NOVEL APPROACHES FOR HIERARCHICAL CLASSIFICATION WITH CASE STUDIES IN PROTEIN FUNCTION PREDICTION

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT AT CANTERBURY

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY (PHD).

By

Carlos Nascimento Silla Junior

September 2011

UC 9

F224459

Abstract

A very large amount of research in the data mining, machine learning, statistical pattern recognition and related research communities has focused on flat classification problems. However, many problems in the real world such as hierarchical protein function prediction have their classes naturally organised into hierarchies. The task of hierarchical classification, however, needs to be better defined as researchers into one application domain are often unaware of similar efforts developed in other research areas.

The first contribution of this thesis is to survey the task of hierarchical classification across different application domains and present an unifying framework for the task. After clearly defining the problem, we explore novel approaches to the task.

Based on the understanding gained by surveying the task of hierarchical classification, there are three major approaches to deal with hierarchical classification problems. The first approach is to use one of the many existing flat classification algorithms to predict only the leaf classes in the hierarchy. Note that, in the training phase, this approach completely ignores the hierarchical class relationships, i.e. the parent-child and sibling class relationships, but in the testing phase the ancestral classes of an instance can be inferred from its predicted leaf classes. The second approach is to build a set of local models, by training one flat classification algorithm for each local view of the hierarchy. The two main variations of this approach are: (a) training a local flat multi-class classifier at each non-leaf class node, where each classifier discriminates among the child classes of its associated class; or (b) training a local flat binary classifier at each node of the class hierarchy, where each classifier predicts whether or not a new instance has the classifier's associated class. In both these variations, in the testing phase a procedure is used to combine the predictions of the set of local classifiers in a coherent way, avoiding inconsistent predictions. The third approach is to use a global-model hierarchical classification algorithm, which builds one single classification model by taking into account all the hierarchical class relationships in the training phase. In the context of this categorization of hierarchical classification approaches, the other contributions of this thesis are as follows.

The second contribution of this thesis is a novel algorithm which is based on the local classifier per parent node approach. The novel algorithm is the selective representation approach that automatically selects the best protein representation to use at each non-leaf class node.

The third contribution is a global–model hierarchical classification extension of the well known naive Bayes algorithm. Given the good predictive performance of the global–model hierarchical–classification naive Bayes algorithm, we relax the Naive Bayes' assumption that attributes are independent from each other given the class by using the concept of k dependencies. Hence, we extend the flat classification k-Dependence Bayesian network classifier to the task of hierarchical classification, which is the fourth contribution of this thesis.

Both the proposed global-model hierarchical classification Naive Bayes and the proposed global-model hierarchical k-Dependence Bayesian network classifier have achieved predictive accuracies that were, overall, significantly higher than the predictive accuracies obtained by their corresponding local hierarchical classification versions, across a number of datasets for the task of hierarchical protein function prediction.

Acknowledgements

Pursuing a PhD is normally a very hard working task. I have been unfortunate to have as many external problems as I had while pursuing my PhD, but at the same time I have been very fortunate to have all the people who were there for me in different (or all) moments of it. Even though I know I may forget a lot of people (and I do not mean it), I will try to use one of the (many) valuable principles I learned from my PhD supervisor Dr. Alex Alves Freitas and "be as precise as possible".

First and foremost, I would never be here if not by all the love, care and support from my family. Specially my late Grandmother, who was like a mother to me and passed away last year. To her I dedicate this thesis as nothing I write will ever reflect how thankful I am for everything she did to me and how much I will ever keep missing her and wishing she was still here, as she shall forever live in my heart. This has possibly been the worst thing I have had to face during the PhD (and in my life in general) and I will be eternally thankful for the support of all my friends, specially the ones in Brazil!

Secondly my mum, who worried constantly about me and my wellbeing being so far away from home (not that it would be different if I was still living at home). In theory most mums should be like mine, but I still believe my mum to be the best mum in the world (which, I admit, is a very biased opinion), and my greatest role model when it comes to have strength in face of many adversities.

Thirdly, I would like to thank my friend and supervisor, Alex. I knew before coming to the UK that I wanted to study with him, because he was very knowledgeable in his area of expertise. However little did I know that he was such an amazing person as well. If not for his guidance and support, I would have long quit pursuing my PhD at the University of Kent. It is funny, when I attended the initial workshops from the TST (Transferable Skills Training programme), very early in my PhD, people would often ask "You are from Brazil right?". I would proudly answer "Yes! 100% Brazilian!". Then they would follow up asking "Why study in the UK? Why come to Canterbury instead of any of the sunny/warm/fun places in Brazil?". I would always promptly answer "Because of my supervisor!". The student-supervisor relationship in TST was always said to be vital to the success or failure of a postgraduate degree, and I am very glad I made a wise decision to study for my PhD under Alex's guidance.

It was under the TST programm that by accident (it is a long funny story) I ended up into a workshop run by Paul Toombs. Paul has been a key person in my personal support network, believing in me in some very tough moments, and no matter how amazing I say his workshops are (which is why I possibly attended some workshops just because I saw his name on it, as it was a sign of amazing training quality) they are just a shadow of this great man. Also, within my personal support network, I really want to thank Marcos Antonio Dal Collina, Elson Valerio, and more recently a certain special someone for helping me a lot in their own ways. If not for their particular help and support, I wonder if I would have managed to keep pursing my PhD.

For certain, I would have never made it to the University of Kent at Canterbury (UKC) without the scholarship from the Brazilian research-support agency CAPES. I would like to say a special thank you to Joana Ribeiro de Abreu and Vanda Lucena for their patience, politeness and for promptly replying to all of my (many) questions involving the scholarship bureaucracy.

The UKC has been a brilliant place to study at. The period of moving into a new, unknown environment without knowing anyone, which is very familiar to all new students, specially oversea students, has been considerably shortened due to the way the UKC is organised.

First, every PhD student at the school of computing shares an office with other

PhD students. I was lucky to be originally in S15 and later in S109B where I met great people (Fernando, Mudassar, Nick, Lingfang, Sebastien, Radu, Chris, Ben) who today I have the privilege to know as my friends.

Second, there is a small Brazilian Community in Canterbury and I would like to thank all my Brazilian (Fernando, Kadu, Erick, Marjory, Luis, Sonia, Lucas, Joao, Anne, Paulo, Valeschka, Julio, Andre, Gisele, Plastino, Adilson, Jean, Marcio, Alvaro) and pseudo-Brazilian (Gift, Natalia, Rodolfo) friends who in one way or another made some of the last four years way more enjoyable.

Third, the UKC has student societies which are bound to match any student particular interests. This provides us with the ability to meet with people with similar interests as we do or to discover and learn about new things. Thanks to the existence of student societies in the UKC, I had a great time and made a lot of friends within different societies. I would like to thank in special the UKC Latin and Ballroom society (which is now a sports club) and the UKC Salsa Society (Let's Rueda!). These two societies are very special to me, as through them I made very special friends. I am afraid I will have to omit names here, as I do not think I can get away with the departmental guidelines by listing over 300 friends here! However, Angelo Bastiaensz and Terri Jane Clare Dowling deserve a special mention.

During the course of my PhD, the University has allowed me to do seasonal undergraduate teaching under the guidance of the module convenors. I am thankful to Sally Fincher, Bob Eager, Nick Ryan, Ursula Fuller, Olaf Chitil, John Crawford and David Barnes for sharing their teaching experience with me. I am also thankful to Sonnary Var, Angela Doe, Angie Allen and Jenny Oatley for always dealing with their responsibilities (and my many inquires related to them) with a smile.

Outside the UKC I am also very thankful to the following people: My friend and former supervisor Celso Kaestner who, during my early years as an undergraduate computer science student, introduced me to the world of computer science research in the topics of text mining and natural language processing, and later supervised my Masters research. My friend and former co-supervisor Alessandro Koerich who introduced me to the field of music information retrieval. Dr. Joao Gama for providing very useful pointers and information to the k-Dependence Bayesian classification algorithm in the early stages of my PhD.

My PhD viva has been a very interesting learning experience thanks to my thesis examiners Dr. Wenjia Wang and Professor Jon Williamson and their very valuable feedback on my research.

Lastly, I would like to thank the anonymous reviewers from the Data Mining and Knowledge Discovery Journal, the Intelligent Data Analysis journal, the IEEE 2009 Int. Conf. in Data Mining (ICDM) and the IEEE 2009 Int. Conf. on Systems, Man and Cybernetics (SMC) for providing valuable questions, suggestions and feedback to the papers submitted and published in these venues which are part of this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A very large amount of research in the data mining, machine learning, statistical pattern recognition and related research communities has focused on flat classification problems. By flat classification problem we are referring to standard binary or multi-class classification problems. On the other hand, many important real-world classification problems are naturally cast as hierarchical classification problems, where the classes to be predicted are organized into a class hierarchy – typically a tree or a DAG (Directed Acyclic Graph).

One of the application domains that can truly benefit from hierarchical classification is the field of bioinformatics. More precisely the task of protein function prediction. This task is particularly interesting as, although the human genome-sequencing project has ended, the contribution made for knowledge is less clear, because we still do not know the functions of many proteins encoded by genes (Corne and Fogel, 2002). In this work, we propose novel approaches to the task of hierarchical classification.

The task of hierarchical classification, however, needs to be better defined, as it can be overlooked or confused with other tasks, which are often wrongly referred to by the same name. Moreover, the existing literature that deals with hierarchical classification problems is usually scattered across different application domains which are not strongly connected with each other. As a result, researchers in one application domain are often unaware of methods developed by researchers in another domain. Hence, we first present, in the next section, some basic concepts of the hierarchical classification task, based on our recently published survey on this topic (Silla Jr. and Freitas, 2011b).

1.1 What is Hierarchical Classification?

In order to learn about hierarchical classification, one might start searching for papers with the keywords "hierarchical" and "classification"; however, this might be misleading. One of the reasons for this is that, due to the popularity of SVM (Support Vector Machine) methods in the machine learning community (which were originally developed for binary classification problems), different researchers have developed different methods to deal with multi-class classification problems. The most common are the One-Against-One and the One-Against-All schemes (Lorena and Carvalho, 2004). A less known approach consists of dividing the problem in a hierarchical way where classes which are more similar to one another are grouped together into meta-classes, resulting in a Binary Hierarchical Classifier (BHC) (Kumar et al., 2002). For instance, in (Chen et al., 2004) the authors modified the standard SVM, creating what they called a H-SVM (Hierarchical SVM), based on this hierarchical problem decomposition approach.

When we consider the use of meta-classes in the pattern recognition field, they are usually manually assigned, like in (Koerich and Kalva, 2005), where handwritten letters with the same curves in uppercase and lowercase format (e.g. "o" and "O") will be represented by the same meta-class. An automated method for the generation of meta-classes was recently proposed in (Freitas et al., 2008). At first glance the use of meta-classes (and their automatic generation) seems to be related to the hierarchical problem decomposition approach, as one can view the use of meta-classes as a two-level hierarchy where leaf classes are grouped together by similarity into intermediate classes (the meta-classes). This issue is interesting and deserves further investigation, but is beyond the scope of this thesis. In this thesis we take the perspective that this kind of approach is not considered to be a hierarchical classification approach, because it creates new (meta-)classes on the fly, instead of using a pre-established taxonomy. In principle a classification algorithm is not supposed to create new classes, which is related to clustering.

In this thesis we are interested in approaches that cope with a pre-defined class hierarchy, instead of creating one from the similarity of classes within the data (which would lead to higher-level classes that could be meaningless to the user). Let us elaborate on this point. There are application domains where the internal (non-leaf) nodes of the class hierarchy can be chosen based on the data (usually in the text mining application domain), like in (Sasaki and Kita, 1998; Punera et al., 2005; Li et al., 2007; Hao et al., 2007), where they build the hierarchy during training by using some sort of hierarchical clustering method, and then classify new test examples by using a hierarchical classification approach. However, in other domains, like protein function prediction in bioinformatics, just knowing that classes A and B are similar can be misleading, as proteins with similar characteristics (sequences of amino acids) can have very different functions and vice-versa (Gerlt and Babbitt, 2000). Therefore, in this work, we are interested only in hierarchical classification (a type of supervised learning). Hierarchical clustering (a type of unsupervised learning) is out of the scope of this thesis.

Hierarchical classification can also appear under the name of Structured Classification (Seeger, 2008; Astikainen et al., 2008). However, the research field of structured classification involves many different types of problems which are not hierarchical classification problems, e.g. Label Sequence Learning (Altun and Hofmann, 2003; Tsochantaridis et al., 2005). Therefore, hierarchical classification can be seen as a particular type of structured classification problem, where the output of the classification algorithm is defined over a class taxonomy; whilst the term structured classification is broader and denotes a classification problem where there is some structure (hierarchical or not) among the classes.

It is important then to define what exactly is a class taxonomy. (Wu et al., 2005) have defined a class taxonomy as a tree structured regular concept hierarchy defined over a partially order set $(C, \prec)$, where C is a finite set that enumerates all class concepts in the application domain, and the relation $\prec$ represents the "IS-A" relationship. (Wu et al., 2005) define the "IS-A" relationship as both anti-reflexive and transitive. However, we prefer to define the "IS-A" relationship as asymmetric, anti-reflexive and transitive:

- The only one greatest element "R" is the root of the tree.
- $\forall c_i, c_j \in C, if\ c_i \prec c_j\ then\ c_j \nprec c_i$.
- $\forall c_i \in C, c_i \nprec c_i$.
- $\forall c_i, c_j, c_k \in C, c_i \prec c_j\ and\ c_j \prec c_k\ imply\ c_i \prec c_k$.

This definition, although originally proposed for tree structured class taxonomies, can be used to define DAG structured class taxonomies as well. (Ruiz and Srinivasan, 2002) give a good example of the asymmetric and transitive relations: The "IS-A" relation is asymmetric (e.g. all dogs are animals, but not all animals are dogs) and transitive (e.g., all pines are evergreens, and all evergreens are plants; therefore all pines are plants). As for the anti-reflexive property in the class hierarchies there is no arrow from a class node to itself.

Note that, for the purposes of this thesis, any classification problem with a class structure satisfying the aforementioned four properties of the IS-A hierarchy can be considered as a hierarchical classification problem, and in general the hierarchical classification methods surveyed in this work assume (explicitly or implicitly) the underlying class structure satisfies those problems. In the vast majority of works on hierarchical classification, the actual class hierarchy in the underlying problem domain can indeed be called a IS-A hierarchy from a semantical point of view. However, in a few cases the semantics of the underlying class hierarchy might be different, but as long as the aforementioned four properties are satisfied, we would consider the target problem as a hierarchical classification one. For instance, the class taxonomy associated with cellular localization in the Gene Ontology (an ontology which is briefly discussed later) is essentially, from a semantical point of view, a PART-OF class hierarchy, but it still satisfies the four properties of the aforementioned definition of a IS-A hierarchy, so we consider the prediction of cellular location classes according to that class hierarchy as a hierarchical classification problem.

Whether the taxonomy is organized into a tree or a DAG influences the degree of difficulty of the underlying hierarchical classification problem. Notably, as it will be seen in Section 2.7, most of the current literature focus on working with trees as it is an easier problem. One of the main contributions of this thesis is to organize the existing hierarchical classification approaches into a taxonomy, based on their essential properties, regardless of the application domain. One of the main problems, in order to do this, is to deal with all the different terminology that has already been proposed, which is often inconsistent across different works. In order to understand these essential properties, is important to clarify a few aspects of hierarchical classification methods.

Let us consider initially two types of conventional classification methods that cannot directly cope with hierarchical classes: binary and multi-class classifiers. First, the main difference between a binary classifier and a multi-class classifier is that the binary classifier can only handle two-class problems, whilst a multi-class classifier can handle in principle any number of classes. Secondly, there are multi-class classifiers that can also be multi-label, i.e. the answer from the classifier can be more than one class assigned to a given example. Thirdly, since these types of classifiers were not designed to deal with hierarchical classification problems, they will be referred to as flat classification algorithms. Fourthly, in the context of hierarchical classification most approaches could be called multi-label. For instance, considering the hierarchical class structure presented in Figure 1.1 (where R denotes the root node), if the output of a classifier is class 2.1.1, it is natural to say that it also belongs to classes 2 and 2.1, therefore having three classes as the output of the classifier. In (Tikk et al., 2004) this notion of multi-label is used and they call this a particular type of multi-label classification problem. However, since this definition is trivial, as any hierarchical approach

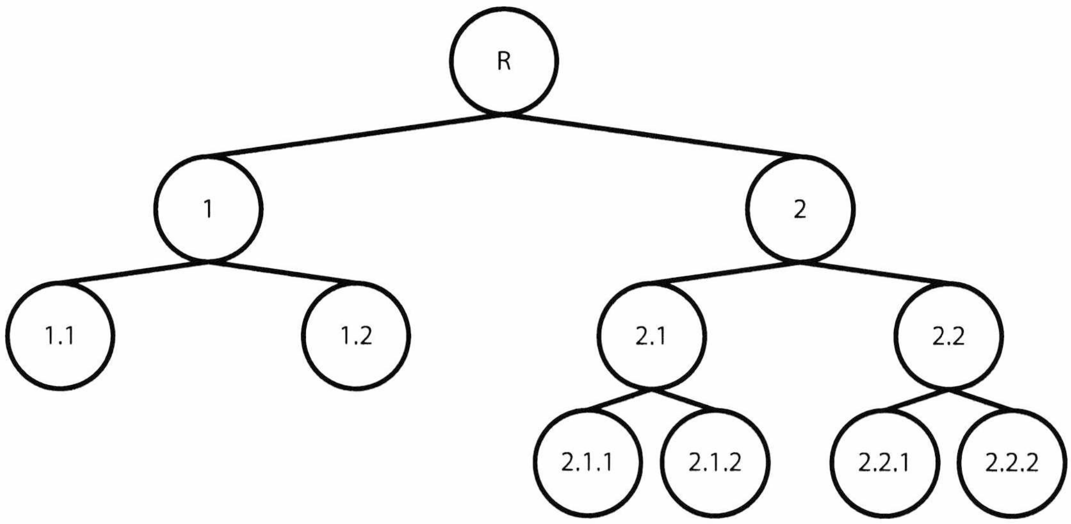


Figure 1.1: An example of a tree-based hierarchical class structure.

could be considered multi-label in this sense, in this work we will only consider a hierarchical classifier to be hierarchically multi-label if it can assign more than one class at any given level of the hierarchy to a given example. This distinction is particularly important, as a hierarchically multi-label classification algorithm is more challenging to design than a hierarchically single-label one. Also, recall that in hierarchical classification we assume that the relation between a node and its parent in the class hierarchy is a "IS-A" relationship.

According to (Freitas and de Carvalho, 2007; Sun and Lim, 2001) hierarchical classification methods differ in a number of criteria. The first criterion is the type of hierarchical structure used. This structure is based on the problem structure and it typically is either a tree or a DAG. Figure 1.2 illustrates these two types of structures. The main difference between them is that in the DAG a node can have more than one parent node. In this figure each node is labelled with a class number (except the root nodes, denoted by "R"), and the edges represent the parent-child relationships among classes. In Figure 1.2 the "." notation is used to separate classes at distinct levels, while the "1-2.1" notation indicates the class at the second level has as parents on the first level both classes 1 and 2.

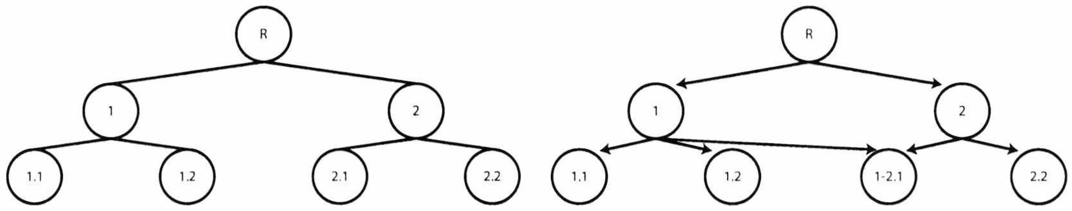


Figure 1.2: A simple example of a tree structure (left) and a DAG structure (right).

The second criterion is related to how deep the classification in the hierarchy is performed. I.e., the hierarchical classification method can be implemented in a way that will always classify a leaf node (which (Freitas and de Carvalho, 2007) refers to as mandatory leaf-node prediction and (Sun and Lim, 2001) refers to as virtual category tree) or the method can consider stopping the classification at any node in any level of the hierarchy (which (Freitas and de Carvalho, 2007) refers to as non-mandatory leaf node prediction and (Sun and Lim, 2001) refers to as category tree). In this chapter we will use the term (non-)mandatory leaf node prediction, which can be naturally used for both tree-structured and DAG-structured class taxonomies.

The third criterion is related to how the hierarchical structure is explored. The current literature often refers to top-down (or local) classifiers, when the system employs a set of local classifiers; big-bang (or global) classifiers, when a single classifier coping with the entire class hierarchy is used; or flat classifiers, which ignore the class relationships, typically predicting only the leaf nodes.

These are the main points which will be discussed in detail in Chapter 2.

As mentioned earlier, the task of protein function prediction can benefit from hierarchical classification approaches. Specially when we consider that the release of the first draft of the human genome (a 3,000,000,000-letter code that distinguishes *Homo sapiens* from other species) was in June 2000 (Corne and Fogel, 2002). This achievement was possible due to a world-wide effort. However, as pointed out in (Corne and Fogel, 2002), the contribution to knowledge was much less clear if we consider questions like "Which genes are involved in the human immune system?". One (out of the many) possible important questions is which genes code which proteins, or more precisely, given an amino acid chain representing a specific protein, what corresponding functions are coded in it?. This is the type of question that we are interested in this work. More precisely, we are interested in creating a machine learning approach that is able to predict the unknown hierarchical functions of proteins based on the knowledge of existing proteins whose functions are known. As it will be seen in Chapter 3 protein functions have been usually organised by biologists into hierarchies. For example, enzymes are proteins whose function is to accelerate chemicals reactions. A hydrolase is a type of enzyme that catalyzes the hydrolysis of a chemical bond. A protease is a more specific type of hydrolase whose function is breaking down large protein molecules into smaller ones in the digestive system of animals, so they can be absorbed by the intestines.

1.2 Aims and Objectives

Broadly speaking, the aims of this thesis are to propose a unifying framework for the hierarchical classification task and to develop novel algorithms for this task, using the the hierarchical protein function prediction problem as a type of case study.

More specifically, the objectives of this thesis are:

- To propose a unifying framework for the hierarchical classification task, including a taxonomy of hierarchical classification problems and methods, based on a detailed survey of this task across different application domains such as protein function prediction, text categorization, music genre classification, etc. This is an important objective because hierarchical classes are common to many application domains, but most of the research on hierarchical classification addresses just one application domain, and researchers in one domain are often unaware of research efforts in other domains.

- To study the impact of different protein representations in the performance of hierarchical classification algorithms for the task of hierarchical protein function prediction.

- To develop novel algorithms for the task of hierarchical classification, and evaluate their performance based on a measure of hierarchical predictive accuracy.

1.3 Original Contributions

A summary of the main contributions of this thesis are presented here.

- A New Unifying Framework for Hierarchical Classification – The task of hierarchical classification is pursed by different research groups in different application domains which are not strongly related to each other. In this work we propose a new unifying framework that classifies all existing approaches into well-defined types of hierarchical classification problems and algorithms represented using a tuple notation, i.e., with each tuple value representing a well-defined type of hierarchical classification problem or algorithm.

- A Novel Local Selective Representation Top-Down Approach – We developed a novel local–model approach that selects at each local decision point (i.e., at each node of the class hierarchy where a classifier has to be built) which particular representation to use.

- Novel Global–model hierarchical–classification Naive Bayes algorithms – We developed two extensions to the flat classification Naive Bayes to deal with hierarchical classification problems in a global–model way. The resulting algorithms are the Global–Model Hierarchical–Classification Naive Bayes (GMNB) and its variant which uses the notion of usefulness (i.e. the fact that deeper predictions are more informative to the user than shallower predictions), called the Global–Model Hierarchical–Classification Naive Bayes with Usefulness (GMNBwU).

- Novel Global–model hierarchical classification k-Dependence Bayesian network classifiers – Given the positive results obtained by GMNB and GMNBwU, we relaxed the attribute independence assumption of the Naive Bayes algorithm, to allow up to k other attributes as parents (besides the class node). This approach is an extension of the flat classification k-DBC classifier to deal with the hierarchical classification problems in a global–model way. Two hierarchical-classification versions of k-DBC, were developed, with and without the notion of usefulness, denoted by VGHCS-k-DBCwU and VGHCS-k-DBC.

1.4 Structure of the Thesis

This thesis is structured in a way that each chapter is self-contained and it is organised as follows:

Chapter 2 introduces the basic concepts of the task of flat classification and presents a thorough discussion of the existing research on hierarchical classification across different application domains.

Chapter 3 describes the task of protein function prediction and different methods to extract features from biological databases. This is needed, as the chapter also present a novel algorithm which is based on the local classifier per parent node approach (LCPN). The novel algorithm is the selective representation approach that aims at each non-leaf node, automatically select the best feature set representation to use.

Chapter 4 describes the "flat" classification Naive Bayes algorithm and how we have modified the flat algorithm in order for it to cope with hierarchical classification problems in a global way. It then presents two novel global–model hierarchical classification algorithms: the global–model hierarchical–classification Naive Bayes (GMNB) and the global–model hierarchical–classification Naive Bayes with Usefulness(GMNBwU). The two proposed global–model algorithms are then compared to two LCPN approaches.

Chapter 5 describes one particular set of extensions for the "flat" Naive Bayes algorithm, that relaxes the Naive Bayes strong assumption about attribute independence. The section presents the concept of unrestricted and class-constrained Bayesian network classifiers. It then presents two novel global–model hierarchical classification Bayesian network algorithms. The two proposed global–model hierarchical classification algorithms are then compared to two LCPN approaches.

Chapter 6 presents the conclusions of this thesis and discusses future research directions.

1.5 Publications Derived From This Research

The following list of publications has resulted from the research presented in this thesis, comprising works published and accepted for publication in the scientific literature:

- C. N. Silla Jr. and A. A. Freitas. A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery. Vol. 22, No. 1–2, pp. 31–72, 2011.
- C. N. Silla Jr. and A. A. Freitas. Selecting different protein representations and classification algorithms in hierarchical protein function prediction. Intelligent Data Analysis Journal. Vol. 15, No. 6, pp. 979–999, 2011.
- C. N. Silla Jr. and A. A. Freitas. A Global-Model Naive Bayes Approach to the Hierarchical Prediction of Protein Functions. Proc. of the IEEE

International Conference on Data Mining (ICDM). Miami, FL, USA, pp. 992–997, December 2009.

- C. N. Silla Jr. and A. A. Freitas. Novel Top-Down Approaches for Hierarchical Classification and Their Application to Automatic Music Genre Classification. Proc. of the IEEE International Conference on Systems, Man and Cybernetics (SMC). San Antonio, TX, USA, pp. 3499–3504, October 2009.

Chapter 2

Hierarchical Classification

This chapter reviews in detail concepts and approaches for hierarchical classification, which is the machine learning task (type of problem) addressed in this thesis. Before discussing the task of hierarchical classification, we present a brief discussion of the flat classification task, also known as supervised pattern recognition or supervised learning. Most of this chapter's contents has been published in a survey of hierarchical classification published in (Silla Jr. and Freitas, 2011b).

2.1 Pattern Recognition Overview

The pattern recognition (PR) task has the objective of giving labels to objects. Since the objects cannot be given as they are to computers, they need to be translated/measured in a way that the computer can handle them. In PR this is done by extracting features (which are measurements/descriptions) of the objects. Because pattern recognition is faced with the challenges of solving real-life problems, in spite of decades of productive research, elegant modern theories still coexist with ad hoc ideas, intuition and guessing (Kuncheva, 2004).

Figure 2.1 shows an overview of the PR task. Consider a hypothetic user who presents us with the problem and a set of data. Our task is to clarify the problem, translate it into pattern recognition terminology, solve it, and communicate the solution back to the User (Kuncheva, 2004).

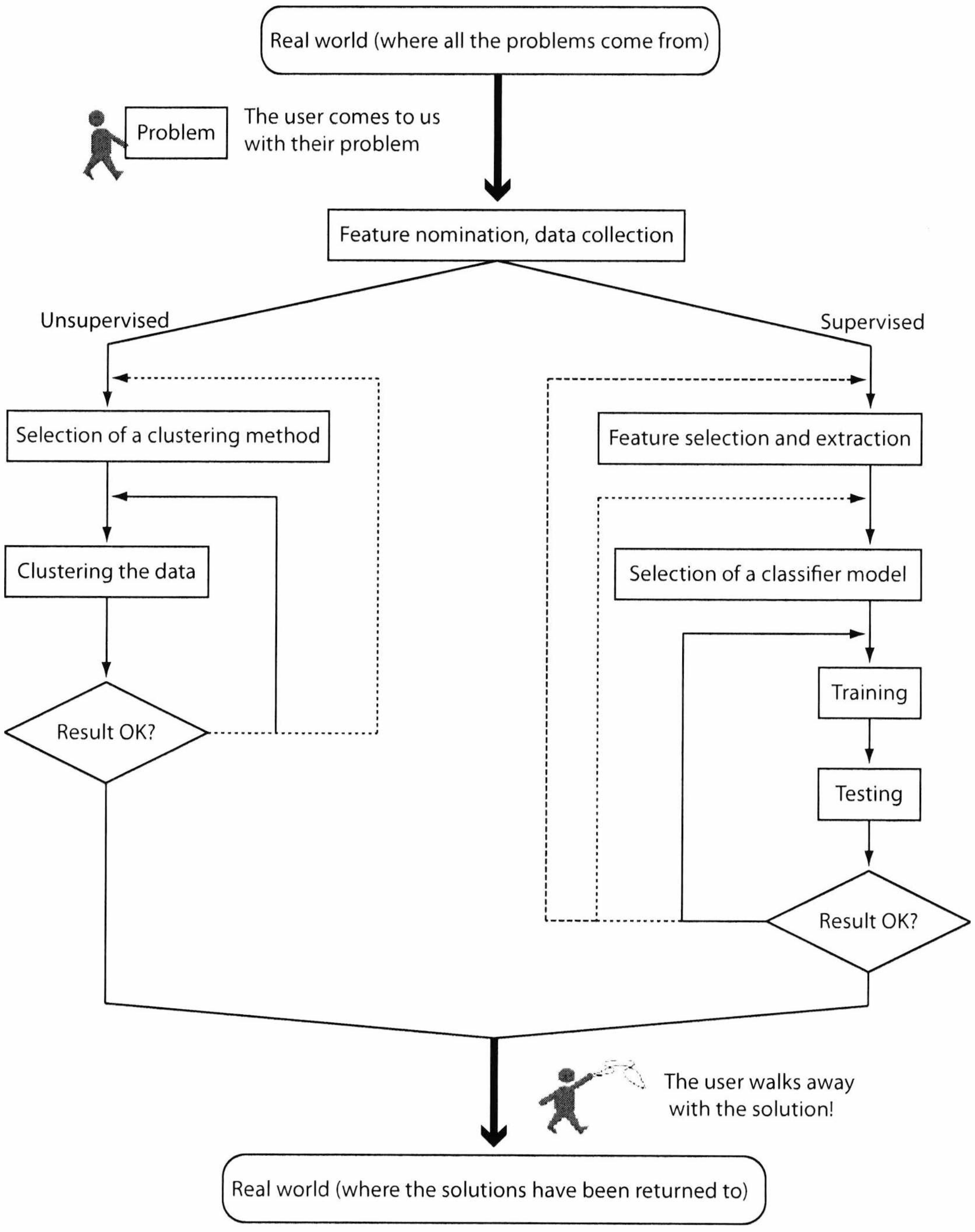

Real world (where all the problems come from)
Problem
The user comes to us with their problem
Feature nomination, data collection
Unsupervised
Supervised
Selection of a clustering method
Clustering the data
Result OK?
Feature selection and extraction
Selection of a classifier model
Training
Testing
Result OK?
The user walks away with the solution!
Real world (where the solutions have been returned to)

Figure 2.1: Overview of the pattern recognition task (Kuncheva, 2004).

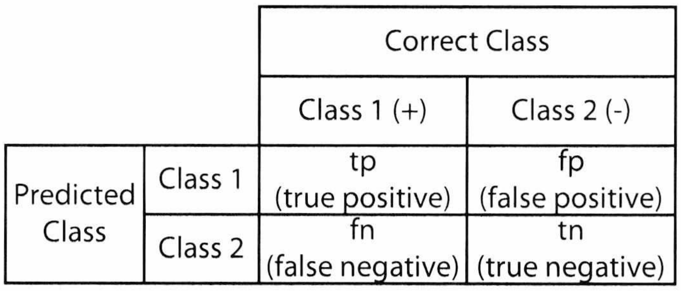
As illustrated in Figure 2.1 there are two main types of problems in pattern recognition: supervised (also known as classification) and unsupervised (also known as clustering). In a supervised learning problem, broadly speaking, the task is given a set of objects (represented by their feature vectors) whose labels are known, to train a classification model and use it to predict the class of objects whose labels are unknown. In an unsupervised learning problem, the task is given a set of unlabelled objects (represented by their feature vectors), and, broadly speaking, the goal is to learn about their structure and relationships.

In this thesis we are interested in supervised learning problems and unsupervised learning is out of the scope of the thesis. Considering the supervised learning stages in Figure 2.1, the first stage is feature selection and extraction. In this work we are not focusing on feature selection and the process of feature extraction will be explained where applicable as there are several approaches to extract features for different applications. As it will be seen in chapter 3 for the task of protein function prediction there are several ways of extracting features from proteins, for example.

The second stage consists of choosing an algorithm to create the classification model (classifier). There are several algorithms that can be used with very different underlying principles. In order for the algorithm to build a classification model, it needs data. At this stage, the dataset is usually split into a training set and a test set.

In the third stage, the training set is used to train a classifier, building the classification model. In the fourth stage, the test set (which has examples not present in the training set) is used to verify how well the built classification model performs.

In order to measure how well the model built performed on the test set, there are several evaluation measures in the machine learning community (Sokolova and Lapalme, 2009). These measures are always defined over a confusion matrix. Figure 2.2 shows an example of a confusion matrix for a two-class problem.



		Correct Class	
		Class 1 (+)	Class 2 (-)
Predicted Class	Class 1	tp (true positive)	fp (false positive)
	Class 2	fn (false negative)	tn (true negative)

Figure 2.2: An example of a confusion matrix.

One commonly employed evaluation measure is the f-measure. The f-measure is defined as: F-measure $= 2 \times \frac{precision \times recall}{precision + recall}$. Where Precision $= \frac{tp}{tp+fp}$ and Recall $= \frac{tp}{tp+fn}$. As precision and recall are defined only over a two-class confusion matrix, in the case of a multiclass problem, the weighted mean (by taking into account the number of examples of each class) over the confusion matrix of all classes are used. In this case for each class, the confusion matrix considers as positive examples, the examples of the class and as negative examples the examples from all the other classes.

Usually, to have a more reliable estimate of the predictive performance of the classifiers, instead of using only one split of the available data into training and test sets, the dataset is divided into 10 folds with approximately the same number of examples of each class in each fold (stratified splitting) and the experiments are run 10 times, by using an alternating 9 folds to train and 1 to test. This procedure is known as stratified 10-fold cross-validation.

It should be noted that conventional classification algorithms can only cope with flat classes (where there is no parent-child relationships among the classes to be predicted), but many real-world classification problems naturally have classes arranged into a hierarchical structure. This motivates research on hierarchical classification, as discussed in the remainder of this chapter.

2.2 Hierarchical Classification

As discussed in Chapter 1, any classification problem with a class taxonomy that satisfies the following four properties can be considered a hierarchical classification problem, and in general the hierarchical classification methods surveyed in this work assume (explicitly or implicitly) the underlying class structure satisfies those properties.

The four properties of a hierarchical classification problem are that the only one greatest element "R" is the root of the tree and the "IS-A" relationship is asymmetric, anti-reflexive and transitive.

As seen in Section 2.2, according to (Freitas and de Carvalho, 2007; Sun and Lim, 2001) hierarchical classification methods differ in a number of criteria. These criteria are the type of hierarchical structure used, how deep the classification in the hierarchy is performed and how the hierarchical structure is explored. However, a closer look at the existing hierarchical classification methods reveals that:

1. The top-down approach is not a full hierarchical classification approach by itself, but rather a method for avoiding or correcting inconsistencies in class prediction at different levels, during the testing (rather than training) phase;

2. There are different ways of using local information to create local classifiers, and although most of them are referred to as top-down in the literature, they are very different during the training phase and slightly different in the test phase;

3. Big-bang (or global) classifiers are trained by considering the entire class hierarchy at once, and hence they lack the kind of modularity for local training of the classifier that is a core characteristic of the local classifier approach.

These are the main points which will be discussed in detail in this chapter.

2.3 Flat Classification Approach

The flat classification approach, which is the simplest one to deal with hierarchical classification problems, consists of completely ignoring the class hierarchy, typically predicting only classes at the leaf nodes. This approach behaves like a traditional classification algorithm during training and testing. However, it provides an indirect solution to the problem of hierarchical classification, because, when a leaf class is assigned to an example, one can consider that all its ancestor classes are also implicitly assigned to that instance (recall that we assume a "IS-A" class hierarchy).

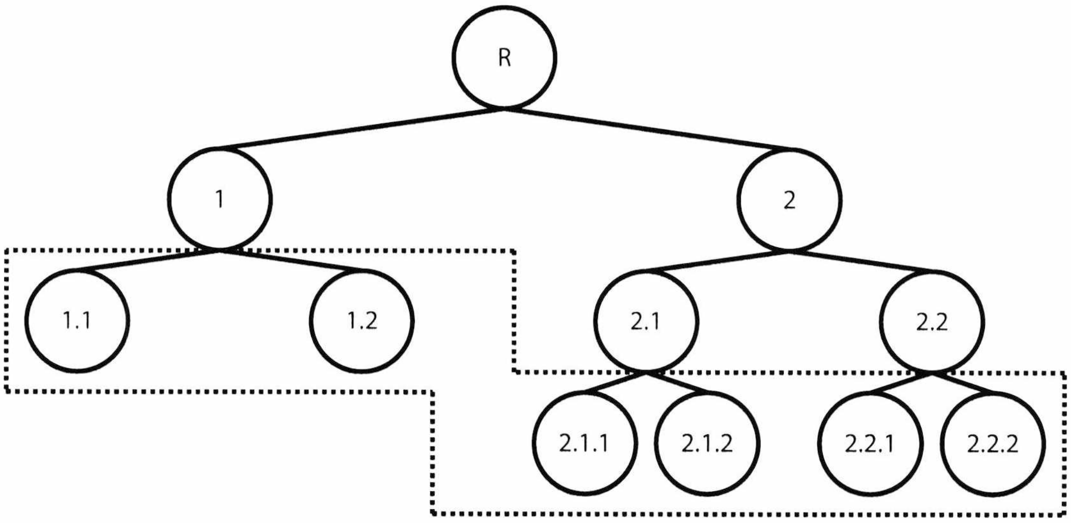


Figure 2.3: Flat classification approach using a flat multi-class classification algorithm to always predict the leaf nodes.

However, this very simple approach has the serious disadvantage of having to build a classifier to discriminate among a large number of classes (all leaf classes), without exploring information about parent-child class relationships present in the class hierarchy. Figure 2.3 illustrates this approach. We use here the term flat classification approach, as it seems to be the most commonly used term in the existing literature, although in (Burred and Lerch, 2003) the authors refer to this approach as "the direct approach", while in (Xiao et al., 2007) this approach is referred to as a "global classifier" – which is misleading as they are referring to this naive flat classification algorithm, and the term global classifier is often used to refer to the "big-bang" approach (Section 2.5).

In (Barbedo and Lopes, 2007) the authors refer to this approach as a "bottom-up" approach. They justify this term as follows: "The signal is firstly classified according to the basic genres, and the corresponding upper classes are consequences of this first classification (bottom-up approach)." In this chapter, however, we prefer to use the term flat classification to be consistent with the majority of the literature.

Considering the different types of class taxonomies (tree or DAG), this approach can cope with both of them as long as the problem is a mandatory-leaf node prediction problem, as it is incapable of handling non-mandatory leaf node prediction problems. In this approach training and testing proceed in the same way as in standard (non-hierarchical) classification algorithms.

2.4 Local Hierarchical Classification Approaches

In the seminal work of (Koller and Sahami, 1997), the first type of local classifier approach (also known as top-down approach in the literature) was proposed. From this work onwards, many different authors used augmented versions of this approach to deal with hierarchical classification problems. However, the important aspect here is not that the approach is top-down (as it is commonly called), but rather that the hierarchy is taken into account by using a local information perspective. The idea behind this reasoning is that in the literature there are several papers that employ this local information in different ways. These approaches, therefore, can be grouped based on how they use this local information and how they build their classifiers around it. More precisely, there seems to exist three standard ways of using the local information: a local classifier per node, a local classifier per parent node and a local classifier per level. In the following subsections we discuss each one of them in detail. Also note that unless specified otherwise, the discussion will assume a tree-structured class hierarchy with a single class label per class level (single-label hierarchical classification for short) and mandatory leaf node prediction.

It should be noted that, although the three types of local hierarchical classification algorithms discussed in the next three sub-sections differ significantly in their training phase, they share a very similar top-down approach in their testing phase. In essence, in this top-down approach, for each new example in the test set, the system first predicts its first-level (most generic) class, then it uses that predicted class to narrow the choices of classes to be predicted at the second level (the only valid candidate second-level classes are the children of the class predicted at the first level), and so on, recursively, until the most specific prediction is made.

As a result, a disadvantage of the top-down class-prediction approach (which is shared by all the three types of local classifiers discussed next) is that an error at a certain class level is going to be propagated downwards the hierarchy, unless some procedure for avoiding this problem is used. If the problem is non-mandatory leaf node prediction, a blocking approach (where an example is passed down to the next lower level only if the confidence on the prediction at the current level is greater than a threshold) can avoid that misclassifications are propagated downwards, at the expense of providing the user with less specific (less useful) class predictions. Some authors use methods to give better estimates of class probabilities, like shrinkage (McCallum et al., 1998) and isotonic smoothing (Punera and Ghosh, 2008). The issues of non-mandatory leaf node prediction and blocking are discussed in Section 2.4.4.

2.4.1 Local Classifier Per Node Approach

This is by far the most used approach in the literature. It often appears under the name of a top-down approach, but as we mentioned earlier, we shall see why this is not a good name as the top-down approach is essentially a method to avoid inconsistencies in class predictions at different levels in the class hierarchy. The local classifier per node approach consists of training one binary classifier for each node of the class hierarchy (except the root node). Figure 2.4 illustrates this approach.

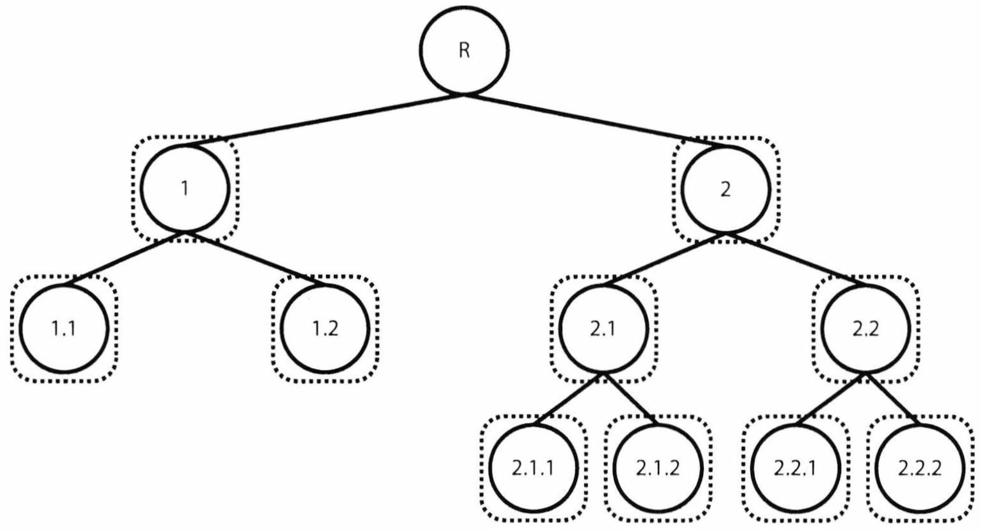


Figure 2.4: Local classifier per node approach (circles represent classes and dashed squares with round corners represent binary classifiers).

There are different ways to define the set of positive and negative examples for training the binary classifiers. In the literature most works often use one approach and studies like (Eisner et al., 2005; Fagni and Sebastiani, 2007) where different approaches are compared are not common. In the work of (Eisner et al., 2005) the authors identify and experiment with four different policies to defining the set of positive and negative examples. In (Fagni and Sebastiani, 2007) the authors focus on the selection of the negative examples and empirically compare four policies (two standard ones compared with two novel ones). However the novel approaches are limited to text categorization problems and achieved similar results to the standard approaches; and for that reason they are not further discussed in this thesis. The notation used to define the sets of positive and negative examples is based on the one used in (Fagni and Sebastiani, 2007) and is presented in Table 2.1.

Table 2.1: Notation for negative and positive training examples.

Symbol	Meaning
Tr	the set of all training examples
$Tr^+(c_j)$	the set of positive training examples of c_j
$Tr^-(c_j)$	the set of negative training examples of c_j
$\uparrow(c_j)$	the parent category of c_j
$\downarrow(c_j)$	the set of children categories of c_j
$\Uparrow(c_j)$	the set of ancestor categories of c_j
$\Downarrow(c_j)$	the set of descendant categories of c_j
$\leftrightarrow(c_j)$	the set of sibling categories of c_j
$*(c_j)$	denotes examples whose most specific known class is c_j

- The "exclusive" policy (as defined by (Eisner et al., 2005)): $Tr^+(c_j) = *(c_j)$ and $Tr^-(c_j) = Tr \setminus *(c_j)$. This means that only examples explicitly labelled as c_j as their most specific class are selected as positive examples, while everything else is used as negative examples. For example, using Fig. 2.4, for $c_j = 2.1$, $Tr^+(c_{2.1})$ consists of all examples whose most specific class is 2.1; and $Tr^-(c_{2.1})$ consists of the set of examples whose most specific class is 1, 1.1, 1.2, 2, 2.1.1, 2.1.2, 2.2, 2.2.1 or 2.2.2. This approach has a few problems. First, it does not consider the hierarchy to create the local training sets. Second, it is limited to problems where partial depth labeling instances are available. By partial depth labeling instances we mean instances whose class label is known just for shallower levels of the hierarchy, and not for deeper levels. Third, using the descendant nodes of c_j as negative examples seems counter-intuitive considering that examples who belong to class $\Downarrow(c_j)$ also implicitly belong to class c_j according to the "IS-A" hierarchy concept.

- The "less exclusive" policy (as defined by (Eisner et al., 2005)): $Tr^+(c_j) = *(c_j)$ and $Tr^-(c_j) = Tr \setminus *(c_j) \cup \Downarrow(c_j)$. In this case, using Fig. 2.4 as example, $Tr^+(c_{2.1})$ consists of the set of examples whose most specific class is 2.1; and $Tr^-(c_{2.1})$ consists of the set of examples whose most specific class is 1, 1.1, 1.2, 2, 2.2, 2.2.1 or 2.2.2. This approach avoids the aforementioned first and third problems of the exclusive policy, but it is still limited to problems where partial depth labeling instances are available.

- The "less inclusive" policy (as defined by (Eisner et al., 2005), it is the same as the "ALL" policy defined by (Fagni and Sebastiani, 2007)): $Tr^+(c_j) = *(c_j) \cup \Downarrow (c_j)$ and $Tr^-(c_j) = Tr \setminus *(c_j) \cup \Downarrow (c_j)$. In this case $Tr^+(c_{2.1})$ consists of the set of examples whose most specific class is 2.1, 2.1.1 or 2.1.2; and $Tr^-(c_{2.1})$. consists of the set of examples whose most specific class is 1, 1.1, 1.2, 2, 2.2, 2.2.1 or 2.2.2.

- The "inclusive" policy (as defined by (Eisner et al., 2005)): $Tr^+(c_j) = *(c_j) \cup \Downarrow (c_j)$ and $Tr^-(c_j) = Tr \setminus *(c_j) \cup \Downarrow (c_j) \cup \Uparrow (c_j)$. In this case $Tr^+(c_{2.1})$ is the set of examples whose most specific class is 2.1,2.1.1 or 2.1.2; and $Tr^-(c_{2.1})$ consists of the set of examples whose most specific class is 1,1.1,1.2,2.2,2.2.1 or 2.2.2.

- The "siblings" policy (as defined by (Fagni and Sebastiani, 2007), and which (Ceci and Malerba, 2007) refers to as "hierarchical training sets"): $Tr^+(c_j) = *(c_j) \cup \Downarrow (c_j)$ and $Tr^-(c_j) = \leftrightarrow (c_j) \cup \Downarrow (\leftrightarrow (c_j))$. In this case $Tr^+(c_{2.1})$ consists of the set of examples whose most specific class is 2.1,2.1.1 or 2.1.2; and $Tr^-(c_{2.1})$ consists of the set of examples whose most specific class is 2.2,2.2.1,2.2.2.

- The "exclusive siblings" policy (as defined by (Ceci and Malerba, 2007) and referred to as "proper training sets"): $Tr^+(c_j) = *(c_j)$ and $Tr^-(c_j) = \leftrightarrow (c_j)$. In this case $Tr^+(c_{2.1})$ consists of the set of examples whose most specific class is 2.1; and $Tr^-(c_{2.1})$ consists of the set of examples whose most specific class is 2.2.

It should be noted that in the aforementioned policies for negative and positive training examples, we have assumed that the policies defined in (Fagni and Sebastiani, 2007) follow the usual approach of using as positive training examples all the examples belonging to the current class node ($*(c_j)$) and all of its descendant classes ($\Downarrow (c_j)$). Although this is the most common approach, several other approaches can be used, as shown by (Eisner et al., 2005). In particular, the exclusive and less exclusive policies use as positive examples only the examples whose most specific class is the current class, without using the examples whose most specific class is a descendant from the current class in the hierarchy. It should be noted that the aim of the work of (Eisner et al., 2005) was to evaluate different ways of creating the positive and negative training sets for predicting functions based on the Gene Ontology, but it seems that they overlooked the use of the siblings policy which is common in the hierarchical text classification domain. Given the above discussion, one can see that it is important that authors be clear on how they select both positive and negative examples in the local hierarchical classification approach, since so many ways of defining positive and negative examples are possible, with subtle differences between some of them.

Concerning which approach one should use, (Eisner et al., 2005) note that as the classifier becomes more inclusive (with more positive training examples) the classifiers perform better. Their results (using F-measure (defined in Section 2.1) as a measure of performance) comparing the different measures are: Exclusive: 0.456, Less Exclusive: 0.528, Less Inclusive: 0.696 and Inclusive: 0.697. In the experiments of (Fagni and Sebastiani, 2007), where they compare the siblings and less-inclusive policies, concerning predictive accuracy there is no clear winner. However, they note that the siblings policy uses considerably less data in comparison with the less-inclusive policy, and since they have the same accuracy, that is the one that should be used. In any case, more research, involving a wider variety of datasets, would be useful to better characterise the relative strengths and weakness of the aforementioned different policies in practice.

During the testing phase, regardless of how positive and negative examples were defined, the output of each binary classifier will be a prediction indicating whether or not a given test example belongs to the classifier's predicted class. One advantage of this approach is that it is naturally multi-label in the sense that it is possible to predict multiple labels per class level, in the case of multi-label problems. Such a natural multi-label prediction is achieved using just conventional single-label classification algorithms, avoiding the complexities associated with the design of a multi-label classification algorithm (Tsoumakas and Katakis, 2007). In the case of single-label (per level) problems one can enforce the prediction of a single class label per level by assigning to a new test example just the class predicted with the greatest confidence among all classifiers at a given level – assuming classifiers output a confidence measure of their prediction. This approach has, however, a disadvantage. Considering the example of Figure 2.4 it would be possible, using this approach, to have an output like Class 1 = false and Class 1.2 = true (since the classifiers for nodes 1 and 1.2 are independently trained), which leads to an inconsistency in class predictions across different levels. Therefore, if no inconsistency correction method is taken into account, this approach is going to be prone to class-membership inconsistency.

As mentioned earlier, one of the current misconceptions in the literature is the confusion between local information-based training of classifiers and the top-down approach for class prediction (in the testing phase). Although they are often used together, the local information-based training approach is not necessarily coupled with the top-down approach, as a number of different inconsistency correction methods can be used to avoid class-membership inconsistency during the test phase. Let us now review the existing inconsistency correction methods for the local classifier per node approach.

The class-prediction top-down approach seems to have been originally proposed by (Koller and Sahami, 1997), and its essential characteristic is that it consists of performing the *testing* phase in a top-down fashion, as follows. For each level of the hierarchy (except the top level), the decision about which class is predicted at the current level is based on the class predicted at the previous (parent) level. For example, at the top level, suppose the output of the local classifier for class 1 is *true*, and the output of the local classifier for class 2 is *false*. At the next level, the system will only consider the output of classifiers predicting classes which are children of class 1. Originally, the class-prediction top-down method was forced to always predict a leaf node (Koller and Sahami, 1997). When considering a non-mandatory leaf-node prediction problem, the class-prediction top-down approach has to use a stopping criterion that allows an example to be classified just up to a non-leaf class node. This extension might lead to the *blocking* problem, which will be discussed in Section 2.4.4.

Besides the class-prediction top-down approach, other methods were proposed to deal with inconsistencies generated by the local classifier per node approach. One such method consists of stopping the classification once the binary classifier for a given node gives the answer that the unseen example does not belong to that class. For example, if the output for the binary classifier of class 2 is *true*, and the outputs of the binary classifiers for classes 2.1 and 2.2 are *false*, then this approach would ignore the answer of all the lower level classifiers predicting classes that are descendant of classes 2.1 and 2.2 and output the class 2 to the user. By doing this, the class predictions respect the hierarchy constraints. This approach was proposed by (Wu et al., 2005) and was referred to as "Binarized Structured Label Learning" (BSLL).

In (Dumais and Chen, 2000) the authors propose two class-membership inconsistency correction methods based on thresholds. In order for a class to be assigned to a test example, the probabilities for the predicted class were used. In the first method, they use a boolean condition where the posterior probability of the classes at the first and second levels must be higher than a user specified threshold, in the case of a two-level class hierarchy. The second method uses a multiplicative threshold that takes into account the product of the posterior probability of the classes at the first and second levels. For example, let us suppose that, for a given test example, the posterior probability for each class in the first two levels in Figure 2.4 were: $p(c_1) = 0.6$, $p(c_2) = 0.2$, $p(c_{1.1}) = 0.55$, $p(c_{1.2}) = 0.1$, $p(c_{2.1}) = 0.2$, $p(c_{2.2}) = 0.3$. Considering a threshold of 0.5, by using the boolean rule the classes predicted for that test example would be class 1 and class 1.1 as both classes have a posterior probability higher than 0.5. By using the multiplicative threshold, the example would be assigned to class 1 but not class 1.1, as the posterior probability of class 1 $\times$ the posterior probability of class 1.1 is 0.33, which is below the multiplicative threshold of 0.5.

In the work of (Barutcuoglu and DeCoro, 2006; Barutcuoglu et al., 2006; DeCoro et al., 2007) another class-membership inconsistency correction method for the local classifier per node approach is proposed. Their method is based on a Bayesian aggregation of the output of the base binary classifiers. The method takes the class hierarchy into account by transforming the hierarchical structure of the classes into a Bayesian network. In (Barutcuoglu and DeCoro, 2006) two baseline methods for conflict resolution are proposed: the first method propagates negative predictions downward (i.e. the negative prediction at any class node is used to overwrite the positive predictions of its descendant nodes) while the second baseline method propagates the positive predictions upward (i.e. the positive prediction at any class node is used to overwrite the negative predictions of all its ancestors). Note that the first baseline method is the same as the BSLL.

Another approach for class-membership inconsistency correction based on the output of all classifiers has been proposed in (Valentini, 2009), where the basic idea is that by evaluating all the classifier nodes' outputs it is possible to make consistent predictions by computing a "consensus" probability using a bottom-up algorithm.

(Xue et al., 2008) proposes a strategy based on pruning the original hierarchy. The basic idea is that when a new document is going to be classified it can possibly be related to just some of the many hierarchical classification classes. Therefore, in order to reduce the error of the top-down class-prediction approach, their method first computes the similarity between the new document and all other documents, and creates a pruned class hierarchy which is then used in a second stage to classify the document using a top-down class-prediction approach.

(Bennett and Nguyen, 2009) proposes a technique called expert refinements. The refinement consists of using cross-validation in the training phase to obtain a better estimation of the true probabilities of the predicted classes. The refinement technique is then combined with a bottom-up training approach, which consists of training the leaf classifiers using refinement and passing this information to the parent classifiers.

So far we have discussed the local classifier per node approach mainly in the context of a single label (per level) problem with a tree-structured class hierarchy. In the multi-label hierarchical classification scenario (where an example can be labelled with several classes at the same level of the class hierarchy), this approach is still directly employable, but some more sophisticated method to cope with the different outputs of the classifiers should be used. For example, in (Esuli et al., 2008) the authors propose the TreeBoost.MH which uses during training at each classification node the AdaBoost.MH base learner. Their approach can also (optionally) perform feature selection by using information from the sibling classes. In the context of a DAG, the local classifier per node approach can still be used in a natural way as well, as it has been done in (Jin et al., 2008; Otero et al., 2009).

2.4.2 Local Classifier Per Parent Node Approach

Another type of local information that can be used, and it is also often referred to as top-down approach in the literature, is the approach where, for each parent node in the class hierarchy, a multi-class classifier (or a problem decomposition approach with binary classifiers like One-Against-One scheme for Binary SVMs) is trained to distinguish between its child nodes. Figure 2.5 illustrates this approach.

In order to train the classifiers the "siblings" policy, as well as the "exclusive siblings" policy, both presented in Section 2.4.1, are suitable to be used.

During the testing phase, this approach is often coupled with the top-down class prediction approach, but this coupling is not necessarily a must, as new class prediction approaches for this type of local approach could be developed. Consider the top-down class-prediction approach and the same class tree example of Figure 2.5, and suppose that the first level classifier assigns the example to the class 2. The second level classifier, which was only trained with the children of the class node 2, in this case 2.1 and 2.2, will then make its class assignment (and

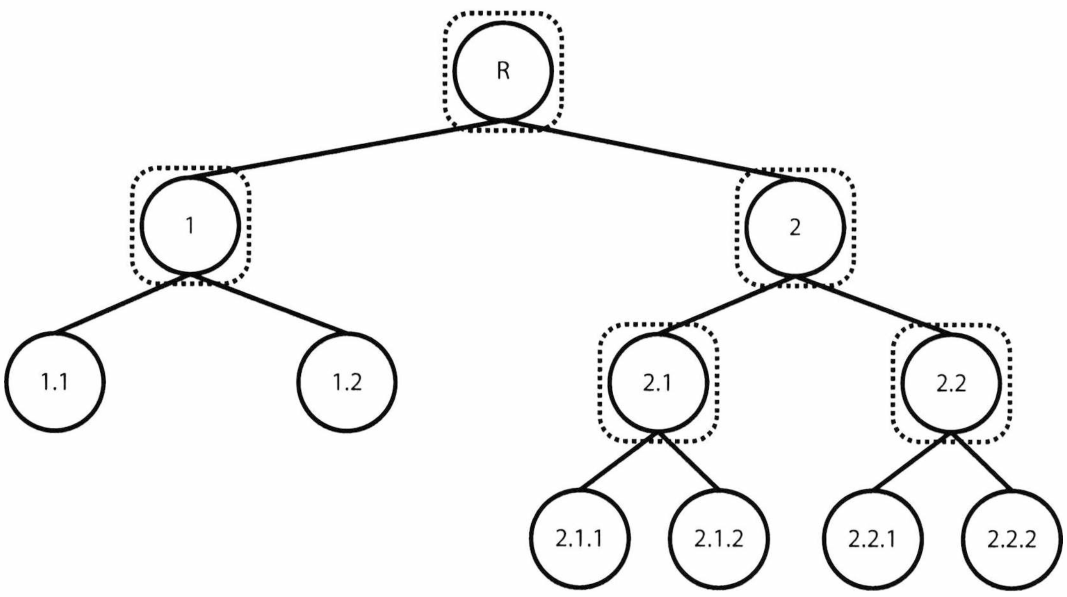


Figure 2.5: Local classifier per parent node (circles represent classes and dashed squares with round corners in parent nodes represent multi-class classifiers – predicting their child classes).

so on, if deeper-level classifiers were available), therefore avoiding the problem of making inconsistent predictions and respecting the natural constrains of class membership.

An extension of this type of local approach known as the "selective classifier" approach was proposed by (Secker et al., 2007). The authors refer to this method as the Selective Top-Down approach, but it is here re-named to "selective classifier" approach to emphasize that what are being selected are the classifiers, rather than attributes as in attribute (feature) selection methods. In addition, we prefer to reserve the term "top-down" to the class prediction method during the testing phase, as explained earlier. Usually, in the local classifier per parent node approach the same classification algorithm is used throughout all the class hierarchy. In (Secker et al., 2007), the authors hypothesise that it would be possible to improve the predictive accuracy of the local classifier per parent node approach by using different classification algorithms at different parent nodes of the class hierarchy. In order to determine which classifier should be used at each node of the class hierarchy, during the training phase, the training set is split into a sub-training and validation set with examples being assigned randomly to each of those datasets. Different classifiers are trained using that sub-training set and are then evaluated on the validation set. The classifier chosen for each parent class node is the one with the highest classification accuracy on the validation set. An improvement over the selective classifier approach was proposed by (Holden and Freitas, 2008), where a swarm intelligence optimization algorithm was used to perform the classifier selection. The motivation behind this approach is that the original selective classifier approach uses a greedy, local search method that has only a limited local view of the training data when selecting a classifier, while the swarm intelligence algorithm performs a global search that considers the entire tree of classifiers (having a complete view of the training data) at once. Another improvement over the selective classifier approach was proposed by (Silla Jr. and Freitas, 2009b), where both the best classifier and the best type of example representation (out of a few types of representations, involving different kinds of predictor attributes) are selected for each parent node classifier. In addition, (Secker et al., 2010) extended their previous classifier-selection approach in order to select both classifiers and attributes at each classifier node.

Recently in the work of (Carvalho et al., 2011) the authors proposed a local classifier per parent node method that use the top-down approach during training and testing of the classifier. The proposed method is called HCGA (Hierarchical Classification Genetic Algorithm). Their approach consists of creating the classification models in a top-down fashion, and providing the current classifier at level l as a starting point to the classifiers at level $l + 1$. Note that for each non-leaf class, starting from the root, a different classification model is built at each node at level l with the information from the previous level classifier at level $l - 1$, except for the root node classifier.

So far we have discussed the local classifier per parent node approach in the context of a single label problem with a tree-structured class hierarchy. Let us now briefly discuss this approach in the context of a multi-label problem. In this multi-label scenario, this approach is not directly employable. There are, at least, two approaches that could be used to cope with the multi-label scenario. One is to use a multi-label classifier at each parent node, as done by (Wu et al., 2005). The second approach is to take into account the different confidence scores provided by each classifier and have some kind of decision thresholds based on those scores to allow multiple labels. One way of doing this would be to adapt the multiplicative threshold proposed by (Dumais and Chen, 2000). When dealing with a DAG-structured class hierarchy, this approach is also not directly employable, as the created local training sets might be highly redundant (due to the the fact that a given class node can have multiple parents, which can be located at different depths). To the best of our knowledge this approach has not yet been used with DAG-structured class hierarchies.

2.4.3 Local Classifier Per Level Approach

This is the type of "local" (broadly speaking) classifier approach least used so far on the literature. The local classifier per level approach consists of training one multi-class classifier for each level of the class hierarchy. Figure 2.6 illustrates this approach. Considering the example of Figure 2.6, three classifiers would be trained, one classifier for each class level, where each classifier would be trained to predict one or more classes (depending on whether the problem is single-label or multi-label) at its corresponding class level. The creation of the training sets here is implemented in the same way as in the local classifier per parent node approach.

This approach has been mentioned as a possible approach by (Freitas and de Carvalho, 2007), but to the best of our knowledge its use has been limited as a baseline comparison method in (Clare and King, 2003) and (Costa et al., 2007b).

One possible (although very naïve) way of classifying test examples using classifiers trained by this approach is as follows. When a new test example is presented to the classifier, get the output of all classifiers (one classifier per level) and use this information as the final classification. The major drawback of this class-prediction

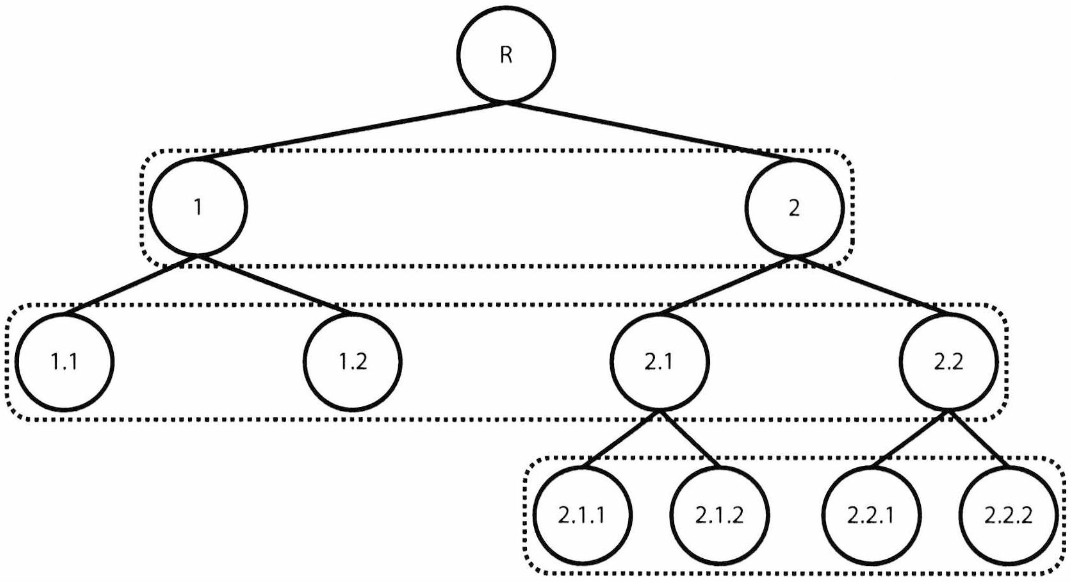


Figure 2.6: Local classifier per level (circles represent classes and each dashed rectangle with round corners encloses the classes predicted by a multi-class classifier).

approach is being prone to class-membership inconsistency. By training different classifiers for each level of the hierarchy it is possible to have outputs like Class 2 at the first level, Class 1.2 at the second level, and Class 2.2.1 at the third level, therefore generating inconsistency. Hence, if this approach is used, it should be complemented by a post-processing method that tries to correct the prediction inconsistency.

To avoid this problem, one approach that can be used is the class-prediction top-down approach. In this context, the classification of a new test example would be done in a top-down fashion (similar to the standard top-down class-prediction approach), restricting the possible classification output at a given level only to the child nodes of the class node predicted in the previous level (in the same way as it is done in the local classifier per parent node approach).

This approach could work with either a tree or a DAG class structure. Although depth is normally a tree concept, it could still be computed in the context of a DAG, but in the latter case this approach would be considerably more complex. This is because, since there can be more than one path between two nodes in a DAG, a class node can be considered as belonging to several class levels, and so there would be considerable redundancy between classifiers at different levels. In the context of a tree structured class hierarchy and multi-label problem, methods based on confidence scores or posterior probabilities could be used to make more than one prediction per class level.

2.4.4 Non-Mandatory Leaf Node Prediction and the Blocking Problem

In the previous sections, we discussed the different types of local classifiers but we avoided the discussion of the non-mandatory leaf node prediction problem. The non-mandatory leaf node prediction problem, as the name implies, allows the most specific class predicted to any given instance to be a class at any node (i.e. internal or leaf node) of the class hierarchy, and was introduced by (Sun and Lim, 2001). A simple way to deal with the non-mandatory leaf-node prediction problem is to use a threshold at each class node, and if the confidence score or posterior probability of the classifier at a given class node – for a given test example – is lower than this threshold, the classification stops for that example. A method for automatically computing these thresholds was proposed by (Ceci and Malerba, 2007).

The use of thresholds can lead to what (Sun et al., 2004) called the *blocking* problem. As briefly mentioned in Section 2.4.1, *blocking* occurs when, during the top-down process of classification of a test example, the classifier at a certain level in the class hierarchy predicts that the example in question does not have the class associated with that classifier. In this case the classification of the example will be "blocked", i.e., the example will not be passed to the descendants of that classifier. For instance, in Figure 1.1 blocking could occur, say, at class node 2, which would mean that the example would not be passed to the classifiers that are descendants of that node.

Three strategies to avoid blocking are discussed by (Sun et al., 2004): threshold reduction method, restricted voting method and extended multiplicative thresholds. These strategies were originally proposed to work together with two binary classifiers at each class node. The first classifier (which they call local classifier) determines if an example belongs to the current class node, while the second classifier (which they call sub-tree classifier) determines whether the example is going to be given to the current node's child-node classifiers or if the system should stop the classification of that example at the current node.

These blocking reduction methods work as follows:

- Threshold Reduction Method: This method consists of lowering the thresholds of the subtree classifiers. The idea behind this approach is that by reducing the thresholds this will allow more examples to be passed to the classifiers at lower levels. The challenge associated with this approach is how to determine the threshold value of each subtree classifier. This method can be easily used with both tree-structured and DAG-structured class hierarchies.

- Restricted Voting: This method consists of creating a set of secondary classifiers that will link a node and its grandparent node. The motivation for this approach is that, although the threshold reduction method is able to pass more examples to the classifiers at the lower levels, it is still possible to have examples wrongly rejected by the high-level subtree classifiers. Therefore, the restricted voting approach gives the low-level classifiers a chance to access these examples before they are rejected. This approach is motivated by ensemble-based approaches and the set of secondary classifiers are trained with a different training set than the original subtree classifiers. This method was originally designed for tree-structured class hierarchies and extending it to DAG-structured hierarchies would make it considerably more complex and more computationally expensive, as in a DAG-structured class hierarchy each node might have multiple parent nodes.

- Extended Multiplicative Thresholds: This method is a straightforward extension of the multiplicative threshold proposed by (Dumais and Chen, 2000) (explained in Section 2.4.1), which originally only worked for a 2-level hierarchy. The extension consists simply of establishing thresholds recursively for every two levels.

2.5 Global Classifier (or Big-Bang) Approach

Although the problem of hierarchical classification can be tackled by using the previously described local approaches, learning a single global model for all classes has the advantage that the total size of the global classification model is typically considerably smaller, by comparison with the total size of all the local models learned by any of the local classifier approaches. In addition, dependencies between different classes with respect to class membership (e.g. any example belonging to class 2.1 automatically belongs to class 2) can be taken into account in a natural, straightforward way, and may even be explicitated (Blockeel et al., 2002). This kind of approach is known as the big-bang approach, also called "global" learning. Figure 2.7 illustrates this approach.

In the global classifier 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 (Freitas and de Carvalho, 2007).

Originally in the work of (Sun and Lim, 2001) the authors stated that there were two approaches to hierarchical classification: top-down and big-bang. This statement has been followed by several works in the field until recently(Costa et al., 2007b; Secker et al., 2007; Alves et al., 2008; Xue et al., 2008). Based on the new perspective about the top-down approach discussed earlier, that approach is essentially a strategy for avoiding class-prediction inconsistencies across

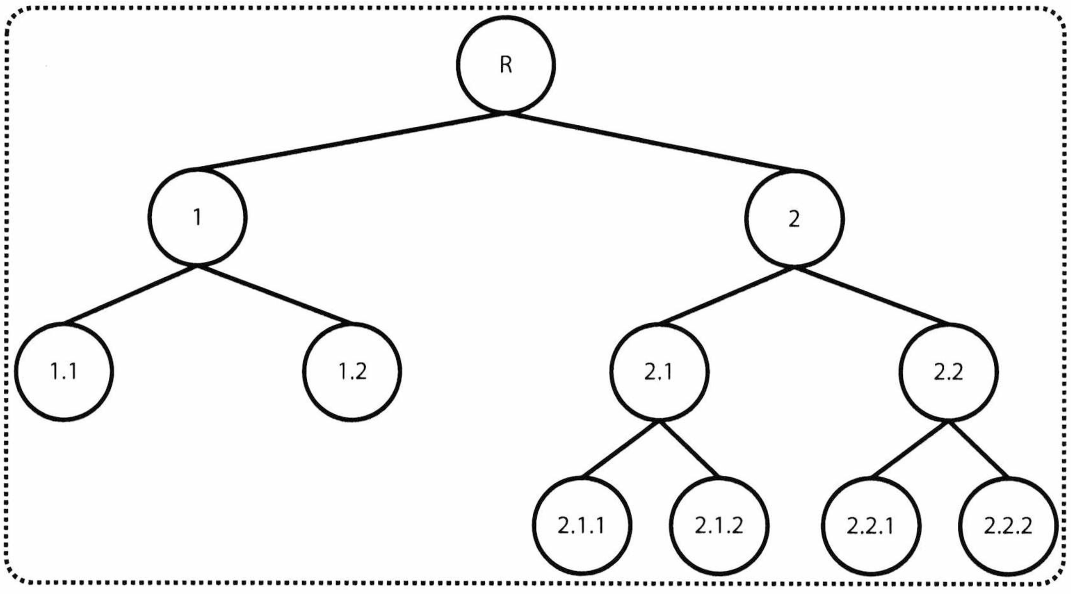


Figure 2.7: Big-Bang classification approach using a classification algorithm that learns a global classification model about the whole class hierarchy.

class levels during the testing (rather than training) phase, when using a local hierarchical classification method. However, we still lack a clear definition for the big-bang approach, as such definition has not been made so far. Usually, by a mutual exclusion criterion, any hierarchical classification method not considered top-down has been called big-bang.

Therefore, one of the contributions of this chapter is clarifying what kinds of approaches can be considered as global classifier approaches. Compared to the local classifier approaches, much less research has been reported using the global classifier approach. Although the latter approach has the advantage of learning, during the training phase, a global model for all the classes in a single step, it has an added complexity to it.

Although there seems to be no specific core characteristic shared by all global classifier approaches, in general global classifiers have two related broad characteristics, as follows. They consider the entire class hierarchy at once (as mentioned earlier) and they lack the kind of modularity for local training of the classifier that is a core characteristic of the local classifier approach. We emphasize that the crucial distinction between the global (big-bang) and local classifier approaches is in the training phase. A global classifier can even use a top-down approach (typically used by local classifier approaches) in its testing phase. In this latter case, as long as the classifier's training phase follows the two aforementioned broad characteristics of the global approach, it should be classified as a global (rather than local) classifier.

From the current state of the art, the main kinds of approaches that are usually considered to be global approaches are as follows. First, there is an approach based on the Rocchio classifier (Rocchio, 1971). This approach uses the idea of class clusters, where new examples are assigned to the nearest class by computing the distance between the new test example and each class.

One example of this approach is found in (Labrou and Finin, 1999). In this work the system classifies web pages into a subset of the Yahoo! hierarchical categories. This method is specific to text mining applications. During the testing phase each new document has its similarity computed with respect to each document topic. The final classification is given based on some threshold.

Another type of global classifiers is based on casting the hierarchical class problem as a multi-label classification problem (Kiritchenko et al., 2005, 2006). In order to be able to predict any class in the hierarchy, the training phase is modified to take into account all the classes in the hierarchy, by augmenting the non-leaf nodes with the information of its ancestor classes. During the test phase, since the algorithm does not take the hierarchy into account, it may suffer from the same limitations of the local classifier per node, that is, it is prone to class-prediction inconsistency. For this reason, in the approach of (Kiritchenko et al., 2005, 2006), the authors have a post-processing step, which takes all the outputs into account in order to ensure that the hierarchical constrains are respected.

Another type of global classifiers consists of modifying existing classifiers to directly cope with the class hierarchy and benefit from this additional information.

Global classifiers of this type are heavily specific to the underlying flat classification algorithm, as the original classification algorithms are modified in some way to take into account the entire class hierarchy. This might represent a disadvantage when compared to the local classifier approaches, which are not specific to a classification algorithm and can be augmented in a number of different ways. However, to the user, the output of a global classifier approach might be easier to understand/interpret than the one from a local classifier approach, due to the typically much smaller size of the classification model produced by the former approach, as mentioned earlier. This is the case for instance in (Vens et al., 2008), where the number of rules generated by the global approach is much smaller than the number of rules generated by the local approaches used in their experiments. Also, the global classifier approach does not suffer from the major drawback of the local classifier approach, namely the fact that a misclassification at a given class level is propagated to the lower levels of the class hierarchy. Different modifications of the base flat classification algorithms have been proposed by different authors, as follows.

In (Wang et al., 2001) an association rule mining algorithm is heavily modified in order to handle hierarchical document categorization. The main modification was to make the algorithm work with a set of labels instead of a single label.

In (Clare and King, 2003) a modified version of the decision tree algorithm C4.5 to handle the class hierarchy (HC4.5) was used. However, there are few details available about how this algorithm is different from the standard C4.5. The only information the authors provide is that they modified the entropy calculation formula to consider some form of weighting. It seems that, other things being equal, deeper nodes are preferred over shallower ones, because deeper nodes provide more specific class predictions to users. In (Silla Jr. and Freitas, 2009a) the authors have used the same principle to create a global-model Naive Bayes classifier.

In (Blockeel et al., 2002, 2006; Vens et al., 2008) the authors present the Clus-HMC algorithm, which is based on predictive cluster trees. The main idea of the

Table 2.2: Global classifier approaches and their underlying flat counterpart.

Base Algorithm	Global Approach
Ant-Miner	Otero et al. (2009)
Association Rule-based Classifier	Wang et al. (2001)
C4.5	Clare and King (2003)
Naive Bayes	Silla Jr. and Freitas (2009a)
Predictive Clustering Trees	Blockeel et al. (2006); Vens et al. (2008)
Kernel Machines	Cai and Hofmann (2004, 2007); Dekel et al. (2004a,b); Rousu et al. (2005, 2006); Seeger (2008); Qiu et al. (2009); Wang et al. (2009)

method is to build a set of classification trees to predict a set of classes, instead of only one class. To do this, the authors transform the classification output into a vector with boolean components corresponding to the possible classes. They also need to take into account some sort of distance-based metric to calculate how similar or dissimilar the training examples are in the classification tree. Originally the metric used was the weighted Euclidian Distance. In the work of (Aleksovski et al., 2009) the authors investigated the use of other distance measures, namely the Jaccard distance, the SimGIC distance and the ImageClef distance. They concluded that there was no statistically significant difference between the different distance metrics. Also, in (Dimitrovski et al., 2008) the authors have proposed the use of two ensembles approaches (bagging and random forests) applied to the Clus-HMC algorithm and concluded that the use of ensembles improves the classification accuracy.

In (Otero et al., 2009) the authors proposed the hAnt-Miner algorithm, a global-model hierarchical Ant-Miner classification method (a type of swarm intelligence method based on the paradigm of ant colony optimization) to cope with DAGs.

Table 2.2 lists the original flat classification algorithm and which authors have modified it in order to create global classification approaches.

2.6 A Unifying Framework for Hierarchical Classification

Based on our discussion so far, there are very many types of hierarchical classification algorithms and a number of different types of hierarchical classification problems. Hence, there is a clear need for a more precise way of describing (using a standardized terminology as much as possible) which kind of hierarchical classification problem is being solved, and what are the characteristics of the hierarchical classification algorithm being used. For this reason, in this section we propose a unifying framework for hierarchical classification problems and algorithms.

2.6.1 Categorization of the Different Types of Hierarchical Classification Problems

In the proposed framework a hierarchical classification problem is described as a 3-tuple $< \Upsilon, \Psi, \Phi >$, where:

- Υ specifies the type of graph representing the hierarchical classes (nodes in the graph) and their interrelationships (edges in the graph). The possible values for this attribute are:
    - T (tree), indicating that the classes to be predicted are arranged into a tree structure;
    - D (DAG), indicating that the classes to be predicted are arranged into a DAG (Directed Acyclic Graph).
- Ψ indicates whether a data instance is allowed to have class labels associated with a single or multiple paths in the class hierarchy. For instance, in the tree-structured class hierarchy of Fig. 2.4, if there is a data instance whose most specific labels are, say, both 2.1.1 and 2.2.1, that instance has multiple paths of labels. This attribute can take on two values, as follows (the values' names are self-explained):

- *SPL* Single Path of Labels. This term is equivalent to the term "single label per class level" which was used in the previous sections of this chapter (to be consistent with some works in the literature). In the proposed unifying framework we prefer the new term because it can be naturally applied to both trees and DAGs, whilst the definition of "class level" is not so clear in the case of DAGs.
- *MPL* Multiple Paths of Labels. This term is equivalent to the term "hierarchically multi-label" which was used in the previous sections.

• Φ describes the label depth of the data instances, as follows.

- The value *FD* (Full Depth Labeling) indicates that all instances have a full depth of labeling, i.e. every instance is labelled with classes at all levels, from the first level to the leaf level.
- The value *PD* (Partial Depth Labeling) indicates that at least one instance has a partial depth of labeling, i.e. the value of the class label at some level (typically the leaf level) is unknown. In practice it is often useful to know not only that a dataset has at least one instance with a partial depth of labeling, but also the precise proportion of instances with such partial depth of labeling. Hence, in the problem-describing tuple of the proposed framework, the value of this attribute can be specified in a more precise way as $PD_{\%}$, where % means the percentage of the instances that have partial depth labeling.

2.6.2 Categorization of Different Types of Hierarchical Classification Algorithms

A hierarchical classification algorithm is described as a 4-tuple $< \Delta, \Xi, \Omega, \Theta >$, where:

• Δ indicates whether or not the algorithm can predict labels in just one or multiple (more than one) different paths in the hierarchy. For instance, in the tree-structured class hierarchy of Fig. 2.4, if the algorithm can predict both class 1.1 and 1.2 to a given instance, which is equivalent to predicting the paths R-1-1.1 and R-1-1.2, then the algorithm is capable of multiple label path prediction. This attribute can take on two values, as follows:

- *SPP* (Single Path Prediction) indicates that the algorithm can assign to each data instance at most one path of predicted labels.
- *MPP* (Multiple Path Prediction) indicates that the algorithm can potentially assign to each data instance multiple paths of predicted labels.

Note that this attribute is conceptually similar to the aforementioned Ψ attribute used to describe hierarchical classification problems; but they refer to different entites (algorithms vs. problems). If the target problem is a SPL (Single Path of (True) Labels) one, it would be more natural to use a SPP (Single Path Prediction) algorithm, since a MPP (Multiple Path Prediction) algorithm would have "too much flexibility" for the target problem and could produce invalid classifications, wrongly assining multiple paths of labels to some instances. If the target problem is a MPL (Multiple Paths of (True) Labels) one, then one should use a MPP algorithm, since a SPP algorithm would clearly have "too little flexibility" for the target problem, not predicting true labels to some instances. In practice, however, in order to avoid the complexities associated with MPP algorithms, some works simply transform an original MPL problem into a simpler SPL problem, and then apply a SPP algorithm to the simplified data set. This kind of transformation can be achieved by using, for instance, variations of the methods for transforming flat multi-label problems into flat single-label ones described by (Tsoumakas and Katakis, 2007), with proper adaptations for the context of hierarchical classification. In any case, when such a problem simplification is done, it should be clearly indicated in the work.

- Ξ is the prediction depth of the algorithm. It can have two values:

- *MLNP* (Mandatory Leaf-Node Prediction) which means the algorithm always assign leaf class(es).
- *NMLNP* (Non-Mandatory leaf-node prediction) which means the algorithm can assign classes at any level (including leaf classes).

  Again, there is a natural relationship between this Ξ attribute for describing algorithms and its counterpart Φ attribute for describing problems. If the target problem is a FD (Full Depth Labeling) one, one should of course use a MLNP algorithm, since a NMLNP algorithm would have "too much flexibility" and would "under-classify" some instances. If the target problem is a PD (Partial Depth Labeling) one, one should of course use a NMLNP algorithm, since a MLNP algorithm would have "too little flexibility" and would "over-classify" some instances.

- Ω is the taxonomy structure the algorithm can handle. It has two values:
  - *T* (tree), indicating that the classes to be predicted are arranged into a tree structure;
  - *D* (DAG), indicating that the classes to be predicted are arranged into a DAG (Directed Acyclic Graph).

  In principle an algorithm designed for coping with DAGs can be directly applied (without modification) to trees. However, the converse is not true, i.e., if an algorithm was designed for coping with tree-structured class hierarchies only, it would have to be significantly extended to cope with DAGs, as discussed across earlier sections of this chapter.

- Θ is the categorization of the algorithm under the proposed taxonomy of local or global approaches (Sections 2.4 and 2.5) and has the values:
  - LCN (Local Classifier per Node). Within this category, there is also another argument that needs to be specified, which is the strategy used

for selecting negative and positive examples. It can have the following values (most of them defined previously in section 2.4.1):

  * E (Exclusive).
  * LE (Less Exclusive).
  * LI (Less Inclusive).
  * I (Inclusive).
  * S (Siblings).
  * ES (Exclusive Siblings).
  * D (Dynamic) for the cases where the positive and negative examples are selected in a dynamic way (like in (Fagni and Sebastiani, 2007)), but in this case the paper should clearly state how the examples are chosen.

- LCL (Local Classifier per Level).
- LCPN (Local Classifier per Parent Node).
- GC (Global Classifier).

Hence, researchers in hierarchical classification can use this unifying framework to make precisely clear what are the main characteristics of the problem they are solving and also the main characteristics of the hierarchical classification algorithm being used.

Table 2.3: Summary of characteristics of different hierarchical classification approaches, at a high level of abstraction.

Hierarchical Approach	Advantages	Disadvantages
Flat Classifier	Simplicity;	Completely Ignores the class hierarchy;

Local Classifier per Node (training phase)	Simplicity;
	
	Naturally Multi-label;
		May suffer from the blocking problem;
		Prone to inconsistency;
		Employs a greater number of classifiers;

Local Classifier per Parent Node (training phase)	Simplicity; Employs fewer classifiers than Local Classifier per Node;	May suffer from the blocking problem; Prone to inconsistency;
Local Classifier per Level (training phase)	Simplicity; Employs a small number of classifiers;	Prone to inconsistency; A classifier might have to discriminate among a large number of classes (at deep levels); Ignores parent-child class relationships during training;
(Any) Local Classifier with the top-down class prediction approach	Preserves natural constrains in class membership; Considers the class hierarchy during testing and during the creation of the training sets; Generality (can be used with any base classifier);	May suffer from the blocking problem; Depending on the problem at hand, can create a very complex set of cascade of classifiers, which in turn leads to a complex classification model; Misclassification at a given class node is propagated downwards to all its descendant classes;
Global Classifier	Preserves natural constrains in class membership; Considers the class hierarchy during training and testing; Single (although complex) decision model;	Classifier-specific;

Table 2.4: Categorization of hierarchical classification methods proposed in the literature according to the taxonomy proposed in this thesis.

Approach (Θ)	Class Structure (Ω)	List of Works
Flat Classifier	Tree	Barbedo and Lopes (2007); Furnkranz and Sima (2010)
	DAG	Hayete and Bienkowska (2005)
Local Classifier per Node	Tree	D´ Alessio et al. (2000); Dumais and Chen (2000); Sun and Lim (2001); Mladenic and Grobelnik (2003); Sun et al. (2003, 2004); Liu et al. (2005); Wu et al. (2005); Cesa-Bianchi et al. (2006a,b); Cesa-Bianchi and Valentini (2009); Esuli et al. (2008); Punera and Ghosh (2008); Xue et al. (2008); Bennett and Nguyen (2009); Binder et al. (2009); Valentini (2009); Valentini and Re (2009)
	DAG	Barutcuoglu and DeCoro (2006); Barutcuoglu et al. (2006); DeCoro et al. (2007); Guan et al. (2008); Jin et al. (2008)
Local Classifier per Parent Node	Tree	Koller and Sahami (1997); Chakrabarti et al. (1998); McCallum et al. (1998); Weigend et al. (1999); D´ Alessio et al. (2000); Ruiz and Srinivasan (2002); Burred and Lerch (2003); Tikk and Biró (2003); Tikk et al. (2003); McKay and Fujinaga (2004); Li and Ogihara (2005); Brecheisen et al. (2006a); Tikk et al. (2007); Holden and Freitas (2005, 2006, 2008, 2009); Xiao et al. (2007); Secker et al. (2007, 2010); Costa et al. (2008); Silla Jr. and Freitas (2009b); Gauch et al. (2009); Keshtkar and Inkpen (2009); Carvalho et al. (2011); Albornoz et al. (2010); Ghazi et al. (2010); Keshtkar and Inkpen (2011); Zimek et al. (2010)
	DAG	Kriegel et al. (2004)
Local Classifier per Level	Tree	Clare and King (2003)
	DAG	
Global Classifier	Tree	Labrou and Finin (1999); Wang et al. (1999, 2001); Clare and King (2003); Blockeel et al. (2006); Cai and Hofmann (2004, 2007); Dekel et al. (2004a,b); Peng and Choi (2005); Rousu et al. (2005, 2006); Astikainen et al. (2008); Seeger (2008); Silla Jr. and Freitas (2009a); Qiu et al. (2009)
	DAG	Kiritchenko et al. (2005, 2006); Alves et al. (2008); Dimitrovski et al. (2008); Vens et al. (2008); Aleksovski et al. (2009); Otero et al. (2009); Wang et al. (2009)

2.7 Conceptual and Empirical Comparison Between Different Hierarchical Classification Approaches

In the previous sections, we provided a critical review of the existing approaches for the task of hierarchical classification. Therefore, it is interesting to compare the existing approaches on an abstract level. Table 2.3 provides a summary of the different approaches, considering their advantages and disadvantages. In that table, the three rows referring to the three types of local classifiers consider only the training phase of those local approaches. The next row considers the testing phase of any of those three types of local classifiers using the top-down approach. For each row in the table, the description of advantages and disadvantages is self-explanatory.

Also, it is interesting to verify what kinds of approaches have been investigated and what kinds of class structure (tree or DAG) have been used so far in the literature. Table 2.4 classifies the works reviewed in this thesis according to the new proposed taxonomy. The analysis of Table 2.4 shows that the majority of the research carried out so far deals with tree-structured classification problems, rather than DAG-structured ones. Also, the number of papers found in the literature using local classifiers is more than twice the number of papers using global classifiers. This is expected as developing new global classifiers is more complicated than using local approaches with well-known classifiers.

Considering the issues of single/multiple path predictions and prediction depth, a more detailed analysis is carried out in Table 2.5. Note, however, that this table contains only the papers in the literature which provide clear information about these two issues. Therefore, Table 2.5 refers to fewer papers than Table 2.4, although the papers which are mentioned in the former are reported in more detail, according to the standardized terminology of the proposed unified framework. It should be noted that a significant number of papers that are mentioned

Table 2.5: A more detailed categorization of some hierarchical classification works, according to the following attributes of the proposed unifying framework: Approach (Θ), Class Structure (Ω), Label Cardinality Prediction (Δ) and Prediction Depth (Ξ).

$< \Theta, \Omega, \Delta, \Xi >$	List of Works
$< LCN, T, SPP, NMLNP >$	Punera and Ghosh (2008); Binder et al. (2009)
$< LCN, T, MPP, NMLNP >$	Dumais and Chen (2000); Cesa-Bianchi et al. (2006a,b); Cesa-Bianchi and Valentini (2009); Bennett and Nguyen (2009); Valentini (2009); Valentini and Re (2009)
$< LCN, D, MPP, NMLNP >$	Barutcuoglu and DeCoro (2006); Barutcuoglu et al. (2006); DeCoro et al. (2007); Guan et al. (2008); Jin et al. (2008)
$< LCPN, T, SPP, MLNP >$	Koller and Sahami (1997); Chakrabarti et al. (1998); Weigend et al. (1999); Ruiz and Srinivasan (2002); Burred and Lerch (2003); Tikk and Biró (2003); McKay and Fujinaga (2004); Li and Ogihara (2005); Holden and Freitas (2005, 2006, 2008, 2009); Xiao et al. (2007); Secker et al. (2007, 2010); Costa et al. (2008); Silla Jr. and Freitas (2009b); Gauch et al. (2009); Keshtkar and Inkpen (2009); Carvalho et al. (2011); Albornoz et al. (2010); Ghazi et al. (2010); Keshtkar and Inkpen (2011)
$< LCPN, T, SPP, NMLNP >$	Tikk et al. (2007)
$< GC, T, SPP, MLNP >$	Qiu et al. (2009)
$< GC, T, SPP, NMLNP >$	Labrou and Finin (1999); Silla Jr. and Freitas (2009a)
$< GC, T, MPP, NMLNP >$	Clare and King (2003); Blockeel et al. (2006); Rousu et al. (2005, 2006); Dimitrovski et al. (2008); Aleksovski et al. (2009)
$< GC, D, SPP, NMLNP >$	Otero et al. (2009)
$< GC, D, MPP, NMLNP >$	Alves et al. (2008); Vens et al. (2008)

in Table 2.4 are not mentioned in Table 2.5 because those papers did not provide clear information about some characteristics of the corresponding hierarhical classification problem or algorithm. This reinforces the need for the hierarchical classification community in general to be clearer on which kind of problem and what type of algorithms they are using, and the proposed unifying framework offers a standardized terminology and a taxonomy for this purpose.

Although the great majority of research has been carried out on local classifiers, one question that naturally arises is whether a particular type of approach is better than the others or not. In order to investigate that, a compilation of the existing literature (based on the conclusions of the authors of each paper) is shown in Table

2.6, where the symbols $\uparrow$, $\downarrow$, $\sim$ represent whether each approach (corresponding to a given row in the table) obtained a better ($\uparrow$), worse ($\downarrow$) or similar ($\sim$) predictive performance than the approach shown in the corresponding column. The names of the approaches in the rows and columns of this table are abbreviated as follows: LCN is the Local Classifier per Node, LCPN is the Local Classifier per Parent Node and LCL is the Local Classifier per Level. It should be noted that when a particular approach is compared against itself, e.g. LCPN against LCPN, this represents the case where the authors propose a new method within the same broad approach and use the standard approach of that type as a baseline. Also, the lack of any comparisons in a given table cell should not be interpreted as no comparisons were done in the corresponding cell. Sometimes this is the case, while in others the authors compare only different variations of their own approach (e.g. parameter tuning) and not to other approaches.

A careful analysis of the data compiled in Table 2.6 shows that, taking into account the works that compare their hierarchical approaches against flat classification, the hierarchical approaches are usually better than the flat classification approach. As shown in Table 2.6 there is one exception to that. In (Zimek et al., 2010) the authors use a flat classification ensemble approach known as ensembles of nested dichotomies (ENDs) and compare it against the LCPN approach with ENDs as base classifiers at each non-leaf node. In their experiments the LCPN with ENDs was beatean by the flat classification approach with ENDs. The authors in (Zimek et al., 2010) conclude that powerful ensemble classification approaches should be used as baseline for hierarchical classification problems. Another work where the authors have used ensembles for hierarchical classification was in (Costa et al., 2008) however the authors did not compare their approach (a LPCN approach with an ensemble of classifiers at each internal non-leaf node) against a flat classification scenario. However as recently stated by (Deng et al., 2010) when working with huge amounts of data, the use of more sophisticated approaches is unfeasible due to the high computational costs.

Table 2.6: An analysis of how the hierarchical classification methods proposed in the literature performed when compared to other approaches.

Approach	Work	Result when compared against				
		Flat	LCN	LCPN	LCL	GC
LCN	Brecheisen et al. (2006a)	~				
	D´ Alessio et al. (2000)	↑				
	Liu et al. (2005)	↑				
	Cesa-Bianchi and Valentini (2009)	↑				
	DeCoro et al. (2007)	↑				
	Guan et al. (2008)	↑				
	Cesa-Bianchi et al. (2006a,b)	↑	↑			
	Valentini (2009)	↑	↑			
	Valentini and Re (2009)	↑	↑			
	Sun et al. (2004)		↑			
	Barutcuoglu and DeCoro (2006)		↑			
	Punera and Ghosh (2008)		↑			
	Bennett and Nguyen (2009)		↑			
	DeCoro et al. (2007)			↑		
LCPN	Zimek et al. (2010)	↓				
	Koller and Sahami (1997)	~				
	Burred and Lerch (2003)	~				
	Chakrabarti et al. (1998)	↑				
	McCallum et al. (1998)	↑				
	Dumais and Chen (2000)	↑				
	Kriegel et al. (2004)	↑				
	McKay and Fujinaga (2004)	↑				
	Li and Ogihara (2005)	↑				
	Xiao et al. (2007)	↑				
	Jin et al. (2008)	↑				
	Gauch et al. (2009)	↑				
	Keshtkar and Inkpen (2009)	↑				
	Albornoz et al. (2010)	↑				
	Ghazi et al. (2010)	↑				
	Keshtkar and Inkpen (2011)	↑				
	Ruiz and Srinivasan (2002)	↑				~
	Secker et al. (2007)			↑		
	Costa et al. (2008)			↑		
	Holden and Freitas (2008)			↑		
LCL	Clare and King (2003)					~
GC	Dekel et al. (2004a,b)	↑		↑		
	Wang et al. (2001)	↑				
	Peng and Choi (2005)	↑				
	Rousu et al. (2005, 2006)	↑				
	Blockeel et al. (2006)	↑				
	Cai and Hofmann (2004, 2007)	↑				
	Wang et al. (1999)	↑				
	Astikainen et al. (2008)	↑				
	Wang et al. (2009)	↑				
	Kiritchenko et al. (2005, 2006)	↑		~		
	Vens et al. (2008)		↑			
	Otero et al. (2009)			↑		
	Silla Jr. and Freitas (2009a)			↑		
	Clare and King (2003)				~	
	Aleksovski et al. (2009)					~
	Blockeel et al. (2006)					↑
	Dimitrovski et al. (2008)					↑
	Qiu et al. (2009)					↑

Although it is clear that in most cases the local approach is better (more accurate) than the flat classification approach, it is less clear if the global approach is better or worse to deal with hierarchical classification problems than the local approach.

Two studies that tried to answer that question were (Costa et al., 2007b) and (Ceci and Malerba, 2007). In (Costa et al., 2007b) an evaluation comparing: the flat classification approach, the local classifier per level approach, the local classifier per parent node approach and a global approach (using HC4.5 (Clare, 2004)) was performed using two biological datasets with the same base classifier (C4.5). In those experiments, the local classifier per parent node with the top-down class prediction approach performed better on the first dataset while the global approach performed better on the second dataset.

In (Ceci and Malerba, 2007) the authors investigated the use of flat classifiers against the hierarchical local classifier per parent node approach using the same base classifier: SVM or Naive Bayes depending on the experiment. In their experiments, using accuracy as the evaluation measure, the flat SVM obtained better results than its hierarchical counterpart. In a deeper analysis of the misclassified instances, the authors used a combination of four measures into one. The idea behind the use of different measures was to verify different types of errors in a hierarchical classification scenario (e.g. sibling classification error; predicting only higher-level classes (and not the most specific class) for a given example, etc.). After this deeper analysis of the misclassification errors, the authors noticed that although the flat SVM is more accurate, it commits more serious errors than its hierarchical counterpart. Therefore, it seems that whether one particular approach is better than another remains an open question. Hence, the issue of whether one particular approach is better than another naturally depends on the evaluation measure used.

Regardless of which approach is better, the analysis of (Ceci and Malerba, 2007) raises an important concern: How to evaluate hierarchical classification algorithms? Since the use of flat classification measures might not be enough to give us enough insight at which algorithm is really better. Before trying to answer this question, we analysed how the evaluation was carried out in the surveyed papers. The analysis shows that most researchers used standard flat classification evaluation measures, while recognizing that they are not ideal, because the errors at different levels of the class hierarchy should not be penalized in the same way. Other authors propose their own hierarchical classification evaluation measures, which are often only used by the ones who propose it, and in some cases there is not a clear definition of the evaluation measure being suggested. There are also cases when researchers use more than one existing evaluation measure and also propose their own! A good review of hierarchical classification evaluation measures is found in (Sun et al., 2003), although it is out of date now. A more recent survey on evaluation measures for hierarchical classification was presented in (Costa et al., 2007a), however it was limited to tree-structured problems with single-label per level class predictions. An evaluation measure that can cope with multi-label prediction in tree-structured problems was proposed in (Cesa-Bianchi et al., 2006b), called h-loss (for hierarchical loss) as opposed to the traditional zero-one loss. The h-loss however cannot cope with DAGs.

There seems to be no studies that empirically compare the use of the different hierarchical classification evaluation measures, in different application domains (which is important as they have very different class structures), against the flat classification accuracy measure. This would be particularly interesting because most of the approaches currently use flat classification evaluation measures. When comparing a hierarchical classification approach against a flat classification approach authors usually report small gains in accuracy, while when using the hierarchical evaluation measures proposed in (Kiritchenko et al., 2006) the difference in predictive accuracy over the flat approach in the worst case was of 29.39%. This poses an interesting question, if hierarchical approaches overall show similar or better results against the flat classification approach when using flat classification evaluation measures, couldn't the results be actually much better if a hierarchical classification evaluation measure was used instead?

This question naturally leads to the question of which hierarchical classification measure to use? Based on our experience, we suggest the use of the metrics of hierarchical precision (hP), hierarchical recall (hR) and hierarchical f-measure (hF) proposed in (Kiritchenko et al., 2005). They are defined as follows: $hP = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{P}_i|}$, $hR = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{T}_i|}$, $hF = \frac{2*hP*hR}{hP+hR}$, where $\hat{P}_i$ is the set consisting of the most specific class(es) predicted for test example i and all its(their) ancestor classes and $\hat{T}_i$ is the set consisting of the true most specific class(es) of test example i and all its(their) ancestor classes. The summations are of course computed over all test examples. Note that these measures are extended versions of the well known metrics of precision, recall and f-measure but tailored to the hierarchical classification scenario. To determine if there is statistically significant difference between different algorithms, the interested reader is referred to (García and Herrera, 2008).

Although no hierarchical classification measure can be considered the best one in all possible hierarchical classification scenarios and applications, the main reason for recommending the hP, hR and hF measures is that, broadly speaking, they can be effectively applied (with a caveat to be discussed later) to any hierarchical classification scenario; i.e., tree-structured, DAG-structured, single path of labels (SPL), multiple paths of labels (MPL), mandatory leaf-node prediction or non-mandatory leaf-node prediction problems. Let us elaborate on these points, in the context of the categorization of different types of hierarchical classification problems and algorithms proposed in the previous section.

First, the hP, hR and hF measures can be applied not only to tree-structured classes, but also to DAG-structured classes. In the latter case, although in a DAG a node can have multiple paths, one can still compute the set of all ancestors of a node (possibly involving multiple paths from the node to the root) without any ambiguity, and this set of ancestors is basically what is needed to compute these hierarchical classification measures. Secondly, these measures can be applied not only to SPL problems, but also to MPL problems, since one can also compute the set of all ancestors of multiple nodes without any ambiguity. Thirdly, the hP, hR and hF measures can also be naturally applied to full depth labeling problems, associated with mandatory leaf-node prediction algorithms.

The fourth case to be considered here, and the most interesting and complex one, is the case of partial depth labeling problems associated with non-mandatory leaf-node prediction algorithms. This is a scenario where the application of these measures faces some problems, in particular due to a relationship between the concepts of hierarchical precision and hierarchical recall and the concepts of generalization and specialization errors presented in (Ceci and Malerba, 2007). In the latter work, a generalization error refers to the case where the most specific class predicted for an example is more generic than the true most specific known class associated with the example; e.g., predicting only class R.1 for an example whose most specific known class is R.1.1. A specialization error refers to the case where the most specific class predicted for an example is more specific than the true most specific known class associated with the example; e.g. predicting class R.1.1 for an example whose most specific known class is R.1.

To illustrate some issues associated with the hP, hR and hF measures in the context of generalization and specialization errors, let us consider some hypothetical examples. Consider the following three cases of generalization errors:

- A) Predicted Classes: "R.1", True Known Classes: "R.1.2"
- B) Predicted Classes: "R.1", True Known Classes: "R.1.2.1"
- C) Predicted Classes: "R.1", True Known Classes: "R.1.2.1.1"

In these cases the values of hP and hR will be, respectively:

- A) hP = 1/1; hR = 1/2
- B) hP = 1/1; hR = 1/3
- C) hP = 1/1; hR = 1/4

Hence, one can see that for a fixed predicted class, the larger the generalization error (corresponding to a deeper true known class), the lower the hR value, whilst the hP value remains constant. Now let us consider the following cases of specialization errors:

- D) Predicted Classes: "R.2.2", True Known Class: "R.2"
- E) Predicted Classes: "R.2.2.1", True Known Class: "R.2"
- F) Predicted Classes: "R.2.2.1.3", True Known Class: "R.2"

In these cases the values of hP and hR will be, respectively:

- D) hP = 1/2; hR = 1/1
- E) hP = 1/3; hR = 1/1
- F) hP = 1/4; hR = 1/1

Hence, one case see that for a fixed true known class, the larger the specialization error (corresponding to a deeper predicted class), the lower the hP value, whilst the hR value remains constant. Therefore, after careful consideration

In summary, the hF measure, which aggregates hP and hR into a single formula, seems to be able to effectively penalize both generalization and specialization errors, at first glance.

However, there is a problem associated with the use of the hP measure in the context of the so-called "specialization error", as follows. Suppose that the most specific true known class for an example is R.1, and the algorithm predicts to that example the class R.1.1, leading to a hP value of 1/2. Is this penalization fair? Can we be sure that this kind of over-specialized prediction is really an error? This seems to depend on the application. In some applications perhaps we could consider this penalization fair, and consider this over-specialization as an error, if we interpret the most specific true known class as representing the absolute truth about the classes associated with the example. In practice, however, in many applications this interpretation seems unfair, because the most specific true known class associated with an example represents, as emphasized by the use of the keyword "known", just our current state of knowledge, which includes current uncertainties about deeper classes that might be solved later, as more knowledge about the example's classes becomes available.

To consider a concrete example, a major application of hierarchical classification is in the prediction of protein functions where classes are terms in the Gene Ontology, which is briefly reviewed in the next Section. In this application, many proteins are currently annotated with very generic classes only. However, this does not mean the protein really does not have more specific classes, it just means the more specific classes of the protein are not known at present, but might very well be discovered later by biologists. In this kind of application, if the most specific known class of an example is R.1, if the algorithm predicts for that example class R.1.1, the only thing we can really say for sure is that the prediction was correct at the first level, we simply do not know if the prediction was correct or not at the second level, since the true class of the example at the second level is unknown. Therefore, in this kind of application there is an argument to modify the definition of hP in such a way that over-specialized predictions are not considered as errors and so are not penalized.

Despite the above problem, overall the measures of hP, hR and hF seem effective measures of hierarchical classification across a broad range of scenarios, as discussed above, which justifies their recommendation. In (Sokolova and Lapalme, 2009) the authors also consider these measures to be adequate as they do not depend on subjective and user-specific parameters like the distance-based or semantics-based measures. It should also be noted that, in contrast to the hP, hR and hF measures, some other measures of hierarchical classification face some problems in their computation when applied to DAG-structured and/or MPL problems. For instance, although a distance-based measure can naturally be applied to a tree-structured class hierarchy, the concept of the distance between two nodes faces some ambiguity in a DAG, where there can be multiple paths between two nodes. In that case, it is not clear if the distance between two nodes should be given by the shortest, longest or average distance among all paths connecting those two nodes.

2.8 Major Applications of Hierarchical Classification

2.8.1 Text Categorization

The use of hierarchical classification in the field of text categorization dates back to at least 1997, when (Koller and Sahami, 1997) proposed the use of a local classifier per parent node for training coupled with the top-down class-prediction method for testing. There are different types of motivation to work with hierarchical classification in this field. The first one is due to the large growing of the number of electronic documents, and a natural way to handle them is to organize them into hierarchies. Indeed, virtually any type of electronic document can be organized into a taxonomy, e.g. webpages, digital libraries, patents, e-mails, etc. For instance, in (Chakrabarti et al., 1998) the authors propose an interesting example showing how the use of hierarchies can improve the use of information retrieval systems. The example they use is to search for the keywords *jaguar* (and other related words to the animal) on web-search websites. They note that for the user it would be very difficult to retrieve the information he/she was seeking, as a huge amount of information about the car was returned. However, if the user could limit his/her search within a hierarchy (e.g. search for jaguar in the part of the hierarchy rooted at animals), that would help to disambiguate polysemous terms. Figure 2.8 illustrates one example of a document-related class hierarchy.

Figure 2.8: A small part of one of the two taxonomies used in (Chakrabarti et al., 1998) that represents a portion of the US patent database taxonomy.

Another application of hierarchical classification within text categorization is the hierarchical classification of emotions in text. In (Keshtkar and Inkpen, 2009, 2011) the authors address the task of classifying blog posts by mood. That is given a blog post, they wanted to predict the most likely state of mind in which the post was written, e.g. depressed, cheerful, bored, etc. In their experiments they used a LCPN approach with SVM classifiers on a dataset of 815,494 blog posts from Livejournal, a free weblog service used by millions of people to create weblogs. The used hierarchy contains 132 moods organised in up to 5 levels. In (Ghazi et al., 2010) the authors also performed experiments on hierarchical classification of emotions in text using a LCPN approach with SVM but on smaller datasets (with a hierarchy of up to 3 levels).

2.8.2 Protein Function Prediction

In bioinformatics, particularly in the task of protein function prediction, the classes to be predicted (protein functions) are naturally organized into class hierarchies. Examples of these hierarchies are the Enzyme Commision (Barret, 1997) and the Gene Ontology (Ashburner et al., 2000). The Enzyme Commision class hierarchy is – as suggested by its name – specific to enzymes (proteins that speed up chemical reactions), but the Gene Ontology class hierarchy is extremely generic, and can be applied to potentially any type of protein. Protein function prediction is important because this type of information can be potentially used to develop drugs and for better diagnosis and treatment of diseases, since many diseases are caused by or related to malfunctioning of proteins. Figure 2.9 illustrates a very small part of the the Gene Ontology hierarchy. It is important to note that other hierarchical classification schemes exist to annotate proteins, e.g. the MIPS FunCat (Ruepp et al., 2004). These different hierarchies have been used by different authors (Clare and King, 2003; Kriegel et al., 2004; Wu et al., 2005; Barutcuoglu et al., 2006; Blockeel et al., 2006; Rousu et al., 2006; Alves et al., 2008; Guan et al., 2008; Costa et al., 2008; Vens et al., 2008; Otero et al., 2009).

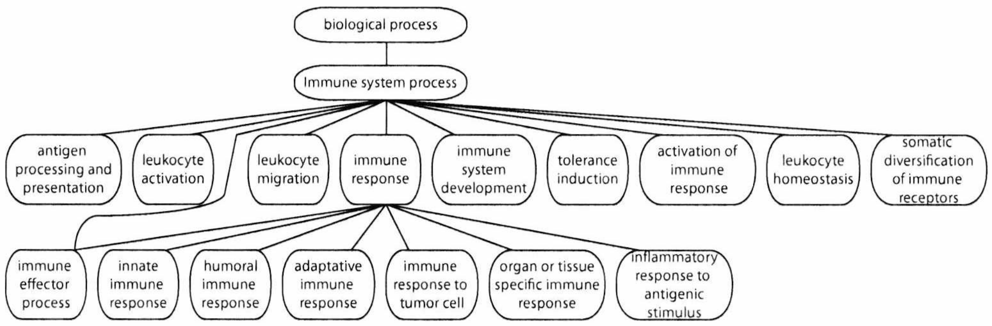

biological process
Immune system process
antigen processing and presentation
leukocyte activation
leukocyte migration
immune response
immune system development
tolerance induction
activation of immune response
leukocyte homeostasis
somatic diversification of immune receptors
immune effector process
innate immune response
humoral immune response
adaptative immune response
immune response to tumor cell
organ or tissue specific immune response
inflammatory response to antigenic stimulus

Figure 2.9: Illustration of the top level structure of the immune system processes in the Gene Ontology (Ashburner et al., 2000).

2.8.3 Music Genre Classification

In organizing and retrieving music information, the genre plays an important concept, as there are studies that show that genre is one of the most used concepts to search for music in music information systems (Downie and Cunningham, 2002; Lee and Downie, 2004). As with other applications, having the genres organized into a class hierarchy helps users to browse and retrieve this information. So far most of the work in this area is only concerned with music genres as a flat classification problem, although many researchers acknowledge the possibility of using class hierarchies in their future works. Some of the works that have used class hierarchies in this application domain are: (Burred and Lerch, 2003; McKay and Fujinaga, 2004; Li and Ogihara, 2005; Brecheisen et al., 2006a; Barbedo and Lopes, 2007; DeCoro et al., 2007; Silla Jr. and Freitas, 2009b). The idea of using the hierarchy for browsing and retrieval has been explored so far in two existing tools for organizing music collections: (Zhang, 2003) demonstrates an end-user system based on the use of hierarchies to organize music collections; and (Brecheisen et al., 2006b) allows the system to have user feed-back in order to re-organize the pre-existing class hierarchy as the users see fit. Figure 2.10 illustrates the audio class hierarchy used in (Burred and Lerch, 2003).

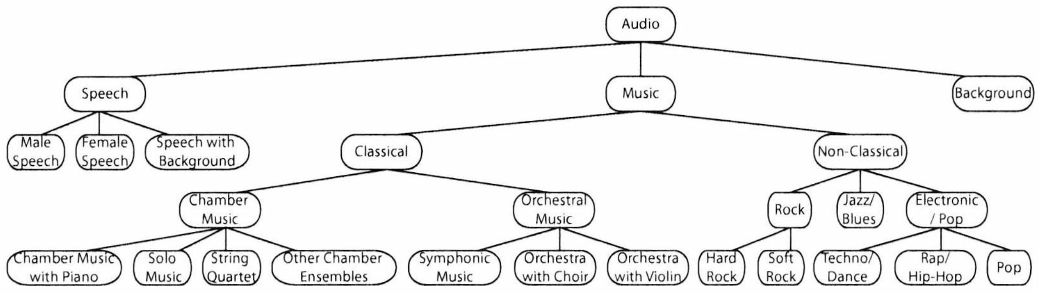


Figure 2.10: The audio class hierarchy used in (Burred and Lerch, 2003).

2.8.4 Other Applications

Although the existing literature has used hierarchical classification methods to deal with the types of applications described in the previous sections, of course the use of those methods is not limited to those applications. In this section, we briefly review some projects that use hierarchical classification approaches to deal with different types of applications.

In (Dekel et al., 2004b) the authors use a large margin classifier in the task of hierarchical phoneme classification. This task consists of classifying the phonetic identity of a (typically short) speech utterance (Dekel et al., 2004b). In this context the class hierarchy plays the role of making the misclassifications less severe. Figure 2.11 illustrates the phonetic hierarchy for American English.

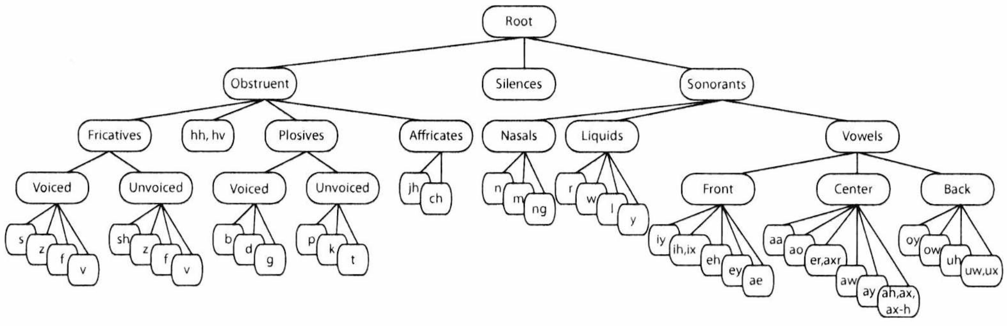


Figure 2.11: The phonetic tree of American English (Dekel et al., 2004b).

In (Barutcuoglu and DeCoro, 2006) the authors use their Bayesian Network aggregation with k-NN base classifiers in the task of 3D shape classification. The motivation to use hierarchical approaches to this problem is that in 3D shape classification scenarios classes are arranged in a hierarchy from most general to most specific shapes. Moreover, a common problem in shape analysis involves assigning semantic meaning to geometry by using a pre-existing class hierarchy. In their experiments they used the Princeton Shape Benchmark (Shilane et al., 2004), which has a 4 level depth hierarchy and 198 leaf classes. Figure 2.12 illustrates the sub-tree Animals of the hierarchy. Other works that deal with hierarchical image classification are (Dimitrovski et al., 2008) and (Binder et al., 2009).

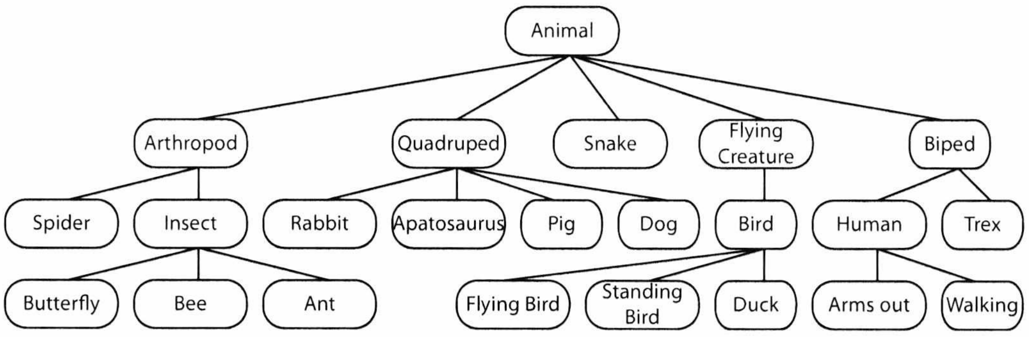


Figure 2.12: The animal branch of the Princeton shape database (Shilane et al., 2004) used by (Barutcuoglu and DeCoro, 2006).

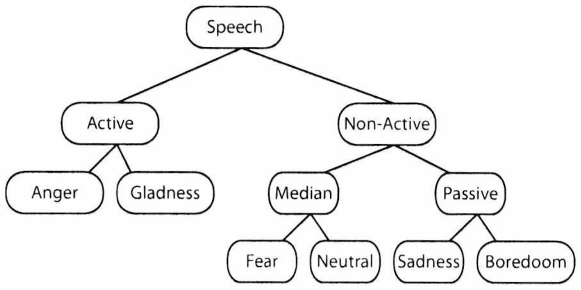


Figure 2.13: The hierarchy used for mood classification based on speech in the Berlin dataset (Burkhardt et al., 2005) used by (Xiao et al., 2007).

Table 2.7: Summary of the existing literature on hierarchical classification according to the type of application domain and the type of hierarchical classification approach.

Type of Application	Hierarchical Classification Approach (Θ)	List of Works
Text Categorization	LCN	D´ Alessio et al. (2000); Sun and Lim (2001); Mladenic and Grobelnik (2003); Sun et al. (2003, 2004); Wu et al. (2005); Cesa-Bianchi et al. (2006b,a); Esuli et al. (2008); Jin et al. (2008); Punera and Ghosh (2008); Xue et al. (2008); Bennett and Nguyen (2009)
	LCPN	Koller and Sahami (1997); Chakrabarti et al. (1998); McCallum et al. (1998); Weigend et al. (1999); D´ Alessio et al. (2000); Dumais and Chen (2000); Ruiz and Srinivasan (2002); Tikk and Biró (2003); Tikk et al. (2003, 2007); Kriegel et al. (2004); Gauch et al. (2009); Keshtkar and Inkpen (2009); Ghazi et al. (2010); Keshtkar and Inkpen (2011); Furnkranz and Sima (2010)
	GC	Labrou and Finin (1999); Wang et al. (1999, 2001); Dekel et al. (2004a); Cai and Hofmann (2004, 2007); Rousu et al. (2005, 2006); Kiritchenko et al. (2005, 2006); Peng and Choi (2005); Seeger (2008); Qiu et al. (2009)
Protein Function Prediction	LCN	Wu et al. (2005); Barutcuoglu et al. (2006); Guan et al. (2008); Valentini (2009); Valentini and Re (2009); Cesa-Bianchi and Valentini (2009)
	LCPN	Holden and Freitas (2005, 2006, 2008, 2009); Secker et al. (2007, 2010); Costa et al. (2008); Kriegel et al. (2004); Carvalho et al. (2011); Zimek et al. (2010)
	LCL	Clare and King (2003)
	GC	(Clare and King, 2003; Blockeel et al., 2006; Alves et al., 2008; Rousu et al., 2006; Vens et al., 2008; Astikainen et al., 2008; Otero et al., 2009; Silla Jr. and Freitas, 2009a; Aleksovski et al., 2009; Wang et al., 2009)
Music Genre Classification	LCN	DeCoro et al. (2007)
	LCPN	Burred and Lerch (2003); McKay and Fujinaga (2004); Li and Ogihara (2005); Brecheisen et al. (2006a); Silla Jr. and Freitas (2009b)
Image Classification	LCN	Barutcuoglu and DeCoro (2006); Binder et al. (2009)
	GC	Dimitrovski et al. (2008)
Emotional Speech Classification	LCPN	Xiao et al. (2007); Albornoz et al. (2010); Xiao et al. (2011)
Phoneme Classification	GC	Dekel et al. (2004a,b)

In (Xiao et al., 2007) the authors build a class hierarchy for the task of hierarchical classification of emotional speech. The database used in this paper is Berlin emotional speech database (Burkhardt et al., 2005). They create a 3 level depth hierarchy to distinguish between 6 leaf classes (which are types of emotion): anger, boredom, fear, gladness, sadness and neutral. They use a local classifier per parent node approach with a Multi Layer Perceptron (MLP) Neural Network with sequential forward feature selection. Figure 2.13 illustrates their class hierarchy.

A summary of the literature cited above according to their application domain and type of hierarchical classification approach used is presented in Table 2.7.

Chapter 3

A New Data Representation Selection Approach for Local Hierarchical Classification

The task of computational prediction of protein function based on the protein's amino acid sequence is an active area of research in the field of proteomics (Friedberg, 2006; Zhao et al., 2008). One approach that can be used to infer proteins functions is supervised machine learning – more precisely, the classification task of machine learning or data mining. The goal is to use a set of proteins whose functions are known to build a classification model that can be used to predict the functions of proteins whose functions are unknown. The use of supervised machine learning (classification) algorithms is common practice in the field (Syed and Yona, 2003; Hayete and Bienkowska, 2005; Al-Shahib et al., 2005; Secker et al., 2007).

There are two major problems in the task of computational protein function prediction with classification algorithms, which are the choice of the protein representation and the choice of the classification algorithm. Those are open problems, even in the conventional scenario of "flat" classification (where there are no hierarchical relationships among classes), as there are many choices and it is not clear which representation and classification algorithm are the best. In the hierarchical classification scenario addressed in this chapter, where protein functional classes are organised into a hierarchy, these problems are aggravated, due to the large number of classes and classification sub-problems (where different algorithms and different representations might be best for different class levels).

There are several ways of extracting features from a protein, and the choice of the feature representation might be as important as the choice of the classification algorithm. Apart from a few works, such as (King et al., 2001), the issue of which feature representation to use is often overlooked as the authors are usually more focused on which classification algorithm to use or related issues. One particular challenge is that not all feature sets are available for every experiment, as some biological databases are highly specialized in one particular organism and the same information might not be available for other organisms.

According to (Davies et al., 2008b) there are two broad types of representations that can be derived for proteins: alignment-independent, which are features computed from the sequence by using some computational method without performing sequence alignment, and alignment-dependent, which are features obtained from biological databases of motifs or domains that were typically discovered by performing sequence alignment on a large-scale, in order to identify conserved regions in the sequences of homologous proteins.

In this chapter we present a new local classifier per parent node approach for hierarchical classification that tries to select the best representation at each local parent node of the hierarchy. The experiments are run using protein function prediction datasets with different feature representations. Most of this chapter is based on a paper published in the Intelligent Data Analysis (IDA) Journal (Silla Jr. and Freitas, 2011a).

3.1 From Genes to Proteins

According to (Cristianini and Hahn, 2007), the simplest way to explain how proteins are made - ignoring many details and condensing complicated cellular reactions - is a very simple diagram that shows what is known as the *central dogma*:

$$\text{DNA} \rightarrow \text{RNA} \rightarrow \text{Protein}$$

The arrows on the diagram can be read as "is converted into". Therefore DNA is converted into RNA which in turn is converted into Proteins (which then help to produce DNA again). The central dogma is not always right, but it is right so much of the time that showing where it went wrong is rewardable with Nobel Prizes (Cristianini and Hahn, 2007). An exception to the central dogma are the "auto-replicating" (loosely speaking) proteins responsible for the Mad Cow Disease known as *Prions*. In the remaining of this section, each element responsible for the protein synthesis according to the central dogma will be briefly reviewed.

3.1.1 DNA

A genome is the set of all DNA contained in a cell and is formed by one or more long stretches of DNA strung together into chromosomes (Cristianini and Hahn, 2007). The DNA, which has some parts consisting of genes or coding regions and some other parts consisting of non-coding regions, is a molecule that consists of a chain of smaller molecules called *nucleotides*. The *nucleotides* are distinct from each other only in a chemical element called base and they contain the bases adenine (A), cytosine (C), guanine (G), and thymine (T) (Alberts et al., 2002). A DNA sequence is double-stranded (hence, the "double helix"), and the two strands are complementary to each other. The complementary pairings are: A-T, C-G, G-C and T-A.

3.1.2 Amino Acids

There are 20 amino acids from which about half can be made by vertebrates; the others must be supplied in the diet (Alberts et al., 2002). The amino acids which need to be supplied in the diet are also called *essential amino acids*. They are made in other organisms, usually by long and energetically expensive pathways that have been lost in the course of vertebrate evolution (Alberts et al., 2002). Table 3.1 lists the 20 amino acids and their standard symbols.

Table 3.1: The 20 amino acids and their standard symbols (Cristianini and Hahn, 2007). An * marks the essential amino acids (Alberts et al., 2002).

A	Alanine	Q	Glutamine	L	Leucine*	S	Serine
R	Arginine	E	Glutamic acid	K	Lysine*	T	Threonine*
N	Asparagine	G	Glycine	M	Methionine*	W	Tryptophan*
D	Aspartic acid	H	Histidine*	F	Phenylalanine*	Y	Tyrosine
C	Cysteine	I	Isoleucine*	P	Proline	V	Valine*

3.1.3 Transcription and Translation

The process of going from a gene encoded in DNA to a protein made up of amino acids is divided conceptually into two steps: Transcription, which is the copying of DNA sequences into RNA sequences, and Translation, which is the process of directing the synthesis of specific proteins. During transcription, the cell transcribes the DNA encoding genes into messenger RNA (mRNA) and sends it to the ribosome. The ribosome takes the sequence of nucleotides and translate it into a string of amino acids (Cristianini and Hahn, 2007). The ribonucleic acid (RNA) has two chemical differences to the DNA (Alberts et al., 2002): (1) the sugar-phosphate backbone of RNA contains ribose instead of a deoxyribose sugar, and (2) the base thymine (T) is replaced by the very closely related base uracil (U).

In order to perform the translation, every organism uses a three-nucleotide unit (often referred to as a *codon*) to specify amino acids. The translation of each codon to a specific amino acid (or into a special punctuation called stop codons)

Table 3.2: The standard genetic code (Cristianini and Hahn, 2007).

	A		G		C		U	
A	AAA	K	AGA	R	ACA	T	AUA	I
	AAG	K	AGG	R	ACG	T	AUG	M
	AAC	N	AGC	S	ACC	T	AUC	I
	AAU	N	AGU	S	ACU	T	AUU	I
G	GAA	E	GGA	G	GCA	A	GUA	V
	GAG	E	GGG	G	GCG	A	GUG	V
	GAC	D	GGC	G	GCC	A	GUC	V
	GAU	D	GGU	G	GCU	A	GUU	V
C	CAA	Q	CGA	R	CCA	P	CUA	L
	CAG	Q	CGG	R	CCG	P	CUG	L
	CAC	H	CGC	R	CCC	P	CUC	L
	CAU	H	CGU	R	CCU	P	CUU	L
U	UAA	*	UGA	*	UCA	S	UUA	L
	UAG	*	UGG	W	UCG	S	UUG	L
	UAC	Y	UGC	C	UCC	S	UUC	F
	UAU	Y	UGU	C	UCU	S	UUU	F

by the cell is done by using a process that can be conceptually summarized by a look-up table (Cristianini and Hahn, 2007). This look-up table is called the genetic code and can be seen in Table 3.2. There are exceptions, such as the human mitochondria, even to this look-up table (Cristianini and Hahn, 2007). An example of how a DNA sequence can be organized into codons to specify an amino acids sequence is shown in Figure 3.1. An overview of the process is shown in Figure 3.2.

As seen in (Cristianini and Hahn, 2007) even with genetic code in hand, how does the ribosome know where to start reading the mRNA sequence? This is particularly important as depending on where one starts reading a DNA sequence

DNA	CTT	GTG	CCC	GGC	TGC	GGC	GGT	TGT	ATC	CTG
Protein	L	V	P	G	C	G	G	C	I	L

Figure 3.1: An example of the codon-mapping (Cristianini and Hahn, 2007).

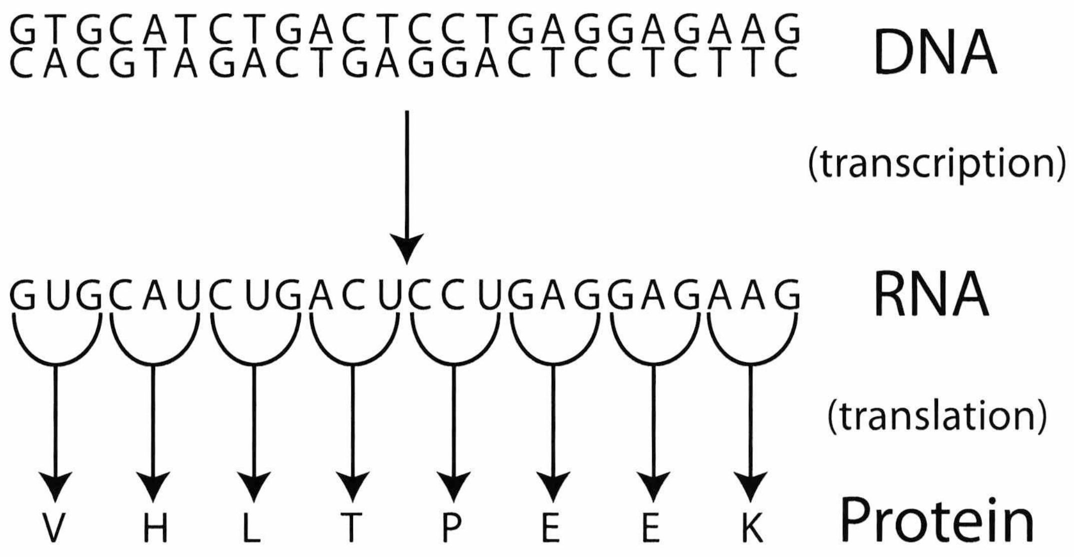


Figure 3.2: An overview of the biological process of protein synthesis (Madprime, 2007).

there are three different ways to decompose it into codons. For example, considering the mRNA sequence of ...AGCUACGUAGCUACGU... it could be decomposed in three different ways: ...AGC-UAC-GUA-GCU-ACG-U...; ...AG-CUA-CGU-AGC-UAC-GU...; or ...A-GCU-ACG-UAG-CUA-CGU... . In each of these cases the resulting protein would be completely different. Therefore, the ribosome must know where to start translating the mRNA into amino acids. Each non-overlapping decomposition of a DNA sequence into codons is called a *reading frame*. The starting point of a reading frame of every protein is specified by the amino acid methionine. This way the ribosome knows that the first AUG in the mRNA specifies the reading frame for the rest of the protein and will read it until a stop codon is found (UGA, UAA, or UAG). This stretch of DNA that has a start codon followed by a run of stop-codon-free DNA in the same reading frame, followed by a stop codon, is called an *open reading frame*.

3.1.4 Proteins

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 (Freitas and de Carvalho, 2007). It is amazing that the same basic chemical structure – a chain of amino acids – can form so many different structures: an efficient rubber like material (elastin), a steel-like cable (collagen), or the wide variety of catalytic surfaces on the globular proteins that function as enzymes (Alberts et al., 2002). Because of the variety of their amino acid side chains, proteins are remarkably versatile with respect to the type of structures they can form (Alberts et al., 2002). A typical protein contains 200-300 amino acids, but some are much smaller (30-40) and some are much larger reaching tens of thousands of amino acids (Cristianini and Hahn, 2007).

Although proteins can be read as one-dimensional objects, they do not have a linear form in the cell: an amino acids chain rapidly folds itself into a three dimensional shape that ultimately determines its function. This processes is known as protein folding. Also, the exact identity and order of the amino acids in the chain will determine the final shape of a protein.

Proteins can have four distinct aspects according to their structure (Alberts et al., 2002) (Freitas and de Carvalho, 2007):

- Primary Structure: The linear amino acid sequence.
- Secondary Structure: 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) created by hydrogen-bond interactions within contiguous stretches of polypeptide chains.
- Tertiary Structure: Consists of the three-dimensional structure of the protein.
- Quaternary Structure: Is the complete structure of protein molecules formed

as a complex of more than one polypeptide chain. Not all proteins have a quaternary structure.

In the section 3.3 we will describe in detail the types of protein representation used in this thesis which are all based on the primary structure.

3.2 (Automated) Protein Function Prediction

Currently, the usual approach for a biologist to obtain some insight about a protein whose function is unknown, is to use a technique known as sequence comparison. In order to illustrate how sequence comparison works/why it is important, let's use a very interesting example from (Gibas and Jambeck, 2001) called eye of the fly.

Fruit flies (*Drosophila melanogaster*) have a gene called eyeless, which, if it's "knocked out" (i.e., eliminated from the genome using molecular biology methods), results in fruit flies with no eyes. There is a human gene responsible for a condition called aniridia. Humans without this gene develop eyes without irises. If the gene for aniridia is inserted into an eyeless drosophila "knock out," it causes the production of normal drosophila eyes. In order to verify if this was just a coincidence or if there could be more similarity between the eyeless gene and the gene for aniridia, in a first moment, a biologist will compare their sequences. As in a second moment, more careful experimentation would be needed to get a more definitive answer.

This raises the question of how to compare two (or more) protein sequences. In order to compare two protein sequences, first they need to be aligned. Protein alignment can be done in a variety of ways, and its basic concept is as follows (Gibas and Jambeck, 2001): Two sequences are matched up in an arbitrary way. The quality of the match is scored. Then one sequence is moved with respect to the other and the match is scored again, until the best-scoring alignment is found. The main use of sequence comparison is to determine if two proteins are homologous, i.e., if the two proteins share an evolutionary common ancestor.

Figure 3.3 shows an example of a simple protein alignment (Gibas and Jambeck, 2001). The top line contains a small subset of the eyeless sequence. The bottom line is a small subset of the aniridia sequence. The middle line shows where the sequences match. If there is a letter on the middle line, the sequences match exactly at that position. If there is a plus sign on the middle line, the two sequences are different at that position, but there is some chemical similarity between the amino acids (e.g., S and A, Serine and Alanine). If there is nothing on the middle line, the two sequences don't match at that position.

```
I E R L P S L E D M A H K
I   R   P +       M +
I P R P P A R A S M Q N S
```

Figure 3.3: An example of protein alignment (Gibas and Jambeck, 2001).

The task of how to perform automated protein sequence alignment and comparison is a research area by itself (Rosenberg, 2009). The reason for this is that homology-based transfer, using programs such as BLAST (Altschul et al., 1990), is probably the most widely used form of computational function-prediction method; assigning unannotated proteins with the function of their annotated homologues (Sleator and Walsh, 2010). The rationale for this approach is based on the assumption that two sequences with a high degree of similarity most likely evolved from a common ancestor and thus must have similar functions (Sleator and Walsh, 2010).

In fact, for (Pandey et al., 2007) the design of sequence similarity systems such as BLAST were the first major breakthrough in the field of computational biology. In any case, the automated protein function prediction tools normally used by biologists and bioinformaticians - namely, tools implementing BLAST or one of its many variants - solve just a flat (rather than hierarchical) classification problem, as follows. Those tools typically are based on a sequence similarity-based approach, where the system computes the similarity between the sequence of a new protein (with unknown function) and the sequence of each of the proteins in the database, and then the system assigns, to the new protein, the functional class of its most similar protein(s) in the database.

In (Freitas et al., 2010) it is discussed that this sequence similarity-based approach shares core characteristics with the supervised classification paradigm known as instance-based learning (also called the nearest neighbour method or lazy learning) for flat classification. In this approach, the training phase essentially consists of storing known-function proteins, and the learning occurs in the testing phase, where an algorithm is used to identify the training sequence most similar to the target protein sequence, therefore ignoring the hierarchical structure of protein functions.

The use of machine learning approaches to automatically predict protein functions is not limited to the instance-based learning paradigm, and any type of classifier can be employed for the task (with different degrees of success). More precisely, each protein corresponds to an instance (example). From each protein are extracted protein descriptors which are used as features (or attributes) and the protein functions are the classes to be predicted. By using supervised learning paradigms different from instance-based learning, it is also possible to go beyond the limitation of predicting function only for similar sequences.

Besides not dealing with hierarchical classification problems, another drawback of inferring protein function based on similarity search is that proteins with similar characteristics (sequences of amino acids) can have very different functions and vice-versa (Gerlt and Babbitt, 2000). For that reason, some biological databases such as PROSITE (Hulo et al., 2006) contain common patterns (motifs) of residues of sets of proteins. Such a pattern (or motif) appears in a family of related proteins usually because of the requirements of binding sites that constrain the evolution of a protein family and they often indicate distant relationships not otherwise detectable by comparing sequences (Lesk, 2002).

3.3 Protein Feature Types

In this section we describe the protein representations used and evaluated in this work. The protein representations in Sub-sections 3.3.1, 3.3.2, 3.3.3, 3.3.4 are alignment-independent representations. The protein representations in Sub-section 3.3.5 are alignment-dependent.

3.3.1 Sequence Length and Molecular Weight

The sequence length is a numerical value which is simply the count of amino acids of a protein. The molecular weight is the sum of the molecular weights of all amino acids in the protein.

These features have been used (with other attributes) in (King et al., 2001; Al-Shahib et al., 2005; Holden and Freitas, 2006). Since these features are believed to be important for protein functional prediction and they are easily available, we always use them in conjunction with the other protein representation studied in this work.

3.3.2 Z-Values

The z-values (Secker et al., 2007; Davies et al., 2007), also known as Sandberg Descriptors (Sandberg et al., 1998; Lapinsh et al., 2002), are the principal components of 26 different physicochemical measured and calculated properties of amino acids, and essentially represent hydrophobicity/hydrophilicity (z1), steric/ bulk properties and polarizability (z2), polarity (z3), and electronic effects (z4 and z5) of the amino acids (Lapinsh et al., 2002).

In (Secker et al., 2007) 5 z-values are used to represent each amino acid of the protein sequence. For example, the Alanine (A) amino acid has 5z values: z1 = 0.24, z2 = -2.32, z3 = 0.60, z4 = -0.14, z5 = 1.30. Therefore a protein sequence of length n would be represented by n*5 features. In (Davies et al., 2007; Secker et al., 2007) the authors suggested that the z-values for all amino acids of each protein are averaged so that a protein is represented by just 5 z-values, instead of 5*n. This is needed because most machine learning methods cannot cope with instances (in this case proteins) which have varying number of features (in this case the z-values). It should be noted that they tried more complicated ways of aggregating z-values, but they had better results with this simpler method.

Originally in (Secker et al., 2007) the authors used the averaged z-values from the whole amino acid sequence. After some experimental research they found out that in order to classify GPCR (G-Protein Coupled Receptor) proteins, it would be better to use 15 z-values (Davies et al., 2007). These z-values are then computed as follows: 5-values are computed and averaged over the whole protein sequence. Another 5 z-values are computed from the N-terminus (the first 150 amino acids of the protein sequence) of the protein and further 5 z-values are computed from the C-terminus (the last 150 amino acids of the protein sequence). The number of 150 amino acids was found, in previous experiments, to give the largest improvement in accuracy (Davies et al., 2007).

In this work we use both 5 z-values and 15 z-values.

3.3.3 Amino Acid Composition (AA)

Another feature which is very simple to compute based on the protein sequence is the percentage occurrence of each amino acid within a protein sequence. This will create a feature set of 20 features, each of them with the percentage of how many times a particular amino acid occurs within the protein's amino acid sequence.

This type of feature has been used in (Hobohm and Sander, 1995; King et al., 2001; Syed and Yona, 2003; Al-Shahib et al., 2005).

3.3.4 Local Descriptors (LD)

The local descriptors, also known as global protein sequence descriptors (Dubchak et al., 1995), were used in (Cai et al., 2003; Cui et al., 2007; Davies et al., 2008a; Tong and Tammi, 2008).

There are three types of local descriptors used in the aforementioned works (and also used in our own experiments): Composition, Transition and Distribution, which are computed based on the variation of occurrence of functional groups of amino acids within the primary sequence of the protein. The functional groups used were: hydrophobic (amino acids CVLIMFW), neutral (amino acids GASTPHY), and polar (amino acids RKEDQN).

Composition accounts for the percentage composition (relative frequency) of a particular functional group within the amino acid sequence. Therefore, there are three composition features, one for each functional group of amino acids.

Transition features represent the relative frequency in which an amino acid from a particular functional group is followed by an amino acid from another functional group. More precisely, the following transitions are considered: Polar $\rightarrow$ Neutral or Neutral $\rightarrow$ Polar; Polar $\rightarrow$ Hydrophobic or Hydrophobic $\rightarrow$ Polar; and Neutral $\rightarrow$ Hydrophobic or Hydrophobic $\rightarrow$ Neutral.

Distribution features are computed based on the percentage of how many amino acids of a particular functional group are present on the first, 25%, 50%, 75% and 100% of the amino acid sequence.

In total there would be 21 features (3 composition, 6 transition, 12 distribution) if they were computed from the whole amino acid sequence. However, in (Davies et al., 2008a; Tong and Tammi, 2008) the authors divided the protein sequence into 10 descriptor regions (A-J) as follows: Regions A,B,C and D are obtained by dividing the entire protein sequence into four equal-length regions. Regions E and F are obtained by diving the protein sequence in two equal-length regions. Region G represents the middle with 50% of the sequence. Region H represents the first 75% of the sequence, Region I the final 75% of the sequence and Region J the middle with 75% of the sequence. For each region the 21 local descriptors are extracted, resulting in a 210 feature vector. These regions are illustrated in Figure 3.4.

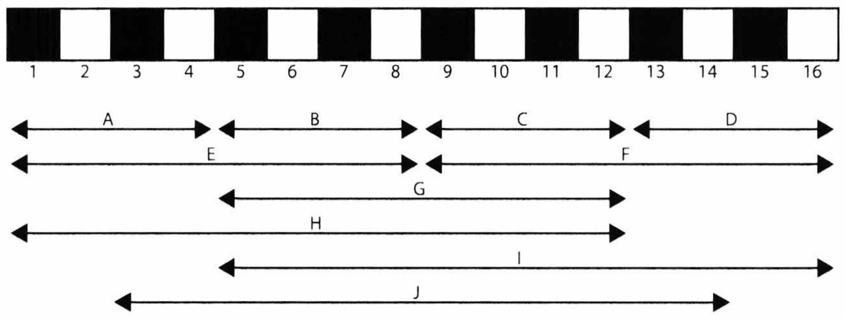


Figure 3.4: The 10 regions used by the Local Descriptor technique as used in (Tong and Tammi, 2008; Davies et al., 2008a)

3.3.5 Motif-Based Features

Instead of computing features directly from the protein sequence, like in the previously described protein representations, it is possible to use features obtained from biological databases. In (Drawid and Gerstein, 2000; Ben-Hur and Brutlag, 2003, 2006; Hayete and Bienkowska, 2005; Zhang et al., 2005; Holden and Freitas, 2006, 2009; Kong et al., 2007; Nariai et al., 2007) the authors use the absence/presence of a particular type of protein signatures ("motifs") as binary features.

In this work we use protein signatures from four different databases as features. The employed signatures are PROSITE patterns (Hulo et al., 2006), which use regular expressions to encode the motifs; Fingerprints from the PRINTS (Attwood, 2002) database, which are created by considering several motifs to be present in the same protein; motifs from the PFAM (Bateman et al., 2004) database, which are created by using hidden Markov models (HMMs); and entries from the InterPro (Mulder et al., 2007; Hunter et al., 2009) database.

The PROSITE patterns are encoded as regular expressions, and the rationale behind its development is that a protein family could be characterized by a single most conserved motif within a multiple alignment of its members sequences, as this would likely encode a key biological feature (Higgs and Attwood, 2005). For example, the small sequence from the eyeless gene shown in Figure 3.3 could be described by the following PROSITE pattern: I-X(1)-R-X(1)-P-[SA]-X(3)-M-[AQ]-X(2). In this pattern the protein sequence must have an Isoleucine

(I) followed by one other amino acid (denoted by X(1)), followed by an Arginine (R) followed by one other amino acid (X(1)), followed by a Proline (P) followed by either a Serine or an Alanine ([SA]) followed by any other three amino acids (X(3)), followed by a Methionine (M) followed by either an Alanine or a Glutamine ([AQ]) followed by any other two amino acids (X(2)). However, as pointed out in (Higgs and Attwood, 2005), most protein families are characterized not by one, but by several conserved motifs. This is the rationale behind the development of the fingerprints motifs used in the PRINTS database which is another approach to characterize protein families that adopts the principle that the variable regions between conserved motifs also contain valuable information. Therefore as a motif, the PRINTS fingerprints uses for each protein family a set of patterns whose number will depend on protein family. In the PFAM database, instead of using patterns as motifs, hidden Markov Models are used to encode profiles. Although there is some overlap between these three databases, their content is significantly different. Also, as pointed out in (Higgs and Attwood, 2005), these motifs have different areas of application, e.g.: PROSITE patterns are unreliable in the identification of members of highly divergent superfamilies (where HMMs excel); fingerprints perform relatively poorly in the characterization of very short motifs (where PROSITE patterns do well); and HMMs are less likely to give specific subfamily diagnoses (where fingerprints excel). For these reasons, the curators of these databases (among others) decided to combine efforts in the creation of the INTERPRO database, which combines the information from all these and other databases.

3.3.6 Summary of Protein Features Used in this work

Table 3.3 presents a summary of the feature types and the respective number of features used in this work. As explained earlier, the top 4 features in Table 3.3 are alignment-independent features, whilst the bottom 4 features are alignment-dependent. In this table EC and GPCR refer to the Enzyme and GPCR datasets whose creation is explained in detail in section 3.5.2.

Table 3.3: Summary of number of features per type.

Protein Feature Type	# features
5 Z-Values (5z)	5
15 Z-Values (15z)	15
Amino Acid Composition (AA)	20
Local Descriptors (LD)	210
Prosite Patterns	582 for EC, 127 for GPCR
Prints Fingerprints	380 for EC, 281 for GPCR
Pfam Profiles	706 for EC, 73 for GPCR
Interpro Entries	1,214 for EC, 448 for GPCR

3.4 Hierarchical Protein Function Prediction

Protein functions are often specified in a functional class hierarchy, with more generic functions at higher levels and more specific functions at deeper levels. For instance, Figure 3.5 illustrates a small part of the Enzyme Commission hierarchy. On the first level of the hierarchy, there are 6 classes. The meaning of each class is as follows: EC 1 = Oxidoreductases, EC 2 = Transferases, EC 3 = Hydrolases, EC 4 = Lyases, EC 5 = Isomerases, EC 6 = Ligases. The remaining classes shown on Figure 3.5 have the following functions: EC 1.1 = Acting on the CH-OH group of donors, EC 1.1.1 = With NAD or NADP as acceptor, EC 1.1.1.1 = alcohol dehydrogenase, EC 1.1.1.2 = alcohol dehydrogenase (NADP+), EC 1.1.1.3 = homoserine dehydrogenase.

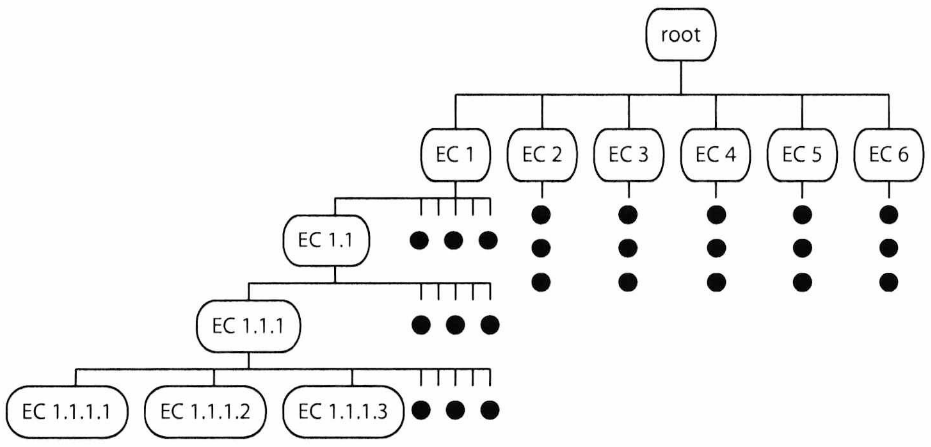


Figure 3.5: An excerpt of the Enzyme Commission class hierarchy

Since protein functions are often specified in a hierarchy, it is natural to apply hierarchical classification algorithms to this type of data. In the literature however, this issue is often ignored and many papers focus on predicting just one single level (usually the first) of the class hierarchy (Jensen et al., 2003; Weinert and Lopes, 2004).

3.4.1 The Selective Classifier Approach

In (Secker et al., 2007; Davies et al., 2007) the authors hypothesise that it would be possible to improve the predictive accuracy of the local, top-down approach by using different classification algorithms at different nodes of the class hierarchy. The choice of which classifier to use at a given class node is made on a data-driven manner using the training set. More precisely, in order to determine which classifier should be used at each node of the class hierarchy, during the training phase, the training set is randomly split into mutually-exclusive sub-training and validation sets. Different classifiers are then trained using this sub-training set and are then evaluated on the validation set. The classifier chosen for the current class node is the one with the highest classification accuracy on the validation set. In this approach the protein representation is fixed, i.e. all classifiers are trained with the same feature set. This approach is referred to as the Selective Classifier (Sel. C.) approach.

In this work as components of the Sel. C. approach we have employed the k nearest neighbor (k-NN) with $k = 3$ (Cover and Hart, 1967), Naive Bayes (NB) (Duda and Hart, 1973) and Support Vector Machines (SVM) (Platt, 1998). All these classifiers were used with the WEKA Data mining Tool (Witten and Frank, 2005) with default parameters. The rationale behind the choice of these particular classifiers is that they are well-known classifiers which have been successfully used in flat (non-hierarchical) protein function prediction problems and also they have very different inductive biases, meaning that they will construct different classification models, therefore insuring a diversity of predictions to be exploited by the Sel. C. approach.

3.4.2 A Novel Local Selective Representation Approach

One interesting aspect of research that has been little investigated in the literature is the development and evaluation of new strategies to handle different feature representations in hierarchical classification. The motivation behind this idea arises from the questions: "Do the features used to distinguish between different classes have the same importance at different levels of the hierarchy?" Moreover, "would different types of features at different class nodes improve the classification accuracy / interpretability of the results"?

Inspired by the selective classifier approach, we propose a novel method in this thesis which is referred to as the Selective Representation (Sel. R.) approach (by contrast to the Sel. C. approach proposed in (Secker et al., 2007)). The former uses a strategy similar to the one used by the latter, but instead of selecting the best classifier at each parent node in the class tree (like in (Secker et al., 2007)), it selects the best feature representation at each parent node of the class hierarchy. In this approach the classifier is fixed, i.e. at all class nodes the same type of classifier is trained with each of the different types of feature set, and the best type of feature (on the validation set) is chosen at each class node. Algorithm 1 presents the training phase of the Sel. R. approach. During the test phase the Sel. R. representation approach uses the top-down strategy for testing and uses at each non-leaf node the feature representation chosen for that particular non-leaf node.

The motivation behind this approach can be explained by doing an analogy with the biological taxonomy of animals. The latter is a hierarchy that consists of eight levels. At each level of the hierarchy, groups of animals are distinguished from one another by their dissimilarities. To illustrate this argument, Table 3.4 contains four animals (two kinds of cats and two kinds of horses) and their respective classifications using the biological taxonomy. The important question here is: "Are the features that distinguish between Carnivore and Perissodactyla (e.g. shape of the hoofs, type of alimentation, type of teeth), the same as the

Algorithm 1: The selective representation approach – training phase.

```
Input: A set of $n$ training datasets $D_1, \ldots, D_n$ where each dataset contains the
       same instances in the same order but with a different feature
       representation.
Input: A supervised flat classification learning algorithm.
Input: A hierarchy of the class labels: $H$.
Output: A set of local per parent node classifiers and their respective
        representations created by the Sel. R. approach.
1 foreach Non-leaf node in $H$ do
2     foreach dataset $D_i, 1 \leq i \leq n$ do
3         Split the training set $D_i$ into sub-training (using 90% of the data) and
          sub-validation sets (using the remaining 10% of the data);
4         Train the flat classification algorithm with the sub-training set;
5         Evaluate the predictive performance of the flat classification algorithm
          with the sub-validation set;
6         Save the performance of this classifier for the $i$-th feature representation;
7     end
8     Choose the feature representation with the highest performance among the
      sub-validation sets of all classifiers trained for this non-leaf node;
9 end
```

features that distinguish between a Persian cat and a Siamese cat (e.g. length of the fur, thickness of the fur, color of the fur, shape of the skull)?". Moreover, are the features that distinguish between the different cats the same as the ones that distinguish between the different horses (height, weight, thickness of the hair (which is referred as fur in small animals))? From this analysis, it is clear that the classification of different objects (in this example, the animals), benefits from different representations at different levels of the hierarchy.

Table 3.4: Four animals according to their biological taxonomy.

	Animal			
	Persian Cat	Siamese Cat	Breton Horse	Arab Horse
Kingdom	Animalia	Animalia	Animalia	Animalia
Phylum	Chordata	Chordata	Chordata	Chordata
Class	Mammalia	Mammalia	Mammalia	Mammalia
Order	Carnivore	Carnivore	Perissodactyla	Perissodactyla
Family	Felidae	Felidae	Equidae	Equidae
Genus	*Felix*	*Felix*	*Equus*	*Equus*
Species	*F. domesticus*	*F. domesticus*	*E. caballus*	*E. caballus*
Breed	Persian	Siamese	Breton	Arab

Note that, at a very high level of abstraction, the idea of representation selection seems similar to the well-known idea of feature selection in data mining (Liu and Motoda, 2007). In (Secker et al., 2010) the authors have employed a feature selection strategy with the Sel. C. approach. However, their results has shown that this procedure was not efficient (from a classification perspective) as the results achieved by this approach were 69.97% against 70.46% (measured by predictive accuracy) with using just the Sel. C. approach. In this work the motivation for representation selection rather than feature selection in a hierarchical classification scenario is explained by the following reasons: (a) it is much more efficient (faster) to select a representation at each class node than to perform feature selection at each class node; (b) Representation selection produces results at a coarser grain of information, possibly providing new insights to biologists, that is, it might reveal that some broad type of representations (sets of features of the same type, rather than single features) are particularly more effective to classify protein functions at particular levels. It also differs from feature selection as different representations in a dataset are actually just "candidate representations", because just one will be chosen, unlike in feature selection where any subset of features could be chosen.

3.5 Experimental Setup

3.5.1 Evaluation Metrics for Hierarchical Predictive Accuracy

Unfortunately, in the task of hierarchical classification there are no standard measures to evaluate the results. Comprehensive reviews of hierarchical classification measures can be found in (Sun and Lim, 2001; Costa et al., 2007a). An aspect that can be criticized in the field is that most researchers still use flat classification measures to evaluate their hierarchical classification algorithms. Therefore, the question that naturally arises, since there is no consensus in the literature, is

"What evaluation metric to use?". In order to evaluate the algorithms we have used the metrics of hierarchical precision (hP), hierarchical recall (hR) and hierarchical f-measure (hF) proposed in (Kiritchenko et al., 2005) (See Section 2.7). These measures are extended versions of the well known metrics of precision, recall and f-measure but tailored to the hierarchical classification scenario.

3.5.2 Data Preparation

The protein datasets used in this work were originally developed in (Holden and Freitas, 2009). These datasets were originally created from the information about two types of proteins (Enzymes and GPCRs – G-Protein Coupled Receptors) obtained from different protein databases. For both datasets, the classes (protein functions) form a tree where each node represents a class. An excerpt of the class tree associated with the Enzymes dataset was shown in Figure 3.5, where classes at different levels are separated by a "." E.g., as shown in that figure, there are 6 classes at the first level, each of them sub-divided into sub-classes, and so on, until the fourth class level. Each class essentially refers to the type of chemical reaction catalyzed by an enzyme. In the case of the GPCR dataset, each class essentially denotes the type of ligand that binds to the GPCR (GPCRs essentially transmit signals received from ligands outside the cells to other molecules inside the cell). For further details of the meaning of the functional classes in these two datasets, see (Holden and Freitas, 2009).

It should be noted that proteins obtained from biological databases contain non-standard amino acids and in such cases we have made the following substitutions, as it has been done in (Davies et al., 2007): B (either an asparagine or aspartic acid) $\rightarrow$ N (asparagine); Z (either a glutamine or a glutamic acid) $\rightarrow$ Q (glutamine); X (unknown residues) $\rightarrow$ A (alanine); U (selenocysteine) $\rightarrow$ C (cysteine).

Originally there were 8 datasets (4 for Enzymes and 4 for GPCR) created based on protein data available in the Uniprot database and motif information obtained from the Interpro, Pfam, Prints and Prosite databases. Each of those datasets contained only one type of motif representation. For example, the EC-Interpro dataset had as predictive attributes only the Interpro entries motifs.

One of the objectives of this work is to evaluate the impact of the many different types of representations discussed in Section 3.3. Therefore, we expanded the number of representations used in each of the original eight datasets by extracting the alignment-independent attributes described in Section 3.3. This means that each of these 8 datasets now has 5 representations (5z,15z,AA,LD,one type of motif). These datasets are hereafter referred to as single-motif datasets.

Although these datasets allow us to verify the impact of each of the alignment-independent features against each of the motif representations, they do not allow us to verify if there is any difference in the predictive power of the different motifs representations. For this reason, we have also created two new datasets, which we refer to as "multiple-motif EC" and "multiple-motif GPCR" which were created from the common proteins that appeared in all four corresponding specific datasets, i.e. the four datasets about EC or the four datasets about GPCR. Therefore, each multiple-motif dataset has 8 representations (5z,15z,AA,LD,Interpro motifs, Pfam motifs, Prints motifs and Prosite motifs).

Table 3.5 presents a summarized description of the datasets. The last column of Table 3.5 presents the number of classes at each level of the hierarchy (1st/2nd/3rd/ 4th levels). Note that concerning the number of protein representations, the multiple-motif datasets are more comprehensive than their single-motif counterpart datasets, because the 5 candidate representations used in a single-motif dataset are a proper subset of the 8 candidate representations used in the corresponding multiple-motif dataset. However, the motivation for performing the experiments on both single-motif and multiple-motif datasets is that the latter datasets have a reduced number of examples (specially in the case of Enzymes), since a protein is included in a multiple-motif dataset only if it appears in all the four single-motif datasets for the protein in question (Enzymes or GPCRs). Hence, the single-motif datasets have considerably more examples, offering a better statistical support to some experiments. The datasets used in the experiments are available at: http://sites.google.com/site/carlossillajr/resources/.

Table 3.5: Dataset details.

Dataset	# of Examples (Proteins)	# Classes per Level
Multiple-motif EC	5,221	6/35/47/70
EC-Interpro	14,027	6/41/96/187
EC-Pfam	13,987	6/41/96/190
EC-Prints	14,025	6/45/92/208
EC-Prosite	14,041	6/42/89/187
Multiple-motif GPCR	5,156	7/42/74/49
GPCR-Interpro	7,444	12/54/82/50
GPCR-Pfam	7,053	12/52/79/49
GPCR-Prints	5,404	8/46/76/49
GPCR-Prosite	6,246	9/50/79/49

3.6 Computational Results and Discussion

In this section, we will first discuss the impact of the different protein representations (which is less investigated in the literature) on the task of hierarchical protein function prediction and we will also discuss the impact of the different classifiers. Also, all the experiments were performed using 10-fold cross-validation.

3.6.1 Impact of the Different Protein Representations

Results for the Single-Motif Datasets

One of the main contributions of this chapter is to assess the impact of the choice of a type of protein representation on the hierarchical protein function prediction problem. Table 3.6 presents the results obtained by each representation on each single-motif dataset. It should be noted that in the cells at the intersection of a row where the feature type is the Sel. R. approach and the column for the Sel. C. approach, the results shown are based on the combination of both these approaches together.

However, verifying the particular importance of each protein representation is not straightforward, since as seen in section 3.3.5 different motif representations

Table 3.6: Hierarchical F-measure (hF) for the single-motif datasets with 5 protein representations.

Dataset	Type of Feature	k-NN hF	SVM hF	NB hF	Sel.C. hF
EC-Interpro	5z	42.36	18.56	21.68	40.85
	15z	50.44	20.98	24.39	47.80
	AA	51.86	23.92	25.29	49.14
	LD	57.76	31.52	15.71	55.79
	Interpro Motif	84.28	83.02	77.01	83.01
	Sel. R.	84.26	83.00	79.65	82.97
EC-Pfam	5z	40.39	18.74	21.98	39.40
	15z	50.30	21.15	24.64	47.60
	AA	53.13	23.86	25.76	50.29
	LD	58.94	33.45	17.81	57.28
	Pfam Motif	83.94	82.36	76.30	82.73
	Sel. R.	83.95	82.49	78.77	82.60
EC-Prints	5z	39.17	19.32	21.93	39.25
	15z	49.84	22.48	21.61	50.35
	AA	53.04	25.63	25.50	53.29
	LD	59.83	41.10	24.50	60.24
	Prints Motif	83.10	80.63	79.96	82.04
	Sel. R.	83.19	81.17	81.39	83.08
EC-Prosite	5z	42.92	16.60	19.73	43.65
	15z	51.82	21.21	22.91	52.52
	AA	53.23	23.42	24.34	54.50
	LD	59.26	32.60	14.14	58.52
	Prosite Motif	85.19	83.57	81.96	85.26
	Sel. R.	85.25	83.85	83.16	85.48
GPCR-Interpro	5z	60.80	45.06	46.98	60.58
	15z	73.07	57.11	51.35	72.93
	AA	78.03	63.56	53.19	77.95
	LD	82.12	77.51	60.35	82.27
	Interpro Motif	79.44	74.36	65.80	79.52
	Sel. R.	86.16	81.66	74.72	86.39
GPCR-Pfam	5z	62.24	46.40	48.29	62.10
	15z	74.82	59.43	52.85	74.78
	AA	79.68	65.72	55.55	79.80
	LD	83.54	78.79	62.06	83.57
	Pfam Motif	68.06	59.07	57.44	67.27
	Sel. R.	85.19	84.00	74.70	85.23
GPCR-Prints	5z	67.91	50.56	52.09	67.67
	15z	77.25	60.35	56.01	77.29
	AA	80.97	66.21	55.93	81.08
	LD	83.30	79.02	61.30	83.86
	Prints Motif	76.64	72.02	64.54	76.64
	Sel. R.	83.33	81.09	74.31	83.90
GPCR-Prosite	5z	67.05	49.45	50.95	66.87
	15z	76.27	58.14	54.65	76.21
	AA	80.79	64.09	53.97	80.83
	LD	82.69	78.47	61.73	82.92
	Prosite Motif	64.54	53.56	49.80	64.54
	Sel. R.	82.69	78.52	63.67	82.97

have very different rationale behind their development. For this reason, in the analysis of the different protein representations based on the single-motif datasets, we break down the analysis by the type of motif. That is, for each of the 4 types of motif, we analyse the result for both EC and GPCR datasets with that motif as a candidate representation to be selected. E.g., taking into account the results on both EC-Interpro and GPCR-Interpro, as they have the same type of motif-based protein representation.

For the Interpro-motif based datasets, considering all the representations (including the selective representation method), the average ranking of the protein representations (computed by the Friedman statistical test, considering the hierarchical f-measure values) is: Sel. R. (1.375), Interpro motifs (2.0), LD (3.0), AA (3.875), 15z (4.875) and 5z (5.875) (the smaller the rank number, the better the method). This ranking provides an overall order of the effectiveness of each protein representation across all datasets without going into the merit of wins/loses in individual datasets (Demsar, 2006). In order to identify on which pairwise comparisons there is a statistical difference between the results, we conduct a post-hoc test. As strongly recommended in (García and Herrera, 2008) we use the Shaffer static procedure for $\alpha = 0.05$. This combination of Friedman statistical test and Shaffer post-hoc test was used to produce all results shown in Figures 3.6 to 3.11. Figure 3.6 shows the result of this test in a graphical way as suggested in (Demsar, 2006). In Figure 3.6 the bold horizontal lines connect the representations whose results are not found to be statistically significantly different. (This graphical representation is also used in Figures 3.6 through 3.11.) The analysis of the results in Figure 3.6 shows that there is no statistical difference, when comparing the Sel. R., Interpro Motifs, LD and AA. There is a statistical difference when comparing the Sel. R., Interpro Motifs and LDs with 5z and 15z.

For the Pfam-motif based datasets, the average ranking of the protein representations is: Sel. R. (1.25), LD (2.875), Pfam motifs (3.125), AA (3.5), 15z (4.5) and 5z (5.875). Figure 3.7 shows the graphical result of the Shaffer static post-doc test. The analysis of the results in Figure 3.7 shows that there is no statistical

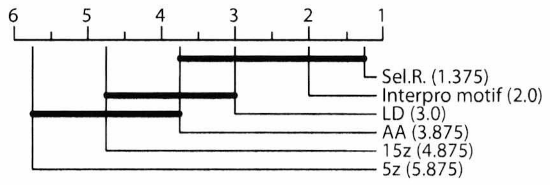


Figure 3.6: Analysis of relative protein representation importance on the Interpro-motif based datasets.

difference, when comparing the Sel. R., LD, Pfam Motifs and AA. There is a statistical difference when comparing the Sel. R., Pfam Motifs and LDs with 5z and 15z.

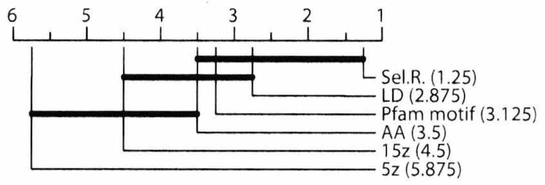


Figure 3.7: Analysis of relative protein representation importance on the Pfam-motif based datasets.

For the Prints-motif based datasets, the average ranking of the protein representations is: Sel. R. (1.0), LD (2.75), Prints motifs (2.875), AA (3.75), 15z (4.75) and 5z (5.875). The analysis of the results in Figure 3.8 shows that there is no statistical difference, when comparing the Sel. R., LD, Pfam Motifs and AA. There is a statistical difference when comparing the Sel. R., Prints Motifs and LDs with 5z and 15z.

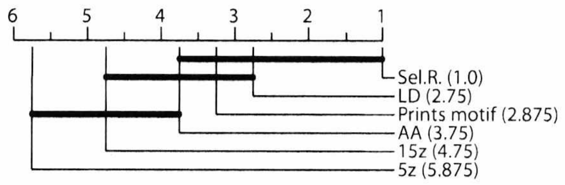


Figure 3.8: Analysis of relative protein representation importance on the Prints-motif based datasets.

For the Prosite-motif based datasets, the average ranking of the protein representations is: Sel. R. (1.0625), LD (2.8125), AA (3.5), Prosite motifs (3.875), 15z (4.25) and 5z (5.5). The analysis of the results in Figure 3.9 shows that there is no statistical difference, when comparing the Sel. R., LD, and AA. There is a statistical difference when comparing the Sel. R. with Prosite Motifs, 5z and 15z.

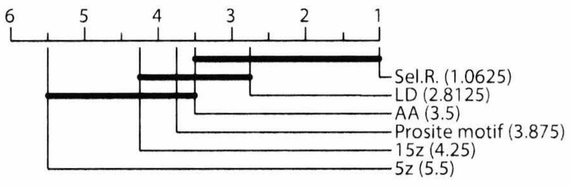


Figure 3.9: Analysis of Relative protein representation importance on the Prosite-motif based datasets.

Table 3.7: Hierarchical F-measures (hF) for the multiple-motif datasets with 8 protein representations.

Dataset	Type of Feature	k-NN hF	SVM hF	NB hF	Sel.C. hF
Multiple-motif EC	5z	54.92	33.39	40.99	63.56
	15z	62.55	36.97	45.00	65.66
	AA	63.70	40.11	47.34	73.32
	LD	65.67	56.44	39.94	74.42
	Interpro	79.91	79.59	79.07	81.93
	Pfam	79.43	79.07	77.39	79.37
	Prints	79.59	78.56	81.18	82.34
	Prosite	79.44	78.51	79.64	79.09
	Sel. R.	79.82	79.56	82.33	81.58
Multiple-motif GPCR	5z	68.63	51.85	53.39	68.56
	15z	77.97	61.87	57.39	77.96
	AA	81.58	67.66	57.08	81.65
	LD	83.58	79.60	62.49	84.30
	Interpro	79.79	75.47	64.54	79.89
	Pfam	63.32	51.70	51.79	62.66
	Prints	76.83	71.90	65.13	76.46
	Prosite	65.82	54.91	52.29	65.78
	Sel. R.	86.61	83.53	74.91	86.94

Results for the Multiple-Motif Datasets

Recall that apart from the single-motif datasets, we have also created two multiple-motif datasets in order to evaluate the performance of each particular type of motif against the others as well as against the alignment-independent features and the Sel. R. approach. Table 3.7 presents the hierarchical f-measure of each

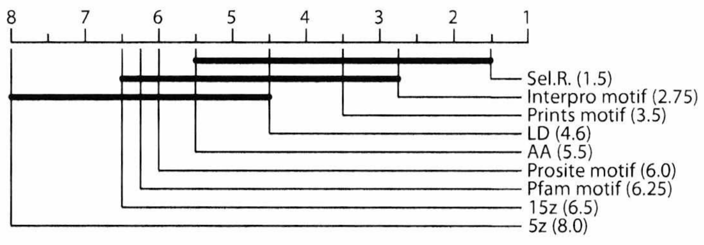


Figure 3.10: Analysis of relative protein representation importance on the multiple-motif datasets.

representation on the multiple-motif datasets. The average ranking of the protein representations (computed by the Friedman statistical test, considering the hierarchical f-measure values) is: Sel. R. (1.5), Interpro motifs (2.75), Prints motifs (3.5), LD (4.625), AA (5.5), Prosite motifs (6.0), Pfam motifs (6.625), 15z (6.5), 5z (8.0). Again, this ranking provides an overall order of the effectiveness of each protein representation across all datasets and classification algorithms without going into the merits of individual wins/loses. Figure 3.10 shows the graphical result of the Shaffer static post-doc test. The analysis of the results in Figure 3.10 shows that there is no statistical difference, when comparing the Sel. R., Interpro motifs, Prints motifs, LD, and AA. There is a statistical difference when comparing the Sel. R. with Pfam and Prosite Motifs, 5z and 15z.

Discussion of Results for Different Protein Representations

The overall analysis of the results shows some interesting points. First, although not statistically significantly different from some representations, the Sel. R. has ranked 1st in all experiments, meaning that it is an interesting approach to deal with the problem of hierarchical protein function prediction.

Second, the result that 15z is better than 5z (although not statistically significant) corroborates with the experiments of (Davies et al., 2008a; Secker et al., 2010) where the authors came to the same conclusion. Note, however, that in their experiments they used only one GPCR dataset, while in this study we have employed 4 GPRC and 4 Enzyme datasets. Our work therefore, validates their initial proposal in a larger number of datasets. According to (Secker et al., 2007), the

Table 3.8: Percentage of times each representation is selected by the Sel. R. method per class level per dataset for 5 protein representations.

Rep.	Dataset	Class Level			
		1	2	3	4
5z	EC-Interpro	0	0	8	22
	EC-Pfam	0	0	8	21
	EC-Prints	0	0	7	14
	EC-Prosite	0	0	4	13
	GPCR-Interpro	0	7	10	7
	GPCR-Pfam	0	8	10	8
	GPCR-Prints	0	8	11	8
	GPCR-Prosite	0	17	11	8
	Average	0	5.625	10.875	14.375
15z	EC-Interpro	0	4	6	11
	EC-Pfam	0	0	5	11
	EC-Prints	0	0	9	17
	EC-Prosite	0	0	5	9
	GPCR-Interpro	0	8	15	17
	GPCR-Pfam	0	11	16	19
	GPCR-Prints	0	17	13	15
	GPCR-Prosite	0	5	18	16
	Average	0	5.625	10.875	14.375
AA	EC-Interpro	0	3	12	14
	EC-Pfam	0	2	11	19
	EC-Prints	0	0	14	19
	EC-Prosite	0	0	10	18
	GPCR-Interpro	0	32	15	29
	GPCR-Pfam	0	31	20	30
	GPCR-Prints	0	29	18	33
	GPCR-Prosite	0	23	22	32
	Average	0	15	15.25	24.25
LD	EC-Interpro	0	1	7	7
	EC-Pfam	0	6	12	8
	EC-Prints	0	12	14	16
	EC-Prosite	0	1	8	13
	GPCR-Interpro	0	17	44	36
	GPCR-Pfam	0	28	54	41
	GPCR-Prints	0	22	40	31
	GPCR-Prosite	75	43	48	42
	Average	9.375	16.25	28.375	24.25
Motifs	EC-Interpro	100	92	67	46
	EC-Pfam	100	92	64	41
	EC-Prints	100	88	56	34
	EC-Prosite	100	99	73	47
	GPCR-Interpro	100	36	16	11
	GPCR-Pfam	100	22	0	2
	GPCR-Prints	100	24	18	13
	GPCR-Prosite	25	12	1	2
	Average	90.625	58.125	36.875	24.5

z-values representation provides a numerical description of the proteins' physico-chemical properties that potentially results in a higher predictive accuracy than the use of amino acid sequence composition. However, in our experiments we have empirically verified that this is not the case, since the AA features are always ranked above both 5z and 15z features (although this difference is not statistically significant).

Third, the best performing alignment-independent feature is the LD. Its results are better than all the other alignment-independent features (although only statistically significantly different from 5z on some motif-based datasets).

Fourth, the use of the alignment-dependent features (motifs) on the single-motif datasets have ranked 2nd for Interpro motifs, 3rd for Pfam and Prints motifs and 4th for the Prosite motifs. On the multiple-motif datasets the alignment-dependent features (motifs) have ranked 2nd (Interpro), 3rd (Prints), 6th (Prosite) and 7th (Pfam). Considering the rankings it is clear that the use of Interpro motifs lead to higher predictive accuracies than the use of other types of motifs. This is an expected result, since (as previously discussed) Interpro is a joint effort from curators of all its members databases which includes Prosite, Prints and Pfam among others.

Note that the Sel. R. was the best protein representation on both experiments (single-motif datasets and multiple-motif datasets). Hence, it is interesting to analyse which features were selected the most by the selective representation approach at each level of the class hierarchy. Tables 3.8 and 3.9 present the percentage of how many times a particular protein representation was selected in each dataset at each level of the class hierarchy for the datasets with 5 and 8 representations, respectively, corresponding to single-motif and multiple-motif datasets, respectively.

The analysis of Table 3.8 shows that for the single-motif datasets, the motif features are highly predictive for the classes at the first level of the class hierarchy being selected on average in 90.6% of the time. In fact, the only dataset

Table 3.9: Percentage of times each representation is selected by the Sel. R. method per class level per dataset for 8 protein representations.

Rep.	Dataset	Class Level			
		1st	2nd	3rd	4th
5z	Multiple-motif EC	0	0	5	23
	Multiple-motif GPCR	0	15	11	9
	Average	0	7.5	8	16
15z	Multiple-motif EC	0	7	5	4
	Multiple-motif GPCR	0	7	13	14
	Average	0	7	9	9
AA	Multiple-motif EC	0	5	11	33
	Multiple-motif GPCR	0	17	14	31
	Average	0	11	12.5	32
LD	Multiple-motif EC	0	5	8	12
	Multiple-motif GPCR	0	28	32	28
	Average	0	16.5	20	20
Interpro	Multiple-motif EC	95	37	49	23
	Multiple-motif GPCR	73	26	22	15
	Average	84	31.5	35.5	19
Pfam	Multiple-motif EC	0	12	2	0
	Multiple-motif GPCR	25	7	0	0
	Average	12.5	9.5	1	0
Prints	Multiple-motif EC	0	13	18	1
	Multiple-motif GPCR	2	0	8	3
	Average	1	6.5	13	2
Prosite	Multiple-motif EC	5	21	2	4
	Multiple-motif GPCR	0	0	0	0
	Average	2.5	10.5	1	2

where other type of protein representation is selected at this level is the GPCR-Prosite dataset, where the LD representation is selected 75% of the time. For the other three class levels, it seems that the motifs are often selected for the EC datasets, while a combination of alignment-independent features are selected for GPCRs. An explanation for this was presented in (Davies et al., 2008b) were the authors claim that there are several instances where the application of alignment-independent techniques have been proven to be more effective than alignment-dependent techniques. And the GPCRs are an example of this, because they have a great structural and/or functional homology but a low degree of sequence similarity.

For the multiple-motif datasets presented in Table 3.9 the same conclusions can be drawn. That is, the motif features are highly discriminative at the 1st level of the class hierarchy, specially the Interpro motifs. There is a significant difference in the number of times that motif-based and alignment-independent features are selected for the Enzyme and GPCRs datasets. These results confirm that the Sel. R. approach effectively determines which protein representation is the best to be used with each classifier across different levels in the class hierarchy structure.

3.6.2 Impact of the Different Classifiers

It is a well-known fact in machine learning that there is "no free-lunch", i.e. a classifier which is the best for all applications does not exist. Recall that in this work we are employing the selective classifier approach with three classification algorithms: k-NN, SVM and NB. To measure the performance of the classifiers we consider their average ranking over all datasets and over all representations (computed by the Friedman statistical test, considering the hierarchical f-measure values). The resulting ranking is: Sel. C. (1.5454), k-NN (1.560606), SVM (3.3181), NB (3.5757).

Again we employ the Shaffer static post-hoc test and the graphical representation of the result of the test is shown in Figure 3.11. The analysis of the results in Figure 3.11 shows that the Sel. C. and k-NN are both (statistically significant) better than SVM and NB, but there is no statistically significant difference between the results of Sel. C. and k-NN. Also, there is no statistically significant difference between the results of SVM and NB.

Considering we are using the Sel. C. approach and it gives results just slightly better than the k-NN classifier, a question that naturally arises is if in the internal classifier selection procedure of the Sel. C. method the k-NN classifier is almost always chosen. Table 3.10 presents the relative classifier importance for each dataset, i.e. the number of times a particular classifier is selected at each

Table 3.10: Percentage of times each classifier is selected by the Sel. C. method per class level per dataset.

Classifier	Dataset	Class Level			
		1	2	3	4
SVM	EC-Interpro	20	35	41	59
	EC-Pfam	20	36	41	59
	EC-Prints	0	30	39	47
	EC-Prosite	8	23	37	50
	GPCR-Interpro	0	51	50	56
	GPCR-Pfam	18	49	42	44
	GPCR-Prints	0	49	47	60
	GPCR-Prosite	0	44	43	44
	Multiple-motif EC	12	57	52	66
	Multiple-motif GPCR	14	36	36	47
	Average	9.2	41	42.8	53.2
KNN	EC-Interpro	80	65	49	20
	EC-Pfam	80	64	50	15
	EC-Prints	100	70	44	27
	EC-Prosite	92	77	61	23
	GPCR-Interpro	100	36	42	30
	GPCR-Pfam	82	40	48	38
	GPCR-Prints	100	40	44	28
	GPCR-Prosite	100	46	45	39
	Multiple-motif EC	88	26	37	15
	Multiple-motif GPCR	86	55	56	43
	Average	90.8	51.9	47.6	27.8
NB	EC-Interpro	0	0	10	21
	EC-Pfam	0	0	9	26
	EC-Prints	0	0	17	26
	EC-Prosite	0	0	2	27
	GPCR-Interpro	0	13	8	14
	GPCR-Pfam	0	11	10	18
	GPCR-Prints	0	11	9	12
	GPCR-Prosite	0	10	12	17
	Multiple-motif EC	0	17	11	19
	Multiple-motif GPCR	0	9	8	10
	Average	0	7.1	9.6	19

class level. The analysis of Table 3.10 reveals that at the first level the k-NN classifier is selected in about 90% of the experiments. This result corroborates with the experiments reported in (Secker et al., 2010) where for one GPCR dataset the k-NN classifier was always selected at the first class level. For the other class levels it seems that, although the Sel. C. approach actually selects different classifiers, this does not impact significantly on the results. In the few cases that the Sel. C. performs worse than k-NN, it is possible that the Sel. C. approach is overfitting during the training phase given its greedy nature to select the best classifier. Other studies on hierarchical protein function prediction that employed the Sel. C. approach achieved similar conclusions (Secker et al., 2007; Holden and Freitas, 2008), i.e. the Sel. C. is better than most classifiers but is not statistically significantly different from a k-NN classifier, even though the former employs several classifiers, which has the disadvantage of considerably increasing the training time of the hierarchical classification system. Therefore, it seems that the use of the Sel. C. approach does not bring the same benefits as the Sel. R. approach.

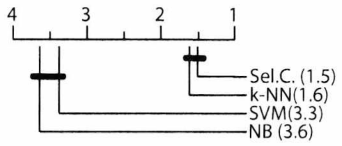


Figure 3.11: Analysis of relative classifier importance over all datasets.

Moreover, the negative impact of using a bad representation even with a good classifier (e.g. 5z with k-NN on EC-Interpro has an hierarchical f-measure of 42.36%) seems to be greater than the impact of using a bad classifier with a good representation (e.g. Motif with NB on EC-Interpro has an hierarchical f-measure of 77.01%). Interestingly, most papers in protein function prediction are more concerned with trying different classification algorithms than different protein representations (type of features) and their impact on predictive accuracy.

Chapter 4

Global-Model Hierarchical-Classification Naive Bayes

Most of the classification research in machine learning focuses on the development and improvement of flat classification methods. In addition, in the field of hierarchical classification most approaches use a local-model approach with a top-down class prediction approach. As seen in Chapter 2 the global–model approach is still under-explored in the literature and it deserves more investigation because it builds a singular coherent classification model. Even though a single model produced by the global-model approach will tend to be more complex (larger) than each of the many classification models produced by the local–model approach, intuitively the single global model will tend to be much simpler (smaller) than the entire hierarchy of local classification models. There is also empirical evidence for this intuitive reasoning (Costa et al., 2007b), (Vens et al., 2008).

In this Chapter we present a novel type of global-model hierarchical-classification algorithm. This new global–model hierarchical–classification algorithm is a modified version of the Naive Bayes classifier which was extended to deal with hierarchical classification problems. The choice of extending a naive Bayes classifier to deal with hierarchical classification problems, instead of extending other machine learning algorithms, is justified by the following reasons: First, in the field of protein function prediction there is a need for white box model approaches, i.e. algorithms whose reasoning can be understood/interpreted by biologists (Szafron et al., 2004; He et al., 2006; Freitas et al., 2010). Second, Naive Bayes has the advantage of being a very fast classification algorithm, requiring computational time which is linear in both the number of examples and number of attributes. This important in hierarchical classification problems, which typically require much more time for building classification models than flat classification problems. In addition, note that the fact that the local model version of Naive Bayes had the worst performance when compared to the k-NN and SVM classifiers in Chapter 3 does not mean that a global model version would also have a relatively bad performance, since the local and global model versions of Naive Bayes are very different. Hence, there is no way of knowing how the global version of these classifiers will perform in hierarchical classification problems without creating them. This chapter is an extension of one of the published papers during the research (Silla Jr. and Freitas, 2009a).

4.1 Flat Naive Bayes Algorithm

The flat Naive Bayes algorithm is well known in the field of data mining for being a simple approach, with clear semantics, to representing, using and learning probabilistic knowledge (Witten and Frank, 2005). The flat Naive Bayes algorithm works as follows:

Given a training set (Tr) with attributes and its respective class labels. The Naive Bayes classifier uses the Bayes theorem (Han and Kamber, 2006) to classify a new test example ($\mathbf{X} = a_{1j}, a_{2j}, \ldots, a_{n_a j}$). It computes the posterior probability ($P(C_k|\mathbf{X})$)) for each of the classes $C_k, k = 1 \ldots n_c$, where n_c is the number of classes. The class with the highest posterior probability is chosen, as the class label:

Table 4.1: Symbols used in this chapter.

Symbol	Meaning
Tr	the set of all training examples.
A	Set of attributes.
A_i	i-th attribute.
a_{ij}	i-th attribute with j-th value.
C	Set of classes.
C_k	k-th class.
$P(C\|e)$	Probability of the class C given the evidence e.
$\mathbf{X}$	A test example containing a set of attributes.
n_c	the number of classes.
n_a	the number of attributes.
n_e	the number of examples.
n_{vi}	the number of attribute values for A_i.

$$classify(\mathbf{X}) = \arg\max_{C_k} P(\mathbf{X}|C_k) \times P(C_k) \tag{1}$$

where:

- $P(\mathbf{X}|C_k)$ is the likelihood;
- $P(C_k)$ is the Prior;

The Naive Bayes classifier assumes that all attributes are independent of each other given the class, hence:

$$P(\mathbf{X}|C_k) = \prod_{i=1}^{n_a} P(A_i = a_{ij}|C_k) = P(a_{1j}|C_k) \times P(a_{2j}|C_k) \ldots \times P(a_{n_aj}))|C_k) \tag{2}$$

Considering that A_i is categorical, then $P(A_i|C_k)$ is estimated by the number of examples of class C_k in Tr having the value a_{ij} for A_i, divided by the number of examples of class C_k in Tr.

In the case that A_i is continuous there are two options to deal with it. The first option is to estimate the underlying probability of the given attribute. The second option is to discretise (convert/transform) the continuous attribute into a categorical one. How to discretise a numerical attribute is an area of research by itself. However for Naive Bayes classification, it is a well known fact that discretisation is recommended over trying to estimate an unknown real-world underlying probability (Dougherty et al., 1995). There are many different types of discretisation algorithms, but they can be broadly classified (with regard to their class awareness) into supervised and unsupervised discretisation methods (Dougherty et al., 1995). An excellent review of discretisation methods for Naive Bayes is presented in (Yang and Webb, 2009). In this work, due to the skewed nature of the the data, as it will be discussed in Section 4.4, we will discretise all numerical attributes using an unsupervised discretisation method known as Equal Frequency Discretization (EFD) (Yang and Webb, 2009).

In practice another step that is needed to be taken into account when estimating $P(A_i|C_k)$ is the case of when in the training data, a particular value of a_{ij} does not occur. In this case this value would be zero (0) and would drastically affect the performance of the classifier. In this work, when such case happens, we estimate this value to be $\frac{1}{n_e}$.

For each of the classes $C_k, k = 1, \ldots, n_C$, the prior probability of the class is estimated by using the number of examples in Tr with class C_k divided by the total number of examples in Tr, as shown in Eq. 3:

$$P(C_k) = \frac{\#(C_k)}{\#Tr} \tag{3}$$

where:

- $\#(C_k)$ is the total number of examples in Tr with class C_k.
- $\#Tr$ is the total number of examples in Tr.

The main advantages of the Naive Bayes classifier are its simplicity and computational efficiency (an important point in hierarchical classification, given the large number of classes to be predicted), however it is sensitive against redundant attributes (Witten and Frank, 2005).

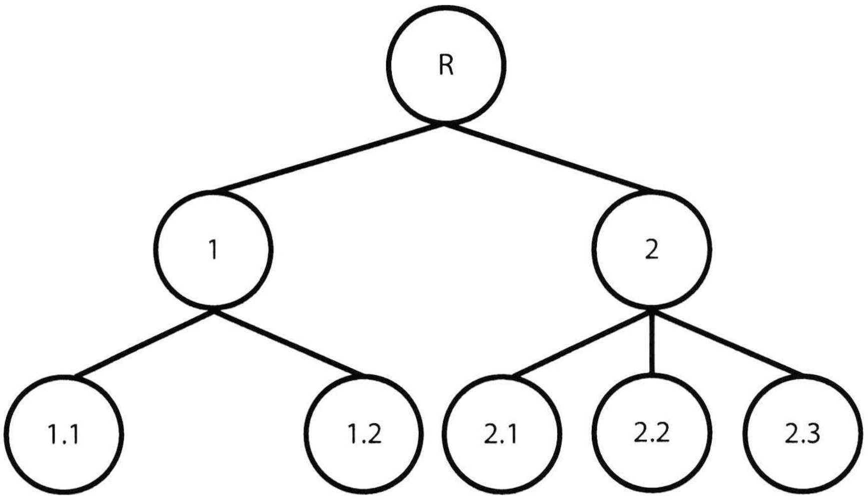


Figure 4.1: Hierarchical class structure example.

4.2 Global-Model Hierarchical-Classification Naive Bayes

In the previous section we have reviewed how the flat classification Naive Bayes works. In this section we will show how we have modified the algorithm to cope with hierarchical classification problems.

Considering Figure 4.1, where each node in the tree corresponds to a class, the flat Naive Bayes algorithm has to be modified to take into account all the classes in the hierarchy (and their parent-child relationships) in order to compute the prior probabilities and the likelihoods.

In the prior probabilities computation, using Figure 4.1 as an example, the flat Naive Bayes algorithm would compute the probabilities for only the leaf classes, i.e.: P(1.1), P(1.2), P(2.1), P(2.2), P(2.3). The prior probability computation needs to be modified to take into account all the classes in the hierarchy. That is, we want the algorithm to compute the prior probability for all the classes in the hierarchy, i.e.: P(1), P(2), P(1.1), P(1.2), P(2.1), P(2.2), P(2.3).

In order to modify the prior probability calculations to take the hierarchy into account, during the training phase, we will assume that any example which belongs to class C_k will also belong to all its ancestor classes. For example, if a training example belongs to a certain class (say class 2.1), this means that the prior probabilities of both that class and its ancestor classes (i.e. classes 2 and 2.1 in this case) are going to be updated. (This is because we are dealing with a "IS-A" class hierarchy, as usual.)

Hence, for the global model Naive Bayes (GMNB), the prior computation ($P(C_k)$) for each of the classes $C_k, k = 1, \ldots, n_C$, is estimated by using the number of examples in Tr with class C_k (as in flat Naive Bayes) plus the number of examples in Tr which are descendants of C_k divided by the total number of examples in Tr, as shown in Eq. 4:

$$P(C_k) = \frac{\#(C_k) + \#(\Downarrow (C_k))}{\#Tr} \tag{4}$$

where:

- $\#(C_k)$ is the total number of examples in Tr whose most specific class is C_k.
- $\#(\Downarrow (C_k))$ is the total number of examples in Tr whose most specific class is descendant of C_k.
- $\#Tr$ is the total number of examples in Tr.

In the likelihood computation, using Figure 4.1 as an example, the flat Naive Bayes algorithm would compute the likelihood for only the leaf classes, i.e.: P(a_{ij}|1.1), P(a_{ij}|1.2), P(a_{ij}|2.1), P(a_{ij}|2.2), P(a_{ij}|2.3). This is computed for each attribute A_i, with each value a_{ij} belonging to the domain of A_i, i = 1,...,n_a, j = 1,...,n_{vi}, where n_a is the number of attributes and n_{vi} is the number of values in the domain of the A_{i-th} attribute. The likelihood computations need to be modified to compute the likelihood for all classes in the hierarchy, i.e.: P(a_{ij}|1), P(a_{ij}|2), P(a_{ij}|1.1), P(a_{ij}|1.2), P(a_{ij}|2.1), P(a_{ij}|2.2), P(a_{ij}|2.3).

As with the prior calculations modifications, we will assume that any example which belongs to class C_k will also belong to all its ancestor classes. This way,

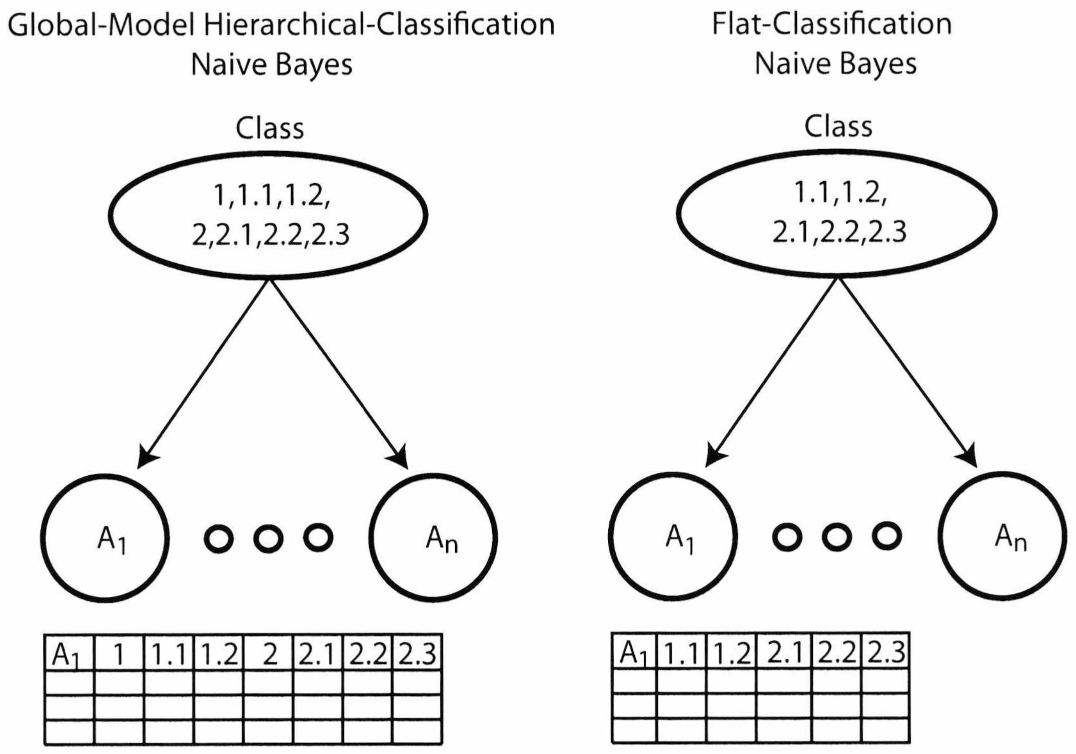

Global-Model Hierarchical-Classification Naive Bayes
Class
1,1.1,1.2, 2,2.1,2.2,2.3
A1
An
A1 1 1.1 1.2 2 2.1 2.2 2.3
Flat-Classification Naive Bayes
Class
1.1,1.2, 2.1,2.2,2.3
A1
An
A1 1.1 1.2 2.1 2.2 2.3

Figure 4.2: Example to illustrate difference between GMNB and NB.

when a training example is processed, its attribute-value pair counts are added to the counts of the given class and its ancestor classes. As in the previous example, if the training example belongs to class 2.1, the attribute-value counts are added to the counts of both classes 2 and 2.1.

These modifications during the training phase, will allow the algorithm to predict classes at any level of the hierarchy during the testing phase. In order to classify a new example, the global model Naive Bayes simply assigns the class with maximum value of the posterior probability, as seen in Eq. 1, and outputs as the class label the class with the maximum posterior probability and its ancestor classes. Figure 4.2 illustrates the difference between the GMNB and the NB considering a problem with the class structure of Figure 4.1.

According to the taxonomy for classifying hierarchical classification algorithms, presented in Chapter 2, where a hierarchical classification algorithm is described as a 4-tuple $< \Delta, \Xi, \Omega, \Theta >$, the classification of the global model Naive Bayes is as follows:

- Δ = SPP (Single Path Prediction), since the algorithm can only assign to each data example one path of predicted labels.
- Ξ = NMLP (Non-Mandatory leaf-node prediction), since the algorithm can assign classes at any level (including leaf classes).
- Ω = D (DAG). Although the algorithm has only been illustrated with tree-structured class problems so far, it can also cope with SPL (Single Path of Labels) DAG-structured class problems. In the case of the DAG-structured problem, the algorithm would compute all the priors and likelihoods in the same way as previously discussed.
- Θ = GC (Global Classifier), as the algorithm takes the hierarchy into account during the training phase and can predict any class during the test phase.

4.3 Global-Model Hierarchical-Classification Naive Bayes with Usefulness

In section 4.2 we presented the Global-Model Naive Bayes algorithm which uses the MAP (Maximum a posteriori) rule to predict the final class label. Although the predictions of deeper classes are often less accurate (since deeper classes have fewer examples to support the training of the classifier than shallower classes), deeper class predictions tend to be more useful to the user, since they provide more specific information than shallower class predictions. If we only consider the posterior class probability (the product of likelihood $\times$ prior class probability) we would not take into account the usefulness to the user. It is interesting therefore to select a class label which has a high posterior probability and is also useful to the user. Therefore an optional step in the proposed method is to predict the class with maximum value of the product of posterior probability × usefulness. The question that arises is how to evaluate the usefulness of a predicted class?

Given that predictions at deeper levels of the hierarchy are usually more informative than the classes at shallower levels, some sort of penalization for shallower class predictions is needed. In Clare's work (Clare, 2004) the original formula for entropy was modified to take into account two aspects: multiple labels and prediction depth (usefulness) to the user. In this work we have modified part of the entropy-based formula described in (Clare, 2004). The main reason to modify this formula is that while Clare was using a decision tree classifier based on entropy, in this work we are using a Bayesian algorithm that makes use of probabilities. Therefore, we need to adapt the "usefulness" measure from (Clare, 2004) to the context of our algorithm. Also, all that we need is a measure to assign different weights to different classes at different class levels. Therefore, we adapt Clare's measure of usefulness by using a normalized usefulness value based on the position of each class level in the hierarchy. Moreover, we only use the normalized value of the Clare's equation to measure the usefulness:

$$usefulness(C_k) = 1 - (\frac{a(C_k) log_2 treesize(C_k)}{max}) \tag{5}$$

where:

- $treesize(C_k) = 1+$ number of descendant classes of C_k (1 is added to represent C_k itself)
- $a(C_k) = 0$, if $p(C_k) = 0$; $a(C_k) =$ a user defined constant (default=1) otherwise.
- max is the highest value obtained by computing $a(C_k) log_2 treesize(C_k)$ for all classes $C_k, k = 1, \ldots, n_c$ (where n_c is the number of classes) and it is used to normalize all the other values into the range $[0, 1]$.

To make the final class prediction, the global-model Naive Bayes with Usefulness (GMNBwU) assigns, to the current test example, the class label which maximizes the product of the posterior probability and usefulness as shown in Equation 6.

$$classify(\mathbf{X}) = \arg\max_{C_k} \prod_{i=1}^{n_a} (P(A_{i=a_{ij}}|C_k) \times P(C_k)) \times Usefulness(C_k) \qquad (6)$$

4.4 Experimental Set Up

4.4.1 Establishing a Baseline Method

An important issue when dealing with hierarchical classification is how to establish a meaningful baseline method. Since we are dealing with a problem where the classifier's most specific class prediction for an example can be at any level of the hierarchy (non-mandatory leaf node prediction – see Chapter 2), it is fair to have a comparison against a method whose most specific class prediction can also be at any level in the class hierarchy.

Therefore, in this work, as a baseline method, we use the same broad type of classifier (Naive Bayes), but with a conventional local classifier per parent node approach with a top-down class prediction testing approach. More precisely, during the training phase, for every non-leaf class node, a Naive Bayes multi-class classifier was trained to distinguish between the node's child classes. To implement the test phase, we used the top-down class prediction strategy (see Chapter 2) in the context of a non-mandatory leaf-node class prediction problem. The criterion for deciding at which level to stop the classification during the top-down classification process is based on the usefulness measure (see Section 4.3).

Since we already have the measure for usefulness of a predicted class, we decided to use the following stopping criterion: If $p(C_k) \times usefulness(C_k) > p(C_j) \times usefulness(C_j)$ for all classes C_j that are a child of the current class C_k, then stop classification. In other words, if the posterior probability times the usefulness (given by Equation 5) computed by the classifier at the current class node is higher than the posterior probability times the usefulness computed for each of its child class nodes, then stop the classification at the current class node – i.e., make that class the most specific (deeper) class predicted for the current test example. This approach is hereafter referred to as Local Model Naive Bayes approach with usefulness (LMNBwU).

In order to verify how effective is the usefulness criterion used by the LMNBwU, we have also implemented the same Local Model Naive Bayes with the standard top-down approach. That is, an approach that always classify a test example with a leaf node. This approach is hereafter referred to as Local Model Naive Bayes (LMNB).

4.4.2 Protein Function Prediction Datasets Used in the Experiments

In this work we have used datasets about two different proteins families: Enzymes and GPCRs (G-Protein-Coupled Receptors). Enzymes are catalysts that accelerate chemical reactions while GPCRs are proteins involved in signalling and are particularly important in medical applications as it is believed that from 40% to 50% of current medical drugs target GPCR activity (Filmore, 2004). In each dataset, each example represents a protein.

Each dataset contains different types of predictor attributes (with no missing values), and in each dataset the classes to be predicted are hierarchical protein functions (see Chapter 3). More precisely:

- Datasets whose name ends with 5z specify that the dataset contains 5-Z values as predicting attributes.
- Datasets whose name ends with 15z specify that the dataset contains 15-Z values as predicting attributes.
- Datasets whose name ends with AA specify that the dataset contains Amino

Acid Composition as predicting attributes.

- Datasets whose name ends with LD specify that the dataset contains Local Descriptors as predicting attributes.
- Datasets whose name does not contain any special ending contains "protein signature" (or motif) as predicting attributes. Which particular type of motif is present in the dataset is identified by its name. For example EC-Interpro, means Enyzme Commission with Interpro Entries. The motifs used in this work were: Interpro Entries, FingerPrints from the Prints database, Prosite Patterns and Pfam.

Before performing the experiments, the following pre-processing steps were applied to the datasets:

1. Every class with fewer than 10 examples was merged with its parent class. If after this merge the class still had fewer than 10 examples, this process would be repeated recursively until the examples would be labeled to the Root class.
2. All examples whose most specific class was the Root class were removed.
3. An unsupervised discretisation algorithm based on Equal Frequency Discretisation (using 20 intervals) (Dougherty et al., 1995) was applied to all numerical attributes, including the molecular weight and sequence length attributes, which are present in all datasets. For this discretisation step we have used the implementation available in the WEKA framework (Witten and Frank, 2005). Due to the skewed nature of the datasets, it was not possible to apply any supervised (or class-aware) discretisation algorithms.

Table 4.2 present the datasets' main characteristics after these pre-processing steps. The 4th column of Table 4.2 presents the number of classes at each level of the hierarchy (1st/2nd/3rd/ 4th levels). The 5th, 6th and 7th columns of Table 4.2 presents the classification for a hierarchical classification problem, according to

Table 4.2: Bioinformatics datasets details.

Dataset	n_a	n_e	n_c/Level	Υ	Ψ	Φ
EC-Interpro	1,216	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-5z	7	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-15z	17	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-AA	22	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-LD	212	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Pfam	708	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-5z	7	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-15z	17	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-AA	22	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-LD	212	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Prints-Motif	382	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-5z	7	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-15z	17	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-AA	22	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-LD	212	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prosite	585	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-5z	7	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-15z	17	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-AA	22	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-LD	212	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
GPCR-Interpro	450	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-5z	7	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-15z	17	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-AA	22	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-LD	212	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Pfam	75	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-5z	7	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-15z	17	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-AA	22	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-LD	212	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Prints	283	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-5z	7	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-15z	17	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-AA	22	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-LD	212	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prosite	129	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-5z	7	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-15z	17	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-AA	22	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-LD	212	6,246	9/50/79/49	T	SPL	$PD_{8\%}$

the taxonomy presented in Chapter 2, where a hierarchical classification problem is described as a 3-tuple $< \Upsilon, \Psi, \Phi >$, where:

- $\Upsilon = T$ (tree), indicating that the classes to be predicted are arranged into a tree structure;
- $\Psi = SPL$ (Single Path of Labels);
- $\Phi = PD$ (Partial Depth Labeling), where $PD_{\%}$ means the percentage of the examples in the dataset that have partial depth labeling.

The pre-processed version of the datasets (as they were used in the experiments) are available at: http://sites.google.com/site/carlossillajr/resources/.

4.5 Computational Results

In this section, we are interested in answering the following questions by using controlled experiments: (a) How does the choice of a local (with top-down class prediction approach) or global (with the proposed method) approach affect the predictive performance of the algorithms? (b) How does the inclusion of the usefulness criterion (Equation 5) affect the global model Naive Bayes algorithm? (c) How does the inclusion of the usefulness criterion (Equation 5) affect the local model Naive Bayes algorithm with the top-down strategy? These questions are addressed in the next subsections.

All the experiments reported in this section were obtained by using the datasets presented in Section 4.4.2, using stratified ten-fold cross-validation (Witten and Frank, 2005). In order to evaluate the algorithms we have used the metrics of hierarchical precision (hP), hierarchical recall (hR) and hierarchical f-measure (hF) proposed in (Kiritchenko et al., 2005) (See Section 2.7).

To measure if there is any statistically significant difference between the hierarchical classification methods (measured by hierarchical f-measure) being compared, we have employed the Friedman test with the post-hoc Shaffer's static procedure (with $\alpha = 0.05$) for comparison of multiple classifiers over many datasets as strongly recommended by (García and Herrera, 2008). The results of the statistical test are presented in a graphical way (as suggested in (Demsar, 2006)) in Figure 4.3 in order to summarize the pairwise comparisons made by the test. In Figure 4.3 the bold horizontal lines connect the representations whose results are not found to be statistically significantly different.

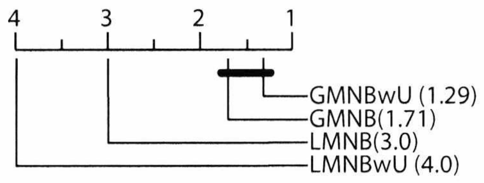


Figure 4.3: Statistical tests for the hierarchical naive Bayes classifiers with $\alpha = 0.05$.

4.5.1 Evaluating the Local Vs. Global Model Approaches

We first evaluate the impact of the usefulness component in the different types of hierarchical classification algorithms. Table 4.3 presents the results comparing the baseline local-model Naive Bayes approaches (LMNBwU and LMNB) described in Section 4.4.1 with the proposed global-model Naive Bayes (GMNBwU and GMNB), both with and without usefulness.

For all forty datasets the proposed global-model with usefulness obtained significantly better results than the local-model with usefulness. The statistical significance of the detailed results shown in Table 4.3 is confirmed by the statistical test shown in Figure 4.3. The same result is achieved by the global-model Naive Bayes without usefulness and the Local-model Naive Bayes without usefulness (LMNB).

These results corroborate with the ones reported in (Vens et al., 2008) where a global–model decision-tree approach was also better than a local-model one. Most previous studies comparing the local–model and global–model approaches have focused on mandatory leaf node prediction problems (Costa et al., 2007b; Ceci and Malerba, 2007), which is a simpler scenario – since there is no need

Table 4.3: Hierarchical Precision (hP), Recall (hR) and F-Measure (hF) on the hierarchical protein function datasets.

	LMNB			LMNBwU			GMNB			GMNBwU		
Dataset	hP	hR	hF	hP	hR	hF	hP	hR	hF	hP	hR	hF
EC-Interpro	87.40	92.54	89.90	93.77	61.58	74.34	94.76	89.80	92.20	93.82	93.04	93.43
EC-Interpro-5z	32.15	34.28	33.18	36.88	23.97	29.06	46.90	40.61	43.53	44.95	44.67	44.81
EC-Interpro-15z	34.13	36.37	35.22	38.87	24.57	30.11	53.05	44.77	48.55	51.93	48.80	50.31
EC-Interpro-AA	35.99	38.17	37.05	40.36	26.10	31.70	59.12	51.24	54.89	57.57	55.41	56.47
EC-Interpro-LD	37.17	39.24	38.18	40.98	25.69	31.58	58.41	48.32	52.88	58.14	50.56	54.07
EC-Pfam	86.12	90.99	88.49	92.66	59.73	72.64	95.10	86.92	90.80	93.56	92.28	92.91
EC-Pfam-5z	31.67	33.80	32.70	36.25	23.57	28.56	47.22	40.95	43.86	45.66	45.36	45.51
EC-Pfam-15z	34.85	37.11	35.95	39.52	24.83	30.49	53.66	45.34	49.14	52.57	49.41	50.94
EC-Pfam-AA	35.39	37.58	36.45	39.73	25.61	31.14	60.18	52.14	55.87	58.83	56.46	57.62
EC-Pfam-LD	37.63	39.69	38.63	41.47	25.79	31.80	58.79	48.58	53.19	58.53	50.86	54.42
EC-Prints	84.71	89.96	87.25	89.64	75.49	81.96	92.18	87.24	89.64	90.88	90.60	90.74
EC-Prints-5z	34.33	36.95	35.59	37.53	31.19	34.07	52.48	46.39	49.24	50.87	50.19	50.53
EC-Prints-15z	36.83	39.67	38.19	39.92	33.31	36.31	58.69	50.42	54.23	57.57	53.81	55.62
EC-Prints-AA	41.05	43.93	42.44	44.11	36.22	39.78	66.54	59.08	62.59	65.43	62.59	63.98
EC-Prints-LD	42.93	45.33	44.10	44.77	37.61	40.87	65.12	55.32	59.81	64.63	57.03	60.58
EC-Prosite	87.32	92.31	89.74	96.51	32.20	48.28	95.10	89.58	92.26	93.31	92.56	92.93
EC-Prosite-5z	31.52	33.54	32.49	42.95	15.66	22.94	48.37	41.94	44.92	46.51	46.24	46.38
EC-Prosite-15z	34.45	36.47	35.43	44.88	16.27	23.88	55.77	47.31	51.19	55.03	52.00	53.47
EC-Prosite-AA	36.67	38.67	37.64	46.56	16.66	24.53	60.73	53.12	56.67	59.87	57.85	58.84
EC-Prosite-LD	36.66	38.57	37.59	45.19	15.56	23.15	59.07	48.88	53.49	58.86	51.45	54.90
GPCR-Interpro	75.19	76.62	75.90	89.86	37.66	53.07	87.40	71.17	78.45	84.12	74.70	79.13
GPCR-Interpro-5z	48.87	49.82	49.34	66.59	28.77	40.18	65.99	51.29	57.72	60.68	54.04	57.17
GPCR-Interpro-15z	56.76	57.94	57.35	70.74	31.05	43.15	74.20	58.77	65.58	70.46	61.08	65.43
GPCR-Interpro-AA	60.56	61.51	61.03	73.11	33.10	45.56	80.20	68.25	73.74	78.31	71.49	74.75
GPCR-Interpro-LD	60.99	61.39	61.19	67.09	31.45	42.82	80.27	67.03	73.04	79.42	67.65	73.05
GPCR-Pfam	60.12	61.94	61.01	90.45	38.83	54.33	77.32	57.58	66.00	70.40	60.19	64.89
GPCR-Pfam-5z	50.20	51.31	50.75	68.39	29.57	41.28	67.96	52.98	59.54	62.32	55.31	58.61
GPCR-Pfam-15z	58.43	59.75	59.08	72.61	32.66	45.05	75.57	59.75	66.72	71.79	62.07	66.57
GPCR-Pfam-AA	61.68	62.74	62.20	74.07	34.05	46.65	81.87	69.58	75.22	79.96	72.71	76.16
GPCR-Pfam-LD	61.81	62.47	62.14	68.30	34.01	45.41	80.87	67.36	73.49	79.73	67.84	73.29
GPCR-Prints	72.17	74.15	73.14	89.37	33.82	49.07	87.03	69.55	77.30	82.99	72.91	77.62
GPCR-Prints-5z	54.12	55.72	54.91	77.05	29.22	42.36	71.82	56.18	63.04	65.72	59.17	62.27
GPCR-Prints-15z	58.90	60.64	59.76	77.16	30.26	43.46	78.69	61.15	68.81	73.74	63.65	68.31
GPCR-Prints-AA	62.98	64.66	63.81	78.29	31.20	44.61	83.58	71.02	76.79	80.70	73.98	77.19
GPCR-Prints-LD	62.10	62.99	62.54	70.58	30.14	42.23	83.02	68.07	74.79	81.39	69.19	74.79
GPCR-Prosite	55.38	57.24	56.29	84.63	30.48	44.81	75.71	53.82	62.91	66.55	56.72	61.23
GPCR-Prosite-5z	54.35	55.77	55.05	74.88	28.90	41.69	71.25	55.74	62.54	65.53	59.04	62.12
GPCR-Prosite-15z	59.58	60.99	60.28	75.28	29.63	42.52	77.11	60.88	68.02	72.43	63.33	67.57
GPCR-Prosite-AA	62.33	63.68	63.00	76.00	30.43	43.44	81.80	70.22	75.57	79.58	73.41	76.36
GPCR-Prosite-LD	62.31	63.23	62.77	70.28	28.52	40.57	82.49	68.48	74.82	81.51	69.93	75.26

to decide at which level the classification should be stopped for each example and there is no need to consider the trade-off between predictive accuracy and usefulness. Moreover, even when compared to the LMNB which employs the usual top-down prediction approach, making classification all the way to the leaf nodes, both global approaches (GMNB and GMNBwU) have higher hierarchical f-measure and this gain in accuracy is statistically significant as it can been seen in Figure 4.3.

4.5.2 Evaluating the Impact of the Usefulness Measure in the Global-Model Hierarchical-Classification Naive Bayes

Let us now evaluate the impact of the optional usefulness criterion in the proposed global-model Naive Bayes, which considers the trade-off between accuracy and usefulness when deciding what should be the most specific class predicted for a given test example. Table 4.3 shows the hierarchical measures of precision, recall and f-measure of the global-model Naive Bayes without (GMNB) and with the usefulness criterion (GMNBwU).

The analysis of the results corroborate with our previous statements. That is, the GMNB has an overall higher hierarchical precision than the GMNBwU, while the GMNBwU has a higher overall hierarchical recall than the GMNB. This means that that by adding the usefulness to the global-model Naive Bayes, the classifier is really making deeper predictions at the cost of their precision. It should be noted however, that there is no statistically significant difference between the hF measure values of the two classifiers, as shown in Figure 4.3.

The decision of which version of the classifier to use will depend on the type of protein being studied and the costs associated with the biological (laboratory) experiments in order to verify if the predictions are correct.

4.5.3 Evaluating the Impact of the Usefulness Measure in the Local-Model Naive Bayes

Let us now evaluate the impact of the usefulness criterion in the local-model Naive Bayes approach. Contrary to the global-model approach, the usefulness criterion is used in conjunction with the top-down approach, in order to create a non-mandatory top-down approach (LMNBwU), as discussed in Section 4.4.1. However, by comparing this approach with the standard top-down approach (LMNB) it can be seen (by analysing the results of Table 4.3) that using this criterion produces a higher hierarchical precision, but a lower hierarchical recall. The reason for this might be that the current criterion leads to the blocking problem, as discussed in Section 2.4.4. Nevertheless the use of the LMNB is a better choice than the LMNBwU even making prediction up to leaf node, where clearly the classifier is making deeper predictions. Also, according to the statistical tests shown in Figure 4.3, the LMNB is significantly better than the LMNBwU.

4.6 Discussion and Related Work

To the best of our knowledge, this work is the first to modify the Naive Bayes algorithm directly to cope with hierarchical classification problems, since the proposed algorithm, can be applied in any application domain. Indirectly however, some authors have improved on the Naive Bayes classifier for hierarchical classification problems, in particular in the text categorization task.

One such approach was developed by (McCallum et al., 1998) for the text categorization problem. In text categorization, the task is (usually, although not limited to) to label documents according to their topics. In order to represent the document, as attributes, the words from the document collection are used, and as values of this attributes, there are different ways to represent them (as with any other classification problem). One possible approach is the absence/presence of a particular word. Considering that the number of features is huge in this type of problem, the attribute vectors end up being sparse (many attributes with the value zero). And with a huge amount of sparse data, the usual way of computing probabilities become less reliable. In order to overcome this limitation, they employed a statistical technique known as *shrinking* which aims at reducing the variance in the attribute values' frequencies that are used to compute probabilities in the Naive Bayes algorithm. Moreover they have applied this technique to a hierarchical classification setting. However their approach can only cope with mandatory-leaf node problems.

In (Xue et al., 2008; Gauch et al., 2009) the authors study different approaches for selecting training examples for Naive Bayes when mining large document hierarchies. In (Xue et al., 2008) the authors propose a two-stage algorithm consisting of a search stage and a classification stage. In the search stage, given a new document without its label, the algorithm finds a subset of categories from the original hierarchy, that are similar to the given document. Based on the similar categories, the algorithm produces a pruned hierarchy. A Naive Bayes classifier in then built on this pruned hierarchy and used to classify the document into the classification stage. In order to prune the hierarchy the authors consider three strategies: (1) using a flat hierarchy; (2) using a pruned hierarchy consisting of the similar categories and their parents and grandparents; (3) using the ancestor-assistant strategy, which consists of pruning the hierarchy, and then consolidating the parents and grandparents of the category creating a modified flat hierarchy space. The main limitation of this method is that it builds one particular pruned hierarchy (and hence train one classifier) for each test example (unlabeled document).

In (Gauch et al., 2009) the authors propose the use of document pooling strategies. That is instead of using all the documents available, they experiment with different centroid-based approaches. Centroid-based approaches are often employed to measure how similar a document is to each other. In order to decided which classifier to use, the authors tested the performance of several classifiers, one of them was Naive Bayes. However for performing their experiments they opted to use a Rocchio classifier.

In (Punera and Ghosh, 2008) the authors propose a post-processing strategy for local classifier per node approach approach based on a technique called Isothonic smoothing, that aims to give better probabilities estimates taking into account the hierarchy. In their research they have employed both a Naive Bayes classifier and a SVM classifier, although their approach works with any probabilistic classifier.

The flat Naive Bayes classifier has also been used as a base classifier in several works along with the Local Classifier Per Node (LCN) and Local Classifier Per Parent Node (LCPN) in the literature. It has been used as a base classifier in LCN (Mladenic and Grobelnik, 2003; Sun et al., 2003; Wu et al., 2005; Jin et al., 2008) and LCPN (Chakrabarti et al., 1998; Ceci and Malerba, 2007; Carvalho et al., 2011) approaches. It has been used by different authors, as one of the base classifiers, in the Selective Classifier approach (Holden and Freitas, 2008; Secker et al., 2007, 2010; Silla Jr. and Freitas, 2009b, 2011a).

Chapter 5

Global-Model Hierarchical-Classification k-Dependence Bayesian Classifier

In the previous chapter we have presented the Global-Model Hierarchical-Classification Naive Bayes classifier. Over the last decades of research there were many attempts to improve the flat Naive Bayes classifier. For a survey of the main approaches the reader is referred to (Jiang et al., 2007). In this chapter we are interested in investigating one of the main extensions to the flat Naive Bayes classifier which are the approaches that focus on structural extension, that is approaches that extend the structure of the Naive Bayes classifier to represent dependencies among attributes (Jiang et al., 2007).

One sound mathematical formalism to represent (in)dependencies between attributes (or domain variables) are Bayesian networks (Pearl, 1988). Bayesian networks are probabilistic graphical models that encode (in)dependencies between attributes in a Directed Acyclic Graph (DAG). A Bayesian network is a directed acyclic graph that has two components, the Bayesian network structure and the parameters of the network (also known as the conditional probabilities tables (CPTs)) (Sebastiani et al., 2010).

In the work of (Friedman et al., 1997) the authors investigated and proposed

Table 5.1: Symbols used in this chapter.

Symbol	Meaning
Tr	the set of all training examples
A	Set of attributes
A_i	i-th attribute
C	Set of classes
C_k	k-th class
$P(C\|e)$	Probability of the Class C given the evidence e.
X	A data instance containing a set of Attributes.
n_c	the number of classes
n_a	the number of attributes
n_e	the number of examples
n_{nl}	the number of non-leaf class in C
n_{cl}	the number of class levels
β	Bayesian Network
B_S	Bayesian Network Structure
B_P	Bayesian Network Conditional Probabilities or Parameters
π_i	The set of parents of A_i in B_S
θ_{ijk}	The conditional probability that a variable A_i in B_S has the value A_{ik}
A_{ik}	i-th attribute with k-th value
r_i	Number of attribute values for A_i
q_i	Number of configuration values for the parents of A_i

algorithms that use Bayesian networks for classification tasks. This type of classifiers are known as Bayesian Network Classifiers, and since (Friedman et al., 1997) have received considerable attention from the scientific community in order to develop better, faster, more accurate and comprehensible Bayesian network classifiers. All these approaches can be classified into two broad groups, hereafter refereed to a class-constrained and unrestricted Bayesian network classifiers.

In the class-constrained approach, the class node is treated as a special node in the Bayesian network classifier, as it is used as the parent attribute of all the other attributes. In the unrestricted approach (referred to as general Bayesian network classifier in (Cheng and Greiner, 1999, 2001)), the class node is treated as a regular node, and the Bayesian network classifier is inferred by using the Markov Blanket of the class node (defined in section 5.1.1).

In this Chapter we review the approaches for constructing class–constrained and unrestricted Bayesian network classifiers and present a novel type of class-constrained global-model hierarchical-classification Bayesian network classifier algorithm. This new global-model hierarchical-classification Bayesian network classifier algorithm is a modified version of one the class-constrained Bayesian network classifiers which was extended to deal with hierarchical classification problems.

5.1 (Flat) Bayesian Network Classifiers

As previously discussed there are two broad approaches for constructing Bayesian network classifiers, class-constrained and unrestricted. Regardless of the approach, since all Bayesian network classifiers are based on the Bayesian Network framework, we will now present the main works in each approach.

5.1.1 (Flat) Unrestricted Bayesian Network Classifiers

Unrestricted Bayesian network classifiers (Friedman et al., 1997) (also known as General Bayesian networks (Cheng and Greiner, 1999, 2001) and Selective Unrestricted Bayesian network (Pernkopf, 2005)), are built by using standard approaches for Bayesian network learning from data. A Bayesian network is a directed acyclic graph that has two components, the Bayesian network structure and the parameters of the network (also known as the conditional probabilities tables (CPTs)) (Sebastiani et al., 2010), hence a Bayesian Network $\beta = (B_S, B_P)$. B_S is a directed acyclic graph (DAG) where the nodes correspond to the attributes (or domain variables) $A_1, \ldots, A_{n_a}$ and the arcs represent direct dependencies between the variables.

Table 5.2: Dataset example from (Buntine, 1996).

case	A_1	A_2	A_3
1	T	F	T
2	T	T	T
3	F	T	T
4	F	T	T

Considering the dataset example presented in Table 5.2, where each attribute A_1 to A_3 is binary (that is, it can have only two attribute values), the question that arises is how to build a Bayesian Network from it. Initially Bayesian networks were hand-crafted and applied to different problems (Cooper and Herskovits, 1992), however the prohibitive costs of having expertise knowledge to build the networks as well as the need to develop Bayesian networks to analyse data where expertise is minimum lead to the development of automatically constructing Bayesian network classifiers from the data. The most used approach for learning a Bayesian network from data consists of the induction of its two different components (Sebastiani et al., 2010):

1. The graphical structure of conditional dependencies (model selection), that is determining B_S; This automatic identification process requires two components: a scoring metric to select the best model and a search strategy to explore the space of possible, alternative models.

2. The conditional distributions quantifying the dependency structure (parameter estimation), that is determining B_P;

Let us evaluate each of these points in more detail. In order to build a Bayesian Network model from the data, given the number of attributes n_a, the number of possible network models is $2^{n_a(n_a-1)/2}$ (Buntine, 1996) considering the number of different undirected arcs one can add to the model.

As the number of attributes grows it is unfeasible to generate all possible Bayesian network models, hence why heuristics are widely employed. One of the earlier algorithms to build a Bayesian network from data is known as the K2 algorithm (Cooper and Herskovits, 1992). The K2 algorithm assumes that the attributes are ordered in a way that each node can have as a parent node, only attributes which came before it in the ordering. It also limits the number of parents a given attribute node can have by using a user-defined constant k. The K2 algorithm is shown in Algorithm 2.

Algorithm 2: The K2 algorithm

```
   Input: A Dataset $D$ with an ordered set of Attributes, $A_1, \ldots, A_{n_a}$.
   Input: The maximum number of parents, given by $k$.
   Output: An unrestricted Bayesian network $\beta$.
 1 foreach node $A_i, 1 \leq i \leq n_a$ do
 2     $B_S \leftarrow A_i$;
 3     Compute $Score(B_S, D)$; // Evaluate the current model.
       // find $\pi_{A_i}$, as follows:
 4     $\pi_{A_i} \leftarrow \emptyset$ // The parent set of $A_i$ starts empty.
 5     $NotDone \leftarrow True$;
 6     while $NotDone = True \wedge |\pi_{A_i}| < k$ do
           // Find a link to add from $A_z$ to $A_i$:
 7         Find $A_z$ that maximizes $Score(B_S \bigcup (\pi_{A_i} \leftarrow A_z), D)$;
 8         if $Score(B_S \bigcup \pi_{A_i} \leftarrow A_z), D) > Score(B_S, D)$ then
               // Permanently add the link from $A_z$ to $A_i$ in $B_S$:
 9             $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow A_z)$;
10         else
11             $NotDone \leftarrow False$;
12         end
13     end
14 end
```

Line 7 in the K2 algorithm, makes the evaluation of different models by applying $Score(B_S \bigcup (\pi_{A_i} \leftarrow A_z), D)$. The question that arises is how to evaluate the different Bayesian network models. According to (Rubio and Gamez, 2011) model scoring functions can be divided in two main groups: filter and wrapper approaches.

In the filter approach a score metric that is independent from the classifier is considered as a quality measure of the Bayesian network, given the training data. Examples of score metrics are the K2 metric (Cooper and Herskovits, 1992), the Minimum description length (MDL) (Lam and Bacchus, 1994), the Aikake Information Criterion (AIC) (Akaike, 1974), the Bayesian information criteria (BIC) (Schwarz, 1978) and the BDe score(Heckerman et al., 1995).

In the wrapper approach, for each possible arc addition, a classifier is built based on the data and the current model. In this case the data is split into subTraining and subValidation sets. The subTraining set is used to train the classifier while the measure of the classifier performance (e.g. f-measure) on the

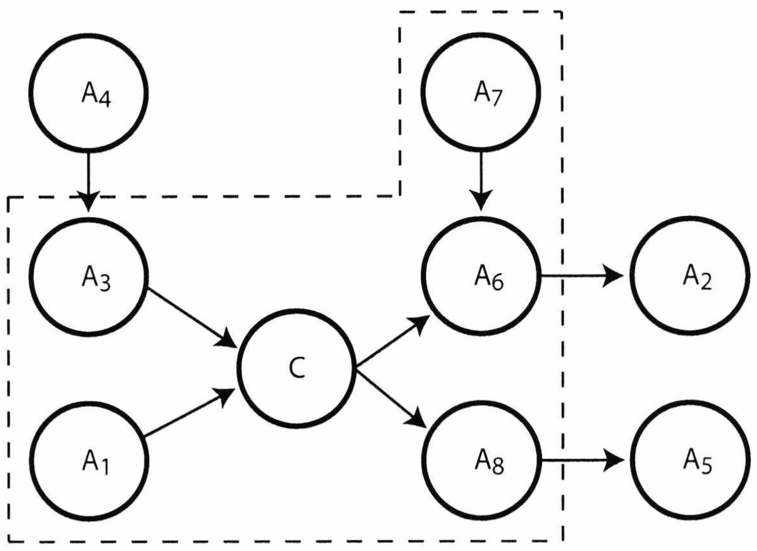


Figure 5.1: An example of an Unrestricted Bayesian network Classifier. The dashed lines represent the Markov Blanket of the class node C.

subValidation set is used to score the different models. By using an unrestricted Bayesian network for classification, the classifier is built by using the Markov Blanket of the class node. The Markov Blanket of A_i is given by the parents of A_i, the children of A_i and the parents of the children of A_i in B_S (Sebastiani et al., 2010). An example of a classification problem with 8 Attributes is shown in Figure 5.1, the dashes lines represent the Markov Blanket of the class node C.

As mentioned earlier a Bayesian network has two components, the B_S and the B_P. So far, we have only discussed how to search for the B_S. Now let's look at how to estimate the B_P given the data. Let θ_{ijk} denote the conditional probability that a variable A_i in B_S has the value a_{ik}, for some k from 1 to r_i, where r_i is the number of possible values of A_i. The number of configurations for the parents of A_i will be denoted by q_i. The configuration number j for π_i (set of parents of A_i) will be denoted by w_{ij}. Then $\theta_{ijk} = P(A_i = k|\pi_i = w_{ij})$ is termed a network conditional probability (Madden, 2009). The simplest form of parameter estimation is based on frequency counts as shown in Equation 7.

$$E(\theta_{ijk}|D, B_S) = \frac{N_{ijk}}{N_{ij}} \tag{7}$$

where:

$$N_{ij} = \sum_{k=1}^{r_i} N_{ijk} \tag{8}$$

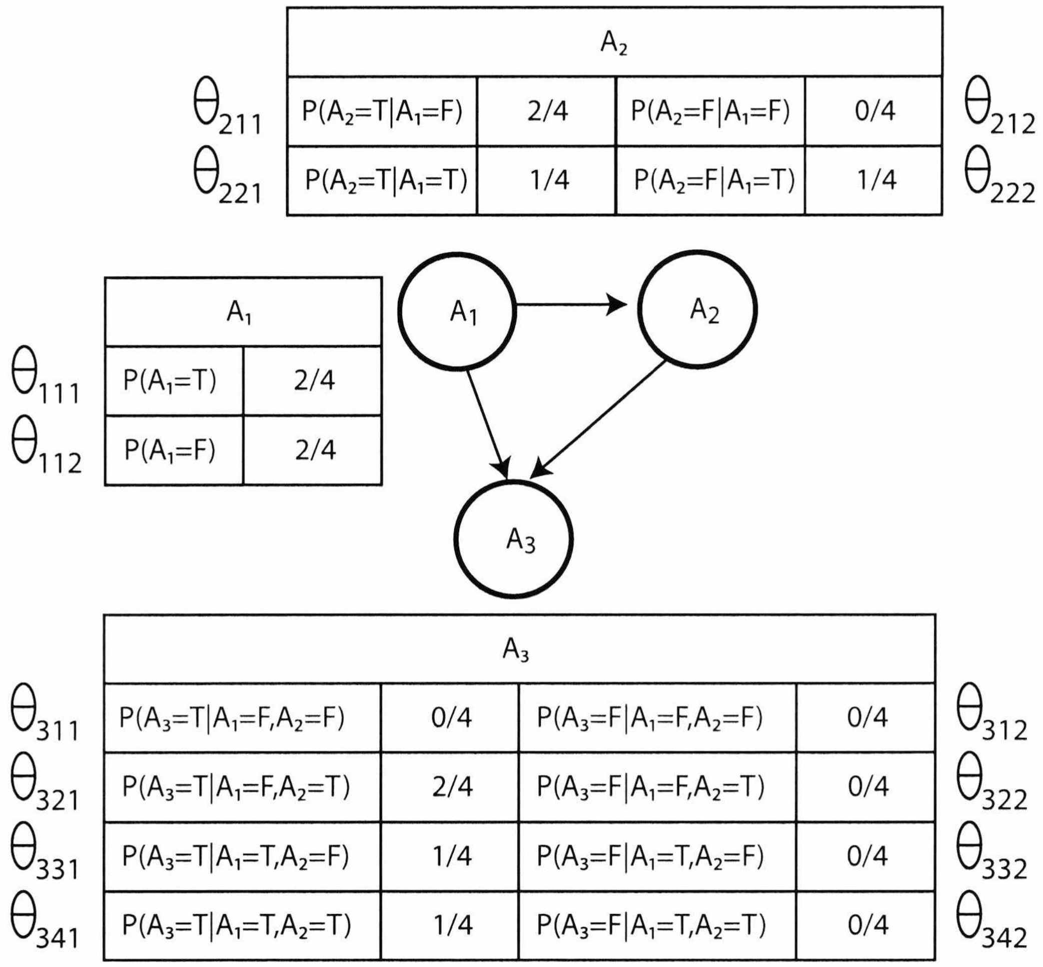
where N_{ijk} is the the number of examples having the k-th value of the i-th attribute and the j-th value of the configuration of the parents of A_i.

Basically, Equation 7 counts the number of occurrences of $A_i = a_{ik}$ given that its parents (if any) have the values of the $j-th$ configuration. Let us illustrate this problem with a simple example. Considering the Table 5.2 we have constructed the Bayesian network shown in Figure 5.2. Besides the network structure (B_S), this example also has the conditional probabilities tables (CPTs), which are the B_P.

The example in Figure 5.2 allow us to further clarify the notation used so far. Let us assume that we are interested in computing the probability of θ_{311}. In this case $q_i = 4$ as there are four possible ($A_1 = F$,$A_2 = F$; $A_1 = F$,$A_2 = T$; $A_1 = T$,$A_2 = F$; $A_1 = T$,$A_2 = T$) configurations for π_i. Since $j = 1$ in our example, this means we will be looking at the 1st configuration, which in this case is $A_1 = F$,$A_2 = F$. $N_{311} = 0/4$ and $N_{31} = \sum_1^{r_i} N_{31k} = N_{311} + N_{312} = 0/4 + 0/4$. Although this example makes clear the use of the notation for counting, it also points to a problem in the construction of the CPTs when certain combination of attribute values are not present in the examples in the dataset. In such cases we will use the value of $1/n_e$ to avoid multiplications by zero.

5.1.2 (Flat) Class–Constrained Bayesian Network Classifiers

In the previous section we have presented the unrestricted Bayesian network classifier. The main difference between a unrestricted Bayesian network classifier



	A₂				
θ_{211}	$P(A_2=T\|A_1=F)$	2/4	$P(A_2=F\|A_1=F)$	0/4	θ_{212}
θ_{221}	$P(A_2=T\|A_1=T)$	1/4	$P(A_2=F\|A_1=T)$	1/4	θ_{222}

	A₁	
θ_{111}	$P(A_1=T)$	2/4
θ_{112}	$P(A_1=F)$	2/4

$$\begin{tikzcd} A_1 \arrow[r] \arrow[dr] & A_2 \arrow[d] \\ & A_3 \end{tikzcd}$$

	A₃				
θ_{311}	$P(A_3=T\|A_1=F,A_2=F)$	0/4	$P(A_3=F\|A_1=F,A_2=F)$	0/4	θ_{312}
θ_{321}	$P(A_3=T\|A_1=F,A_2=T)$	2/4	$P(A_3=F\|A_1=F,A_2=T)$	0/4	θ_{322}
θ_{331}	$P(A_3=T\|A_1=T,A_2=F)$	1/4	$P(A_3=F\|A_1=T,A_2=F)$	0/4	θ_{332}
θ_{341}	$P(A_3=T\|A_1=T,A_2=T)$	1/4	$P(A_3=F\|A_1=T,A_2=T)$	0/4	θ_{342}

Figure 5.2: An example of Bayesian network and the respective counts.

and a class-constrained Bayesian network classifier is that in the class-constrained Bayesian network classifier, the class node is treated as special node and it is the parent of all the attributes. The Naive Bayesian classifier presented in Chapter 4 can be seen as a special case of a Bayesian network classifier, as shown in Figure 5.3. Observe that in Figure 5.3 all attributes have the class node as a parent. Moreover, since the Naive Bayes algorithm assumes independence between the attributes given the class, there are no arcs between the attributes showing graphically that all attributes are independent of each other.

Applying the formalism of Bayesian networks, Friedman et al. (Friedman et al.,

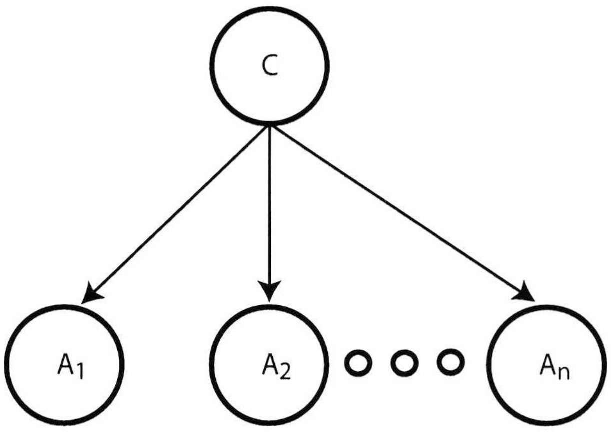


Figure 5.3: The Naive Bayes classifier as a special case of a general Bayesian network classifier.

1997) proposed novel approaches for building class-constrained Bayesian network classifiers. One of these approaches is the Tree Augmented Naive Bayes (TAN). In a TAN the attributes form a tree. Figure 5.4 shows an example of a TAN classifier built for a problem with 5 attributes. Note that except for the randomly selected attribute A_4, all the other attributes have exactly one other attribute as a parent besides the class node. As pointed out in (Keogh and Pazzani, 1999), one disadvantage of the TAN classifier is that it forces attribute dependencies even when they do not necessarily exist, as there are always $n_a - 1$ arcs added.

The algorithm for creating a TAN classifier is as follows:

1. Calculate $I(A_i, A_j|C) = \sum_{i=1}^{n_a} \sum_{j=1}^{n_a} \sum_{k=1}^{n_c} p(A_i, A_j, C_k) log \frac{p(A_i,A_j|C_k)}{p(A_i|C_k)p(A_j|C_k)}$ with $i < j, j = 2, \ldots, n_a$, where $I(A_i, A_j|C)$ is the mutual information between attributes A_i and A_j given the class attribute C.

2. Build an undirected complete graph, where the nodes correspond to the predictor variables: $A_1, \ldots, A_{n_a}$. Assign to the edge connecting variables A_i and A_j the weight $I(A_i, A_j|C)$.

3. Assign the largest two branches to the tree to be constructed.

4. Examine the next largest branch and add it to the tree unless it forms a

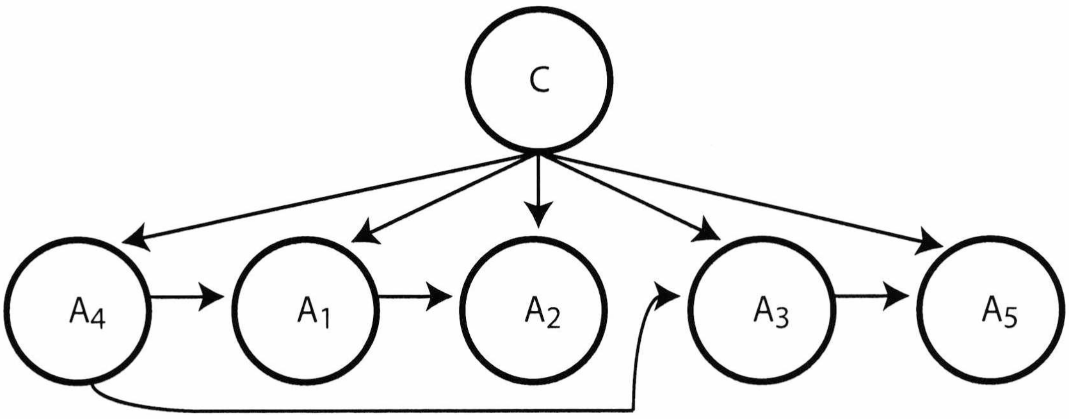


Figure 5.4: An example of a TAN classifier in a problem with 5 attributes.

loop. In the latter case discard it and examine the next largest branch.

5. Repeat Step 4 until $n_a - 1$ branches have been added.

6. Transform the undirected graph in a directed one, by choosing a random variable as the root.

7. Build the TAN structure adding a node labeled as C, and later add one arc from C to each of the predictor variables $A_i (i = 1, \ldots, n_a)$.

Inspired by the good performance of the TAN classifier, alternative approaches to the construction of TAN classifiers were proposed in (Keogh and Pazzani, 1999, 2002). The first approach, hereafter called HCS-TAN (Hill Climbing Search TAN) works by using a greedy hill climbing search algorithm to construct the network, as shown in Algorithm 3.

The HCS-TAN has the main disadvantage of being computationally more expensive than the original TAN due to the hill climbing search strategy and the constant re-evaluation of the adding arcs operations. However, it overcomes the limitation of TAN of introducing dependencies between all attributes, creating dependencies (i.e. adding arcs) only between attributes which improve the classifier. Moreover, the HCS-TAN achieved better classification results than TAN and Naive Bayes in (Keogh and Pazzani, 1999, 2002).

Algorithm 3: The HCS-TAN algorithm (Keogh and Pazzani, 1999, 2002).

```
Input: A Dataset $D$ with an unordered set of Attributes, $A_1, \dots, A_{n_a}$ and
       the class attribute $C$.
Output: A HCS-TAN classifier.
// Initialize the network to Naive Bayes:
1  $B_S \leftarrow C$;
2  foreach node $A_i, 1 \le i \le n_a$ do
       // Add the class node as the parent of node $A_i$:
3      $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow C)$;
4  end
5  $NotDone \leftarrow True$;
6  while $NotDone$ do
7      Compute $Score(B_S, D)$; // Evaluate the current model.
8      Find $(\pi_{A_j} \leftarrow A_i)$ that maximizes
       $Score(B_S \bigcup (\pi_{A_j} \leftarrow A_i), D), |\pi(A_j) \backslash C| < 1$;
9      if $Score(B_S \bigcup (\pi_{A_j} \leftarrow A_i), D) > Score(B_S, D)$ then
           // Permanently add the arc from $A_i$ to $A_j$ in $B_S$:
10         $B_S \leftarrow B_S \bigcup (\pi_{A_j} \leftarrow A_i)$;
11     else
12         $NotDone \leftarrow False$;
13     end
14 end
```

In the hope of improving the classification accuracy further, in (Keogh and Pazzani, 1999, 2002) the authors propose a heuristic search method for creating TAN classifiers, known as Super Parent TAN (SP-TAN). The SP-TAN algorithm is shown in Algorithm 4.

In the experiments of (Keogh and Pazzani, 1999, 2002) SP-TAN has the same accuracy as HCS-TAN, however its search procedure is more efficient. SP-TAN has a complexity of $O(n^2)$ against a complexity of $O(n^3)$ for HCS-TAN. An improvement over the SP-TAN was developed in (Zhang and Ling, 2001) where the authors proposed a method known as Stump Network classifier, hereafter called SN-TAN, which is a constant faster than SP-TAN and has similar predictive accuracy. The Stump network algorithm works as follows:

1. Initialize β to Naive Bayes and calculate its predictive accuracy;

2. Let node set N be all attributes (except C) and $TreeStump$ queue be empty;

Algorithm 4: The SP-TAN algorithm (Keogh and Pazzani, 1999, 2002).

```
Input: A Dataset $D$ with an unordered set of Attributes, $A_1, \ldots, A_{n_a}$ and
       the class attribute $C$.
Output: A SP-TAN classifier.
// Initialize the network to Naive Bayes:
1  $B_S \leftarrow C$;
2  foreach node $A_i, 1 \leq i \leq n_a$ do
       // Add the class node as the parent of node $A_i$:
3      $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow C)$;
4  end
5  Compute $Score(B_S, D)$; // Evaluate the current model.
   // Initialize the list of orphans $O$ to the full set of nodes:
6  $O \leftarrow A_1, \ldots, A_{n_a}$;
7  $NotDone \leftarrow True$;
8  while $NotDone$ do
       // Consider making each node a SuperParent.
       // Let $A_{sp}$ be the SuperParent which increases the accuracy
          the most:
9      Find $A_{sp}$ that maximizes $Score(B_S \bigcup \forall A_j, A_j \leftarrow A_{sp}, D), |\pi(A_j) \backslash C| < 1$;
       // Consider adding an Arc from $A_{sp}$ to each orphan in $O_i$
10     Find $O_i$ that maximizes $Score(B_S \bigcup \pi_{O_i} \leftarrow A_{sp}, D)$;
11     if $Score(B_S \bigcup (\pi_{O_i} \leftarrow A_{sp}), D) > Score(B_S, D)$ then
           // Permanently add the arc from $A_{sp}$ to $O_i$ in $B_S$:
12         $B_S \leftarrow B_S \bigcup (\pi_{O_i} \leftarrow A_{sp})$;
           // Remove $O_i$ from the set of Orphan Nodes:
13         $O \leftarrow O \backslash O_i$;
14     else
15         $NotDone \leftarrow False$;
16     end
17 end
```

3. Make a tree stump for each node in N. Let T_S be the tree stump with the highest improvement on the predictive accuracy.

4. For each arc on T_S, if the predictive accuracy does not decrease after deleting the arc, remove it from T_S.

5. Put T_S in *TreeStump* queue.

6. Remove all nodes of T_S from N. If N is not empty, go to 4.

7. Go over *TreeStump* queue once, for each tree stump in the queue, add a

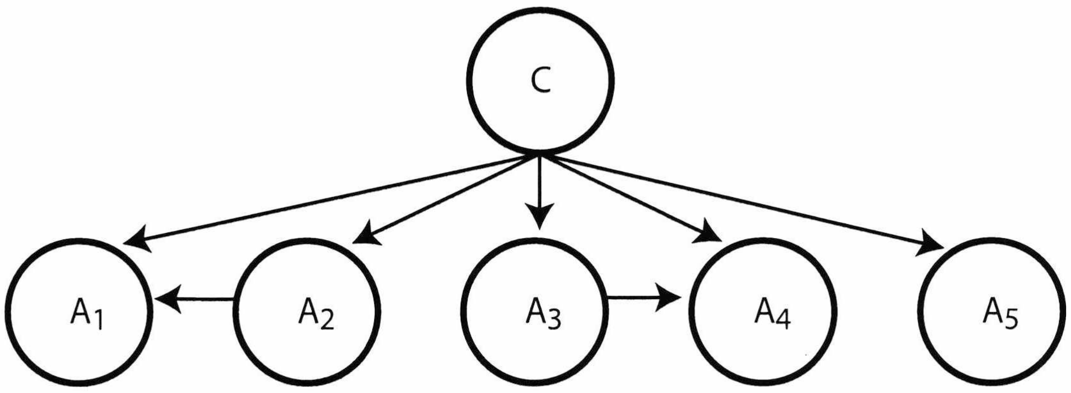


Figure 5.5: An example of a Super Parent TAN (SP-TAN) classifier in a problem with 5 attributes.

link from a leaf of the previous tree stumps in the queue to the root of this tree stump, if the predictive accuracy increases.

Although HCS-TAN, SP-TAN and SN-TAN all improve the accuracy of the built classifier when compared to the original TAN algorithm proposed by Friedman et al. (Friedman et al., 1997), this gain in accuracy comes with the drawback of performing costly structure searches. Moreover in (Keogh and Pazzani, 1999, 2002) the authors investigate if the gain in accuracy gained by SP-TAN when compared to TAN is because of adding less arcs. Their experiments shown that the addition of less arcs is not the main reason for the better performance of SP-TAN, the reason for improvement lies in the way the classifier is built (that is the search strategy for selecting and adding arcs). It should be noted that in all algorithms discussed so far (TAN, HCS-TAN, SP-TAN, SN-TAN) the maximum number of parents an attribute can have is 1 (besides the class node which is the parent of all attributes, as discussed earlier).

Inspired by the SP-TAN algorithm, in (Webb et al., 2005) the authors propose the algorithm known as Averaged One Dependence Estimators (AODE). In AODE instead of performing model search which is the main computational drawback of HCS-TAN, SP-TAN and SN-TAN the algorithm creates a set of n_a models, setting in each model the A_i attribute as the "superparent" to all other attributes. Figure

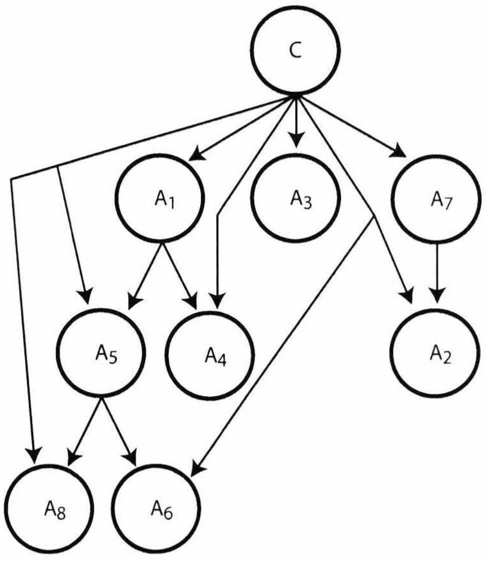


Figure 5.6: An example of a Stump Network (SN-TAN) classifier in a problem with 8 attributes.

5.7 illustrates this approach. Unlike the previous algorithms, since AODE needs to ensemble the decision of the different models, instead of just applying the MAP rule to classify a new example, the AODE algorithm uses Equation 9.

$$argmax_C \left(\sum_{i:1\leq i\leq n_a \wedge F(A_i)\geq m} \hat{P}(C, A_i) \prod_{j=1}^{n_a} \hat{P}(A_j|C, A_i) \right) \tag{9}$$

If $\neg\exists i : 1 \leq i \leq n_a \wedge F(A_i) \geq m$, AODE defaults to NB.

where $F(A_i)$ is a count of the number of training examples having attribute-value A_i and is used to enforce the limit m that is placed to the support needed in order to accept a conditional probability estimate. Originally the value of m was set to 30 ($m = 30$) but it has been later discovered that it could be safely simplified (Cerquides and de Mantaras, 2005), and the value of m set to 1 ($m = 1$) has been used in subsequent research (Yang et al., 2005; Zheng and Webb, 2007). Further research on AODE has been carried out to improve on the accuracy of AODE by performing model selection and model weighting instead of using the m parameter (Yang et al., 2007). This approach decreases the computational

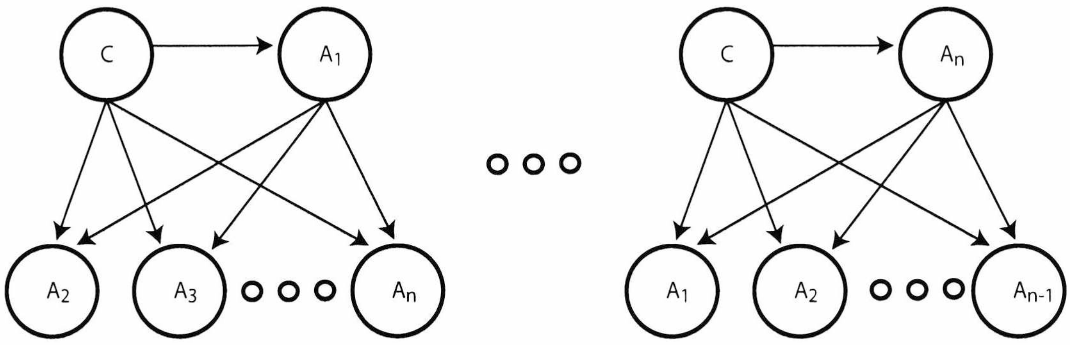


Figure 5.7: The Averaged One Dependence Estimators (AODE) classifier.

efficiency of the original AODE classifier, but brings gains in predictive accuracy.

In the work of (Sahami, 1996), the author proposed the k-Dependence Bayesian Network Classifier (k-DBC) where k denotes the number of attributes a node can have as parents (besides the class node). Let us now review the literature on Bayesian network classifiers based on this concept of k dependencies.

The k-Dependence Bayesian Classifier (k-DBC) was originally proposed in (Sahami, 1996) and works as follows:

- Calculate $I(A_i, C)$ and $I(A_i, A_j|C)$ for each pair of variables.
- At every iteration, add the A_{max} variable not included in the model with the highest $I(A_i, C)$
- Set C and the k variables with the highest $I(A_j, A_{max}|C)$ as the parents of A_{max}.

The k-DBC algorithm main limitations, as pointed out in (Rubio and Gamez, 2011), are:

- The algorithm is guided by a greedy ordering obtained by using knowledge of marginal probabilities between the attributes and the class only. This solution is likely to be a suboptimal one given that interactions between predictive attributes are not considered.

- Since k is the same for all nodes, it is unpractical to set it to values higher than 5 or 6 in almost every scenario. This results in overly complex networks, as all nodes are given k parents. This unnecessary complexity harms classifier performance, as it leads to overfitting. Furthermore, it is not possible to model cases where some nodes have a large number of dependencies, whereas others just have a few.

Although k-DBCs can be built by using the original algorithm, in (Castillo and Gama, 2009) the authors have used a hill climbing search procedure to build a k-DBC classifier, hereafter referred to as HCS-k-DBC. The HCS-k-DBC algorithm proposed in (Castillo and Gama, 2009) is shown in Algorithm 5.

Note that in Line 8 of the HCS-k-DBC algorithm, the evaluation function takes on the form of $Score(B_S, D)$. This is the same scoring function discussed in Section 5.1.1, and can be any of the filter or wrapper approaches discussed so far.

It should be noted that the HCS-k-DBC algorithm overcomes the main limitations of the original k-DBC algorithm, at the cost of computational efficiency. Since the addition of only one arc is considered at each time (Line 8 of the algorithm), it is possible to have attributes that have at most k parents instead of always having k parents as in the original k-DBC algorithm. Also, it is independent of attribute ordering.

Another type of class-constrained Bayesian network algorithm that can be found in the literature, is the Bayesian Network Augmented Naive Bayes (BAN) (Cheng and Greiner, 1999, 2001) which consists of using any unrestricted Bayesian network learning algorithm but with the class node as the parent attribute of every other node. In essence, although built by a different procedure any BAN is a k-DBC where $k = n_a$.

Algorithm 5: The HCS-k-DBC algorithm (Castillo and Gama, 2009).

```
Input: A Dataset $D$ with an unordered set of Attributes, $A_1, \ldots, A_{n_a}$ and
       the class attribute $C$.
Input: The value of $k$ for the maximum number of $\pi_{A_i}$.
Output: A HCS-$k$-DBC classifier.
// Initialize the network to Naive Bayes:
1  $B_S \leftarrow C$;
2  foreach node $A_i, 1 \leq i \leq n_a$ do
       // Add the class node as the parent of node $A_i$:
3      $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow C)$;
4  end
5  $NotDone \leftarrow True$;
6  while $NotDone$ do
7      Compute $Score(B_S, D)$; // Evaluate the current model.
8      Find $(\pi_{A_i} \leftarrow A_j)$ that maximizes
       $Score(B_S \bigcup (\pi_{A_i} \leftarrow A_j), D), |\pi(A_i) \backslash C| < k$;
9      if $Score(B_S \bigcup (\pi_{A_i} \leftarrow A_j), D) > Score(B_S, D)$ then
           // Permanently add the arc from $A_j$ to $A_i$ in $B_S$:
10         $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow A_j)$;
11     else
12         $NotDone \leftarrow False$;
13     end
14 end
```

5.1.3 Bayesian Multinets

In the work of (Friedman et al., 1997), besides proposing the original TAN algorithm, it was also proposed what is known as Bayesian Multinets. In a Bayesian Multinet, a Bayesian network classifier is created for each class. The procedure for creating TAN multinets (Friedman et al., 1997) is as follows:

1. Split D into n_c partitions $D_1, \ldots, D_{n_c}$, such that D_i contains all the instances in D where $C = c_i$.

2. Apply the procedure for TAN on D_i to construct B_i.

It should be noted, that although the original algorithm, uses the procedure for creating a TAN classifier, in theory any algorithm discussed so far (TAN, HCS-TAN, SP-TAN, SN-TAN, AODE, k-DBC, HCS-k-DBC) could be used to create the Bayesian Multinet. Experiments comparing TAN and Bayesian Multinets were performed in (Friedman et al., 1997) but none of the approaches was found to be superior, and the Bayesian Multinet has a higher computational cost since it always builds one Bayesian network for each class. Figure 5.8 illustrates a Bayesian Multinet with 4 attributes, note that each class has its own network structure.

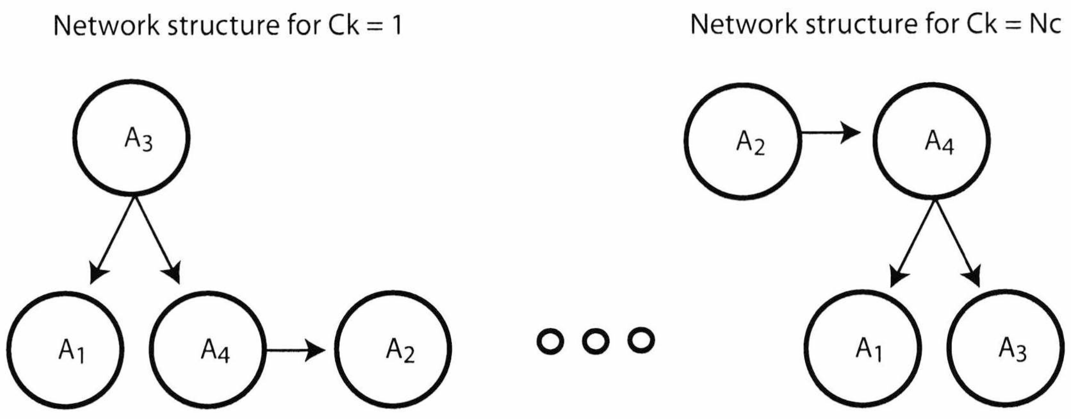


Figure 5.8: An example of a Bayesian Multinet classifier in a problem with 4 attributes.

5.2 Global-Model Hierarchical-Classification k-Dependence Bayesian Classifier

As seen in the previous section, regardless of the approach (class-constrained or unrestricted), all Bayesian network classifiers have two main components. The

Bayesian network structure (B_S) and the Bayesian network parameters (B_P). In this section we will discuss how we have adapted the learning of these two components for hierarchical classification and justify the design choices we made since there are several approaches to building Bayesian network classifiers.

The first design choice concerning the Bayesian network structure B_S is whether to use a class-constrained or an unrestricted approach. The earlier work in the field done by Friedman et al. (Friedman et al., 1997), compared the performance of TAN and an unrestricted Bayesian network classifier using as $Score(S, D)$ the MDL metric. The result of this comparison was that TAN was found to be superior to the unrestricted Bayesian network classifier. Friedman et al. (Friedman et al., 1997) explanation for this fact was that the unrestricted Bayesian network classifier aimed at creating a good Bayesian network structure as a whole, while the class-constrained TAN aimed at maximizing classification accuracy. This has motivated us to pursue the path of class-constrained approaches. It is noteworthy that in a paper published in 2009 (Madden, 2009), the author verified that the earlier experiments performed in (Friedman et al., 1997) were unfair on the unrestricted Bayesian network classifier approach. The reason being that in (Friedman et al., 1997), the authors did not apply any *smoothing* operator to avoid multiplications by zero. Furthermore, the results of (Madden, 2009) show that when *smoothing* is applied to the unrestricted Bayesian network classifier using the MDL score, none of the approaches dominate. However, in a recent study (Santos et al., 2011), employing different approaches to build unrestricted Bayesian network classifier, the TAN algorithm is ranked number one when compared against five others (four being unrestricted Bayesian network classifiers).

After deciding that the algorithm would use a class-constrained approach to the B_S, the next step was deciding which type of algorithm to use. In this case we have chosen the k-Dependence Bayesian Network classifier (k-DBC) because of the flexibility given by how choosing different values for k determines the levels of attribute (in)dependence. As seen in the previous section, there are two approaches to build the B_S (not considering the use of BANs), the original k-DBC algorithm and the HCS-k-DBC algorithm. As discussed earlier (and pointed out in (Rubio and Gamez, 2011)) the original algorithm has some limitations that are overcome by the HCS-k-DBC. For this reason, we have chosen to modify the HCS-k-DBC to deal with hierarchical classification problems. Moreover, the original k-DBC algorithm uses class conditional mutual information tests $I(A_i, A_j|C)$ which are undefined for hierarchical classification problems.

The hill climbing search (HCS) which has been employed in the HCS-TAN and HCS-k-DBC is a greedy search procedure. The HCS works as follows. Given a current search state with network structure B_S it searches for all the possible arc inclusions considering adding only one arc at a time and respecting the number of k parents constrain. However, in some of our earlier experiments, the use of the HCS had a very high computational cost. More precisely it would take four days on an iCore7 processor to run 1 (out of 10) cross-validation folds for our smallest dataset (GPCR-Prints-5z). Interestingly enough, it seems that this study is the first one, in the field of Bayesian network classification research, to use a large amount of classes (due to the hierarchy) and a large amount of training data (examples) as well. In order to illustrate this point, Table 5.3 shows the existing datasets used in Bayesian network classifier research with their number of attributes (n_a), number of examples (n_e), number of classes (n_c) and the list of works where each dataset is used.

Table 5.3: Characteristics of the datasets used for flat Bayesian network classification research.

Dataset	n_a	n_e	n_c	Used in
Abalone	9	4,177	3	(Yang et al., 2007; Zheng and Webb, 2007)
Adult	13	48,842	2	(Cheng and Greiner, 1999, 2001; Webb et al., 2005; Yang et al., 2005; Zheng and Webb, 2007; Madden, 2009)
Annealing	39	898	6	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Rubio and Gamez, 2011)
Audiology	70	226	24	(Yang et al., 2007; Zheng and Webb, 2007)
Automobile	26	205	7	(Yang et al., 2007; Zheng and Webb, 2007)
Australian	14	690	2	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Pernkopf, 2005; Madden, 2009; Pernkopf and Bilmes, 2010)
Breast	10	683	2	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010)
Bank	20	1,162	2	(Zhang and Ling, 2001)
Balance Scale	5	625	3	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011; Rubio and Gamez, 2011)

Table 5.3 – continued from previous page

Dataset	n_a	n_e	n_c	Used in
Car	6	1,728	4	(Cheng and Greiner, 1999; Zheng and Webb, 2007; Madden, 2009)
Chess	36	3,196	2	(Sahami, 1996; Friedman et al., 1997; Cheng and Greiner, 1999, 2001; Madden, 2009; Pernkopf and Bilmes, 2010)
Chess	40	551	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Cleve	13	296	2	(Friedman et al., 1997; Madden, 2009; Pernkopf and Bilmes, 2010)
Corral	6	128	2	(Sahami, 1996; Friedman et al., 1997; Pernkopf and Bilmes, 2010)
CRX	15	653	2	(Friedman et al., 1997; Pernkopf and Bilmes, 2010)
Contact Lenses	5	24	3	(Zheng and Webb, 2007)
Credit Approval	15	690	2	(Yang et al., 2005, 2007; Zheng and Webb, 2007)
Diabetes	8	768	2	(Friedman et al., 1997; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010)
DNA	180	3,186	3	(Sahami, 1996; Madden, 2009)
DNA	60	3,186	3	(Cheng and Greiner, 1999, 2001)
Dmplexer	15	1,000	2	(Zheng and Webb, 2007)
Echocardiogram	7	131	2	(Webb et al., 2005; Yang et al., 2007; Zheng and Webb, 2007)
E.coli	7	336	8	(Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Flores et al., 2010, 2011)
Exclusive-or	10	500	2	(Keogh and Pazzani, 1999, 2002)
Flare	10	1,066	2	(Friedman et al., 1997; Cheng and Greiner, 1999; Pernkopf, 2005; Madden, 2009; Pernkopf and Bilmes, 2010)
German	20	1,000	2	(Friedman et al., 1997; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Pernkopf and Bilmes, 2010)
Glass	9	214	7	(Friedman et al., 1997; Pernkopf, 2005; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010; Rubio and Gamez, 2011)
Glass2	9	163	2	(Friedman et al., 1997; Pernkopf, 2005; Madden, 2009; Pernkopf and Bilmes, 2010)
Hayes-roth	4	160	4	(Flores et al., 2010, 2011)
Heart	13	270	2	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Pernkopf, 2005; Webb et al., 2005; Yang et al., 2005; Zheng and Webb, 2007; Madden, 2009; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010)
Heart Disease (cleveland)	14	303	2	(Yang et al., 2007; Zheng and Webb, 2007)
Hepatitis	19	80	2	(Friedman et al., 1997; Madden, 2009; Pernkopf and Bilmes, 2010)
Hepatitis	20	155	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Horse Colic	23	368	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
House-Votes-84	17	435	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Hungarian	14	294	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Hypothyroid	30	3,772	4	(Yang et al., 2005, 2007; Zheng and Webb, 2007)
Iris	4	150	3	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Pernkopf and Bilmes, 2010)
Ionosphere	35	351	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011; Rubio and Gamez, 2011)
KRvsKP	37	3,196	2	(Yang et al., 2005, 2007; Zheng and Webb, 2007)
Kdd-JapanV	14	29,961	9	(Flores et al., 2010, 2011)
LED7	7	3,200	10	(Sahami, 1996; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Letter	16	20,000	26	(Friedman et al., 1997; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010)
Labor negotiations	17	57	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Liver Disorders	7	345	2	(Yang et al., 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011)
Lung Cancer	56	32	3	(Keogh and Pazzani, 1999, 2002; Webb et al., 2005; Yang et al., 2007; Zheng and Webb, 2007)
Lymphography	18	148	4	(Friedman et al., 1997; Yang et al., 2007; Zheng and Webb, 2007; Madden, 2009; Pernkopf and Bilmes, 2010)
Mushroom	22	8,124	2	(Cheng and Greiner, 1999, 2001; Yang et al., 2007; Zheng and Webb, 2007)
Mofn-3-7-10	10	1,324	2	(Friedman et al., 1997; Madden, 2009; Pernkopf and Bilmes, 2010)

Table 5.3 – continued from previous page

Dataset	n_a	n_e	n_c	Used in
Mfeat	6	2,000	10	(Rubio and Gamez, 2011)
Mfeat-factors	216	2,000	10	(Flores et al., 2010, 2011)
Mfeat-fourrier	76	2,000	10	(Flores et al., 2010, 2011)
Mfeat-karh	64	2,000	10	(Flores et al., 2010, 2011)
Mfeat-morph	7	2,000	10	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Mfeat-zenirke	47	2,000	10	(Flores et al., 2010, 2011)
Mnist	196	70,000	10	(Pernkopf and Bilmes, 2010)
Nettalk(Phoneme)	8	5,438	50	(Yang et al., 2007; Zheng and Webb, 2007)
New-Thyroid	6	215	3	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Nursery	8	12,960	5	(Cheng and Greiner, 1999, 2001; Madden, 2009)
Optical Digits	49	5,620	10	(Yang et al., 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011)
Page Blocks	11	5,473	5	(Yang et al., 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011)
Pima	8	768	2	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Pernkopf, 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Pernkopf and Bilmes, 2010)
Post-op	9	90	3	(Keogh and Pazzani, 1999, 2002; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Pen Digits	17	10,992	10	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011; Rubio and Gamez, 2011)
Primary Tumor	18	339	22	(Yang et al., 2007, 2005; Zheng and Webb, 2007)
Promoter Gene Sequences	58	106	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Satimage	36	6,435	6	(Friedman et al., 1997; Webb et al., 2005; Yang et al., 2007; Zheng and Webb, 2007; Pernkopf and Bilmes, 2010)
Segment	19	2,310	7	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Webb et al., 2005; Yang et al., 2007; Zheng and Webb, 2007; Madden, 2009; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010; Rubio and Gamez, 2011)
Shuttle-small	9	5,800	7	(Friedman et al., 1997; Pernkopf and Bilmes, 2010)
Sick-euthyroid	30	3,772	2	(Yang et al., 2007)(Zheng and Webb, 2007)
Sign	9	12,546	3	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Sonar	61	208	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011; Rubio and Gamez, 2011)
Soybean-large	35	562	19	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Madden, 2009; Pernkopf and Bilmes, 2010)
Spambase	57	946	2	(Flores et al., 2010, 2011)
Splice-junction Gene Sequences	62	3,190	3	(Yang et al., 2007; Zheng and Webb, 2007)
Syncon	61	600	6	(Webb et al., 2005; Zheng and Webb, 2007)
SurfInsp	40	450	3	(Pernkopf and Bilmes, 2010)
Text	3,440	1,538	3	(Sahami, 1996)
Tic-Tac-Toe	10	958	2	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Rubio and Gamez, 2011)
Usps	256	11,000	10	(Pernkopf and Bilmes, 2010)
Vehicle	18	846	4	(Friedman et al., 1997; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Pernkopf, 2005; Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007; Madden, 2009; Flores et al., 2010, 2011; Pernkopf and Bilmes, 2010; Rubio and Gamez, 2011)
Vote	16	435	2	(Sahami, 1996; Friedman et al., 1997; Cheng and Greiner, 1999; Keogh and Pazzani, 1999, 2002; Zhang and Ling, 2001; Pernkopf, 2005; Madden, 2009; Pernkopf and Bilmes, 2010)
Volcanoes	4	1,520	4	(Zheng and Webb, 2007)
Vowel	14	990	11	(Yang et al., 2007, 2005; Zheng and Webb, 2007)
Waveform-21	21	5,000	3	(Friedman et al., 1997; Madden, 2009; Pernkopf and Bilmes, 2010)
Waveform-5000	41	5,000	3	(Yang et al., 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011)
Wine Recognition	14	178	3	(Webb et al., 2005; Yang et al., 2005, 2007; Zheng and Webb, 2007)
Zoo	18	101	7	(Yang et al., 2007; Zheng and Webb, 2007; Flores et al., 2010, 2011)

The analysis of Table 5.3 shows that the dataset with the highest number of classes is the nettalk (phoneme) dataset used in (Yang et al., 2007) (Zheng and Webb, 2007). The nettalk dataset has 8 attributes, 5,438 examples and 50 classes while our smallest dataset the GPCR-Prints-5z has 7 attributes, 5,505 examples and 8/46/76/49 classes at the 1st/2nd/3rd/4th levels respectively. This made us consider employing different search procedures, however alternatives approaches to the HCS used in the Bayesian classification literature such as floating search (Pernkopf and O'Leary, 2003) and tabu search (Bouckaert, 1995) are computationally more intense than the HCS. The solution we came to use is hereafter referred to as Very Greedy Hill Climbing Search (VGHCS). The VGHCS works as follows. Given a state of B_S the VGHCS verifies what are all the possible arc inclusions considering adding only one arc at a time and respecting the k number of parents constrain. It then randomly chooses one of the valid possible arc inclusions and compares the performance of this arc inclusion against the current model. If the inclusion of this arc improves the $Score(B_S, D)$ the VGHCS adds this arc to the model and start the next iteration of the search, where the current B_S includes the just added arc. If the randomly selected arc does not improve the $Score(B_S, D)$, it randomly selects another one, until one of them improves the $Score(B_S, D)$ or there are no more arcs to select. In the case none of the valid arcs improve the $Score(B_S, D)$, then the search ends and the current model is returned. Algorithm 6 illustrates this approach.

In order to modify the HCS-k-DBC to deal with hierarchical classification problems, the next question is how to deal with the $Score(B_S, D)$ to evaluate the different B_S models. In this work we have opted for a wrapper approach but instead of using a measure of predictive performance for flat classification problems, we use the hierarchical f-measure. Moreover, by using a wrapper approach with predictive performance as the objective function was found to produce the best performing network structures in (Pernkopf and Bilmes, 2010).

Algorithm 6: The VGHCS-k-DBC algorithm.

```
Input: A Dataset $D$ with an unordered set of Attributes, $A_1, \ldots, A_{n_a}$ and the
       class attribute $C$.
Input: The value of $k$ for the maximum number of $\pi_{A_i}$.
Output: A VGHCS-$k$-DBC classifier.
// Initialize the network to Naive Bayes:
1  $B_S \leftarrow C$;
2  foreach $node\ A_i, 1 \leq i \leq n_a$ do
       // Add the class node as the parent of node $A_i$:
3      $B_S \leftarrow B_S \bigcup (\pi_{A_i} \leftarrow C)$;
4  end
5  $NotDone \leftarrow True$;
6  while $NotDone$ do
7      Compute $Score(B_S, D)$; // Evaluate the current model.
       // Compute the number of valid Bayesian structures:
8      $V \leftarrow validStructures(B_S)$;
9      $keepSearching \leftarrow true$;
10     while $keepSearching = true \wedge V \neq \emptyset$ do
           // Randomly select an arc from V:
11         $(\pi_{A_j} \leftarrow A_i) \leftarrow getRandomArc(V)$;
12         if $Score(B_S \bigcup (\pi_{A_j} \leftarrow A_i), D) > Score(B_S, D)$ then
13             $keepSearching \leftarrow false$;
14             $B_S \leftarrow B_S \bigcup (\pi_{A_j} \leftarrow A_i)$;
15         end
           // Remove the selected arc from V:
16         $V \leftarrow V - (\pi_{A_j} \leftarrow A_i)$;
17     end
18     if $V = \emptyset$ then
           // There are no mode arcs to add, return the current model:
19         $NotDone \leftarrow False$;
20     end
21 end
```

So in this work, we are using a class-constrained Bayesian network classifier where all attributes are children of the class node and that allows up to k dependencies between the attributes. The search strategy is the VGHCS procedure and the $Score(B_S, D)$ is the hierarchical f-measure.

So far we have discussed the issues relative to the B_S of the proposed Bayesian classifier and apart from the use of the hierarchical f-measure as the $Score(B_S, D)$, all other issues are common to any classification problem, hierarchical or not. In order to adapt our algorithm to handle hierarchical classification problems, we will modify the way we compute the probabilities that is the B_P component of the algorithm.

For doing this we will build on the knowledge we acquired in developing the Global-Model Hierarchical-Classification Naive Bayes. As seen earlier the Naive Bayes classifier is a special case of a class-constrained Bayesian network classifier. Therefore, all the modifications we presented in Chapter 4 to take the hierarchy into account can be used for the k-Dependence Bayesian network classifier. The only difference, is that by allowing k attribute dependencies on the Bayesian network structure of the classifier, the CPTs will be conditioned on the Bayesian network structure of the classifier, as it was shown in Figure 5.2.

The classification of a new unseen example is given by Equation 10. This approach is hereafter referred to as GM-k-DBC (Global Model k-Dependence Bayesian network classifier).

$$P(\mathbf{X}|C_k) = \prod_{i=1}^{n_a} P(A_i = a_{ij}|\pi_i, C_k) = P(a_{1j}|\pi_i, C_k) \times \ldots \times P(a_{n_a j}|\pi_i, C_k) \qquad (10)$$

As seen in Chapter 4 there is an important trade-off between classification accuracy (measured by hierarchical f-measure) and usefulness (prediction depth). Therefore, we also evaluate a GM-k-DBCwU (with Usefulness), whose classification of a new unseen example is given by Equation 11.

$$P(\mathbf{X}|C_k) = \prod_{i=1}^{n_a} P(A_i = a_{ij}|\pi_i, C_k) \times Usefulness(C_k) = \\ (P(a_{1j}|\pi_i, C_k) \times \ldots \times P(a_{n_a j}|\pi_i, C_k)) \times Usefulness(C_k) \tag{11}$$

According to the taxonomy for classifying hierarchical classification algorithms, presented in Chapter 2, where a hierarchical classification algorithm is described as a 4-tuple $< \Delta, \Xi, \Omega, \Theta >$, the classification of the global-model hierarchical-classification k-Dependence Bayesian network classifier is as follows:

- Δ = SPP (Single Path Prediction), since the algorithm can only assign to each data example one path of predicted labels.
- Ξ = NMLP (Non-Mandatory leaf-node prediction), since the algorithm can assign classes at any level (including leaf classes).
- Ω = D (DAG). Although the algorithm has only been illustrated with tree-structured class problems so far, it can also cope with SPL (Single Path of Labels) DAG-structured class problems. In the case of the DAG-structured problem, the algorithm would compute all the priors and likelihoods in the same way as previously discussed.
- Θ = GC (Global Classifier), as the algorithm takes the hierarchy into account during the training phase and can predict any class during the test phase.

5.3 Experimental Set Up

5.3.1 Establishing a Baseline Method

An seen in the previous chapter, an important issue when dealing with hierarchical classification is how to establish a meaningful baseline method. In this section we introduce the two baseline approaches used in the experiments.

Both local classifier approaches are trained in the same way and differ in the method used in the testing phase. More precisely, during the training phase, for every non-leaf class node, a flat VGHCS-k-DBC classifier with $Score(B_S, D)$ being the flat classification f-measure (See Section 2.1) was trained to distinguish between the node's child classes. To implement the test phase, we have employed the same strategies introduced in Chapter 4, that is using the usefulness measure as a stopping criterion (this approach will be referred to as LM-k-DBCwU) and the one that always predicts a leaf node (referred to as LM-k-DBC) where k indicates the maximum number of parents a node can have, and LM stands for local model.

5.3.2 Protein Function Prediction Datasets Used in the Experiments

In this chapter, we have used a subset of the datasets used in Chapter 4, presented in Table 5.4. The main reason for not using all the datasets is due to the high number of classes and attributes values. As mentioned earlier, this is a valid concern as most studies within Bayesian network classifiers address problems with either a small number of attributes, a small number of classes or a small number of examples. The datasets characteristics are the same as described in Chapter 4 including the same discretisation process. Moreover, a recent work (Flores et al., 2011) comparing different discretisation approaches for Bayesian network classifiers concluded that regardless of the discretisation approach, all the methods behave in the same way. That is the n-th best classifier, is still the n-th best classifier regardless of the discretisation algorithm used.

5.4 Computational Results

In this section, we are interested in answering the following questions by using controlled experiments: (a) Does introducing the concept of k dependencies in

Table 5.4: Bioinformatics datasets details.

Dataset	n_a	n_e	n_c/Level	Υ	Ψ	Φ
EC-Interpro-5z	7	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-15z	17	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Interpro-AA	22	14,027	6/41/96/187	T	SPL	$PD_{21\%}$
EC-Pfam-5z	7	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-15z	17	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Pfam-AA	22	13,987	6/41/96/190	T	SPL	$PD_{21\%}$
EC-Prints-5z	7	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-15z	17	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prints-AA	22	14,025	6/45/92/208	T	SPL	$PD_{21\%}$
EC-Prosite-5z	7	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-15z	17	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
EC-Prosite-AA	22	14,041	6/42/89/187	T	SPL	$PD_{19\%}$
GPCR-Interpro-5z	7	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-15z	17	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Interpro-AA	22	7,444	12/54/82/50	T	SPL	$PD_{7\%}$
GPCR-Pfam-5z	7	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-15z	17	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Pfam-AA	22	7,053	12/52/79/49	T	SPL	$PD_{7\%}$
GPCR-Prints-5z	7	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-15z	17	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prints-AA	22	5,404	8/46/76/49	T	SPL	$PD_{10\%}$
GPCR-Prosite-5z	7	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-15z	17	6,246	9/50/79/49	T	SPL	$PD_{8\%}$
GPCR-Prosite-AA	22	6,246	9/50/79/49	T	SPL	$PD_{8\%}$

the algorithm affect the predictive performance of the local (with the top-down class prediction approach) or global (with the proposed method) approaches? (b) How does the inclusion of the usefulness criterion (Equation 11) affect the global model k-dependence Bayesian classification algorithm?

These questions are addressed in this section.

All the experiments reported in this section were obtained by using the datasets presented in Section 5.3.2, using stratified ten-fold cross-validation (Witten and Frank, 2005). In order to evaluate the algorithms we have used the metrics of hierarchical precision (hP), hierarchical recall (hR) and hierarchical f-measure (hF) proposed in (Kiritchenko et al., 2005) (See Section 2.7).

To measure if there is any statistically significant difference between the hierarchical classification methods (measured by hierarchical f-measure) being compared, we have employed the Friedman test with the post-hoc Shaffer's static procedure (with $\alpha = 0.05$) for comparison of multiple classifiers over many datasets as strongly recommended by (García and Herrera, 2008). Due to the high number of classifiers employed (12 in total), the results of the statistical test are presented in a graphical way (as suggested in (Demsar, 2006)) in Figure 5.9 in order to summarize the pairwise comparisons made by the test. In Figure 5.9 the bold horizontal lines connect the representations whose results are not found to be statistically significantly different.

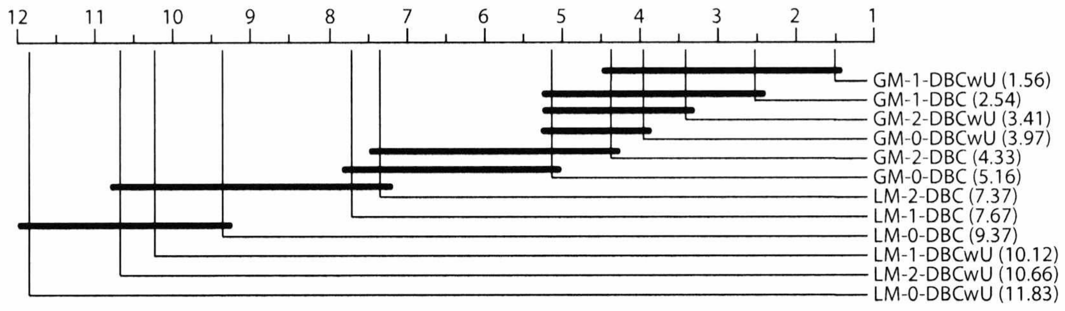


Figure 5.9: Statistical tests for the hierarchical k-DBC classifiers with $\alpha = 0.05$.

5.4.1 Evaluating the Effect of Introducing k Dependencies in Local and Global Model Approaches

We first evaluate the impact of the k dependence in the different types of hierarchical classification algorithms. Table 5.5 presents the results for the baseline local-model k-dependence Bayesian classifier approaches (LM-k-DBCwU and LM-k-DBC) described in Section 5.3.1. Table 5.6 presents the results for the proposed global-model k-dependence Bayesian classifier (GM-k-DBCwU and GM-k-DBC), both with and without usefulness. In both Tables, the range of the value k is from 0 to 2 (where $k = 0$ corresponds to the Naive Bayes classifier).

Table 5.5: Hierarchical F-measure results on the bioinformatics datasets with the LM-k-DBC algorithm.

	LM-k-DBC			LM-k-DBCwU		
Datasets	$k = 0$	$k = 1$	$k = 2$	$k = 0$	$k = 1$	$k = 2$
EC-Interpro-5z	33.18	39.08	40.28	29.06	33.20	34.22
EC-Interpro-15z	35.22	41.65	41.12	30.11	34.92	34.46
EC-Interpro-AA	37.05	41.73	41.59	31.70	34.95	34.75
EC-Pfam-5z	32.70	38.35	39.32	28.56	32.46	33.34
EC-Pfam-15z	35.95	41.30	40.33	30.49	34.51	33.63
EC-Pfam-AA	36.45	41.30	41.98	31.14	34.21	34.92
EC-Prints-5z	35.59	41.09	40.88	34.07	39.06	38.81
EC-Prints-15z	38.19	42.47	42.74	36.31	40.42	40.28
EC-Prints-AA	42.44	45.70	46.27	39.78	42.64	43.10
EC-Prosite-5z	32.49	38.79	40.67	22.94	24.57	25.23
EC-Prosite-15z	35.43	41.03	40.39	23.88	25.08	24.37
EC-Prosite-AA	37.64	42.00	42.76	24.53	25.65	24.92
GPCR-Interpro-5z	49.34	52.88	53.28	40.18	40.58	40.32
GPCR-Interpro-15z	57.35	61.45	60.07	43.15	43.90	43.14
GPCR-Interpro-AA	61.03	64.80	65.86	45.56	46.42	44.96
GPCR-Pfam-5z	50.75	53.76	55.15	41.28	41.67	41.99
GPCR-Pfam-15z	59.08	62.76	61.44	45.05	45.89	44.43
GPCR-Pfam-AA	62.20	66.36	67.27	46.65	48.05	45.96
GPCR-Prints-5z	54.91	58.39	58.65	42.36	43.27	42.96
GPCR-Prints-15z	59.76	65.37	64.73	43.46	45.15	44.73
GPCR-Prints-AA	63.81	67.59	70.01	44.61	45.93	45.85
GPCR-Prosite-5z	55.05	58.48	58.95	41.69	42.31	42.43
GPCR-Prosite-15z	60.28	64.24	63.87	42.52	43.83	43.34
GPCR-Prosite-AA	63.00	66.59	68.34	43.44	44.59	43.83

For all approaches, local or global with or without usefulness, the introduction of k dependencies allows the algorithm to achieve better prediction. This can be seen from the average ranking of the algorithms, shown in Figure 5.9 (recall that the lower the rank the better the predictive performance of the algorithm). However for all approaches, this difference is not statistically significant within each approach. That is, in Figure 5.9 when comparing the same approach with different values of k, there is always a bold horizontal line connecting the approaches, which means that the results are not found to be statistically significantly different.

The analysis of Figure 5.9 shows that by introducing k dependencies on the

Table 5.6: Hierarchical F-measure results on the bioinformatics datasets with the GM-k-DBC algorithm.

	GM-k-DBC			GM-k-DBCwU		
Datasets	$k=0$	$k=1$	$k=2$	$k=0$	$k=1$	$k=2$
EC-Interpro-5z	42.72	44.91	44.22	44.19	46.08	45.23
EC-Interpro-15z	47.94	50.31	48.60	49.57	51.65	51.19
EC-Interpro-AA	54.33	55.43	53.03	55.98	57.21	54.07
EC-Pfam-5z	43.22	44.98	43.98	44.91	46.35	45.25
EC-Pfam-15z	48.43	49.46	48.01	50.16	50.83	49.92
EC-Pfam-AA	55.09	55.87	53.22	56.68	57.30	55.51
EC-Prints-5z	48.70	49.66	47.99	49.81	50.54	49.66
EC-Prints-15z	53.25	55.48	53.40	54.94	56.63	54.34
EC-Prints-AA	62.00	62.18	60.78	63.19	63.24	62.07
EC-Prosite-5z	44.18	45.58	45.31	45.50	46.89	45.87
EC-Prosite-15z	50.22	51.95	50.13	52.37	53.38	51.83
EC-Prosite-AA	56.04	56.70	54.41	57.89	58.43	56.44
GPCR-Interpro-5z	57.37	58.23	57.56	57.02	58.57	58.33
GPCR-Interpro-15z	64.62	66.16	65.57	64.44	65.32	64.48
GPCR-Interpro-AA	73.16	74.27	72.62	74.26	74.06	73.19
GPCR-Pfam-5z	59.34	60.65	60.22	58.32	59.45	59.03
GPCR-Pfam-15z	66.08	66.75	66.20	65.80	66.80	67.23
GPCR-Pfam-AA	74.78	75.65	74.83	75.80	75.80	75.12
GPCR-Prints-5z	62.72	64.59	64.63	61.92	63.36	62.50
GPCR-Prints-15z	68.18	70.33	69.49	67.62	69.25	68.85
GPCR-Prints-AA	75.92	77.28	76.44	76.13	77.48	77.18
GPCR-Prosite-5z	61.97	64.39	63.69	61.70	62.90	62.64
GPCR-Prosite-15z	67.43	68.41	68.70	67.05	68.72	68.60
GPCR-Prosite-AA	74.86	76.20	75.89	75.65	76.24	76.20

LM-k-DBC both LM-1-DBC and LM-2-DBC results are not statistically significant different from LM-0-DBC. By allowing attribute dependencies on the local models, their predictive performance was improved. The reason for this lies in the fact that in every non-leaf node a k-DBC classifier is built. This allows for the LM-k-DBC classifier to use different models at different non-leaf nodes. Note that this is not the case for LM-0-DBC which uses a fixed NB structure at each non-leaf node.

For the global model approaches, using $k = 1$ provides better results for both

GM-1-DBCwU and GM-1-DBC. Indeed, they are both ranked as the top best performing algorithms respectively. The use of $k = 2$ in the global model approaches is still better than using the GM-0-DBC but its results are worse than the ones achieved by $k = 1$ at a much higher computation cost. This result (that $k = 1$ produces better results) corroborates with a recent study (Flores et al., 2011) of Bayesian network classifiers on flat classification problems.

When comparing the results of the local vs. global approaches, the use of the GM-k-DBCwU (with $k = 0, 1$ or 2) and the GM-1-DBC are always (statistically) better than any of the local model approaches. The GM-0-DBC and the GM-2-DBC although achieving better predictive performance than all local mode approaches, are not statistically better than LM-1-DBC and LM-2-DBC. The reason for this lies in the fact that the LM-k-DBC with $k = 1$ or 2 is able to create different Bayesian network structures at different non-leaf nodes which augments considerably its predictive performance.

In summary, two main conclusions can be drawn from these experiments. First, although varying the value of parameter k did not result in statistically significantly different predictive accuracies in each of the four versions of k-DBC (considering a combination of local vs. global and "with usefulness" vs. "without usefulness" approaches), values of $k = 1$ or 2 consistently led to higher predictive accuracies than $k = 0$ (corresponding to a Naive Bayes classifier) in all four versions of k-DBC. As mentioned earlier, $k = 1$ seems to represent the best trade-off between maximizing predictive accuracy and reducing computational time. Secondly, in the ranking of the 12 algorithms (referring to four versions of k-DBC with three different k values for each version), the top six algorithms in the rank were exactly the six global versions, whilst the bottom six were the local versions. In addition, each of the top four global versions of k-DBC obtained a statistically significantly higher predictive accuracy than each of the six local versions. Hence, these results validate the effectiveness of the global hierarchical versions of k-DBC proposed in this thesis.

5.5 Complexity Analysis

So far we have discussed and compared the different global and local approaches based on their predictive performance using the hierarchical f-measure as a scoring function. Let us now analyse the time complexity of the different approaches presented in this chapter.

The flat naive Bayes algorithm has the training time complexity of $O(n_t n_a)$ (Webb et al., 2005) where n_t is the number of training examples and n_a is the number of attributes. As the LMNB algorithms create one flat NB classifier for each non-parent node, its training time complexity is $O(n_t n_a n_{nl})$ where n_{nl} is the number of non-leaf classes in C.

In the case of the LMNBwU it is necessary to calculate the time complexity of the usefulness measure (note that this time complexity is going to be the same for all the algorithms that employ the usefulness measure). The calculation of the usefulness measure is done in a pre-processing step during the training and has time complexity of $O(n_c n_{cl})$ where n_{cl} is the number of class levels in the class hierarchy. Therefore the worst case time complexity of LMNBwU is $O(n_t n_a n_{nl} + n_c n_{cl})$.

For the GMNB the training time complexity is $O(n_t n_a n_{cl})$. The main difference between the LMNB and the GMNB is that the latter during the training phase creates a table with all the classes in the hierarchy and uses the information from each training example to update the counts of the tables relative to all the classes the example belongs (there are n_{cl} such classes, since an example is assigned one class for each class level). In the case of the GMNBwU, as seen previously, the usefulness computation during training has a time complexity of $O(n_c n_{cl})$, therefore the time complexity of the GMNBwU is $O(n_t n_a n_{cl} + n_c n_{cl})$.

So far we have discussed the training time complexity of the flat and global–model version of the naive Bayes classifier. Recall that in these approaches there is no structure search. Now let us examine the training time complexity of the local and global model VGHCS-k-DBC. Although the VGHCS-k-DBC is a heuristic to speed up the training time of the algorithms, in the worst case scenario the VGHCS-k-DBC will perform as the HCS-k-DBC and for this reason we will focus on the analysis of the HCS-k-DBC. The main training time bottleneck of the HCS-k-DBC is the main loop responsible for the structure search in lines 6-14 of Algorithm 5.

In the context of flat classification, in line 7, computing the score function for a given candidate structure, requires first learning the parameters of the network given the structure, and then classifying the examples in order to measure the classification accuracy (assuming the score function is based on classification accuracy). Learning the parameters requires scanning the training set once, with time $O(n_t n_a)$. Using the structure and the parameters to classify a single example has time $O(n_a n_c k)$, since it requires to scan the n_a attribute values in the example and each attribute can have in the worst case k parents (requiring the access to k parents in order to compute the conditional probability for that attribute). The classification process is repeated for all examples in the validation set and this process takes $O(n_t n_a n_c k)$. Hence, the total time complexity for line 7 is $O(n_t n_a) + O(n_t n_a n_c k)$, which is dominated by $O(n_t n_a n_c k)$. Line 8 involves trying to add an arc and evaluating the new structure with that arc with time $O(n_t n_a n_c k)$, which has to be done on the order of n_a times (one arc for each attribute). Hence, the total time complexity of line 8 is $O(n_t n_a^2 n_c k)$. Lines 9-13 have time complexity $O(1)$, which can be ignored. Hence, the time complexity of lines 7-12 is dominated by the term $O(n_t n_a^2 n_c k)$, and since the while loop at line 6 is repeated for at most n_a iterations, the total time of the loop, which is also the total time complexity of the HCS-k-DBC algorithm, is $O(n_t n_a^3 n_c k)$. Therefore the worst-case time complexities for the LM-k-DBC and LM-k-DBCwU are $O(n_t n_a^3 n_c k n_{nl})$ and $O(n_t n_a^3 n_c k n_{nl} + n_c n_{cl})$ respectively.

Concerning the use of the VGHCS-k-DBC algorithm in hierarchical classification, the basic pseudocode is essentially the one shown in Algorithm 5, with just one difference. In the hierarchical version of the algorithm, the score function has to compute a measure of predictive accuracy in hierarchical classification, rather than a measure of predictive accuracy in flat classification. This involves two steps: first, computing the precision and recall for each class in the tree hierarchy; and then aggregating those results into a single hierarchical precision/recall/F-measure value. The time complexity is dominated by that first step, so that the aggregation step can be ignored in the time complexity analysis. The computation of precision and recall for each class involves accessing not only that current class but also its ancestors in the class hierarchy. The worst-case complexity of this operation occurs when computing the precision and recall for leaf classes, and each of those leaf classes has, in the worst case, n_{cl} ancestors. Since the precision and recall measures have to be computed for each class, computing those measures for all classes takes a time proportional to the product $n_c n_{cl}$. In other words, the time complexity for computing the score in hierarchical classification is proportional to $n_c n_{cl}$, whilst the time complexity for computing the score in flat classification is proportional to n_c. The other parts of the time complexity analysis for the hierarchical classification version of VGHCS-k-DBC are essentially the same as for the flat classification version of VGHCS-k-DBC, and so the time complexity analysis for the hierarchical classification version of VGHCS-k-DBC is $O(n_t n_a^3 n_{cl} n_c k)$. Therefore, the worst- case time complexity for the GM-k-DBC is $O(n_t n_a^3 n_{cl} n_c k)$. Concerning the version of the algorithm with the usefulness measure, the time complexity of GM-k-DBCwU is $O(n_t n_a^3 n_{cl} n_c k + n_c n_{cl})$.

5.6 Discussion and Related Work

In this chapter we have introduced the concept of global-model hierarchical classification k-Dependence Bayesian classifier, which is an extension of the k-Dependence Bayesian classifier to deal with hierarchical classification problems. By allowing attribute dependencies, we have obtained a gain in predictive performance measured by hierarchical f-measure, but these results were not found to be statistically significant different from the global-model hierarchical-classification Naive Bayes. This might be due to the search strategy employed, the very greedy hill climbing search, which might be converging to a local optimum during the model search process. However the use of other more costly searches was unfeasible, as for our smaller dataset, performing 10-fold cross-validation would require 40 days of cpu time. The final decision about which global-algorithm should be used will depend on the target problem. For instance, in experiments such as the one done in (Deng et al., 2010), which has over 10,000 image categories to be classified, the use of the GMNB(wU) would be preferred over the VGHCS-k-DBC(wU) with values of k greater than 0 (recall that $k = 0$ corresponds to a Naive Bayes model), since the former is much faster and much more scalable to a large number of classes. Concerning which version of a given type of algorithm to use in non-mandatory leaf node prediction problems, that is, whether to use the version which takes usefulness into account or not, based on our experimental results, we would recommend to always use the versions with usefulness, as they always provide the best results in our experiments.

The use of k-Dependence Bayesian network classifiers for hierarchical classification was explored by (Koller and Sahami, 1997) which employed the k-DBC algorithm as a base classifier in a LCPN (local classifier per parent node) approach in the text classification domain. In order to perform structure learning at each node in the hierarchy of classifiers, they have applied feature selection as a pre-processing step. The issue of feature selection in hierarchical classification problems is still understudied, but the novel algorithms proposed in this thesis can certainly benefit from them.

Although not a Bayesian network classifier per se, in the work of Barocuoglu et al. (Barutcuoglu and DeCoro, 2006; Barutcuoglu et al., 2006; DeCoro et al., 2007; Guan et al., 2008) the authors employed a Bayesian network to ensure prediction consistency while using binary classifiers in a LCN (local classifier per node) approach in different application domains.

In (Campos et al., 2006) the authors present a theoretical framework for using Bayesian networks to the task of web categorization in hierarchical directories using Bayesian networks. In their approach, the Bayesian network structure is fixed (i.e. there is no search involved). Every class node in the hierarchy is a node in the Bayesian network. Each document (example) in the training set is a node in the Bayesian network. Each term in the training set (after the usual pre-processing steps for text classification) is a node in the Bayesian network. Note that in web categorization the presence or absence of a particular term (which are words after pre-processing steps such as case folding and stemming) can be used as attributes for the classification task. They make further modifications to this basic structure in order to produce better estimations in the model. However, regardless of the proposed extensions, this approach is limited in the sense that it only works for this particular representation (i.e. the presence/absence) of terms and it is not general enough to deal with any other type of hierarchical classification problem such as hierarchical protein function prediction.

In (Gyftodimos and Flach, 2004) the authors propose an extension to the Bayesian Network formalism known as hierarchical Bayesian networks (HBN) to deal with structured domains. The main idea behind the HBN is that a particular set of nodes can be represented by a "meta" node in the network. To illustrate this approach, in (Gyftodimos and Flach, 2004), the authors use an extension of the PlayGolf example and show how the attributes Outlook, Temperature and Wind can be represented by a "meta" node called Weather. The main advantage of the HBN is that is simplifies the visual aspect of the Bayesian network, as in the final decision model, one has less attributes to look at, but it is equivalent to a "flat" Bayesian network in its representation power.

Another extension to the Bayesian network formalism that takes hierarchical relationships between the attributes into account is known as Recursive Bayesian networks (RBN) (Williamson and Gabbay, 2005). The RBN extends the Bayesian network formalism by allowing the network attributes to be represented with Bayesian networks as values. There are two types of attributes in a RBN, called network attributes (which takes Bayesian Networks as values) and simple attributes (whose values do not contain such structure). The main difference between the RBN and HBN is that in the RBN the network attributes are existing attributes within the dataset (i.e. no "meta" node is created). Moreover, different states of a network attribute are represented by using the same underlying attributes although with a different network structure for each value of a given network attribute. For example, in the PlayGolf mentioned earlier, let's assume the Weather attribute existed within the dataset and it is a network attribute. Assuming the attribute Weather has two possible values (good and bad). The network representing the value of Weather = good could be a structure with no arcs between the attributes Outlook, Temperature and Wind, while the network representing the value of weather = bad could be a structure with an arc from Outlook to both Temperature and Wind.

Chapter 6

Conclusions

In this work we proposed several new approaches for the task of hierarchical classification, extending previous approaches in the area, and evaluating them on the task of hierarchical protein function prediction. Note that the proposed hierarchical classification approaches are not specialized to the application domain of protein function prediction, in principle they are generic enough to be applied to other application domains. We now briefly summarize the contributions of the thesis.

The first contribution of this work was to bring together the hierarchical classification research done in different applications domains under one unifying theoretical framework.

In order to gain hands-on experience of the task of hierarchical classification, we started by proposing a novel local-model per parent node approach that aims at selecting the attribute representation (more precisely, protein representation in the context of this thesis) with the greatest predictive power for training a classifier at a given parent node in the class hierarchy. The development of this method has created the need to develop novel datasets for the problem of hierarchical protein function prediction, which are extensions of the datasets originally developed in (Holden and Freitas, 2006). These datasets have been made publicly available. Moreover, the development of the local-model hierarchical classification approach with selective protein representation was important to gain insights into the task of hierarchical classification.

Besides the novel local–model approach, we have proposed a novel extension of the well known (flat) classification algorithm Naive Bayes to the task of hierarchical classification. The developed algorithm is referred to as global–model hierarchical–classification Naive Bayes (GMNB). We have introduced the notion of usefulness into the algorithm, creating a variation known as Global–Model Hierarchical–classification Naive Bayes with Usefulness (GMNBwU). Experiments comparing both algorithms to conventional local–model approaches using the same base algorithm (Naive Bayes) have shown that the proposed methods perform well in the task and were statistically significant superior to the local-model baseline approach. No statistically significant difference was found between the results of the different global-model approaches.

Given the good performance of the GMNB and GMNBwU algorithms we relaxed the strong attribute independence of the Naive Bayes algorithm by allowing the attributes to have at most k other attributes as parents by using the framework of k-Dependence Bayesian network classifiers. Two novel algorithms for hierarchical classification were developed, the very greedy hill climbing search k-Dependence Bayesian network classifier (VGHCS-k-DBC) and its variation utilizing the usefulness criterion (VGHCS-k-DBCwU). The experiments comparing the performance of these classifiers with two conventional local–model approaches, which also allow for k dependencies between the attributes, have shown that the VGHCS-1-DBCwU and the VHCS-1-DBC were ranked number one and two (out of 12 algorithms) on average across all experiments. Moreover, these two algorithms (with $k = 1$) were statistically significant better than any of the six local approaches. When compared against the other values of k, the value $k = 1$ showed no statistically significant difference in the results. In this thesis we focused on the problem of non-mandatory leaf-node prediction, of course, if the target problem is a mandatory leaf node prediction one, then the classes to be considered should be restricted only to the leaf classes. In the case of a hierarchy where all the leaf classes are on the same depth, the GMNB and GMNBwU would produce exactly the same result, as all the leaf classes would have the same usefulness measure.

6.1 Future work

Considering the way the class hierarchy was introduced in the developed Bayesian network classifiers, the same idea can be used to create other novel bayesian classifiers. Examples of such approach would be using unrestricted Bayesian network algorithms to create hierarchical Bayesian classifiers.

Also, considering that in the experimental results the k value which obtained the best result was one, it would be interesting to verify the performance of a Hierarchical Classification Average One Dependence Estimators (AODE). Note that a hierarchical classification AODE would mitigate the drawback of very long training time of the VGHCS-k-DBC which is the model search strategy and possibly giving similar predictive results.

Another research direction that would greatly speed up the construction of the global-model k-DBC is the development of information theoretic measures, such as mutual information and class-conditional mutual information which can handle hierarchical classification problems. If such measurements existed, the original algorithms for creating TAN and k-DBC classifiers could be used to develop their hierarchical classification versions.

Currently, the proposed algorithms are limited in the type of hierarchical classification problems they can solve. At present, they can only deal with single path predictions. It would be interesting to extend them further by developing multiple path predictions hierarchical Bayesian network classification algorithms.

In this thesis the proposed global hierarchical Bayesian network classifiers have been applied only to tree-structured class hierarchies. Another research direction would be to apply the proposed algorithms to problems with DAG-structured class hierarchies.

All the experiments done in this work were performed in the task of hierarchical protein function prediction. It would be interesting to evaluate the proposed algorithms in other hierarchical classification application domains, such as automatic text categorization, automatic music genre recognition, automatic image classification, etc.

Moreover, it would be interesting to gather and pre-process in the hierarchical-classification arff (harff) format, different datasets for these different application domains, and perform an empirical investigation comparing all the applicable approaches.

Bibliography

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Al-Shahib, A., Breitling, R., and Gilbert, D. (2005). Feature selection and the class imbalance problem in predicting protein function from sequence. *Applied Bioinformatics*, 4(3):195–203.

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D. (2002). *Molecular Biology of the Cell.* Garland Publishing.

Albornoz, E. M., Milone, D. H., and Rufiner, H. L. (2010). *Development of Multimodal Interfaces: Active Listening and Synchrony*, volume 5967 of *Lecture Notes in Computer Science*, chapter Multiple Feature Extraction and Hierarchical Classifiers for Emotions Recognition, pages 242–254. Springer-Verlag, Berlin Heidelberg.

Aleksovski, D., Kocev, D., and Dzeroski, S. (2009). Evaluation of distance measures for hierarchical multilabel classification in functional genomics. In *Proc. of the 1st Workshop on Learning from Multi-Label Data (MLD) held in conjunction with ECML/PKDD*, pages 5–16.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410.

Altun, Y. and Hofmann, T. (2003). Large margin methods for label sequence learning. In *Proc. of the 8th European Conf. on Speech Communication and Technology (EuroSpeech)*.

Alves, R. T., Delgado, M. R., and Freitas, A. A. (2008). Multi-label hierarchical classification of protein functions with artificial immune systems. In *Advances in Bioinformatics and Computational Biology*, volume 5167 of *Lecture Notes in Bioinformatics*, pages 1–12. Springer.

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

Astikainen, K., Holmand, L., Pitkanen, E., Szedmak, S., and Rousu, J. (2008). Towards structured output prediction of enzyme function. *BMC Proceedings*, 2(Suppl 4).

Attwood, T. K. (2002). The prints database: A resource for identification of protein families. *Briefings in Bioinformatics*, 3(3):252–263.

Barbedo, J. G. A. and Lopes, A. (2007). Automatic genre classification of musical signals. *EURASIP Journal on Advances in Signal Processing*, 2007:12.

Barret, A. J. (1997). Nomenclature committee of the international union of biochemistry and molecular biology (nc-iubmb). enzyme nomenclature. recommendations 1992. supplement 4: corrections and additions (1997). *European journal of biochemistry*, 250(1):1–6.

Barutcuoglu, Z. and DeCoro, C. (2006). Hierarchical shape classification using bayesian aggregation. In *Proc. of the IEEE Conf. on Shape Modeling and Applications*.

Barutcuoglu, Z., Schapire, R. E., and Troyanskaya, O. G. (2006). Hierarchical multi-label prediction of gene function. *Systems Biology*, 22:830–836.

Bateman, A., Coin, L., Durbin, R., Finn, R. D., Hollich, V., Griffiths-Jones, S., Khanna, A., Marshall, M., Moxon, S., Sonnhammer, E. L. L., Studholme, D. J., Yeats, C., and Eddy, S. R. (2004). The pfam protein families database. *Nucleic Acids Research*, 32:D138–D141. Database issue.

Ben-Hur, A. and Brutlag, D. (2003). Remote homology detection: a motif based approach. *Bioinformatics*, 19(Suppl. 1):i26–i33.

Ben-Hur, A. and Brutlag, D. (2006). *Feature extraction, foundations and applications*, chapter Protein sequence motifs: Highly predictive features of protein function, pages 625–645. Springer.

Bennett, P. N. and Nguyen, N. (2009). Refined experts: improving classification in large taxonomies. In *Proc. of the 32nd Int. ACM SIGIR conf. on Research and development in information retrieval*, pages 11–18.

Binder, A., Kawanabe, M., and Brefeld, U. (2009). Efficient classification of images with taxonomies. In *Proc. of the 9th Asian Conf. on Computer Vision*.

Blockeel, H., Bruynooghe, M., Dzeroski, S., Ramon, J., and Struyf, J. (2002). Hierarchical multi-classification. In *Proceedings of the First SIGKDD Workshop on MultiRelational Data Mining (MRDM-2002)*, pages 21–35.

Blockeel, H., Schietgat, L., Struyf, J., so Džeroski, S., and Clare, A. (2006). Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *Knowledge Discovery in Databases: PKDD 2006*, volume 4213 of *Lecture Notes in Computer Science*, pages 18–29. Springer.

Bouckaert, R. R. (1995). *Bayesian Belief Networks: from Construction to Inference.* PhD thesis, University of Utrecht.

Brecheisen, S., Kriegel, H.-P., Kunath, P., and Pryakhin, A. (2006a). Hierarchical genre classification for large music collections. In *Proc. of the IEEE 7th Int. Conf. on Multimedia & Expo*, pages 1385–1388.

Brecheisen, S., Kriegel, H.-P., Kunath, P., Pryakhin, A., and Vorberger, F. (2006b). MUSCLE: Music classification engine with user feedback. In Springer, editor, *Proc. of the 10th Int. Conf. on Extending Database Technology*, number 3896 in LNCS, pages 1164–1167.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):195–210.

Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. F., and Weiss, B. (2005). A database of german emotional speech. In *Proc. of the 9th European Conf. on Speech Communication and Technology*, pages 1517–1520.

Burred, J. J. and Lerch, A. (2003). A hierarchical approach to automatic musical genre classification. In *Proc. of the 6th Int. Conf. on Digital Audio Effects*, pages 8–11.

Cai, C. Z., Han, L. Y., Ji, Z. L., Chen, X., and Chen, Y. Z. (2003). Svm-prot: web-based support vector machine software for functional classification of a protein from its primary sequence. *Nucleic Acids Research*, 31(13):3962–3697.

Cai, L. and Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In *Proc. of the 13th ACM Int. Conf. on Information and knowledge management*, pages 78–87.

Cai, L. and Hofmann, T. (2007). Exploiting known taxonomies in learning overlapping concepts. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence*, pages 714–719.

Campos, L. M. d., Fernandez-Luna, J. M., and Huete, J. F. (2006). *Soft Computing in Web Information Retrieval: Models and Applications*, volume 197 of *Studies in Fuzziness and Soft Computing*, chapter A Theoretical Framework for Web Categorization in Hierarchical Directories using Bayesian Networks, pages 25–43. Springer.

Carvalho, R. V., Brunoro, G., and Pappa, G. L. (2011). Hcga: A genetic algorithm for hierarchical classification. In *Proc. of the IEEE Congress on Evolutionary Computation.*

Castillo, G. and Gama, J. (2009). Adaptive bayesian network classifiers. *Intelligent Data Analysis*, 13(1):39–59.

Ceci, M. and Malerba, D. (2007). Classifying web documents in a hierarchy of categories: A comprehensive study. *Journal of Intelligent Information Systems*, 28(1):1–41.

Cerquides, J. and de Mantaras, R. L. (2005). Robust bayesian linear classifier ensembles. In *Proc. of the16th European Conf. on Machine Learning*, volume 3720 of *LNCS*, pages 72–83. Springer.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. (2006a). Hierarchical classification: combining Bayes with SVM. In *Proc. of the 23rd Int. Conf. on Machine learning*, pages 177–184.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. (2006b). Incremental algorithms for hierarchical classification. *The Journal of Machine Learning Research*, 7:31–54.

Cesa-Bianchi, N. and Valentini, G. (2009). Hierarchical cost-sensitive algorithms for genome-wide gene function prediction. In *Third International Workshop on Machine Learning in Systems Biology.*

Chakrabarti, S., Dom, B., Agrawal, R., and Raghavan, P. (1998). Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, 7:163–178.

Chen, Y., Crawford, M. M., and Ghosh, J. (2004). Integrating support vector machines in a hierarchical output space decomposition framework. In *Proc. of the IEEE Int. Symp. on Geoscience and Remote Sensing*, volume 2, pages 949–952.

Cheng, J. and Greiner, R. (1999). Comparing bayesian network classifiers. In *Proc. of the 15th Conf. on Uncertainty in Artificial Intelligence*, pages 101–108.

Cheng, J. and Greiner, R. (2001). Learning bayesian belief network classifiers: Algorithms and system. In *Proc. of the Canadian Conf. on Artificial Intelligence.*

Clare, A. (2004). *Machine learning and data mining for yeast functional genomics.* PhD thesis, University of Wales Aberystwyth.

Clare, A. and King, R. D. (2003). Predicting gene function in saccharomyces cerevisiae. *Bioinformatics*, 19:ii42–ii49. Suppl. 2.

Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347.

Corne, D. W. and Fogel, G. B. (2002). *Evolutionary Computation in Bioinformatics*, chapter An Introduction to Bioinformatics for Computer Scientists, pages 3–18. Morgan Kaufmann.

Costa, E., Lorena, A., Carvalho, A., and Freitas., A. (2007a). A review of performance evaluation measures for hierarchical classifiers. In *Evaluation Methods for Machine Learning II: papers from the 2007 AAAI Workshop*, pages 1–6. AAAI Press.

Costa, E., Lorena, A., Carvalho, A., Freitas, A. A., and Holden., N. (2007b). Comparing several approaches for hierarchical classification of proteins with decision trees. In *Advances in Bioinformatics and Computational Biology*, volume 4643 of *Lecture Notes in Bioinformatics*, pages 126–137. Springer.

Costa, E. P., Lorena, A. C., de Carvalho, A., and Freitas, A. A. (2008). Top-down hierarchical ensembles of classifiers for predicting g-protein-coupled-receptor functions. In *Advances in Bioinformatics and Computational Biology*, volume 5167 of *Lecture Notes in Bioinformatics*, pages 35–46. Springer.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27.

Cristianini, N. and Hahn, M. W. (2007). *Introduction to Computational Genomics*. Cambridge.

Cui, J., Han, L. Y., Li, H., Ung, C. Y., Tang, Z. Q., Zheng, C. J., Cao, Z. W., and Chen, Y. Z. (2007). Computer prediction of allergen proteins from sequence-derived protein structural and physicochemical properties. *Molecular Immunology*, 44:514–540.

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

Davies, M., Secker, A., Freitas, A., Clark, E., Timmis, J., and Flower, D. (2008a). Optimizing amino acid groupings for GPCR classification. *Bioinformatics*, 24(18):1980–1986.

Davies, M., Secker, A., Freitas, A., Mendao, M., Timmis, J., and Flower, D. (2007). On the hierarchical classification of G protein-coupled-receptors. *Bioinformatics*, 23(23):3113–3118.

Davies, M., Secker, A., Freitas, A., Timmis, J., Clark, E., and Flower, D. (2008b). Alignment-independent techniques for protein classification. *Current Proteomics*, 5(4):217–223.

DeCoro, C., Barutcuoglu, Z., and Fiebrink, R. (2007). Bayesian aggregation for hierarchical genre classification. In *Proc. of the 8th Int. Conf. on Music Information Retrieval*, pages 77–80, Vienna, Austria.

Dekel, O., Keshet, J., and Singer, Y. (2004a). Large margin hierarchical classification. In *Proc. of the 21th Int. Conf. on Machine learning*.

Dekel, O., Keshet, J., and Yoram Singer, Y. (2004b). An online algorithm for hierarchical phoneme classification. In *Proc. of the 1st Machine Learning for Multimodal Interaction Workshop*, volume 3361 of *Lecture Notes in Computer Science*, pages 146–158.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Deng, J., Berg, A. C., Li, K., and Fei-Fei, L. (2010). What does classifying more than 10,000 image categories tell us? In *Proc. of the 11th European Conf. on Computer Vision*, volume 6315 of *LNCS*, pages 71–84.

Dimitrovski, I., Kocev, D., Loskovska, S., and Dzeroski, S. (2008). Hierarchical annotation of medical images. In *Proc. of the 11th Int. Multiconference Information Society*, volume A, pages 174–177.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proc. of the 12th Int. Conf. on Machine Learning*, pages 194–202.

Downie, J. S. and Cunningham, S. J. (2002). Toward a theory of music information retrieval queries: System design implications. In *Proc. of the 3rd Int. Conf. on Music Information Retrieval*, pages 299–300.

Drawid, A. and Gerstein, M. (2000). A bayesian system integrating expression data with sequence patterns for localizing proteins: Comprehensive application to the yeast genome. *Journal of Molecular Biology*, 301:1059–1075.

Dubchak, I., Muchnik, I., Holbrook, S. R., and Kim, S.-H. (1995). Prediction of protein folding class using global description of amino acid sequence. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 92, pages 8700–8704.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons.

Dumais, S. T. and Chen, H. (2000). Hierarchical classification of Web content. In Belkin, N. J., Ingwersen, P., and Leong, M.-K., editors, *Proc. of the 23rd ACM Int. Conf. on Research and Development in Information Retrieval*, pages 256–263.

Eisner, R., Poulin, B., Szafron, D., Lu, P., and Greiner, R. (2005). Improving protein function prediction using the hierarchical structure of the gene ontology. In *Proc. of the IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–10.

Esuli, A., Fagni, T., and Sebastiani, F. (2008). Boosting multi-label hierarchical text categorization. *Information Retrieval*, 11(4):287–313.

Fagni, T. and Sebastiani, F. (2007). On the selection of negative examples for hierarchical text categorization. In *Proc. of the 3rd Language Technology Conference*, pages 24–28.

Filmore, D. (2004). Its a gpcr world. *Modern Drug Discovery*, 7(11):24–28.

Flores, M. J., Gámez, J. A., Martiinez, A. M., and Puerta, J. M. (2011). Handling numeric attributes when comparing bayesian network classifiers: does the discretization method matter? *Applied Intelligence*, 34(3):372–385.

Flores, M. J., Gámez, J. A., Martínez, A. M., and Puerta, J. M. (2010). Analyzing the impact of the discretization method when comparing bayesian classifiers. In *Proc. of the 23rd Int. Conf. on Industrial Engineering and Other Applications of Applied Intelligent Systems*, volume 6096 of *LNCS*, pages 570–579. Springer.

Freitas, A. A. and de Carvalho, A. C. P. L. F. (2007). *Research and Trends in Data Mining Technologies and Applications*, chapter A Tutorial on Hierarchical Classification with Applications in Bioinformatics, pages 175–208. Idea Group.

Freitas, A. A., Wieser, D. C., and Apweiler, R. (2010). On the importance of comprehensible classification models for protein function prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):172–182.

Freitas, C. O. A., Oliveira, L. S., Aires, S. B. K., and Bortolozzi, F. (2008). Metaclasses and zoning mechanism applied to handwriting recognition. *Journal of Universal Computer Science*, 14(2):211–223.

Friedberg, I. (2006). Automated protein function prediction – the genomic challenge. *Briefings in Bioinformatics*, 7(3):225–242.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29:131–163.

Furnkranz, J. and Sima, J. F. (2010). On exploiting hierarchical label structure with pairwise classifiers. *SIGKDD Explorations*, 12(2):21–25.

García, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

Gauch, S., Chandramouli, A., and Ranganathan, S. (2009). Training a hierarchical classifier using inter document relationships. *Journal of the American Society for Information Science and Technology*, 60(1):47–58.

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

Ghazi, D., Inkpen, D., and Szpakowicz, S. (2010). Hierarchical versus flat classification of emotions in text. In *Proc. of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 140–146.

Gibas, C. J. and Jambeck, P. (2001). *Developing Bioinformatics Computer Skills*. O'Reilly.

Guan, Y., Myers, C. L., Hess, D. C., Barutcuoglu, Z., Caudy, A. A., and Troyanskaya, O. G. (2008). Predicting gene function in a hierarchical context with an ensemble of classifiers. *Genome Biology 2008*, 9(Suppl 1: S3).

Gyftodimos, E. and Flach, P. A. (2004). Hierarchical bayesian networks: A probabilistic reasoning model for structured domains. In *Proc. of Methods and Applications of Artificial Intelligence, Third Hellenic Conf. on AI*, pages 291–300.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Hao, P.-Y., Chiang, J.-H., and Tu, Y.-K. (2007). Hierarchically SVM classification based on support vector clustering method and its application to document categorization. *Expert Systems with Applications*, 33:627–635.

Hayete, B. and Bienkowska, J. (2005). Gotrees: Predicting go associations from protein domain composition using decision trees. In *Proc. of the Pacific Symp. on Biocomputing*, pages 127–138.

He, J., Hu, H.-J., Harrison, R., Tai, P. C., and Pan, Y. (2006). Transmembrane segments prediction and understanding using support vector machine and decision tree. *Expert Systems with Applications*, 30:64–72.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning bayesian networks: the combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243.

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

Hobohm, U. and Sander, C. (1995). A sequence property approach to searching protein databases. *Journal of Molecular Biology*, 251:390–399.

Holden, N. and Freitas, A. A. (2005). A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In *Proc. of the 2nd IEEE Swarm Intelligence Symposium*, pages 100–107.

Holden, N. and Freitas, A. A. (2006). Hierarchical classification of g-protein-coupled receptors with a pso/aco algorithm. In *Proc. of the 3rd IEEE Swarm Intelligence Symposium*, pages 77–84.

Holden, N. and Freitas, A. A. (2008). Improving the performance of hierarchical classification with swarm intelligence. In *Proc. 6th European Conf. on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio)*, volume 4973 of *Lecture Notes in Computer Science*, pages 48–60. Springer.

Holden, N. and Freitas, A. A. (2009). Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Computing Journal*, 13:259–272.

Hulo, N., A., A. B., Bulliard, V., Cerutti, L., De Castro, E., Langendijk-Genevaux, P., M., P., and Sigrist, C. (2006). The prosite database. *Nucleic Acids Research*, 34:D227–D230.

Hunter, S., Apweiler, R., Attwood, T. K., Bairoch, A., Bateman, A., Binns, D., Bork, P., Das, U., Daugherty, L., Duquenne, L., Finn, R. D., Gough, J., Haft, D., Hulo, N., Kahn, D., Kelly, E., Laugraud, A., Letunic, I., Lonsdale, D., Lopez, R., Madera, M., Maslen, J., McAnulla, C., McDowall, J., Mistry, J., Mitchell, A., Mulder, N., Natale, D., Orengo, C., Quinn, A. F., Selengut, J. D., Sigrist, C. J. A., Thimma, M., Thomas, P. D., Valentin, F., Wilson, D., Wu, C. H., and Yeats, C. (2009). Interpro: the integrative protein signature database. *Nucleic Acids Research*, 37:D211–D215. Database issue.

Jensen, L. J., Gupta, R., Staerfeldt, H.-H., and S., B. (2003). Prediction of human protein function according to gene ontology categories. *Bioinformatics*, 19(5):635–642.

Jiang, L., Wang, D., Cai, Z., and Yan, X. (2007). Survey of improving naive bayes for classification. In *Proc. of the 3rd Int. Conf. on Advanced Data Mining and Applications*, volume 4632 of *Lecture Notes in Computer Science*, pages 134–145. Springer.

Jin, B., Muller, B., Zhai, C., and Lu, X. (2008). Multi-label literature classification based on the gene ontology graph. *BMC Bioinformatics*, 9(525).

Keogh, E. J. and Pazzani, M. J. (1999). Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the Seventh International Workshop on AI and Statistics*.

Keogh, E. J. and Pazzani, M. J. (2002). Learning the structure of augmented bayesian classifiers. *International Journal of Artificial Intelligence Tools*, 11(4):587–601.

Keshtkar, F. and Inkpen, D. (2009). Using sentiment orientation features for mood classification in blogs. In *Proc. of the IEEE Int. Conf. on Natural Language Processing and Knowledge Engineering*.

Keshtkar, F. and Inkpen, D. (2011). A hierarchical approach to mood classification in blogs. *Natural Language Engineering*, To Appear.

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

Kiritchenko, S., Matwin, S., and Famili, A. F. (2005). Functional annotation of genes using hierarchical text categorization. In *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*.

Kiritchenko, S., Matwin, S., Nock, R., and Famili, A. F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Proc. of the 19th Canadian Conf. on Artificial Intelligence*, volume 4013 of *Lecture Notes in Artificial Intelligence*, pages 395–406.

Koerich, A. L. and Kalva, P. R. (2005). Unconstrained handwritten character recognition using metaclasses of characters. In *Proc. of the IEEE Int. Conf. on Image Processing*, volume 2, pages 542–545.

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proc. of the 14th Int. Conf. on Machine Learning*, pages 170–178.

Kong, W., Tan, T. S., Tham, L., and Choo, K. W. (2007). Improved prediction of allergenicity by combination of multiple sequence motifs. *Silico Biology*, 7(1):77–86.

Kriegel, H.-P., Kroger, P., Pryakhin, A., and Schubert, M. (2004). Using support vector machines for classifying large sets of multi-represented objects. In *Proc. of the SIAM Int. Conf. on Data Mining*, pages 102–114.

Kumar, S., Ghosh, J., and Crawford, M. M. (2002). Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis & Applications*, 5:210–220.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers*. Wiley-Interscience.

Labrou, Y. and Finin, T. (1999). Yahoo! as an ontology – using yahoo! categories to describe documents. In *Proc. of the ACM Conf. on Information and Knowledge Management*, pages 180–187.

Lam, W. and Bacchus, F. (1994). Learning bayesian belief networks: An approach based on the mdl principle. *Computational Intelligence*, 10:269–293.

Lapinsh, M., Prusis, P., Lundstedt, T., and Wikberg, J. E. S. (2002). Proteochemometrics modeling of the interaction of amine g-protein coupled receptors with a diverse set of ligands. *Molecular Pharmacology*, 61(6):1465–1475.

Lee, J. H. and Downie, J. S. (2004). Survey of music information needs, uses, and seeking behaviours: preliminary findings. In *Proc. of the 5th Int. Conf. on Music Information Retrieval*, pages 441–446.

Lesk, A. M. (2002). *Introduction to Bioinformatics*. Oxford University Press.

Li, T. and Ogihara, M. (2005). Music genre classification with taxonomy. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 197–200.

Li, T., Zhu, S., and Ogihara, M. (2007). Hierarchical document classification using automatically generated hierarchy. *Journal of Intelligent Information Systems*, 29(2):211–230.

Liu, H. and Motoda, H., editors (2007). *Computational Methods of Feature Selection*. Chapman & Hall.

Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. (2005). Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1):36–43.

Lorena, A. C. and Carvalho, A. C. P. L. F. (2004). Comparing techniques for multiclass classification using binary svm predictors. In *Proc. of the IV Mexican Int. Conf. on Artificial Intelligence*, volume 2972 of *Lecture Notes in Artificial Intelligence*, pages 272–281.

Madden, M. G. (2009). On the classification performance of tan and general bayesian networks. *Knowledge-Based Systems*, 22(7):489–495.

Madprime (2007 (accessed December 5, 2007)). *Diagram of the central dogma, DNA to RNA to protein, illustrating the genetic code*. Wikipedia. `http://en.wikipedia.org/wiki/Image:Genetic-code.svg`.

McCallum, A., Rosenfeld, R., Mitchell, T. M., and Ng, A. Y. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of the Int. Conf. on Machine Learning*, pages 359–367.

McKay, C. and Fujinaga, I. (2004). Automatic genre classification using large high-level musical feature sets. In *Proc. of the Int. Conf. on Music Information Retrieval*, pages 525–530.

Mladenic, D. and Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Decision Support Systems*, 35:45–87.

Mulder, N. J., Apweiler, R., Attwood, T. K., Bairoch, A., Bateman, A., Binns, D., Bork, P., Buillard, V., Cerutti, L., Copley, R., Courcelle, E., Das, U., Daugherty, L., Dibley, M., Finn, R., Fleischmann, W., Gough, J., Haft, D., Hulo, N., Hunter, S., Kahn, D., Kanapin, E., Kejariwal, A., Labarga, A., Langendijk-genevaux, P. S., Lonsdale, D., Lopez, R., Letunic, I., Madera, M., Maslen, J., Mcanulla, C., Mcdowall, J., Mistry, J., Mitchell, A., Nikolskaya, A. N., Orchard, R., Orengo, C., Petryszak, R., Selengut, J. D., Sigrist, C. J. A., Thomas, P. D., Valentin, F., Wilson, D., Wu, C. H., and Yeats, C. (2007). New developments in the interpro database. *Nucleic Acids Research*, 35:D224–D228. Database Issue.

Nariai, N., Kolaczyk, E. D., and Kasif, S. (2007). Probabilistic protein function prediction from heterogeneous genome-wide data. *PLoS ONE*, 2(3):e377.

Otero, F. E. B., Freitas, A. A., and Johnson, C. G. (2009). A hierarchical classification ant colony algorithm for predicting gene ontology terms. In Pizzuti, C., Ritchie, M., and Giacobini, M., editors, *Proc. of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio)*, volume 5483 of *Lecture Notes in Computer Science*, pages 68–79. Springer.

Pandey, G., Kumar, V., and Steinbach, M. (2007). Computational approaches for protein function prediction: A survey. Technical Report 4, Digital Technology Center – University of Minnesota.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference*. Morgan Kaufmann.

Peng, X. and Choi, B. (2005). Document classifications based on word semantic hierarchies. In *Proc. of the Int. Conf. on Artificial Intelligence and Applications*, pages 362–367.

Pernkopf, F. (2005). Bayesian network classifiers versus selective k-nn classifier. *Pattern Recognition*, 38:1–10.

Pernkopf, F. and Bilmes, J. A. (2010). Efficient heuristics for discriminative structure learning of bayesian network classifiers. *Journal of Machine Learning Research*, 11:2323–2360.

Pernkopf, F. and O'Leary, P. (2003). Floating search algorithm for structure learning of bayesian network classifiers. *Pattern Recognition Letters*, 24:28392848.

Platt, J. (1998). *Advances in Kernel Methods – Support Vector Learning*, chapter Fast Training of Support Vector Machines using Sequential Minimal Optimization, pages 185–208. MIT Press.

Punera, K. and Ghosh, J. (2008). Enhanced hierarchical classification via isotonic smoothing. In *Proc. of the 17th Int. Conf. on World Wide Web*, pages 151–160.

Punera, K., Rajan, S., and Ghosh, J. (2005). Automatically learning document taxonomies for hierarchical classification. In *Proc. of the Int. World Wide Web Conference*, pages 1010 –1011.

Qiu, X., Gao, W., and Huang, X. (2009). Hierarchical multi-class text categorization with global margin maximization. In *Proc. of the Joint Conf. of the 47th Annual Meeting of the ACL and the 4th Int. Joint Conf. on Natural Language Processing of the AFNLP*, pages 165–168. Association for Computational Linguistics.

Rocchio, J. J. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter Relevance feedback in information retrieval, pages 313–323. Prentice Hall.

Rosenberg, M. S., editor (2009). *Sequence Alignment: Methods, Models, Concenpts and Strategies*. University of California Press.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. (2005). Learning hierarchical multi-category text classification models. In *Proc. of the 22nd Int. Conf. on Machine Learning*, pages 744–751.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. (2006). Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626.

Rubio, A. and Gamez, J. A. (2011). Flexible learning of k-dependence bayesian network classifiers. In *Proc. of the 13th Annual Conf. on Genetic and Evolutionary Computation*, pages 1219–1226.

Ruepp, A., Zollner, A., Maier, D., Albermann, K., Hani, J., Mokrejs, M., Tetko, I., Guldener, U., Mannhaupt, G., Munsterkotter, M., and Mewes, H. W. (2004). The funcat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acids Research*, 32(18):5539–5545.

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

Sahami, M. (1996). Learning limited dependence bayesian classifiers. In *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 335–338. AAAI Press.

Sandberg, M., Eriksson, L., Jonsson, J., Sjostrom, M., and Wold, S. (1998). New chemical descriptors relevant for the design of biologically active peptides. A multivariate characterization of 87 amino acids. *Journal of Medical Chemistry*, 41:2481–2491.

Santos, E. B. D., Hruschka Jr., E. R., Hruschka, E. R., and Ebecken, N. F. F. (2011). Bayesian network classifiers: Beyond classification accuracy. *Intelligent Data Analysis*, 15:279–298.

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

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6:461–464.

Sebastiani, P., Abad, M. M., and Ramoni, M. F. (2010). *Data Mining and Knowledge Discovery Handbook*, chapter Bayesian Networks, pages 175–208. Springer, 2nd edition.

Secker, A., Davies, M., Freitas, A., Timmis, J., Mendao, M., and Flower, D. (2007). An experimental comparison of classification algorithms for the hierarchical prediction of protein function. *Expert Update (the BCS-SGAI Magazine)*, 9(3):17–22.

Secker, A., Davies, M., Freitas, A. A., Clark, E., Timmis, J., and Flower, D. R. (2010). Hierarchical classification of g-protein-coupled-receptors with data-driven selection of attributes and classifiers. *International Journal of Data Mining and Bioinformatics*, 4(2):191–210.

Seeger, M. W. (2008). Cross-validation optimization for large scale structured classification kernel methods. *The Journal of Machine Learning Research*, 9:1147–1178.

Shilane, P., Kazhdan, M., Min, P., and Funkhouser, T. (2004). The princeton shape benchmark. In *Proc. of the Shape Modeling International.*

Silla Jr., C. N. and Freitas, A. A. (2009a). A global-model naive bayes approach to the hierarchical prediction of protein functions. In *Proc. of the 9th IEEE Int. Conf. on Data Mining*, pages 992–997.

Silla Jr., C. N. and Freitas, A. A. (2009b). Novel top-down approaches for hierarchical classification and their application to automatic music genre classification. In *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 3599–3604.

Silla Jr., C. N. and Freitas, A. A. (2011a). Selecting different protein representations and classification algorithms in hierarchical protein function prediction. *Intelligent Data Analysis Journal*, 15(6):979–999.

Silla Jr., C. N. and Freitas, A. A. (2011b). A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1–2):31–72.

Sleator, R. D. and Walsh, P. (2010). An overview of in silico protein function prediction. *Archives of Microbiology*, 192:151–155.

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45:427–437.

Sun, A. and Lim, E.-P. (2001). Hierarchical text classification and evaluation. In *Proc. of the IEEE Int. Conf. on Data Mining*, pages 521–528.

Sun, A., Lim, E.-P., and Ng, W.-K. (2003). 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., and Srivastava, J. (2004). Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1305–1308.

Syed, U. and Yona, G. (2003). Using a mixture of probabilistic decision trees for direct prediction of protein function. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 289–300.

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

Tikk, D. and Biró, G. (2003). Experiment with a hierarchical text categorization method on the wipo-alpha patent collection. In *Proc. of the 4th Int. Symp. on Uncertainty Modeling and Analysis*, pages 104–109.

Tikk, D., Biró, G., and Torcsvári, A. (2007). *Emerging Technologies of Text Mining: Techniques and Applications*, chapter A hierarchical online classifier for patent categorization, pages 244–267. Idea Group.

Tikk, D., Biró, G., and Yang, J. D. (2004). A hierarchical text categorization approach and its application to frt expansion. *Australian Journal of Intelligent Information Processing Systems*, 8(3):123–131.

Tikk, D., Yang, J. D., and Bang, S. L. (2003). Hierarchical text categorization using fuzzy relational thesaurus. *Kybernetika*, 39(5):583–600.

Tong, J. C. and Tammi, M. T. (2008). Prediction of protein allergenicity using local description of amino acid sequence. *Frontiers in Bioscience*, 13:6072–6078.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.

Tsoumakas, G. and Katakis, I. (2007). Multi label classification: An overview. *International Journal of Data Warehouse and Mining*, 3(3):1–13.

Valentini, G. (2009). True path rule hierarchical ensembles. In Kittler, J., Benediktsson, J., and Roli, F., editors, *Proc. of the Eighth Int. Workshop on Multiple Classifier Systems*, volume 5519 of *Lecture Notes in Computer Science*, pages 232–241. Springer.

Valentini, G. and Re, M. (2009). Weighted true path rule: a multilabel hierarchical algorithm for gene function prediction,. In *Proc. of the 1st Workshop on Learning from Multi-Label Data (MLD) held in conjunction with ECML/P-KDD*, pages 132–145.

Vens, C., Struyf, J., Schietgat, L., Dzeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214.

Wang, J., Shen, X., and Pan, W. (2009). On large margin hierarchical classification with multiple paths. *Journal of American Statistical Association*, 104(487):1213–1223.

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

Wang, K., Zhou, S., and Liew, S. C. (1999). Building hierarchical classifiers using class proximity. In *In Proc. of the 25th Conf. on Very Large Data Base*, pages 363–374. Morgan Kaufmann Publishers.

Webb, G. I., BOUGHTON, J. R., and WANG, Z. (2005). Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, 58:5–24.

Weigend, A. S., Wiener, E. D., and Pedersen, J. O. (1999). Exploiting hierarchy in text categorization. *Information Retrieval*, 1:193–216.

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

Williamson, J. and Gabbay, D. (2005). *Laws and models in science*, chapter Recursive Causality in Bayesian Networks and Self-Fibring Networks, pages 173–221. With comments, pp. 223–245. King's College Publications, London.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition.

Wu, F., Zhang, J., and Honavar, V. (2005). Learning classifiers using hierarchically structured class taxonomies. In *Proc. of the Symp. on Abstraction, Reformulation, and Approximation*, volume 3607, 313-320. Springer.

Xiao, Z., Dellandréa, E., Dou, W., and Chen, L. (2007). Hierarchical Classification of Emotional Speech. Technical Report RR-LIRIS-2007-006, LIRIS UMR 5205 CNRS/INSA de Lyon/Universit Claude Bernard Lyon 1/Universit Lumire Lyon 2/Ecole Centrale de Lyon.

Xiao, Z., Dellandrea, E., Dou, W., and Chen, L. (2011). Classification of emotional speech based on an automatically elaborated hierarchical classifier. *ISRN Signal Processing*, 2011:15.

Xue, G.-R., Xing, D., Yang, Q., and Yu, Y. (2008). Deep classification in large-scale text hierarchies. In *Proc. of the 31st annual int. ACM SIGIR conf. on Research and development in information retrieval*, pages 619–626.

Yang, Y., Korb, K., Ting, K., and Webb, G. (2005). Ensemble selection for superparent-one-dependence estimators. In *Proc. of the 18th Autralian Joint Conf. on Artificial Intelligence*, volume 3809 of *LNCS*, pages 102–112.

Yang, Y. and Webb, G. I. (2009). Discretization for naive-bayes learning: Managing discretization bias and variance. *Machine Learning*, 74(1):39–74.

Yang, Y., Webb, G. I., Cerquides, J., Korb, K. B., Boughton, J., and Ting, K. M. (2007). To select or to weigh: A comparative study of linear combination schemes for superparent-one-dependence estimators. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1652–1665.

Zhang, H. and Ling, C. X. (2001). An improved learning algorithm for augmented naive bayes. In *Proc. of the 5th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, volume 2035 of *LNCS*, pages 581–586. Springer.

Zhang, T. (2003). Semi-automatic approach for music classification. In *Proc. of the SPIE Conf. on Internet Multimedia Management Systems*, pages 81–91.

Zhang, Z., Kochhar, S., and Grigorov, M. G. (2005). Descriptor-based protein remote homology identification. *Protein Science*, 14:431–444.

Zhao, X. M., Chen, L., and Aihara, K. (2008). Protein function prediction with high-throughput data. *Amino Acids*, 35(3):517–530.

Zheng, F. and Webb, G. I. (2007). Finding the right family: Parent and child selection for averaged one-dependence estimators. In *Proc. of the 18th European Conf. on Machine Learning*, volume 4701 of *LNCS*, pages 490–501.

Zimek, A., Buchwald, F., Frank, E., and Kramer, S. (2010). A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):563–571.