OF WORKS ADDRESSING HIERARCHICAL CLASSIFICATION PROBLEMS IN PROTEIN FUNCTIONAL CLASSIFICATION

Table 1 summarizes key aspects of previous works addressing hierarchical classification problems in the context of the prediction of protein functional classes. The second column of this table indicates the broad category of class hierarchy, either tree-structured or DAG-structured, and the specific protein functional classification scheme addressed in each work. The third column indicates whether the predictions made by the system are flat or hierarchical. If they are hierarchical, the column also indicates which class level(s) is(are) predicted. The fourth column points out whether the classification algorithm being used is a flat or a hierarchical algorithm. Note that for flat predictions the classification algorithm is always flat. However, in the case of hierarchical predictions, the classification algorithm can be either flat (see the Subsection “Hierarchical Class Predictions Using Flat Classification Algorithms” of the previous Section) or hierarchical (see the Subsection “Big-Bang versus Top-Down Hierarchical Classification” of the previous Section).

As can be observed in Table 1, there are several works involving tree-structured class hierarchies where *the original hierarchical classification problem is transformed into a flat classification problem*. Two examples of this approach are mentioned in Table 1:

- (Jensen et al., 2002), (Weinert & Lopes, 2004) predict only classes at the first level of the previously discussed EC code for enzymes.
- (Jensen et al., 2002) predicts only classes at the first level of the Riley's hierarchical classification scheme. This scheme was originally defined for *E. coli* bacteria (Riley, 1993), but it has been modified to describe protein functions in other organisms.

Table 1: Review of Works Involving Hierarchical Classification of Protein Functions

Work	**Kind of class Hierarchy**	**Flat or hierarchical class predictions?**	**Hierarchical or flat classification algorithm?**
(Jensen et al., 2002)	Tree: Riley's scheme, EC code	Flat; only first class level	Flat artificial neural network
(Weinert & Lopes, 2004)	Tree: EC code	Flat; only first class level	Flat artificial neural network
(Clare & King, 2001)	Tree: MIPS	Flat; all class levels, but just one level at a time	Flat decision-tree induction algorithm
(Clare & King, 2003)	Tree: MIPS	Hierarchical; all class levels	Hierarchical (big-bang) decision-tree induction algorithm, compared with flat decision-tree induction algorithm
(Holden & Freitas, 2005)	Tree: EC code	Hierarchical; all class levels	Hierarchical rule set (top-down) discovered using many runs of a flat hybrid particle swarm/ant colony optimisation algorithm
(Holden & Freitas, 2006)	Tree: GPCR classes	Hierarchical; first four class levels	Hierarchical rule set (top-down) discovered using many runs of a flat hybrid particle swarm/ant colony optimisation algorithm
(Jensen et al., 2003)	DAG: GO	Hierarchical; learnability-based prediction, potentially at any class level	Flat artificial neural network
(Laegreid et al., 2003)	DAG: GO	Hierarchical, learnability-based prediction, potentially at any class level	Flat rough set-based rule induction and genetic algorithms; Class hierarchy used to create flat training sets in data pre-processing
(Tu et al., 2004)	DAG: GO	Hierarchical; learnability-based prediction of child classes given their parent class	Flat artificial neural network; Class hierarchy used to create flat training sets in data pre-processing
(King et al., 2003)	DAG: GO	Deep, specific levels; predictions based on other known classes for the same protein	Flat decision-tree induction algorithm and hierarchical Bayesian networks

Let us now turn to *hierarchical class predictions using a flat classification algorithm*. In the context of tree-structured class hierarchies, the approach of predicting each level of the hierarchy independently from the other levels – i.e., by running a separate flat classification algorithm for each level – is found in (Clare & King, 2001), where it was the only hierarchical classification approach used; and in (Clare & King, 2003), where it was compared with a more sophisticated approach based on a hierarchical version of the well-known C4.5 algorithm, as will be discussed later in this section.

Turning to DAG-structured hierarchical classes, Table 1 mentions several works – in particular (Jensen et al., 2003), (Laegreid et al., 2003) and (Tu et al., 2004) – where classes can be predicted at potentially any level of the previously-discussed Gene Ontology (GO) using a flat classification algorithm – although in practice the actual number of predicted classes is relatively small. Let us first review the strategies used by these works to achieve such flexibility, and next discuss their limitations concerning the actual number of classes predicted.

In (Jensen et al., 2003), a flat Artificial Neural Network (ANN) was trained for each GO class at a time. It seems that parent-child relationships between classes were virtually ignored during the ANN training, since each run of the ANN treated the current class as the positive class and apparently considered all the other classes – regardless of their position in the class hierarchy – as negative classes.

(Laegreid et al., 2003) predicts GO classes using a combination of a Rough Set-based rule induction algorithm and a Genetic Algorithm (GA). However, these algorithms do not directly cope with the DAG-structured class hierarchy of GO. Rather, the class hierarchy is used just to pre-process the data into suitable flat classification training sets, as follows. The genes in the original training set were grouped into 23 high-level GO classes. For genes whose most specific annotated class was below the level of the target classes, the more specific classes were ignored. This corresponds to generalizing each of the specific classes to one of the general classes at the level of the target 23 classes. As a result, 23 groups of genes were created, each group associated to one of the 23 classes. Thus, each group was considered as a training set to the combined technique. As a result,

a set of rules was discovered for each of the 23 classes – i.e., each rule predicts a single class.

Note that in both (Jensen et al., 2003) and (Laegreid et al., 2003) the essential result of the training phase is one classification model for each of the GO training classes. The basic difference is the nature of the classification model, which is an ANN in (Jensen et al., 2003) and a set of rules in (Laegreid et al., 2003). However, at a high level of abstraction, we can ignore this difference and focus on the important point: in both works, a classification model was built for each class, one class at a time. Consequently, when a new example in the test set needs to be classified, we can simply apply each of the classification models (each of them associated with a predicted class at potentially any level in the hierarchy) to that test example, and then assign to the test example the best class(es) among the classes predicted by the classification models, based on some measure of confidence in the prediction made by these models. That is why classification models trained for flat classification were able to assign a new test example to potentially any class at any level of the GO hierarchy.

The work of (Tu et al., 2004) also uses a flat-classification ANN as the classification algorithm. However, unlike the work of (Jensen et al., 2003), where the class hierarchy seems to have been virtually ignored, the work of (Tu et al., 2004) applied the ANN in a way that clearly takes into account the parent-child relationships between classes. A set with a parent GO class and its child sub-classes was called by the authors a classification space. At each classification space, a flat-classification ANN was trained to predict the child class for an example that was known to belong to the parent class. This is what the authors called "further prediction", because in order to predict child classes at a given level n it is necessary to know their parent class at level $n - 1$ (where $n \geq 2$). Note that this is in contrast with the works of (Jensen et al., 2003), (Laegreid et al., 2003), which do not require the parent class of a protein to be known in order to predict its corresponding child class.

Observe also that in both (Laegreid et al., 2003) and (Tu et al., 2004) the structure of the class hierarchy – i.e., parent-child class relationships – was essentially used to create flat training sets in a kind of data pre-processing step for the application of a flat classification algorithm.

We now turn to limitations in the actual number of GO classes predicted in the works of (Jensen et al., 2003), (Laegreid et al., 2003) and (Tu et al., 2004). To understand these limitations, we first must bear in mind that the prediction of GO classes is particularly challenging. As discussed earlier, GO classes are arranged in a DAG, and the number of GO classes is very high (more than 13,000), referring to an extremely diverse set of gene/protein functions. As a result, it is not surprising that many works, like (Jensen et al., 2003), (Laegreid et al., 2003) and (Tu et al., 2004), follow an approach that can be named *learnability-based prediction*, a term explicitly introduced by (Tu et al., 2004). The basic idea is to focus on the prediction of the classes which are "more learnable", i.e., which can be predicted with a reasonable accuracy (given the available predictor attributes) and whose prediction is considered interesting and non-trivial. Let us briefly discuss how this approach has been used in the previously mentioned works.

(Jensen et al., 2003) initially tried to predict 347 GO classes, but this number was significantly reduced later in two stages: (a) the majority of the 347 classes was discarded either because they could not be predicted with a reasonable accuracy or because they represented trivial classes whose prediction was not interesting – this reduced the number of classes to be predicted to 26; (b) a number of classes was discarded to avoid redundancy with respect to other classes, which finally left only 14 GO classes to be predicted. As another example of the use of the learnability-based prediction approach, (Tu et al., 2004) considered 44 classification spaces, containing in total 131 classes – including both parent and child classes. The final set of learnable classes was reduced to just 14 classification spaces, containing in total 45 classes – again, including both parent and child classes. In addition, as mentioned earlier, (Laegreid et al., 2003) focused on predicting only 23 general classes of the GO hierarchy.

Let us now discuss two works making *hierarchical predictions based on either the big-bang or the top-down approach*.

(Clare & King, 2003) modified the well-known C4.5 decision-tree induction algorithm to perform hierarchical classification. This modification was not described in detail in the paper, but the basic idea seems to be that the entropy formula – used to decide which attribute will be selected for a given node in the decision tree being built – was weighted. This weighting took into account the facts that shallower (more general) classes tend to

have lower entropy than deeper (more specific) classes, but more specific classes are preferred – since they provide more biological knowledge about the function of a protein. The predictive accuracy of the hierarchical version of C4.5 was compared with the predictive accuracy obtained applying the standard C4.5 separately to each class level. The hierarchical version of C4.5 obtained mixed results in terms of predictive accuracy, outperforming the standard C4.5 in some cases, but being outperformed by the latter in other cases. Since the hierarchical version of C4.5 apparently considered the entire class hierarchy during its training and produced a hierarchical classification model in a single run of the algorithm, this work can be considered an example of the "big-bang" approach.

By contrast, an example of the top-down approach is found in (Holden & Freitas, 2005). This work applied a new hybrid particle swarm optimization/ant colony optimization (PSO/ACO) algorithm to the prediction of the EC code of enzymes. In addition, (Holden & Freitas, 2006) applied a slightly improved version of the PSO/ACO algorithm to the prediction of G-Protein-Coupled-Receptor (GPCR) classes. Since in both works the hierarchical classification method used has essentially the same basic structure, our discussion hereafter will refer only to the work of (Holden & Freitas, 2005), for the sake of simplicity.

In this work the top-down approach was used mainly for the creation of the training sets used by the PSO/ACO algorithm and for the classification of the examples in the test set, as follows. During its training, the PSO/ACO algorithm was run once for each internal (non-leaf) node of the tree hierarchy. At each internal node, the PSO/ACO discovered a set of rules discriminating among all the classes associated with the child nodes of this internal node. For instance, at the root node, the algorithm discovered rules discriminating among the first-level classes 1,2,…,k_0, where k_0 is the number of first-level classes (child nodes of the root node). At the node corresponding to class 1, the algorithm discovered rules discriminating among the second-level classes 1.1, 1.2,…,k_1, where k_1 is the number of child classes of the class 1, and so on. The class hierarchy was used to select a specific set of positive and negative examples for each run of the PSO, containing only the examples directly relevant for that run. For instance, the algorithm run corresponding to class node 1 – i.e., discovering rules to discriminate among classes 1.1, 1.2,…,k_1 – used only examples belonging to classes 1.1, 1.2,… ,k_1. During that run

of the algorithm, when evaluating a rule predicting, say, class 1.1, examples of this class were considered positive examples, and examples of its sibling classes 1.2, 1.3…,k_1 were considered negative examples.

This approach produces a hierarchical set of rules, where each internal node of the hierarchy is associated with its corresponding set of rules. When classifying a new example in the test set, the example is first classified by the rule set associated with the root node. Next, it is classified by the rule set associated with the first-level node whose class was predicted by the rule set at the root ("zero-th") level, and so on, until the example reaches a leaf node and is assigned the corresponding fourth-level class. For instance, suppose the example was assigned to class 1 by the rule set associated with the root node. Next, the example will be classified by the rule set associated with the class node 1, in order to have its second-level class predicted, and so on, until the complete EC code is assigned to the example. This top-down approach for classification of test examples exploits the hierarchical nature of the discovered rule set, but it has the drawback that, if an example is misclassified at level *l*, then clearly the example will also be misclassified at all levels deeper than *l*.

In addition, note that in (Holden and Freitas, 2005) the class hierarchy was used during training just to produce compact sets of positive and negative examples associated with the run of the PSO/ACO algorithm at each node, but the algorithm itself was essentially a flat classification algorithm. I.e., each time the algorithm was run it was solving a flat classification problem, and it is just the many runs of the algorithm – one run for each internal node of the class hierarchy – that produces the hierarchical rule set.

Hence, this work can be considered as a borderline between hierarchical prediction using a top-down hierarchical classification method and hierarchical prediction using a flat classification algorithm. In this chapter we categorize it mainly as belonging to the former group of methods, mainly because it produces a hierarchical rule set that is used to classify test examples according to the top-down approach. Besides, it systematically creates a hierarchical rule set covering the entire class hierarchy. These characteristics are in contrast with methods that are more typical examples of the approach of hierarchical prediction with a flat classification algorithm, such as predicting classes at each level of hierarchy independently from other levels (as in (Clare & King 2001)) or using a flat

classification algorithm to predict just a relatively small subset of the class hierarchy (as in (Jensen et al., 2003), (Laegreid et al., 2003), (Tu et al., 2004)).

Finally, let us discuss the work of (King et al., 2003) for the prediction of GO classes. First of all, this work is quite different from the other works predicting GO classes shown in Table 1, with respect to the predictor attributes used to make the prediction. In this work, when predicting whether or not a particular GO term *g* should be assigned to a gene, the set of predictor attributes consists of all the other terms annotated for this gene, except the terms that are ancestors or descendants of the term *g* in the GO DAG. By contrast, the other works predicting GO classes mentioned in Table 1 address the more usual problem of predicting GO classes based on a set of predictor attributes, which do not involve any previously annotated GO term.

Since the work of (King et al., 2003) requires a gene to have a set of GO term annotations different from the GO term *g* currently being predicted, this work can be said to perform a kind of "further prediction", a term that was also used to describe the work of (Tu et al., 2004). One important difference is that in (Tu et al., 2004) the prediction of a new term requires knowledge of the parent of that term in the GO DAG, whilst in (King et al., 2003) the prediction of a new term *g* requires knowledge of any other term, except the ancestors and descendants of *g* in the GO DAG.

Concerning the algorithms, (King et al., 2003) investigated two algorithms. One was essentially a flat-classification decision-tree induction algorithm. The other, a Bayesian network algorithm, can be called a hierarchical algorithm in the sense of directly taking the class hierarchy into account when generating the Bayesian network. This was achieved by ordering the attributes (random variables for the Bayesian network algorithm) in a way compatible with the GO DAG; more precisely, using an ordering A_1, …, A_m – where m is the number of attributes – in which attribute A_i comes before attribute A_j whenever A_i is a parent of A_j in the GO DAG. Once one of these orderings is found – there are in general many orderings satisfying the mentioned property and the authors did not try to find the optimal one – it is taken into account in the construction of the Bayesian network.

CONCLUSIONS AND FUTURE RESEARCH

This chapter has two main contributions. The first one is to present a tutorial on hierarchical classification, discussing how to categorize hierarchical classification problems and hierarchical classification algorithms. The second contribution of this chapter is to present a review of a number of works applying hierarchical classification techniques to protein function prediction – an important Bioinformatics problem.

Let us now list the main conclusions of the discussion presented in this chapter, followed by corresponding suggested research directions.

First, in hierarchical classification the misclassification costs tend to vary significantly across different levels of the hierarchy and even across the same level. This is an under-explored topic in the literature on hierarchical classification of protein functions. In general, the works mentioned in Table 1 (in the previous section) employ a simple approach to measure predictive accuracy, assuming uniform misclassification costs. We would like to point out that it is important to use a measure of predictive accuracy tailored for hierarchical classification problems, such as distance-based misclassification costs or a measure based on a hierarchical misclassification cost matrix. The use of these measures is expected to be a significant improvement over the current situation of using a simple predictive accuracy measure that ignores the class hierarchy.

Second, when the target hierarchical classification problem is categorized as optional leaf-node prediction, the hierarchical classification algorithm has autonomy to decide how deep in the class hierarchy should be the most specific class assigned to an example. This decision should, in principle, be based on at least two factors, namely:

(a) How much confidence the algorithm has in different class predictions at different levels of the class hierarchy – recall that, in general, the deeper the class level, the more difficult the prediction is, and so the less confident the algorithm will tend to be in the prediction;

(b) The usefulness of different class predictions at different levels of the class hierarchy – recall that, in general, the deeper the level of a predicted class, the more useful the prediction tends to be to the user.

This trade-off between confidence and usefulness of class predictions at different levels of a class hierarchy is still very under-explored in the literature, and a future

research direction would be to try to develop methods for quantifying this trade-off and coping with it during the construction of the classification model. Since maximizing prediction confidence and maximizing prediction usefulness tend to be conflicting objectives, perhaps the use of a multi-objective classification algorithm based on the concept of Pareto dominance would be useful here. The reader is referred to (Deb, 2001) for a comprehensive review of the concept of Pareto dominance and to (Freitas, 2004) for a review of the motivation for the use of this concept in data mining.

Third, as a particularly important special case of the general trade-off between confidence and usefulness just-discussed, we identify the challenging problem of predicting functional classes of the Gene Ontology – a class hierarchy in the form of a DAG. As discussed in Section 5, several works in this area focus on predicting classes in just a relatively small subset of the GO DAG, rather than all classes in the entire DAG, and in particular focus on classes that are more "learnable" – characterizing the so-called "learnability-based prediction framework". The basic idea of this framework consists of predicting classes which are more learnable and whose prediction is more interesting, which is conceptually similar to the idea of trying to maximize the confidence and the usefulness of the predictions made by the hierarchical classification algorithm. The need for this learnability-based framework is clearly understandable, since the GO DAG contains more than 13,000 class nodes and, in a given Bioinformatics application, the user may very well be interested in predicting a relatively small set of GO functional classes. However, it should be pointed out that, in the literature in general, the precise determination of which GO functional classes should be predicted in the learnability-based framework is usually done in an ad-hoc fashion, being very much dependent on the user's intuition. There is a need to develop more formal and effective methods to determine how to quantify the degree of learnability and interestingness of nodes in the GO class hierarchy. This is an important and challenging research direction.

REFERENCES

Aha, D.W., Kibler, D. & Albert, M.K. (1991). Instance-based learning algorithms. *Machine Learning 6*, 37-66.

Aha, D.W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies 36*, 267-287.

Aha, D.W. (1997). *Artificial Intelligence Review – Special issue on lazy learning, 11,* 1-5.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K. & Walter, P. (2002). *The Molecular Biology of the Cell*. (4th Ed.) Garland Press.

Bard, J., Rhee, S. Y. & Ashburner, M. (2005). An ontology for cell types. *Genome Biology 2005, 6(2)*, Article R21.

Blockeel, H., Bruynooghe, M., Dzeroski, S., Ramon, J. & Struyf, J. (2002). Hierarchical multi-classification. *In KDD-2002 Workshop Notes: MRDM 2002 – Workshop on Multi-Relational Data Mining* (pp. 21-35).

Camon, E., Magrane, M., Barrel, D., Binns, D., Fleischmann, W., Kersey, P., Mulder, N., Oinn, T., Maslen, J., Cox, A. & Apweiler, R. (2003). The Gene Ontology Annotation (GOA) project: implementation of GO in Swiss-Prot, TrEMBL and InterPro. *Genome Research*.

Caruana, R. & Niculescu-Mizil, A. (2004). Data mining in metric space: an empirical analysis of supervised learning performance criteria. *In Proceedings of the 6th ACM SIGMOD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*. ACM Press (pp. 69-78).

Clare, A. & King, R. D. (2001). Knowledge discovery in multi-label phenotype data. *In Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery and Data Mining (PKDD-2001), Lecture Notes in Artificial Intelligence 2168,* (pp. 42-53).

Clare, A. & King, R. D. (2003). Predicting gene function in *Saccharomyces Cerevisiae*. *Bioinformatics, Vol. 19, Suppl. 2,* ii42-ii49.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons.

Deng, M., Zhang, K., Mehta, S., Chen, T. & Sun, F. (2002). Prediction of protein function using protein-protein interaction data. *In Proceedings of the IEEE Computer Society Bioinformatics Conference (CSB'02)* (pp. 947–960).

Dressler, D. & Potter, H. (1991). *Discovering Enzymes*. Scientific American Library.

(Dumais & Chen, 2000) S. Dumais and H. Chen. Hierarchical classification of web content. *In Proceedings of the SIGIR-00 23rd ACM International Conference on Research and Development in Information Retrieval*. ACM Press (pp. 256-263).

Freitas, A.A. (2004). A Critical Review of Multi-Objective Optimization in Data Mining: a position paper. *ACM SIGKDD Explorations, 6(2)*, 77-86.

Gerlt, J. A. & Babbitt, P.C. (2000). Can sequence determine function? *Genome Biology 1(5)*, 1-10.

The Gene Ontology Consortium (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics 25*, 25-29.

Gupta R. & Brunak, S. (2002). Prediction of glycosylation across the human proteome and the correlation to protein function. *In Proceedings of the Pacific Symposium on Biocomputing (PSB-2002)* (pp. 310-322).

Hendlich, M., Bergner, A., Gunther, J. & Klebe, G. (2003). Relibase: design and development of a database for comprehensive analysis of protein-ligand interactions. *Journal of Molecular Biology, 326*, 607-620.

Higgs, P.G. & Attwood, T.K. (2005). *Bioinformatics and Molecular Evolution*. Blackwell.

Holden, N. & Freitas, A. A. (2005). A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. *Proceedings of the 2005 IEEE Swarm Intelligence Symposium* (pp. 100-107).

Holden, N. & Freitas, A. A. (2006). Hierarchical classification of G-Protein-Coupled Receptors with a PSO/ACO Algorithm. *To appear in Proceedings of the 2006 IEEE Swarm Intelligence Symposium*.

Jensen, L. J., Gupta, R., Blom, N., Devos, D., Tamames, J., Kesmir, C., Nielsen, H., Staerfeldt, H. H., Rapacki, K., Workman, C., Andersen, C. A. F., Knudsen, S., Krogh, A., Valencia, A. & Brunak, S. (2002). Prediction of human protein function from post-translational modifications and localization features. *J. Mol. Biol., 319,* 1257-1265.

Jensen, L. J., Gupta, R., Staerfeldt, H.-H. & Brunak S. (2003). Prediction of human protein function according to Gene Ontology categories. *Bioinformatics 19(5)*, 635-642.

Jiang, D., Tang, C. & Zhang, A. (2004). Cluster Analysis for Gene Expression Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 16 (11), 1370-1386.

Karwath, A. & King, R. D. (2002). Homology induction: the use of machine learning to improve sequence similarity searches. *BMC Bioinformatics 3:11*.

King, R. D., Karwath, A., Clare, A. & Dehaspe, L. (2001). The utility of different representations of protein sequence for predicting functional class. *Bioinformatics, 17(5)*, 445-454.

King, O.D., Foulger, R. E., Weight, S. D., White, J. V. & Roth, F. P. (2003). Predicting gene function from patterns of annotation. *Genome Research, 13*, 896-904.

Koller, D. & Sahami, M. (1997). Hierarchically classifying documents using very few words. *In Proceedings of the 14th International Conference on Machine Learning (ICML-1997)*. Morgan Kaufmann, (pp. 170-178).

Laegreid, A., Hvidsten, T. R., Midelfart, H., Komorowski, J. & Sandvik, A.K. (2003). Predicting gene ontology biological process from temporal gene expression patterns. *Genome Research 13*, 965-979.

Lenhinger, A. L., Nelson, D. L. & Cox, M. M. (1998). *Principles of Biochemistry with an Extended Discussion of Oxygen-Binding Proteins*. (2nd Ed.) New York: Worth Publishers.

Lewis, S.E. (2004). Gene Ontology: looking backwards and forwards. *Genome Biology 2004, 6*(*1*), Article 103.

Liao, T.W., Zhang, Z. & Mount, C.R. (1998). Similarity measures for retrieval in case-based reasoning systems. *Applied Artificial Intelligence, 12*, 267-288.

Nagl, S.B. (2003). Molecular Evolution. *In:* C.A. Orengo, D.T. Jones and J.M. Thornton (Eds.) *Bioinformatics: genes, proteins, computers* (pp. 1-17).

Pal, D. & Eisenberg, D. (2005). Inference of protein function from protein structure. *Protein Structure, 13*, 121-130.

Riley, M. (1993). Functions of the gene products of *Eschirichia coli*. *Microbiological Reviews, 57*, 862-952.

Sasaki, M. & Kita, K. (1998). Rule-based text categorization using hierarchical categories. *In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* IEEE Press *(*pp. 2827-2830).

Schug, J., Diskin, S., Mazzarelli, J., Brunk, B.P. & Stoeckert Jr., C.J. (2002). Predicting gene ontology functions from ProDom and CDD protein domains. *Genome Research 12*, 648-655.

Slonim, D. K., Tamayo, P., Mesirov, J. P., Golub, T. R. & Lander, E. S. (2000). Class prediction and discovery using gene expression data. *In Proceedings of the 4th Annual Intemational Conference on Computational Molecular Biology (RECOMB)*, Universal Academy Press, Tokyo, Japan (pp. 263-272)

Smith, C. L., Goldsmith, C. A. W. & Eppig, J. T. (2004). The mammalian phenotype ontology as a tool for annotating, analyzing and comparing phenotypic information. *Genome Biology, 6(1)*, *Article R7*.

Stawiski, E. W., Mandel-Gutfreund, Y., Lowenthal, A. C. & Gregoret L.M., (2002). Progress in predicting protein function from structure: unique features of O-glycosidases. *In Proceedings of the 2002 Pacific Symposyum on Biocomputing* (pp. 637-648).

Sun, A. & Lim, E.-P. (2001). Hierarchical text classification and evaluation. *In Proceedings of the 1st IEEE International Conference on Data Mining*. IEEE Computer Society Press (pp. 521-528).

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

Sun, A., Lim, E.-P. & Ng, W.-K. (2003b). Hierarchical text classification methods and their specification. In: A.T.S. Chan, S.C.F. Chan, H.V. Leong, V.T.Y. Ng. (Eds.) *Cooperative Internet Computing* (pp. 236-256). Kluwer.

Sun, A., Lim, E.-P. & Ng, W.-K. (2004). Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering 16(10)*, 1305-1308.

Syed, U. & Yona, G. (2003). Using a mixture of probabilistic decision trees for direct prediction of protein function. *In Proceedings of the 2003 Conference on Research in Computational Molecular Biology (RECOMB-2003)* (pp. 289-300).

Tan, P. N., Steinbach, M. & Kumar, V. (2006). *Introduction to Data Mining*. Addison-Wesley.

Thomas, P. D., Kejariwal, A., Campbell, M. J., Mi, H., Diemer, K., Guo, N., Ladunga, I., Ulitsky-Lazareva, B., Muruganujan, A., Rabkin, S., Vandergriff, J.A. & Doremieux, O. (2003). PANTHER: a browsable database of gene products organized by biological function, using curated protein family and subfamily classification. *Nucleic Acids Research, 31(1)*, 334-341.

Tu, K., Yu, H., Guo, Z. & Li, X. (2004). Learnability-based further prediction of gene functions in Gene Ontology. *Genomics 84,* 922-928.

Weinert, W. R. & Lopes, H.S. (2004). Neural networks for protein classification. *Applied Bioinformatics, 3(1)*, 41-48.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools with Java Implementations*. (2nd Ed.) Morgan Kaufmann.