2012-04-21

Hierarchical Multi-Label Classification: Going Beyond Generalization Trees

Peerapon Vateekul
University of Miami, peerapon.vateekul@gmail.com

UNIVERSITY OF MIAMI

HIERARCHICAL MULTI-LABEL CLASSIFICATION: GOING BEYOND GENERALIZATION TREES

By

Peerapon Vateekul

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

May 2012

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

HIERARCHICAL MULTI-LABEL CLASSIFICATION: GOING BEYOND GENERALIZATION TREES

Peerapon Vateekul

Approved:

VATEEKUL, PEERAPON (Ph.D., Electrical and Computer Engineering)

Hierarchical Multi-Label Classification: Going Beyond Generalization Trees (May 2012)

Abstract of a dissertation at the University of Miami.

Dissertation supervised by Professor Miroslav Kubat.
No. of pages in text. (183)

Traditional computational approach to automated classification assumes that each object should be assigned to only one out of two or more *classes*. However, some real-world applications digress from this generic scenario in two important ways. First, each example can belong to several classes simultaneously (multi-label classification). Second, the classes can be hierarchically ordered in the sense that some are more specific versions of others (hierarchical classification). Seeking to address both of these issues, the presented work deals with "*hierarchical multi-label classification.*"

The task has recently received considerable attention; databases in various fields, including web repositories, digital libraries, or genomics, are known to be organized as hierarchies. Seeking to start with something relatively simple, scientists have focused on the special case where the inter-class relations are captured by tree-structured hierarchy. This, however, is not enough. Very often, some classes have more than one parent, in which case the mutual relations (if they are known) have to be described by a hierarchy structured as a directed acyclic graph (DAG). This dissertation intends to contribute to this more general problem.

Literature survey indicates that, in *non*-hierarchical multi-label classification, good performance is achieved when a Support Vector Machine (SVM) is used to induce each class separately. This said, some experiments suggest that further improvement can be achieved by explicitly dealing with the problem of *imbalanced training sets* because, in most classes, negative examples heavily outnumber positive ones. The author proposes a solution in terms of a technique referred to as *R*-SVM; the idea is to re-adjust the SVM-hyperplane offset accordingly. Experiments in the first part of this dissertation rely on data from domains of text-categorization.

More important, however, is then the second part that focuses on hierarchical multi-label classification. Here, the author proposes a new technique, *HR*-SVM, that essentially constitutes a hierarchical extension of *R*-SVM proceeding in a top-down fashion from more general to more specific classifiers. The weakness of this approach is known as "*error propagation*": examples misclassified at higher levels are propagated down the hierarchy, thus resulting in negative performances at the lower levels. *HR*-SVM contains a mechanism to correct this kind of errors. The system has been subjected to extensive experiments with many domains from the field of gene function prediction. The results show that the new technique compares favorably with other existing approaches along various performance criteria.

To my beloved parents

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this dissertation. First, thanks in large part to the kindness and considerable mentoring provided by Professor Miroslav Kubat, my dissertation advisor. His help, advice, and encouragement guided the way for this long journey. He has absolutely been a pleasure to work with. I want to offer my heart felt thanks to committee members, Professor Kamal Premaratne, Professor Akmal A. Younis, Professor Nigel M. John, and Professor Maria M. Llabre who gave insightful comments and invaluable suggestions to improve this research.

This work would not have been possible without the financial support of Thai Government Scholarship and Behavioral Medicine Research Center (BMRC) at the Department of Psychology. I am especially indebted to Dr. Nunta Vanicsetakul and Dr. Kanoksri Sarinnapakorn, for their invaluable assistance and advice especially in statistical methods in the dissertation. I am very grateful to Dr. Feng Zhao, my supervisor at the Behavioral Medicine Research Center (BMRC).

Professor Mei-Ling Shyu also deserves a great deal of thanks. I thank my colleagues at BMRC and friends at UM Thai Student Organization, the Department of Electrical and Computer Engineering, and the Department of Psychology for their support and encouragement. Particular thanks go to Decho Surangsrirat for his helpful research discussions. Sareewan Dendamrongvit, my best friend, deserves many thanks for her help and support during these past years. In addition, I thank all my beloved and best friends in my life for their encouragement and great friendship.

Nobody has been more important to me in the pursuit of this project than the members of my family. I would like to express my utmost gratitude to my parents, Niphon and Suporn, whose love and guidance are with me in whatever I pursue. Their unlimited support made everything possible to help me make it this far.

Peerapon Vateekul

University of Miami

May 2012

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Access to electronically stored data can be improved by automated categorization, supported by an appropriate indexing scheme. However, manual creation of such a scheme is impractical due to the size of data (e.g., millions of documents in digital repositories) and the costs of human experts (e.g., in gene function annotation). This is why some machine-learning scientists have explored possibilities of automating the process. The idea is to manually classify only a subset of examples, and then induce a mechanism to be used to automatically annotate (classify) the rest.

This dissertation explores techniques for "hierarchical multi-label classification." Two issues have to be considered. First, each instance can belong to more than one class at the same time, and the classifier should correctly identify as many of them as possible without labeling the document with non-relevant ones (*multi-label classification*). Second, the categories are not independent of each other; their mutual relations are expressed by generalization trees or directed acyclic graphs (*hierarchical classification*).

Multi-label classification is encountered in many areas, for instance, a medical patient may suffer from more than one health condition: diabetes, high blood pressure, high cholesterol. Likewise, a book can be categorized into multiple categories: engineering, technology, science, computers. A movie can straddle several film genres: science fiction, horror, action, and adventure. Scene analysis may result in several semantic classes: beach, ocean, sky, sunset, and mountain. This dissertation was originally motivated by data in domains of *text categorization*.

The problem of hierarchical classification has received less attention, and yet it has many important applications, too. Database collections are often organized as hierarchies as typically seen in web-based repositories, digital libraries, patent libraries, email folders, even in product catalogs. The particular application targeted by this dissertation is *gene function prediction*: given an unknown gene (example), the task is to classify its functions (classes) referring to a predefined function hierarchy, such as Gene Ontology (GO). For instance, a gene product "Actin, alpha cardiac muscle 1" is labeled with the "heart contraction" function, but the same gene also belongs to all ancestor functions of the heart contraction function, including blood circulation, heart process, circulator system process, system process, etc. This is what is referred to as "*the hierarchy constraint*".

1.1 Motivation

1.1.1 Multi-label Classification

Perhaps the oldest application of multi-label classification is *text categorization* where the task is to label text documents with predefined categories. The work summarized in this dissertation started with EUROVOC[1], a large multilingual thesaurus contain-

[1]`http://europa.eu/eurovoc/` and `http://langtech.jrc.it/Eurovoc.html`

ing tens of thousands of documents, each described by about one hundred thousand features. Sarinnapakorn and Kubat (2007) reported that 90% of these documents are labeled with 2 to 5 classes from the total of 30 classes (at the highest hierarchical level). An important problem when dealing with data files of this size is the possibly prohibitive computational costs.

In order to deal with this domain, some researchers have developed mechanisms for induction of multi-label classifiers (Chang and Lin 2011), while others preferred to induce a separate binary classifier for each class (Joachims 1998; Kwok 1998)[2]. The success of the latter approach is often impaired by the phenomenon known as "imbalanced training sets"—positive examples outnumbering the negative ones or the other way round. Traditional machine learning techniques are known to operate poorly in domains of this kind.

The dissertation seeks to address both of these problems: the overwhelming size of the data, and the imbalanced representation of classes in them.

1.1.2 Hierarchical Classification

When the number of classes is large, it is often helpful to organize them in groups and subgroups. For instance, large collections of web pages can be organized in conceptual hierarchies such as the one from Figure 1.1. From an information retrieval viewpoint, this hierarchical arrangement is essential because such organization makes it possible for the user to focus on the most appropriate level of detail.

[2]More details of related work in this kind of domains are described in Section 2.4.

Only a few papers have so far dealt with this topic, most of them constraining their work by certain limitations[3]. For example, some papers (Jensen et al. 2002; Riley 1993; Weinert and Lopes 2004; Berman et al. 2000) ignore the hierarchical relationship among classes, and simply transform the hierarchical problem to single-level classification. Others (Sun and Lim 2001; Koller and Sahami 1997) rely on a tree hierarchy where each class has at maximum one parent (Figure 1.1). However, in many real-world databases, some classes have more than one parent; in that case, the inter-class relations can only be captured by *a directed acyclic graph (DAG)* (Figure 1.2). Domains characterized by DAG-structured hierarchy constitute a still under-explored research area.

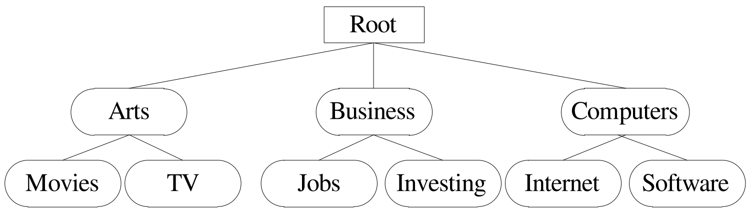


Figure 1.1: Parts of top levels of the open web directory project (ODP) whose hierarchical structure is a tree.

In summary, hierarchical classification can help increase interpretability and accessibility of databases, and the number of real-world domains benefitting from is growing. This said, it is surprising how little has been done in terms of relative research especially when the hierarchy structure is DAG (Table 4 in Silla and Freitas (2010)). This observation motivates this dissertation. From the perspective of application domain, the dissertation will focus on "*gene function prediction*," an important task in the field of bioinformatics.

[3]More information about related work is provided by Section 3.3.

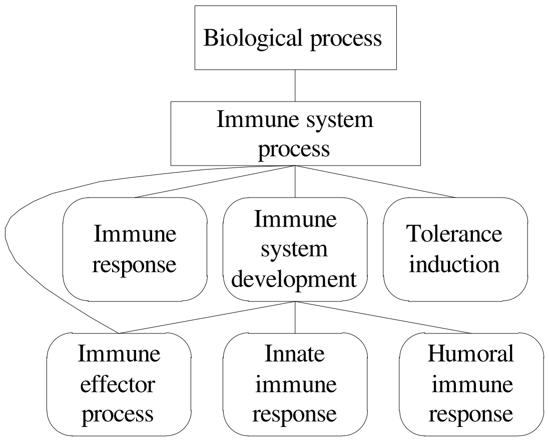


Figure 1.2: Parts of top levels of the immune system processes in Gene Ontology (GO) whose hierarchical structure is a directed acyclic graph (DAG).

Motivation of Gene Function Prediction

The problem of functional annotation in genes is of great importance for the biomedical and bioinformatics communities (Stein 2001). Although the completion of numerous whole-genome sequences have been studied and provided in the past decade, their values are only as good as their annotation. Genomics research, therefore, aims to study behavior (function) of genes in various organisms under different conditions, leading us to an understanding of the underlying mechanisms of diseases and being able to provide indications for the development of target efficient treatment.

However, the lab processes to exploit the functions of genes, such as microarray expression analysis, RNA interference, etc., are very expensive and time-consuming since they involve many manual experiments by experts. Thus, the methods of automated gene function prediction provide hypothetical annotations, which can drive the biological validation and discovery of novel functions of genes and gene product, and, thus, reduce the experimental burden. Figure 1.3[4] illustrates the process of gene function prediction. The functional classes are referred to Gene Ontology (GO) (Ashburner et al. 2000).

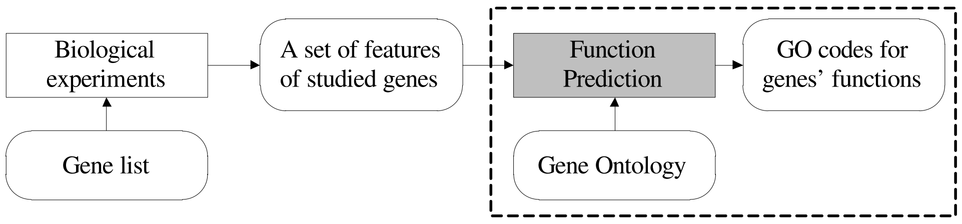

Biological experiments
A set of features of studied genes
Function Prediction
GO codes for genes' functions
Gene list
Gene Ontology

Figure 1.3: Function annotation process. Genes' functions are determined in biological experiments. The goal of the dashed box is to predict these functions represented by GO codes.

In gene function prediction, the number of functional classes is large, and a gene may belong to many classes at the same time; functional classes are structured according to *a hierarchy*; classes are usually unbalanced, with more negative than positive examples; class labels can be uncertain and the annotations largely incomplete.

GO is a hierarchy designed to cover genomes. Current genomes for which GO annotations are available for some organisms such as *yeasts S. cerevisiae and S. pombe, the plant Arabidopsis thaliana, mouse M. musculus, worm C. elegans*, etc. The scheme is a major step towards unifying results from multiple genomes and classifying gene function on a large scale. Its three-way annotation of molecular function, cellular component, and biological process more accurate reflects the current understanding of the different types of function a gene can have. It allows many-to-many relationship on gene-to-function (multi-label) and function-to-function (the DAG structured hierarchy).

[4]The figure is originally from Figure 7.2 of Kiritchenko et al. (2005).

Even with all these different function annotation schemes, there is still the question of the suitability of the current functional classes for functional genomics. Kell and King (2000) conclude that “current lists of functional classes are not driven by data from whole-organisms studies and are *suboptimal* for the purposes of functional genomics,” and recommend that future classification be data driven, and that “inductive methods of machine learning provide the best initial approaches to assign gene functions.”

Recently, there has been some research on hierarchical classification, but most of it has focused on text mining (a tree-structured class hierarchy), and not on bioinformatics (a DAG-structured class hierarchy). In this sense, the hierarchical framework (especially with the DAG structure) is still an under-explored research area.

1.2 Challenges and Research Objective

The objective of this dissertation is the development of a hierarchical multi-label classification framework that achieves promising classification performance. In order to accomplish the objective, the work in this dissertation is divided into two main tasks.

The first task focuses in domains of multi-label classification especially for the needs of a large scale text corpus. This poses the following challenges:

- The number of examples is extremely high.
- The number of feature is enormous. These features represent all classes of the whole data collection. Thus, many of them are not relevant to a specific class.

- The data space is very sparse. For each example, values of most features are equal to 0.
- Multi-label examples are often highly imbalanced; some of the classes are heavily populated while some of the classes only contain a few examples.

These issues result in two serious problems: (*i*) prohibitive computation cost and (*ii*) low predictive power. Thus, the objective of this task is to develop a technique that performs well in real-world domains while being computationally efficient. From the machine learning techniques usually used in similar domains, the Support Vector Machine (SVM) is chosen because it has been shown to to work remarkably well in text categorization (Joachims 1998). The dissertation will describe an enhanced-SVM framework addressing all the above issues in an efficient and effective way.

The second task is to extend the developed multi-label classification to support induction of hierarchical classification, especially in the field of gene function prediction. The following challenges are addressed:

- The prediction of an unknown instance must follow *the hierarchy constraint*.
- The class hierarchies are no longer constrained to trees, but can be DAG.
- An example does not need to belong to classes at the leaf nodes. Thus, the method can consider stopping the classification at any nodes in any levels of the hierarchy, which is called "*non-mandatory leaf node prediction.*"
- The number of classes in domains of hierarchical classification is usually enormous — even larger than the number of classes in domains of multi-label clas-

sification. This means that the imbalance of training data set is extremely high.

There are many approaches suggested in the field of hierarchical classification problem as summarized in Section 3.2. One of them is a top-down approach (Koller and Sahami 1997) by separately generating a binary classifier for each class and proceeding in a top-down fashion. It has many advantages in terms of simplicity, efficiency, and suitability to the problem of our interest, the gene function prediction. Moreover, the baseline classifier for each class can be adopted from our method proposed in the first part of this dissertation. However, the weakness of the top-down approach is known as "*error propagation*" where misclassified examples at the upper level are propagated downwards the hierarchy and directly affects performance of classifiers at the bottom level, and also the overall system. This error must be addressed in the system developed in this part of the dissertation.

For evaluation purposes, the well-known measures of flat classification, such as *precision*, *recall*, and F_1, appear to ignore the hierarchical relationship among classes. Although there have been some alternatives of hierarchical classification measures proposed, none of them are accepted as a standard measure. Thus, another objective of this dissertation in hierarchical classification is to propose a performance measure that can fully address the hierarchical nature.

1.3 Summary of Contributions

For the first part of this dissertation in multi-label classification, all challenges and objectives have been met. There are two classification frameworks contributed to this domain. The first developed system is called "fast decision-tree induction (FDT)" (Vateekul and Kubat 2009). Our experiments showed that FDT performed better than traditional classifiers, such as C4.5 and SVM. Later our preliminary experiments showed that the performance of SVM can be significantly improved by applying the threshold adjustment strategy at slight additional computation costs. This inspired us to study this approach and to propose a new thresholding method called "R-SVM" (Vateekul et al. 2011) in the second framework in the domain of multi-label classification.

The hierarchical extended version of R-SVM based on the top-down approach called "HR-SVM" (H standing for $\underline{\text{h}}$ierarchical) (Vateekul et al. 2012) is presented in the second part of the dissertation. It avoids the *propagated errors* from classifiers at upper levels by fully utilizing the hierarchical information to generate a smaller and less-imbalanced training data set with misclassified data from parent classifiers being added in, so that errors can be learnt and corrected at the current classifier. To evaluate the results, the dissertation introduces a new hierarchical classification measure called "example-label based macro-averaging measure" which is an extended version of the well-known flat classification metrics: *precision*, *recall*, and F_1.

Furthermore, some extra works, which are not directly related to the main task of the dissertation, have also been established and published. First, a new imputation method, "Imputation Tree (ITree)" (Vateekul and Sarinnapakorn 2009), has been proposed to handle data with missing values. Second, the feature selection of FDT is continuous applied to the classifier proposed by Dendamrongvit et al. (2011).

1.4 Organization of the Dissertation

The organization of this dissertation is as follows. Chapter 2 briefly explains an overview of multi-label classification composing the problem statements, general approach, related work, and performance criteria. Chapter 3 summarizes the concept of hierarchical classification in the same way as in the previous chapter. Chapter 4 presents the work in domains of multi-label classification, FDT and *R*-SVM. Chapter 5 discusses the developed hierarchical classification method called *HR*-SVM and introduces the new hierarchical performance measure. Lastly, Chapter 6 concludes the dissertation and discusses possible directions for future research work.

CHAPTER 2

Multi-label Classification

Multi-label classification – where the same example can belong to two or more classes at the same time, is encountered in various fields, including `text`, `biology`, `music`, `image`, and `video`. For example, in text categorization, a text document may belong to many subjects or topics; in biology, one protein may have many effects on a cell; in music categorization, a piece of music shares many musical genres. Due to the needs in the broad domains of these applications, the multi-label classification tasks in these areas are important and have obtained a plethora of research recently.

This chapter gives a definition and summary of work related to multi-label classification. It starts with a formal statement of a multi-label classification problem and some relevant basic definitions. Then two general approaches in dealing with the problem are discussed. Details will be provided on three specific methods that relate to our experiments. Next a survey of previous work related to multi-label classification is presented. Finally, the performance of classifiers in this domain is discussed.

2.1 Problem Statement

Let $\mathcal{X} \subseteq \mathbb{R}^p$ be an instance space of p-dimensional features and $\mathcal{Y}$ be a finite set of labels or classes. Each instance $x \in \mathcal{X}$ has multiple class labels in Y, where $Y \subseteq \mathcal{Y}$. Given a set of training examples consisting of n instances, $S = \{(x_1, Y_1), \ldots, (x_n, Y_n)\}$, where each instance is independent and identically distributed (i.i.d.) drawn from an unknown distribution D, $x_i \in \mathcal{X}$ and labels $Y_i \subseteq \mathcal{Y}$ are known, the goal of multi-label classification is to learn categories' properties from labeled examples and find a multi-label classifier $g : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$ that maps an instance x to its label such that specific performance criteria are optimized. Here, $2^{\mathcal{Y}}$ is the power set of $\mathcal{Y}$, which is the set of all subsets of $\mathcal{Y}$.

Many machine learning algorithms induce decision rules in the form of ranking function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$. For a given instance x, the labels in $\mathcal{Y}$ can be ordered according to the scores or values of $f(x, .)$. This ranking function $f(x, c)$ is thus interpreted as the system's confidence that x belongs to class c. It is said that label c_1 is ranked higher than c_2 if and only if $f(x, c_1) > f(x, c_2)$. If Y is the set of class labels of x, then a good learning algorithm will rank labels in Y higher than those not in Y. A simple classification function, $g(x)$, can be easily obtained from the function $f(x, c)$ and a threshold, $t(x)$, by labeling x with the classes whose ranking values exceed $t(x)$:

$$g(x) \;=\; \{c | f(x, c) > t(x), c \in \mathcal{Y}\}. \tag{2.1}$$

The threshold $t(x)$ is usually chosen to be a constant function. The most common threshold is $t(x) = 0$ which is similar to the sign function.

Classifier's performance depends greatly on the characteristics of the data to be classified. One factor that has a certain bearing on the behavior of a multi-label classifier induced from a data set is the number of labels of each example in the set. In some applications, the number of labels each example has is large compared to the total number of labels in the entire data set, whereas in others it is small. Tsoumakas and Katakis (2007) introduced two pertinent concepts, *label cardinality* and *label density*, that can be used to describe a data set. Denote the cardinality or "size" of a set Y by $|Y|$. Here are the definitions of these two notions:

- Label cardinality of data set S is the average number of labels of the examples in S,

$$LC(S) = \frac{1}{n}\sum_{i=1}^{n}|Y_i|. \tag{2.2}$$

- Label density of data set S is the average proportion of labels of each example,

$$LD(S) = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i|}{|\mathcal{Y}|}. \tag{2.3}$$

Both metrics quantify the number of alternative labels that depict the examples of a multi-label data set. Label cardinality is independent of the total number of labels in the classification problem, while label density takes the total number into account. Two data sets having the same label cardinality but different label density might exhibit different properties that influence the performance of the multi-label classification methods. However, these metrics are reported for the sake of completeness, but they will not be used in the dissertation because the multi-label approach used in this dissertation treats each class separately, and, thus, we will report the percentage of the positive examples for *each class* instead of the measure for *each example.*

2.2 General Approach

Existing multi-label classification methods may be grouped into two main approaches: 1) problem transformation and 2) algorithm adaptation (Tsoumakas and Katakis 2007). *The problem transformation approach* comprises those techniques that transform the multi-label classification problem into one or more conventional single-label classification problems. *The algorithm adaptation approach* comprises any methods that extend specific learning algorithms in order to cope with multi-labeled examples directly.

There are a number of ways that one could use to pursue the problem transformation as follows:

1. Ignore multi-label information. Two straightforward methods are (Boutell et al. 2004a):

    - Randomly or subjectively select one of the multiple labels of each multi-labeled example and dispose of the rest.
    - Simply remove every multi-labeled example from the data set.

    Obviously these methods have a serious shortcoming of discarding a lot of information content of the original multi-label data set.

2. Consider each different set of labels that exists in the multi-label data set as a single label and then train one single-label classifier (Boutell et al. 2004a; Diplaris et al. 2005). One negative aspect of this method is that it may lead to data sets with a large number of classes and few examples per class.

3. Learn $|\mathcal{Y}|$ binary classifiers, one for each different class label c in $\mathcal{Y}$ (Boutell et al. 2004a; Lauser and Hotho 2003). The original data set is transformed into $|\mathcal{Y}|$ data sets such that the c^{th} data set, $c = 1, \ldots, |\mathcal{Y}|$, contains all examples of the original data set, but each example is relabeled as c if the original example has label c, and $\bar{c}$ (not c) otherwise. When categorizing a new instance x, this method gathers and assigns to the new instance the labels output by all binary classifiers. Transforming a multi-label problem into a number of binary classification problems is the most common method of the problem transformation approach. However, this method has some disadvantages (Kang et al. 2006).

   - It treats each class label independently, and therefore does not exploit any correlation among class labels.
   - It does not scale very well to a large number of classes since a binary classifier has to be built for every class.
   - It suffers from the unbalanced data problem, particularly when the number of classes is large.

4. Decompose each example (x, Y) into $|Y|$ examples (x, c) for all $c \in Y$. Then induce one single-label classifier from the transformed data set using a learning method that can give to an instance certainty degrees or probabilities for all labels in $\mathcal{Y}$. A new instance is assigned those labels whose certain degrees are greater than a set threshold. This method increases the size of training examples, which is a downside.

Which one of these approaches (transformation and adaptation) depends on the user's preferences. In the dissertation, the problem transformation approach is preferred and applied to all presented works. Although there are some limitations for each way in the selected approach, it is still superior to the other one in terms of simplicity and generality. By following this approach, all well-known and efficient algorithms in machine learning can be adopted and applied to any domains of multi-label classification. Additionally, the issues occurred in the transformation approach can be alleviated by simple procedures or strategies. For instance, the problem of imbalanced training data can be lessened by the *under*- or *over*- sampling strategies. On the other hand, the problem adaptation approach requires more algorithmic complexity than the chosen one and it is often designed based on some specific contexts.

Among four variants of the problem transformation approach, the dissertation chose to follow the third one (the most common method): generating a separate binary classifier for each class. The first method to ignore multi-label information is obviously the worst one. Comparing to the rest, Rifkin and Klautau (2004) showed that the chosen one is as accurate as any other methods if the baseline classifiers are well-tuned regularized classifier, such as Support Vector Machine in Section 2.3.2.

Apart from the choice of classification methods, there are essentially two different types of categorization decision that one can make, *hard* categorization and *soft* (or ranking) categorization (Sebastiani 2006; Feldman and Sanger 2007). A hard categorization decision refers to a binary decision, yes or no, as to whether an instance x belongs to a category c. A soft categorization decision is the one consisting of attributing a real-valued score or weight to the example-category pair (x, c) indicating goodness-of-fit between the input document and the category. This score reflects the degree of confidence of the classifier in the fact that x belongs to category c, which allows ranking a set of categories in terms of their appropriateness for x, or ranking a set of instances in terms of their appropriateness for category c. A disadvantage of soft categorization or a ranking algorithm is that it does not output a set of labels for the example. A group of top scoring categories or all the categories with the scores above a chosen threshold may be selected. However, the threshold parameter will need tuning in each problem. The two classification functions, $g(x)$ and $f(x, c)$, described in Section 2.1 above are used in hard and soft decisions, respectively. Some classification methods offer the convenience of making both soft and hard decisions, but some methods only permit one decision type.

2.3 Overview of Algorithms

In this section, three state-of-the-art methods commonly used in this domain along those approaches in Section 2.2 are described. In addition, they are related to our prior and ongoing research.

2.3.1 Decision Trees

Decision trees are one of the most popular methods for classification because they provide simple and interpretable rules. A decision tree learns from data to create a predictive model having a tree structure; nodes in the tree represent features, with branches representing conjunctions of features that lead to leaves or terminal nodes representing classifications. Thus, a decision tree can be viewed as a partitioning of the instance space into smaller segments such that each partition, represented by a leaf, contains the instances that are homogeneous and are expected to belong to the same class. Determining the class of an instance from the decision tree is then a matter of tracing the path of nodes and branches starting at the root node of the tree to the terminating leaf. Figure 2.1 shows an example of a decision tree that suggests a suitable transport based on given attributes, the weather and transport conditions.

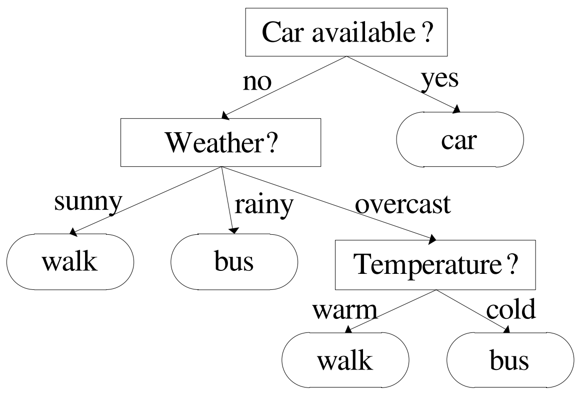


Figure 2.1: A sample decision tree showing decisions at the nodes and final classification at the leaves.

C4.5 is a decision tree generating algorithm developed by Quinlan (1993) and modified further later by Quinlan (1996). It is an extension of Quinlan's earlier ID3 induction algorithm (Quinlan 1986) with a number of improvements to account for unknown attribute values, attributes with differing costs, and bias towards continuous attributes with numerous distinct values. In addition, it is superior to ID3 for many view points including a different criterion for determining the best partitioning of the examples at each decision tree node, "pruned" decision trees to avoid overfitting the data, and the ability to derive classification rules from the unpruned decision tree.

C4.5 builds decision trees from training data using the information entropy concept to measure how informative a node is. It examines the information gain as shown in Equation 2.4, where S is a set of training instances, A is an attribute and a is its value, S_a is a subset of S consisting of the instances with $A = a$. Entropy(S) is defined in Equation 2.5, where $P_S(c_i)$ is the percentage of instances belonging to Class c_i in the database, and $|C|$ is the number of classes. The feature having the highest information gain is the best for discriminating among cases at that node, so it will be chosen to make the decision.

$$IG(S, A) = Info(S) - Info_A(S) = Entropy(S) - \sum_a \frac{|S_a|}{|S|} Entropy(S_a) \tag{2.4}$$

$$Entropy(S) = -\sum_{i=1}^{|C|} P_S(c_i) log P_S(c_i) \tag{2.5}$$

Although information gain is usually a good measure for deciding the relevance of an attribute, it is not perfect. A notable problem occurs when information gain is applied to attributes that can take on a large number of distinct values. For example in Figure 2.2, a primary key attribute, such as Id, has a high information gain because it uniquely identifies each example. Assuming there are two classes (*pass* and *fail*), the decision tree in Figure 2.2 can give an answer to only these ten training records (Id1 - Id10), but cannot predict any other testing data; thus, we do not want to include this kind of attributes in the decision tree. Hence, in C4.5, Quinlan (1996) introduced the gain ratio in Equation 2.6 by normalizing the information encoded in the split itself.

$$GainRatio(S, A) = \frac{IG(S, A)}{SlitInfo(S, A)} \tag{2.6}$$

C4.5 is a good choice for practical classification due to the ease of its interpretability as well as its ability to deal with numeric attributes, missing values, and noisy data. In their IP traffic flow classification study, Williams et al. (2006) compared four

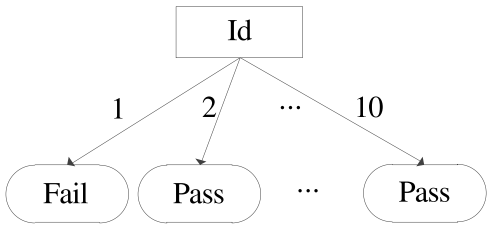


Figure 2.2: An example of the problem in a decision tree based on the information gain criterion.

different machine learning algorithms, Bayesian network, naive Bayes, naive Bayes tree, and *C4.5* decision tree, and concluded that *C4.5* was the best suited for real-time classification tasks. All algorithms provided very similar classification accuracy, however, the *C4.5* algorithm was significantly faster than other methods in terms of classification speed.

2.3.2 Support Vector Machine

This section gives an introduction to the Support Vector Machine (SVM) (Schölkopf et al. 1999; Joachims 1998; Joachims 2003; Kwok 1998). It is currently the state-of-the-art in classification techniques. Benchmarking studies reveal that in general, the SVM performs best among current classification techniques, due to its ability to capture-nonlinearity.

Linear Separable Case: SVM was originally developed for dichotomous domains where each example, $\vec{x}$, is either positive or negative. The data set consists of pairs, $S = \{(\vec{x_1}, y_1), ..., (\vec{x_n}, y_n)\}$, where $\vec{x_i}$ is an attribute vector describing the example, and $y_i \in \{-1, +1\}$ is this example's class label.

From a training set, SVM induces a linear hyperplane separating between two classes as in Equation 2.7, where $\vec{w}$ is a weight vector that determines the hyperplane's orientation, and b (bias) determines its offset relative to the system coordinates.

$$h(\vec{x}) = f(\vec{w}, b) = \vec{w} \cdot \vec{x} + b = 0 \tag{2.7}$$

In the case that two classes can be linearly separable as in Figure 2.3, the linear hyperplane (classifier) H_0 which maximizes the margin can be formulated as set of inequalities over the set of training examples as follows:

$$\begin{aligned} \text{Minimize} &: \tfrac{1}{2}\|w\|^2 \\ \text{subject to} &: y(\vec{x_i} \cdot \vec{w} + b) - 1 \geq 0, \forall x_i \in S \end{aligned} \tag{2.8}$$

From this formulation it can be stated, that all positive examples, for which equality holds, lie on the hyperplane $H_1 : \vec{x_i} \cdot \vec{w} + b = 1$. Similar, all negative examples, for which equality holds, lie on the hyperplane $H_2 : \vec{x_i} \cdot \vec{w} + b = -1$. The width of the margin denotes as $\gamma = \frac{2}{\|w\|^2}$; thus, minimizing $\frac{1}{2}\|w\|^2$ in Equation 2.8 is equivalent to maximizing the margin γ. Those data points for which equality holds are called *support vectors*, which are examples surrounded by circles.

SVM that induces a classifier based on the above set of inequalities is called "*hard margin SVM*" because there are no any examples between the hyperplanes H_1 and H_2. Equation 2.8 can be reformulated by using a Lagrangian formulation of the problem. Positive Lagrange multipliers $\alpha_i, i = 1...|S|$ are introduced, where each α_i is the inequality constraint for an example x_i and $|S|$ denotes the number of examples in the data set S. To form the Lagrangian, the constraint equations are multiplied by the Lagrange multipliers and subtracted from the objective function which gives:

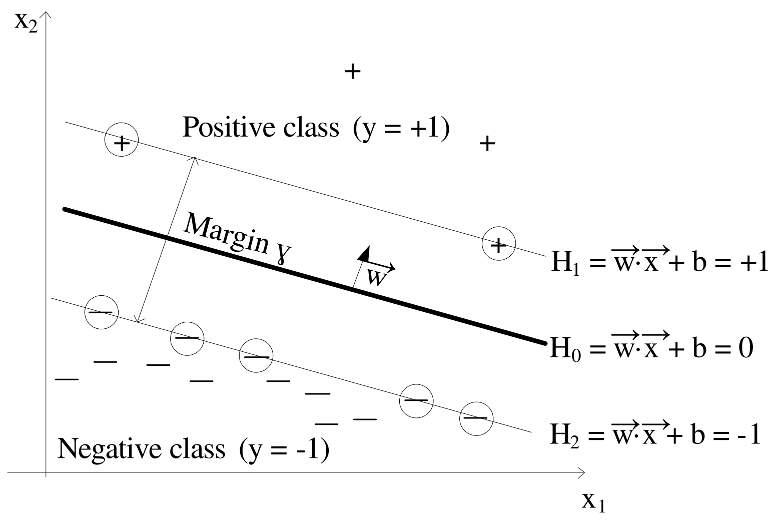


Figure 2.3: Linear SVM classifier in a binary classification problem when two classes can be linearly separated. The separation hyperplane H_0 is also called *a hard decision boundary.*

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{|S|} \alpha_i(y_i(\vec{x_i} \cdot \vec{w} + b) - 1) \tag{2.9}$$

In order to solve $L(\vec{w}, b, \vec{\alpha})$, the problem must be transformed to the dual quadratic optimization problems as follows:

$$\begin{aligned} \text{Maximize} &: \textstyle\sum_{i=1}^{|S|} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \vec{x_i}\vec{x_j} \\ \text{subject to} &: (\textstyle\sum_{i=1}^{|S|} \alpha_i y_i = 0) \wedge (\forall_i, \alpha_i \geq 0) \end{aligned} \tag{2.10}$$

From this optimization problem all α_i can be obtained which gives a separating hyperplane defined by $\vec{w}$, maximizing the margin γ between training examples in the linear separable case. Note that the formulation of these optimization problem replaces $\vec{w}$ with the product of the Lagrangian multipliers and the given training examples $\sum_{i=1}^{|S|} \alpha_i y_i \vec{x_i}$. Thus, the separating hyperplane can be defined only through the given training patterns $\vec{x_i}$. Additionally, support vectors have a Lagrange multiplier of $\alpha_i \geq 0$ whereas all other training examples have a Lagrange multiplier of zero ($\alpha_i = 0$). Support vectors lie on one of the hyperplanes H_1, H_2 and are the critical examples in the training process. Thus, the weight vector $\vec{w}$ which determines the hyperplane's orientation can be written by only the support vectors as follows:

$$\vec{w} = \sum_{\alpha_i \neq 0} \alpha_i y_i \vec{x_i} \tag{2.11}$$

Also, if all training examples with $\alpha_i = 0$ would be removed, retraining the SVM produces to the same separating hyperplane. For the remaining parameter, b, of the hyperplane, it can be solved by using the KKT complementary condition, $\alpha_i(y_i(\vec{w} \cdot \vec{x_i} + b) - 1) = 0$ where $i = 1, ..., |S|$, and $\vec{w}$ is obtained by the above equation. The substitution of each $\{\alpha_i, x_i, y_i\}$ generates different b_i. Thus, it is numerically safer to take the mean value of all resulting b as follows:

$$b = \frac{1}{|S|}(\sum_{i=1}^{|S|}(y_i - \vec{w} \cdot \vec{x_i})) \tag{2.12}$$

Once learning $\vec{w}$ and b, SVM uses them to map testing examples to scalar SVM scores as in Equation 2.13. The predicted class, either positive or negative, is decided by the sign of these SVM scores.

$$s_i = h(\vec{x_i}) = \vec{w} \cdot \vec{x_i} + b \tag{2.13}$$

Non Separable Case: The above equation holds only for the linear separable case; however, it is not feasible in the general case because there are often some outliers within the margin from the separator or even be missclassified as shown in Figure 2.4.

To prevent outliers from affecting the hyperplane, the *slack* variable $\xi_i, i = 1, ..., |S|$, are introduced to relax the hard margin constraints. This kind of SVM is called "*soft margin SVM*" by pursuing the reformulated constraints as follows:

$$\begin{aligned} \text{Minimize} &: \tfrac{1}{2}\|w\|^2 + C\textstyle\sum_{i=0}^{m}\xi_i \\ \text{subject to} &: y_i(\vec{w}\cdot\vec{x_i}+b) \geq (1-\xi_i), (\forall\vec{x_i}\in S \wedge \xi \geq 0) \end{aligned} \tag{2.14}$$

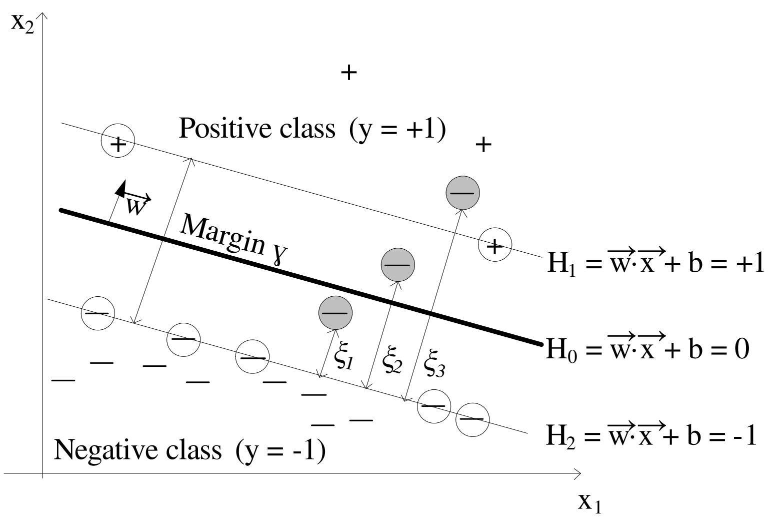


Figure 2.4: Linear SVM classifier in a binary classification problem when two classes cannot be linearly separated. The examples in grey circles are outliers. The separation hyperplane H_0 is also called *a soft decision boundary.*

Whereas the margin was defined by the margin maximization, it now seeks to simultaneously minimize two conditions: (i) the term of $\|w\|^2$ to maximize the margin and (ii) the total classification errors on training data, $\sum_{i=0}^{m}\xi_i$. The trade-off between these conditions is controlled by the parameter C where a larger value leads to small number of misclassifications and smaller margin.

By the definition, ξ_i's have to be larger or equal to 0, so that it really mean *slackness*. Also, we can observe that if $0 < \xi_i \leq 1$, such as ξ_1, it means the example lies somewhere between the margin and the correct side of hyperplane, and if $\xi_i > 1$, such as ξ_2 and ξ_3, it means that the example is missclassified.

Again, by applying Lagrangian multipliers, the dual optimization problem can be formulated as:

$$\begin{aligned} \text{Maximize} &: \textstyle\sum_{i=1}^{|S|} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{x_i} \vec{x_j} \\ \text{subject to} &: (\textstyle\sum_{i=1}^{|S|} \alpha_i y_i = 0) \wedge (\forall_i, 0 \le \alpha_i \le C) \end{aligned} \tag{2.15}$$

The only difference from the hard margin dual form is that α_i is bounded by the trade-off parameter C. Hence, there are three possible solutions for α_i. First, $\alpha_i = 0$ and $\xi_i = 0$, an example $\vec{x_i}$ is correctly classified. Second, $0 < \alpha_i < C$ and $\xi_i = 0$, an example $\vec{x_i}$ is called *an unbounded support vector* which lies on the two margins. Finally, $\alpha_i = C$ and $\xi_i \ge 1$, an example $\vec{x_i}$ is called *an bounded support vector* which lies on the wrong side of the margin.

Non-Linear SVM's: Another possibility for learning linear inseparable data is to map the training data into a higher dimensional space by some mapping function Φ. According to the theory, by applying an appropriate mapping linear inseparable data become separable in a higher dimensional space. Thus, a mapping of the form is applied on the sequence S of training examples transforming them into $S' = \{\langle \Phi(\vec{x_1}), y_1 \rangle ... \langle \Phi(\vec{x_m}), y_m \rangle\}$. Thereby, S' is the new feature space obtained from the original space through the mapping Φ. This is also implicitly done by neural networks (using hidden layers which map the representation).

Figure 2.5 illustrates an example of non-linear mapping on the training data with one attribute, S, to the higher space with three attributes, S'. There are two classes, $Y = \{-1, 1\}$, and three examples, $x_1 = -1$, $x_2 = 0$, and $x_3 = 1$ where their corresponding classes are $y_1 = 1$, $y_2 = -1$, and $y_3 = 1$, respectively. On the one-dimensional space, these examples cannot be linearly separable into two classes. While, the solution, $\vec{h}$, can be found in the three-dimensional space, in which each example is transformed by using the mapping function $\Phi(x) = [\phi_1(x), \phi_2(x), \phi_3(x)]^T = [x^2, \sqrt{2}x, 1]^T$.

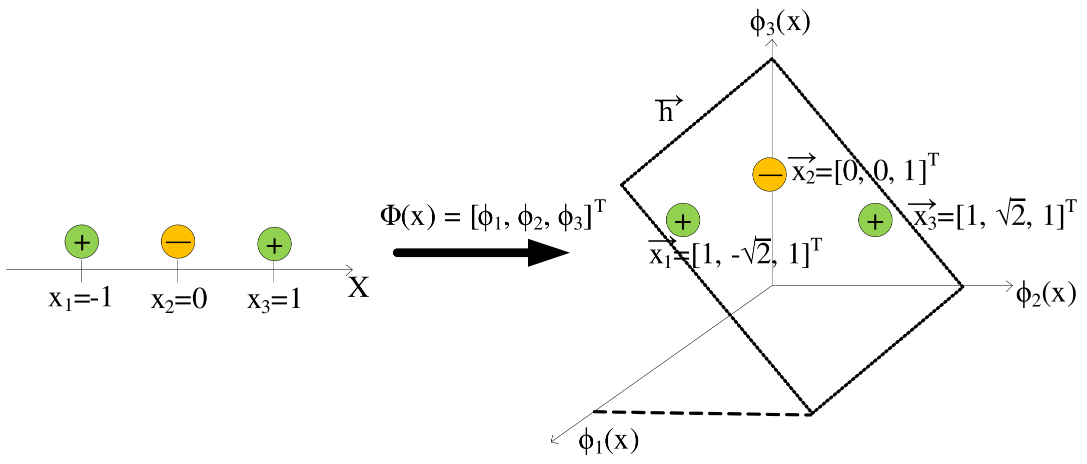

x_1=-1
x_2=0
x_3=1
X
$\Phi(x) = [\phi_1, \phi_2, \phi_3]^T$
$\phi_3(x)$
$\vec{h}$
$\vec{x_2}=[0, 0, 1]^T$
$\vec{x_3}=[1, \sqrt{2}, 1]^T$
$\vec{x_1}=[1, -\sqrt{2}, 1]^T$
$\phi_2(x)$
$\phi_1(x)$

Figure 2.5: Mapping from the original space, S, with one attribute (left) to the higher dimensional space, S', with three attributes (right) by using the function $\Phi(x)$.

One drawback of the mapping is the algorithmic complexity arising from the high dimensional space S', making learning problems virtually intractable. But since learning and testing SVM's is defined by evaluating the inner product between training examples, so called *kernel functions* can be used to reduce algorithmic complexity and making infinite spaces tractable. A kernel function $\kappa(x_i, x_j)$ for a mapping function Φ is defined as $\Phi(\vec{x_i})\Phi(\vec{x_j}) = \kappa(x_i, x_j)$. Thus, the dot product of $x_i \cdot x_j$ in Equation 2.9 can be replaced by $\kappa(x_i, x_j)$ in order to map from the original space to a higher space. The following are common kernel functions:

$$\begin{aligned} \text{Gaussian RBF} &: \kappa(\vec{x_i}, \vec{x_j}) = exp(\tfrac{-\|\vec{x_i}-\vec{x_j}\|^2}{\sigma}) \\ \text{Polynomial} &: \kappa(\vec{x_i}, \vec{x_j}) = ((\vec{x_i}, \vec{x_j}) + b)^d \\ \text{Sigmoid} &: \kappa(\vec{x_i}, \vec{x_j}) = tanh(\kappa(\vec{x_i}, \vec{x_j}) + b) \end{aligned} \tag{2.16}$$

where d, c, σ, and θ are constants.

2.3.3 Associative Classification

Associative classification (AC) is a data mining technique that integrates classification with association rule mining (ARM) (Agrawal and Srikant 1994; Agrawal et al. 1993) to find the rules from classification benchmarks. A generated rule for classification, called “class association rule” (CAR), is an implication of the form of $X \rightarrow c$, where itemset X is non-empty subset of all possible items in the database, $X \subseteq I$, $I = \{i_1, i_2, ..., i_n\}$ where n is the number of itemsets, and c is a class identifier, $c \in \{c_1, c_2, ..., c_m\}$ where m is the number of classes. Let a rule itemset be a pair $< X, c >$, containing the itemset X and a class label c. The rules are discovered in a training data set of transactions D_t. The strength of a rule R can be measured in terms of its support ($sup(R)$) and confidence ($conf(R)$). The support of R is the percentage of the instances in D_t satisfying the rule antecedent and having class label c as shown in Equation 2.17. The confidence of R is the percentage of instances in D_t satisfying the rule antecedent that also have the class label c as shown in Equation 2.18.

$$sup(R) = P(X, c) \tag{2.17}$$

$$conf(R) = \frac{sup(R)}{sup(X)} = \frac{P(X,c)}{P(X)} = P(c|X) \qquad (2.18)$$

Referring to the support and confidence constraints, the rule $X \rightarrow c$ is a class association rule if the following two conditions are satisfied: $sup(X \rightarrow c) \geq minsup$ (minimum support threshold) and $conf(X \rightarrow c) \geq minconf$ (minimum confidence threshold). To find all CARs, many algorithms are commonly decomposed into three major processes: rule generation, rule selection, and classification. First, the rule generation process extracts all CARs that satisfy both minimum support and minimum confidence thresholds from the training data set. Second, the rule selection process applies pruning techniques to select a small subset of high-quality CARs and builds an accurate model of the training data set. Finally, the classification process is used to classify an unknown data instance.

In recent years, some classifiers based on AC have been proposed such as CAEP (Dong et al. 1999), CMAR (Li et al. 2001), CBA (Liu et al. 1998), and ADT (Wang et al. 2000). These algorithms rank the generated CARs using a confidence measure. According to these algorithms, the rule ranking process plays an important role in the classification process since the accuracy is affected directly to the order of CARs. However, the confidence measure has some limitations (Tan et al. 2005). It may not be good enough due to a low discrimination power to the instances in the other classes. The reason is that a confidence measure is defined using a frequency count of the *exact matched* instances on a training data set. On a space of distribution, it is possible that a high confidence rule can be close to many instances in different classes. Thus, this rule can misclassify instances in different classes on a testing set.

Regarding the problem in the traditional AC, Vateekul and Shyu (2008) proposed a novel conflict-based confidence measure for ranking CARs in an AC framework, which better captures the conflict between a rule and a training instance. It quantifies the amount of conflict that a rule possesses with respect to all instances belonging to different classes in the data set. Moreover, the framework also included a presented interleaving ranking strategy that can improve performance of CARs on both balance and imbalance data sets.

2.4 Related Work

This section shows research in domains of multi-label classification. We focus on explaining only works based on the mentioned algorithms in Section 2.2, i.e., decision tree, SVM, and AC.

Decision Tree: In this kind of domains, "multi-label C4.5" (ML-C4.5) (Clare and King 2001) is the most well-known decision tree-based algorithm. The authors modified the C4.5 algorithm for the functional prediction in genomic data. The main difference between ML-C4.5 to the original version is how it calculates an entropy as shown in Equation 2.19, where $P_S(c_i)$ is the percentage of instances belonging to Class c_i in the database, and $|C|$ is the number of classes. The term $(1 - P_S(c_i))log(1 - P_S(c_i))$ is considered in case of the probability belonging to other classes, $(1 - P_S(c_i))$. Gao et al. (2004) extended their binary maximal figure-of-merit learning algorithm to multi-label problems. The method trains all classifiers simultaneously and optimizes performance metrics such as precision and recall. Nonetheless, their discriminant function for classification is still based on individual categories.

$$Entropy(S) = -\sum_{i=1}^{|C|}(P_S(c_i)logP_S(c_i) + (1 - P_S(c_i))log(1 - P_S(c_i))) \tag{2.19}$$

SVM: Several of the recent works in multi-label classification employ SVM, based on the problem transformation approach, by generating a separate SVM classifier for each class. Joachims (1998) and Kwok (1998) showed that SVM is appropriate for learning text classifiers with many desirable properties. SVM is good at handling domains where the number of features exceeds the number of training examples. It eliminates the need for feature selection, saving a complicated preprocessing step. It is well suited for problems with few irrelevant features and sparse example vectors, and it behaves robustly without the need for manual parameter tuning. Ease in incorporating new examples, when available, into an existing trained system is also a plus. In the experiments of Joachims (1998) and Kwok (1998), SVM substantially and significantly outperformed other text classification methods. On the downside, SVM induction is computationally more expensive than that of naive Bayes and k-NN, but roughly comparable to the C4.5 decision tree algorithm. At classification time, SVM is faster than k-NN, however (Colas and Brazdil 2006). An improvement to the standard SVM proposed by Joachims (2006) is a Cutting-Plane Algorithm called SVM-Perf. Compared to existing methods, SVM-Perf is simple and easy to implement. It is faster than the standard one that uses decomposition methods, and in his experiments there was no indication that SVM-Perf was less accurate.

AC: The most common method is MMAC (Multi-class Multi-label Associative Classification) (Thabtah et al. 2004). It is divided into three modules: rules generation, recursive learning and classification. In the recursive learning process, MMAC generates rules that can predict multiple classes using merging strategy while other AC algorithms, CBA and CPAR, produce rules that output only single class. For example, assume there are two rules, $R1 : (x1 \wedge x2) \longrightarrow C1$ and $R2 : (x1 \wedge x2) \longrightarrow C2$, MMAC will merges these rules to the new rule $R : (x1 \wedge x2) \longrightarrow (C1 \vee C2)$.

2.5 Performance Evaluation

2.5.1 Classical Classification Criteria

Before exploring the ways to evaluate multi-label classifiers, we first need to know how to evaluate binary classifiers which label each example as either belonging or not belonging to the given category. Here we rely on the following quantities: TP (true positives), FN (false negatives), FP (false positives), and TN (true negatives). Their values are used in the formulas for *precision (Pr)* and *recall (Re)*:

$$Pr = \frac{TP}{TP + FP} \qquad\qquad Re = \frac{TP}{TP + FN} \tag{2.20}$$

Which of the two really matters depends on the user's needs and preferences. *Precision* is the percentage of truly positive examples among those labeled as such by the classifier; conversely, *recall* gives the percentage of truly positive examples that have been recognized as such by the classifier. Observing that we often want to maximize both criteria while balancing their values, (van Rijsbergen 1979) proposed a way to combine *precision* and *recall* in a single formula, F_β, parameterized by the user-specified $\beta \in [0, \infty)$ that quantifies the relative importance of either metric:

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re} \tag{2.21}$$

The reader can see that $\beta > 1$ apportions more weight to *recall* while $\beta < 1$ emphasizes *precision*. Moreover, F_β converges to *recall* if $\beta \to \infty$, and to *precision* if $\beta = 0$. The situation where *precision* and *recall* are equally relevant is reflected by $\beta = 1$, which leads to the following formula:

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{2.22}$$

2.5.2 Multi-label Classification Criteria

As for the needs of multi-label domains, (Yang 1999) proposed two alternative ways to generalize the above criteria: (1) *macro-averaging*, where *precision* and *recall* are first computed separately for each category and then averaged; and (2) *micro-averaging*, where *precision* and *recall* are obtained by summing over all individual decisions. Which of the two averaging methods really matters depends on the concrete application: whereas micro-F_1 weighs the classes according to their relative frequency, macro-F_1 gives equal weight to each class. The formulas are summarized in Table 2.1 where Pr_i, Re_i, and $F_{1,i}$ stand for *precision*, *recall*, and F_1 for the i-th class (from N classes).

2.6 Conclusion

This chapter summarizes concepts and existing works related to the domain of multi-label classification, where the same example can belong to two or more classes at the

Table 2.1: Macro-averaging and micro-averaging of the performance criteria on the data set with k classes.

Average	Criteria	Equation
Macro	Precision	$Pr^M = \frac{\sum_{i=1}^{k} Pr_i}{k}$
	Recall	$Re^M = \frac{\sum_{i=1}^{k} Re_i}{k}$
	F_1	$F_1^M = \frac{\sum_{i=1}^{k} F_{1,i}}{k}$
Micro	Precision	$Pr^\mu = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} (TP_i + FP_i)}$
	Recall	$Re^\mu = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} (TP_i + FN_i)}$
	F_1	$F_1^\mu = \frac{2 \times Pr^\mu \times Re^\mu}{Pr^\mu + Re^\mu}$

same time. The goal is to induce a classifier from training data to classify testing data in this kind of scenarios such that specific performance criteria are optimized. Existing multi-label classification methods can be grouped into two main approaches: (*i*) problem transformation and (*ii*) problem adaptation. Both approaches have different pros and cons, so the choice of use depends on the user's preferences. The presented works in this dissertation follow the most common approach, the problem transformation one by generating a separate classifier for each class. Details of the common state-of-art baseline classifiers are provided including Decision Trees, Support Vector Machine (SVM), and Associative Classification (AC). Next a survey of previous works based on these three methods in the domain of multi-label classification are presented. To evaluate multi-label classifiers, there are two alternative ways to combine each of single-label classification criteria (*precision (Pr)*, *recall (Re)*, and F_1): (*i*) *macro-averaging* and (*ii*) *micro-averaging*. The first averaging method gives equal weight to all classes while the latter weighs the classes according to their number of examples.

CHAPTER 3

Hierarchical Classification

Hierarchical classification deals with problems where categories (classes) are organized in the form of a hierarchy. This research area has received less attention, yet it has many important applications. Many text collections are organized as hierarchies including (*i*) web repositories and digital libraries, e.g., Dmoz, Wikipedia[5], Yahoo (McCallum et al. 1998; Mladenie 1998), LookSmart (Dumais and Chen 2000), EUROVOC (Sarinnapakorn and Kubat 2007), and Reuters (Lewis et al. 2004a), (*ii*) bio-medical fields, e.g., OHSUMED (Hersh et al. 1994), MIPS' FunCat (Mewes et al. 1997), and Gene Ontology (Harris 2004), (*iii*) image recognition (Stenger et al. 2007), etc. These demonstrate that learning in the presence of class hierarchies is becoming a necessity in many domains of applications.

This chapter aims to show the details of hierarchical classification. First, the specific concepts of problems and tasks in this domain are provided. Then general approaches and its related work are summarized. Finally, the performance of classifiers in this domain will be explained.

[5]The data sets from Wikipedia (www.wikipedia.org) and the ODP web directory (www.dmoz.org) are available at the 2nd Pascal Challenge on Large Scale Hierarchical Text Classification (LSHTC2).

3.1 Specifics of Hierarchical Classification

This section defines some notations specific for a hierarchical classification. In particular, we first give definitions and types of hierarchies, then details of the class relationships are provided, next a problem statement of hierarchical classification is explained, and finally, we will discuss the types of hierarchies in a real-world application: taxonomy and ontology.

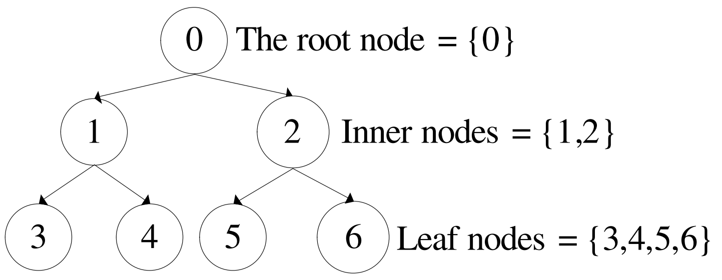
3.1.1 Definition

The following notations and symbols will be used throughout the dissertation.

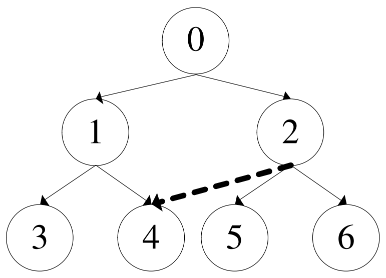
Hierarchy ($\mathcal{H}$): A hierarchy $\mathcal{H} = (\mathcal{N}, \mathcal{E})$ is defined as directed acyclic graph (DAG) consisting of a set of nodes, $\mathcal{N}$, and a set of edges, $\mathcal{E}$, where an edge is an ordered pair of nodes, $(N_p, N_c) \in \mathcal{E} \subseteq \{\mathcal{N} \times \mathcal{N}\}$. The direction of an edge (N_p, N_c) is defined from the parent node N_p to the *direct* child node N_c, specified through the relationship operator $N_p \Rightarrow N_c$ which also called the direct path from N_p to N_c. A path $N_a \rightarrow N_c$ indicates that there exists a path from the ancestor node N_a to the child node N_c. Note that in a hierarchy $\mathcal{H}$ with a path $N_a \rightarrow N_c$ there exists no path $N_c \rightarrow N_a$ since the hierarchy is acyclic.

Additionally there exists exactly one node called *root* node N_r of a graph $\mathcal{H}$ that has no parent. Nodes without any child nodes are called *leaf* nodes. All nodes except the root node and leaf nodes are called *inner* nodes. In Figure 3.1(a), the root node is $\{0\}$, inner nodes are $\{1, 2\}$, and leaf nodes are $\{3, 4, 5, 6\}$.

The structure of hierarchies can be classified into two main categories: *tree* and *Direct Acyclic Graph (DAG)* structures. The main difference between the tree structure (Figure 3.1(a)) and the DAG structure (Figure 3.1(b)) is that each node in the tree hierarchy has just one parent node, while each node in the DAG hierarchy can inherit from many parents.



(a) A tree-structured hierarchy



(b) A DAG-structured hierarchy

Figure 3.1: Examples of hierarchies.

Classes ($\mathcal{C}$): Let $\mathcal{C}$ be a finite set of class labels, and each class C_i consists of a set of examples $\mathcal{X}$. The relationship among classes is defined in the class hierarchy $\mathcal{H}$ where each node N_i within the hierarchy is assigned to exactly one class C_i ($\mathcal{C} \equiv \mathcal{N} \in \mathcal{H}$).

Examples ($\mathcal{X}$): Let $\mathcal{R}^p$ be an example space where each example is described by a vector of p attributes. Let $\mathcal{X} \subset \mathcal{R}^p$ be a finite set of examples. In the hierarchy $\mathcal{H}$, a set of examples is assigned to one or more class labels, $\mathcal{L} = \{C_1, ..., C_l\}$.

3.1.2 Class Relationship

$\mathcal{H}$ is a hierarchical structure defining relationships among all classes $\mathcal{C}$. The assumption behind these constraints is, that $C_{parent} \rightarrow C_{child}$ commonly defines as *a IS-A relationship* in the tree structure and *a Part-of relationship* in the DAG structure among classes whereby C_{parent} has a boarder topic than C_{child} and the topic of a parent class C_{parent} covers all. Additionally, the topics from siblings may be overlapped, the multi-label case. The relationships must follow the properties below:

- *Asymmetric:* $\forall(C_i, C_j) \in \mathcal{C}, \text{if } \ C_i \rightarrow C_j \ \ then \ \ C_j \nrightarrow C_i$. For example, all dogs are mammals, but not all mammals are dog.
- *Anti-reflexive:* $\forall(C_i) \in \mathcal{C}, C_i \nrightarrow C_i$. For example, the parent class of a dog cannot be itself, unnecessary.
- *Transitive:* $\forall(C_i, C_j, C_k) \in \mathcal{C}, \text{if } (C_i \rightarrow C_j) \wedge (C_j \rightarrow C_k) \ \ then \ \ (C_i \rightarrow C_k)$. For instance, all dogs are mammals and all mammals are animals; therefore all dogs are animals.

3.1.3 Problem Statement

The problem in this area is defined as follows:

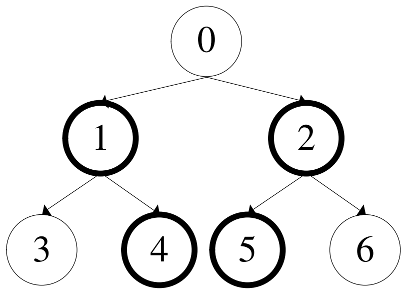
Hierarchical Classification: Given a set of training data, $\mathcal{S} = \{(x_1, C_1), ..., (x_n, C_n)\}$, the goal is to induce a classifier to carry out the mapping $\Phi : \mathcal{X} \rightarrow 2^{\mathcal{C}}$ in a way that maximizes classification performance. The constraint defined by the hierarchy $\mathcal{H}$ must be satisfied as shown in the definitions below.

- **Definition "Hierarchical Consistency (Constraint):**" An example x_i that belongs to a class C_i is automatically assigned to *all* of its ancestor classes $C_{ancestor}$ as shown in the equation below. Note that every example belongs to the top class of the hierarchy; therefore, we always exclude the top node from any ancestor set since including it does not provide any additional information on the example.

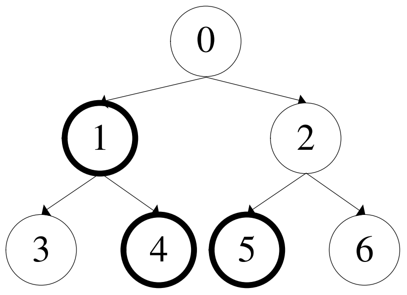
$$\{x_i \in C_i\} \Rightarrow \{\forall(C_{ancestor})|C_{ancestor} \rightarrow C_i \wedge x_i \in C_{ancestor}\}$$
$$\{C_{ancestor}, C_i\} \in \mathcal{H}, \mathcal{C} \tag{3.1}$$

- **Definition "Hierarchical Consistency Requirement:"**. Any class assignments produced by a hierarchical classification system on a given hierarchical tasks has to be consistent with a corresponding class hierarchy.

Figure 3.2 presents examples of consistent and inconsistent class assignments. Figure 3.2(a) gives a case of the consistent assignment since the example x_i which belongs to Classes C_4 and C_5 is labeled along with all their ancestors, Classes C_1 and C_2. While Figure 3.2(b) shows an inconsistent assignment case: the example x_i belongs to Class C_5, but it is not assigned to Class C_2 which is the C_5's parent class. Such situations can occur if we apply conventional classification method to a hierarchical task without any modifications.



(a) Consistent assignment



(b) Inconsistent assigment

Figure 3.2: Examples of consistent and inconsistent class assignments. The circles in bold represent the categories assigned to an example x_i by a classifier.

In some hierarchical classification problems, all examples are associated with classes at the leaf nodes. This kind of problems is called **mandatory leaf-node**

problems (MLNP). When this obligation does not hold, the problem is called **optional leaf-node problems (NMLNP)**. For instance, referring to Figure 3.2(a), an example can belong to any nodes in NMLNP, whereas it must belong to only leaf nodes, $\{3, 4, 5, 6\}$, in MLNP.

3.1.4 Taxonomy and Ontology

Taxonomy is a particular classification arranged in a hierarchical structure. Typically this is organized by supertype-subtype relationships, also called generalization-specialization relationships. Thus, it must be a tree hierarchical structure (cannot be DAG). There is also the *thesaurus.* Thesaurus and Taxonomy are similar in that both are tree hierarchical structures; however, the thesaurus covers more types of relationships than taxonomy which has only the superclass-subclass relationships. Moreover, both taxonomy and thesaurus are commonly used in domains of text categorization. Sometimes taxonomy is also used in domains of biology, such as Functional Category (FunCat) and MIPS provided by The Munich Information Center for Protein Sequences[6] (Mewes et al. 2002).

Ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts. Comparing to taxonomy, ontology is not organized only as a tree structure, but also a DAG structure. Moreover, ontology often uses more complex relationships. The concept of ontology is especially widespread in domains of biology, such as Gene Ontology (GO)[7] (Harris 2004).

[6]http://mips.gsf/proj/yeast/catalogues/funcat

[7]http://www.geneontology.org

3.2 General Approach

Silla and Freitas (2010) summarized three common approaches in domains of hierarchical classification including (*i*) flat classification approach, (*ii*) local classification (or top-down) approach, and (*iii*) global (or big-bang) approach. Note that the notations and figures in this section are based on the survey of Silla and Freitas (2010).

3.2.1 Flat Classification Approach

The simplest approach is to transform the hierarchical problem to a flat classification problem. It ignores the class hierarchy altogether, and deals only with the leaf-node classes (as if the problem were MLNP), whether by a single multi-label classifier or by a set of binary classifiers (a separate one for each leaf node). Traditional approaches, such as neural networks, decision tree, and SVM, may be applied in this context without any modifications. Figure 3.3 gives a scenario of this approach that only the classes in dash boxes, $\{1.1, 1.2, 2.1.1, 2.1.2, 2.2.1, 2.2.2\}$, are conducted classifiers.

The advantage of this approach is its simplicity and the ability to cope with both tree and DAG hierarchical structures. However, it can lead to many serious problems. First, the results of this approach may conflict with the hierarchical consistency in Section 3.1.3 since the parent-child relationships are not taken into account. Second, it cannot support NMLNP. Due to these deficiencies, this approach is not considered in the dissertation.

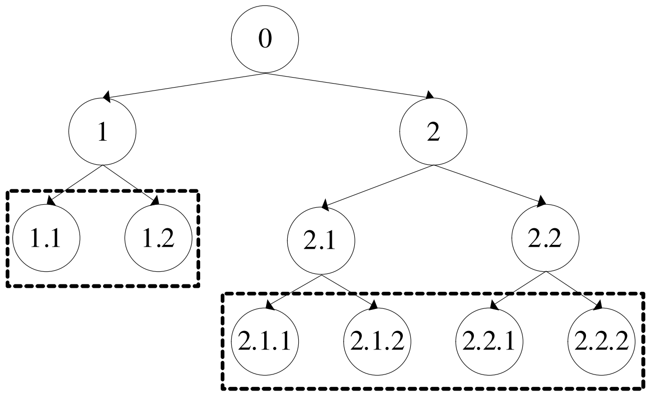


Figure 3.3: Flat classification approach in hierarchical classification. *Circles* represent classes and each *dashed rectangle* encloses the classes predicted by a classifier.

3.2.2 Local Classification (or Top-Down) Approach

This is the most common approach in the domain of hierarchical classification. One or more classifiers are trained for each (*i*) *class node* or (*ii*) *class level* of the hierarchy. This produces a tree of classifiers in the tree structure or a graph of classifiers in the DAG structure. Those classifiers are constructed from "local information". This is also called "top-down approach" (Koller and Sahami 1997) because the training and testing processes proceed in a top-down fashion.

This approach has the advantage in terms of its simplicity, efficiency, and ability to handle the most difficult types of problems that have the characteristics of multi-label, DAG, and NMLNP. Furthermore, the baseline classifier for each class can be adopted from the efficient methods of the simpler domain, multi-label classification. However, there are two main disadvantages. First, inducing a binary classifier on the domain with an excessive number of classes can lead to the circumstance that the negative examples outnumber the positive ones, "highly imbalance of the training data". The second issue is “propagated errors” where misclassified examples at the upper level are propagated down the hierarchy; therefore, this directly affects performance of classifiers at the bottom level, and also the overall system.

In the following, we will explain more details of training data preparation, training, and testing processes. Then related research in this approach will be discussed.

Training Data Preparation Process

Different ways of defining positive and negative training examples for a binary classification method usually give varied classification results. Eisner et al. (2005) and Fagni and Sebastiani (2007) proposed comparative studies among different methods of assigning a training data set. Silla and Freitas (2010) categorized the methods in those studies to five different approaches. Let us explain each of them by giving an example of how to create a set of training data for Class $C_{2.1}$ based on the hierarchy in Figure 3.4. The notations in the explanation are defined by Table 3.1.

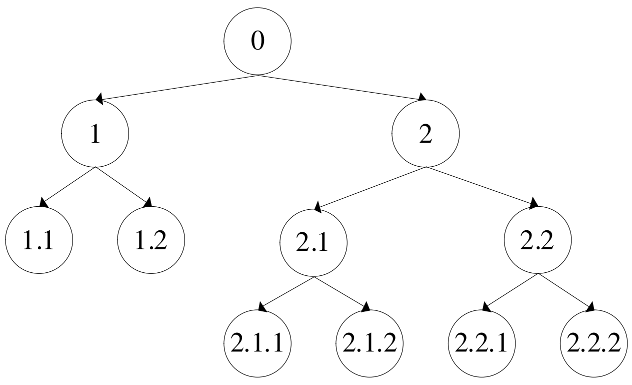


Figure 3.4: A hierarchy for the explanation of the training data preparation process.

Table 3.1: Notations for negative and positive training examples (Silla and Freitas 2010).

Symbol	Meaning
Tr	The set of all training examples
$Tr^+(C_i)$	The set of positive training examples of C_i
$Tr^-(C_i)$	The set of negative training examples of C_i
$\uparrow(C_i)$	The parent classes of C_i
$\downarrow(C_i)$	The children classes of C_i
$\Uparrow(C_i)$	The set of all ancestor classes of C_i
$\Downarrow(C_i)$	The set of all descendant classes of C_i
$\leftrightarrow(C_i)$	The set of sibling classes of C_i
$*(C_i)$	Denotes examples whose "**most specific assigned class**" is C_i
$\backslash$	Denotes as difference, e.g., $Tr\backslash*(C_i)$ represents all examples except C_i

- The "exclusive" policy: $Tr^+(C_i) = *(C_i)$ and $Tr^-(C_i) = Tr\backslash*(C_i)$. The positive examples are only examples explicitly labeled as C_i, while the remaining examples in the database are negative examples. For instance, $Tr^+(C_{2.1})$ consists of examples whose *most specific class* is {2.1}; and $Tr^-(C_{2.1})$ composes of examples of the remaining classes, {1, 1.1, 1.2, 2, 2.1.1, 2.1.2, 2.2, 2.2.1, 2.2.2}.

- The "less exclusive" policy: $Tr^+(C_i) = *(C_i)$ and $Tr^-(C_i) = Tr\backslash\{*(C_i) \cup \Downarrow(C_i)\}$. The positive examples of this policy are similar to those of the exclusive policy, {2.1}, whereas the negative examples do not include examples belonging to the positive class and its descendants. Thus, $Tr^-(C_{2.1})$ is examples in the classes {1, 1.1, 1.2, 2, 2.2, 2.2.1, 2.2.2}.

- The "less inclusive" policy: $Tr^+(C_i) = \{*(C_i) \cup \Downarrow(C_i)\}$ and $Tr^-(C_i) = Tr\backslash\{*(C_i) \cup \Downarrow(C_i)\}$. The set of positive examples also includes examples of its descendant classes, while the rest examples in the database are labeled as negative. For instance, $Tr^+(C_{2.1})$ consists of examples in the classes $\{2.1,\ 2.1.1,\ 2.1.2\}$ and $Tr^-(C_{2.1})$ comprises of examples in the classes $\{1,\ 1.1,\ 1.2,\ 2,\ 2.2,\ 2.2.1,\ 2.2.2\}$.

- The "inclusive" policy: $Tr^+(C_i) = \{*(C_i) \cup \Downarrow (C_i)\}$ and $Tr^-(C_i) = Tr \backslash \{*(C_i) \cup \Downarrow (C_i) \cup \Uparrow (C_i)\}$. The positive examples of this policy are similar to those of the previous policy, but the negative examples exclude any positive examples along with C_i's ancestors and descendants. For instance, $Tr^+(C_{2.1})$ consists of examples in the classes $\{2.1,\ 2.1.1,\ 2.1.2\}$ and $Tr^-(C_{2.1})$ comprises of examples in the classes $\{1,\ 1.1,\ 1.2,\ 2.2,\ 2.2.1,\ 2.2.2\}$, not including $C2$'s examples.

- The "siblings" policy: $Tr^+(C_i) = \{*(C_i) \cup \Downarrow (C_i)\}$ and $Tr^-(C_i) = \leftrightarrow (C_i) \cup \Downarrow (\leftrightarrow (C_i))$. The difference to the previous policy is the set of negative examples that uses examples in C_i's sibling classes and their descendants. For instance, $Tr^+(C_{2.1})$ consists of examples in the classes $\{2.1,\ 2.1.1,\ 2.1.2\}$ and $Tr^-(C_{2.1})$ comprises of examples in the classes $\{2.2,\ 2.2.1,\ 2.2.2\}$.

- The "exclusive siblings" policy: $Tr^+(C_i) = *(C_i)$ and $Tr^-(C_i) = \leftrightarrow (C_i)$. The positive examples are only examples explicitly labeled as C_i, while the examples in the C_i's sibling classes are negative examples. For instance, $Tr^+(C_{2.1})$ consists of $C2.1$'s examples, and $Tr^-(C_{2.1})$ comprises of $C2.2$'s examples.

Concerning which approach should be used, Eisner et al. (2005) concluded that the higher the percentage of positive training data (more inclusive) is, the better the classifier performs. However, the inclusive policy makes the size of training data smaller and can cause in worse results if the training set is *too small.* Their results based on F_1 of classifiers along different policies showed 0.456 in the exclusive policy, 0.528 in the less exclusive policy, 0.696 in the less inclusive policy, and 0.697 in the inclusive policy.

Training Process

There are three standard ways of training classifiers in the local classification approach as follows:

Local classifier per node approach (LCN): This is the most common approach in the literature, including the method proposed in this dissertation. For each class node in the hierarchy (except the root node), a "*binary*" classifier is induced to predict examples whether or not they belong to the class. In Figure 3.5, ten classifiers are induced equal to the number of classes in the hierarchy.

Local classifier per parent node approach (LCPN): This approach generates "*multi-class (multi-label)*" classifiers equal to the number of parent nodes in the hierarchy. The number of outputs for each multi-class classifier is equal to the number of its child nodes. Figure 3.6 illustrates this approach. Note that a classifier at the root node is must be constructed.

Local classifier per level approach (LCL): This approach is commonly used in a tree-structured class hierarchy. For each class level, a "*multi-class (multi-label)*" classifier is induced like in LCPN. Figure 3.7 shows an example of this approach. There are three classifiers induced in the figure. Note that LCL also works with a DAG-structured class hierarchy although the concept of class levels cannot be applied to the DAG structure. If a class node belongs to several class levels, it can be duplicated to be used many times at different class levels.

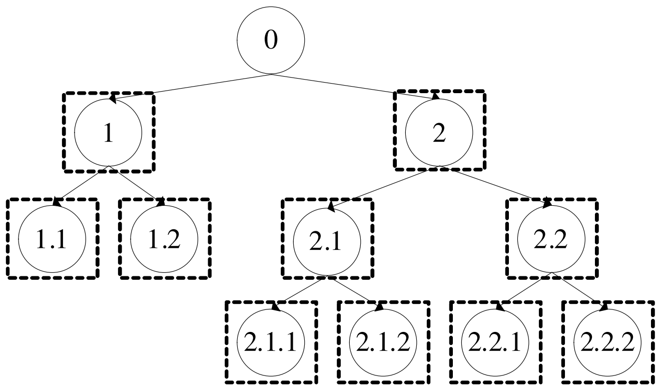


Figure 3.5: Local classifier per node. *Circles* represent classes and each *dashed square* encloses the classes predicted by a binary classifier.

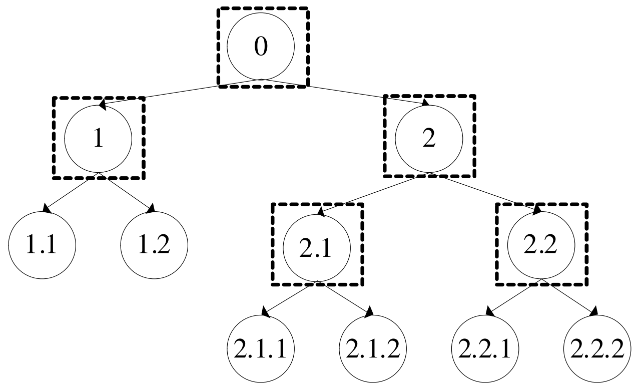


Figure 3.6: Local classifier per parent node. *Circles* represent classes and each *dashed square* encloses the classes predicted by a multi-class (multi-label) classifier.

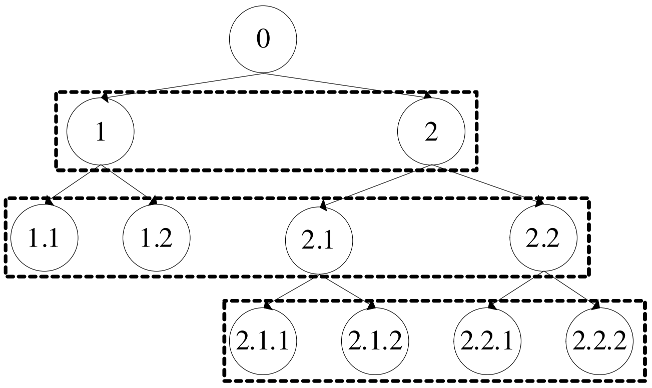


Figure 3.7: Local classifier per level. *Circles* represent classes and each *dashed rectangle* encloses the classes predicted by a multi-class (multi-label) classifier.

Testing Process

Beginning at the root node, an example is classified in a top-down manner. When assigned to one class, the example is then submitted to a new classifier in order to predict to which of this class' subclasses it belongs. This procedure is repeated until a leaf-node class is reached or until *no* additional prediction can be made.

Let us give an example based on the system in Figure 3.5 that each class has its own classifier. Assume that an unknown example is predicted as C_2 but not $C_{2.2}$. Thus, this example does not need to continue classifying at the subclassifiers,$\{2.2.1, 2.2.2\}$.

Taking multi-label classification into consideration, an unknown example is *simultaneously* predicted by many classifiers. For instance, assume that an example is assigned to both $C_{1.1}$ and $C_{2.2.1}$; therefore, the classifiers along both C_1 and C_2 subtrees must be employed to predict this example.

In the DAG hierarchy, the post-process called "a hierarchical criteria correction" must be provided to make the classification results be consistent. Figure 3.8 illustrates the scenario that the correction is necessary because the prediction of Classifier $C_{2.1}$ conflicts to that of its parent classifier, C_1; thus, the result of Classifier $C_{2.1}$ must be corrected to be negative.

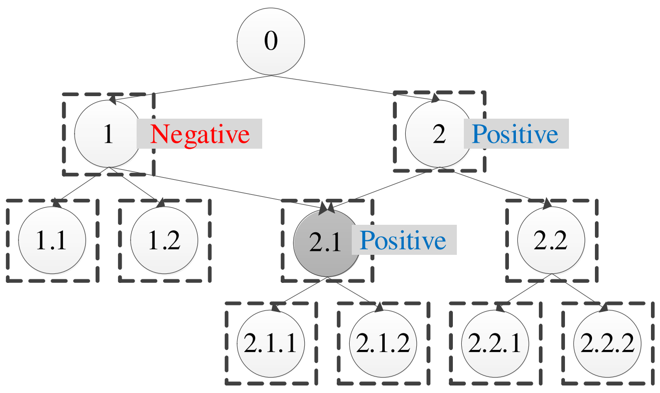


Figure 3.8: The scenario in which the hierarchical criteria correction must be applied. *Circles* represent classes and each *dashed rectangle* encloses the classes predicted by a binary classifier.

3.2.3 Global Classification (or Big-Bang) Approach

This approach constructs one big (global) classification model for all classes of the hierarchy in a "*single*" run of the algorithm as shown in Figure 3.9. It has many advantages: (i) the total size of the global classification model is usually smaller than the total size of all local classification models combined, and (ii) the dependencies among classes in the hierarchy are taken into account. However, there are also some disadvantages: (i) it requires a higher algorithmic complexity than other approaches, and (ii) only a single classifier may not be able to identify unseen data throughout a large number of classes.

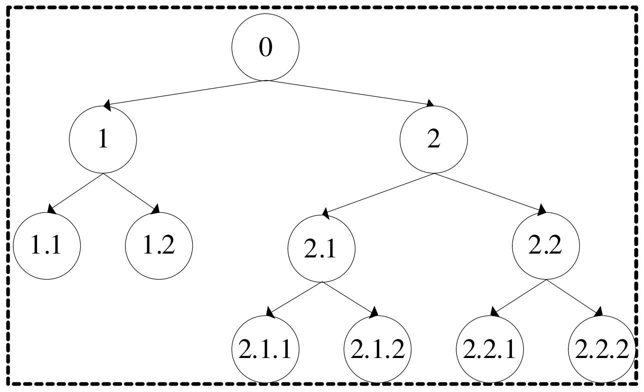


Figure 3.9: Global classification approach. A dashed rectangle represents that this approach uses only a single special classification for the whole class hierarchy.

3.3 Related Work

In this section, we will discuss research work related to the general approaches of hierarchical classification mentioned in Section 3.2.

3.3.1 Related Work on Flat Classification Approach

An ensemble method of a simple Neural Networks called "Protein Feature-Based Prediction" was proposed by Jensen et al. (2002) to predict a class hierarchy of protein functions for the bacteria Escherichia coli (Riley 1993). Optimal combination of parameters for each simple network of the different categories were found using a boot-strap strategy. The experimental result on the category "transport and binding" achieved 90% in terms of sensitivity with 10% of false positive rate.

Then a more complex method using Multilayer Perceptron Neural Networks was invented by Weinert and Lopes (2004) to classify functions and characteristics of enzymes from the Protein Data Bank (PDB) (Berman et al. 2000).

Although those studied showed promising classification performance, they treated each class in the hierarchy independently. Hence, the predictions of one class can conflict with those of other classes regarding the hierarchical constraint.

3.3.2 Related Work on Local (or Top-Down) Classification Approach

Sun and Lim (2001) developed a top-down level-based classification method using binary classifiers, SVM, and experimented on the Reuters collection. The algorithm aimed to support NMLNP and the tree-structured class hierarchy (not the DAG-structured one). For each class node, there are two classifiers: (*i*) local-classifier and (*ii*) subtree-classifier. The *local-classifier* is built to determine whether an example should be assigned to C_i or not. The training data of the first classifier is generated by the siblings policy. The *subtree-classifier* is built to decide whether an example should be assigned to any C_i's subclasses or not in order to support NMLNP. The training data of the second classifier is constructed by the inclusive policy.

The method showed considerably good experimental results along F_1 criteria. However, it is impaired by a concern of the time-consumption for the induction of two classifiers for each class node. Moreover, the subtree classifier does not seem to be necessary. For instance, without using the subtree classifier, if the local classifier predicts an example not belonging to C_i, the system does not need to continue classifying this example to C_i's subclass classifiers to preserve the hierarchical constraint.

Then it was improved by Nguyen et al. (2005) to support a DAG-structured hierarchy by two proposed solutions. First, the "*tree-based solution*" transforms a DAG hierarchy to a tree hierarchy as shown an example in Figure 3.10. Second, the "*DAG-*

based solution" modifies the Sun and Lim (2001)'s method by generating the number of subtree classifiers equal to the number of parent nodes. Experiments were conducted on the Reuters-215878 and AI paper data sets. The results showed that both solutions performed better than the *flat* SVM classification approach. However, the Nguyen et al. (2005)'s method has the same concern of the computational complexity as the Sun and Lim (2001)'s method.

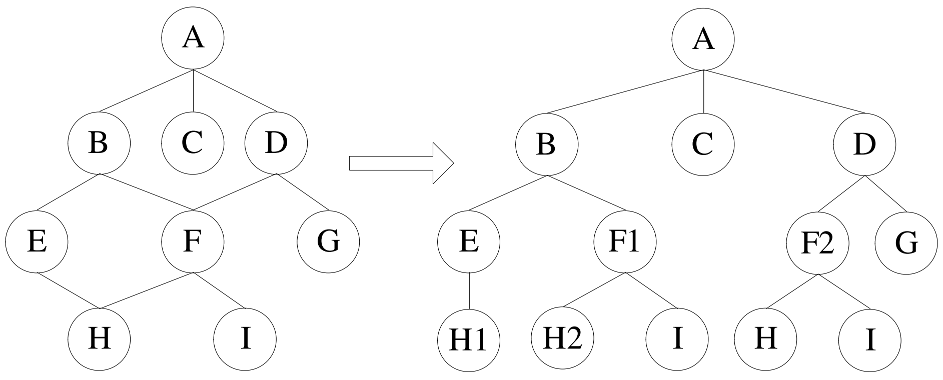


Figure 3.10: DAG-to-Tree transformation (Nguyen et al. 2005).

Secker et al. (2007) proposed a hierarchical classification framework based on the LCPN approach. The authors hypothesize that it would be possible to improve the predictive accuracy by applying heterogeneous classification algorithms at *different* parent nodes of the class hierarchy. There were 10 chosen classifiers: Naive Bayes, Bayesian Network, SMO, 3-NN, and etc. To determine the best classifier for each node, the training set is divided into sub-training and validation sets with examples being assigned randomly to each of those data sets. Different classifiers are trained using the sub-training set and 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. The predictive accuracy at each level of hierarchy was reported and indicated that the selective approach among heterogeneous classifiers outperformed all of those individual methods. However, its induction time must be extremely high.

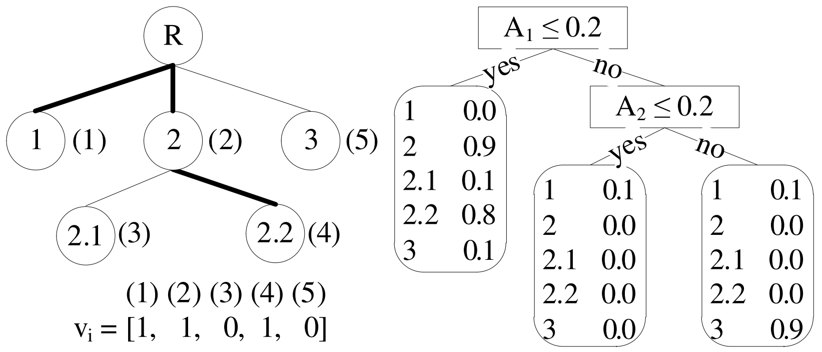
3.3.3 Related Work on Global (or Big-Bang) Classification Approach

Clare and King (2003) proposed a big-bang approach algorithm modified from C4.5 (Quinlan 1986) as shown in Section 2.3.1 to classify functional genomics data. They first introduced a multi-label version of C4.5 (ML-C4.5) as referred in Section 2.4 and then extended it in the hierarchical context. They also preferred more specific classes of the predefined hierarchy and used the weighting concept in the entropy formula to account for this concept.

Blockeel et al. (2006) presented a decision tree based hierarchical classification algorithm called “Clus-HMC” (“Clus” standing for “Clustering tree” and “HMC” standing for “Hierarchical Multi-label Classification”). It is a hierarchical extended version of their own decision tree called “Predictive Clustering Tree (PCT)” (Blockeel et al. 1998). The difference between PCT and other decision trees, such as ID3 and C4.5, is that the PCT’s classification model is constructed using a variance measure instead of the entropy score. Clus-HMC is considered as the global approach since it exploits the given class hierarchy and predicts all classes with a *single* decision tree.

To compute the variance measure in the hierarchical context, the example labels are represented as a vector of Boolean components. In Figure 3.11(a), an example belonging to the classes $\{1, 2, 2.2\}$ is represented by a vector $[1, 1, 0, 1, 0]$. Then the variance of the label vector is computed by the average squared distance between vectors of each examples label v_i and the mean label $\overline{v}$. From these modification, Clus-HMC can generate a decision tree following the same procedure as PCT.

Figure 3.11(b) shows an example of the induced Clus-HMC's tree for the hierarchy in Figure 3.11(a). Each leaf node is represented by the mean $\overline{v}$ of the vectors of its examples. A predefined threshold t is used to interpret predicted classes for each node. If v_i is above the threshold t_i, an example is predicted to belong to class c_i. For instance, if $t = 0.4$ equal to all classes, the left most node of the Clus-HMC tree in Figure 3.11(b) will label examples as C_2 and $C_{2.2}$. The experiments showed that the Clus-HMC tree yield better efficiency and interpretability without suffering a decline in predictive accuracy. However, how the threshold should be chosen was not stated.



(a) A predefined class hierarchy (b) An induced Clus-HMC's tree

Figure 3.11: (a) An example of a class hierarchy with a vector v_i representing an example belonging to the classes $\{1, 2, 2.2\}$, indicating in a bold path. (b) Each leaf of the Clus-HMC's classifier (tree) shows the probability for each class in the hierarchy.

Vens et al. (2008) extended Clus-HMC towards hierarchies structured as DAGs. They showed that inducing one decision tree for simultaneously predicting all functions (the global approach) outperforms learning one tree per function (the local approach). Recently Schietgat et al. (2010) introduced the ensemble version of Clus-HMC called "Clus-HMC-ENS". It constructs multiple trees using a simple bagging (bootstrapping), and then combines the predictions of each classifier. The hierarchical classification system proposed by this dissertation will be compared to Clus-HMC. Although Clus-HMC-ENS outperformed Clus-HMC in terms of "the area under precision and recall measure ($\overline{AUPRC}$)", it was impaired by a large induction time spent on bootstrapping. The suggested number of bootstraps is 50 iterations; this means that the induction time of Clus-HMC-ENS is 50 times that of Clus-HMC.

3.4 Performance Evaluation

In domains with hierarchically organized classes, the classical measures, such as *precision*, *recall*, and F_1, in Section 2.5 appear to be inadequate to fully address the hierarchical nature. Thus, measures that incorporate the problem's hierarchy must be considered. There are several alternatives of hierarchical classification measures, some of which are distance- and semantics-based measures suggested in Sun and Lim (2001) and Costa et al. (2007). However, we omit these measures because acceptable differences and similarities are often specified by users, and, therefore, the results may be subjective and user-specific.

We prefer the use of the metrics proposed by Kiritchenko et al. (2005) as extended versions of, again, *precision*, *recall*, and F_1. In the hierarchical context, each example belongs not only to a class(es), but also to all ancestors in the class hierarchy, excluding the root node. The metrics for the i-th example are summarized in Table 3.2, where P_i and T_i represent sets of predicted and true classes respectively, and

$\hat{P}_i$ and $\hat{T}_i$ stand for P_i and T_i plus their ancestors. *Hierarchical precision* (hPr_i) is an accuracy of the *prediction* path, *hierarchical recall* (hRe_i) is an accuracy of the *true* path, and *hierarchical* F_1 ($hF_{1,i}$) is an equally combined measures of hPr_i and hRe_i. For illustration, referring to the hierarchy from Figure 3.12, let $P_i = \{C2.2\}$ and $T_i = \{C2.2.1\}$. This yields $\hat{P}_i = \{C2, C2.2\}$ and $\hat{T}_i = \{C2, C2.2, C2.2.1\}$, so that $hPr_i = \frac{2}{2} = 1$, $hRe_i = \frac{2}{3}$, and $hF_{1,i} = 0.8$.

Table 3.2: The hierarchical version of *precision*, *recall*, and F_1 for an example i.

Precision	Recall	F_1
$hPr_i = \frac{\lvert\hat{P}_i \cap \hat{T}_i\rvert}{\lvert\hat{P}_i\rvert}$	$hRe_i = \frac{\lvert\hat{P}_i \cap \hat{T}_i\rvert}{\lvert\hat{T}_i\rvert}$	$hF_{1,i} = \frac{2 \times hP \times hR}{hP + hR}$

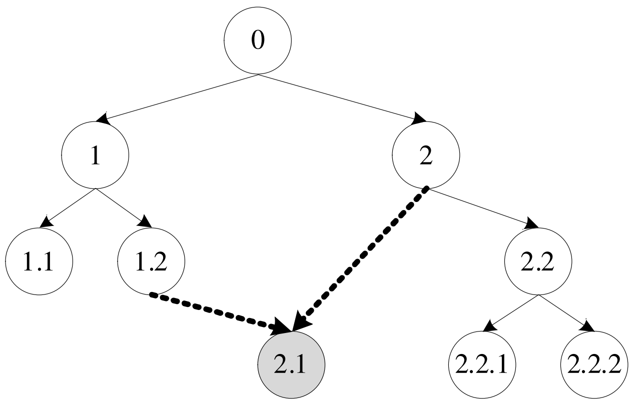


Figure 3.12: An example of a DAG-structured hierarchy.

Let us now compute the hierarchical performance of the data set with n examples labeled by l classes. The authors then chose to combine the performance of all examples by *micro-averaging*, which is shown in Table 3.3. However, this combining method tends to be biased towards examples with *longer* paths.

Table 3.3: The micro-averaging version of *hierarchical precision*, *hierarchical recall*, and *hierarchical* F_1 of n examples.

Precision	Recall	F_1
$hPr^{\mu} = \frac{\sum_{i=1}^{n} \lvert \hat{P}_i \cap \hat{T}_i \rvert}{\sum_{i=1}^{n} \lvert \hat{P}_i \rvert}$	$hRe^{\mu} = \frac{\sum_{i=1}^{n} \lvert \hat{P}_i \cap \hat{T}_i \rvert}{\sum_{i=1}^{n} \lvert \hat{T}_i \rvert}$	$hF_1{}^{\mu} = \frac{2 \times hP \times hR}{hP + hR}$

To see why, consider the following data set where the true labels are as follows: $\mathcal{S} = \{(x_1, C2.2), (x_2, C1), (x_3, C2)\}$. Let the classes be organized according to the hierarchy from Figure 3.12. Suppose there are two classifiers, Φ_1 and Φ_2. Their prediction results on the data set $\mathcal{S}$ are shown in the following list. Note that the underline represents a correctly predicted class by a classifier.

- $\Phi_1(\mathcal{S}) = \{(x_1, \underline{C2.2}), (x_2, C2), (x_3, C1)\}$ (correctly classifying x_1).
- $\Phi_2(\mathcal{S}) = \{(x_1, C1.1), (x_2, \underline{C1}), (x_3, \underline{C2})\}$ (correctly predicting x_2 and x_3).

The hierarchical evaluation for *each example* (hPr, hRe, and hF_1) of those above classifiers on the data set $\mathcal{S}$ is shown in Table 3.4. In this scenario, it is obvious that Φ_2 is better than Φ_1, but the micro-averaging (the performance of *all examples*) indicate that both classifiers perform the same, $hPr^{\mu} = hRe^{\mu} = hF_1{}^{\mu} = \frac{1}{2}$.

Table 3.4: Hierarchical performances for each example (Table 3.2) of classifiers Φ_1 and Φ_2 on the data set $\mathcal{S}$.

Classifier	Example	Precision (hPr)	Recall (hRe)	hF_1
$\Phi_1(\mathcal{S})$	x_1	$\frac{2}{2} = 1$	$\frac{2}{2} = 1$	1
	x_2	$\frac{0}{1} = 0$	$\frac{0}{1} = 0$	0
	x_3	$\frac{0}{1} = 0$	$\frac{0}{1} = 0$	0
$\Phi_2(\mathcal{S})$	x_1	$\frac{0}{2} = 0$	$\frac{0}{2} = 0$	0
	x_2	$\frac{1}{1} = 1$	$\frac{1}{1} = 1$	1
	x_3	$\frac{1}{1} = 1$	$\frac{1}{1} = 1$	1

To reasonably evaluate the classifiers in the previous scenario, we proposed to apply *macro-averaging* to merge hierarchical measures of all examples as shown in Table 3.5. This averaging method then evaluates the performance of Φ_2 higher than that of Φ_1 as follows:

- $\Phi_1(\mathcal{S}) : hPr^M = hRe^M = hF_1{}^M = \frac{1}{3}$.
- $\Phi_2(\mathcal{S}) : hPr^M = hRe^M = hF_1{}^M = \frac{2}{3}$.

Table 3.5: The macro-averaging version of *hierarchical precision*, *hierarchical recall*, and *hierarchical* F_1 of n examples.

Precision	Recall	F_1
$hPr^M = \frac{\sum_{i=1}^{n} hPr_i}{n}$	$hRe^M = \frac{\sum_{i=1}^{n} hRe_i}{n}$	$hF_1^M = \frac{\sum_{i=1}^{n} hF_{1,i}}{n}$

3.5 Conclusion

This chapter addresses the more advanced version of the classification problem where the classes are interrelated so that some classes are generalizations of others. We call this the hierarchical classification task. The goal is not only to induce a classifier seeking for the optimized performance, but also the classification result must pursue "*the hierarchical constraint*": an example does not just belong to some classes, but also their superclasses. In this domain, the class hierarchy can be either tree or directed acyclic graph (DAG). Moreover, when examples must be assigned to classes at the leaf nodes, this kind of problem is called "a mandatory leaf node problem (MLNP)". When this obligation does not hold, the problem is called "a non-mandatory leaf node problem (NMLNP)". The target problem of this dissertation has the characteristics of multi-label, DAG, and NMLNP.

Existing hierarchical classification methods can be grouped into three main approaches: (*i*) flat classification approach, (*ii*) local classification (or top-down) approach, and (*iii*) global (or big-bang) approach. All of their concepts and related works are presented. The proposed works in this dissertation follow the most common approach, the top-down one which generates a separate classifier for each class, and then proceeds its training and testing from the classifiers at upper levels down to bottom levels.

To evaluate hierarchical classifiers, the suggested metrics are an extended version of the classical measures. For the example x_i, the metrics are *hierarchical precision* (hPr_i), *hierarchical recall* (hRe_i), and *hierarchical* F_1 ($hF_{1,i}$). Then there are two alternative ways to combine those criteria on *all examples*: (*i*) *macro-averaging* and (*ii*) *micro-averaging*. The first averaging method gives equal weight to all examples while the latter weighs the examples according to their path length.

CHAPTER 4

Learning from Large Scale, Imbalanced, and Multi-Label Domains

This chapter shows the research in domains of multi-label classification. In domains of this kind, two general paradigms have been explored: (*i*) induction of one large multi-label classifier and (*ii*) induction of sets of binary classifiers, each classifier for a different class label. In our research, we opted for the latter approach, the most commonly used one. It was illustrated by (Rifkin and Klautau 2004) that it is as accurate as any other learning strategy if the baseline classifiers are well-tuned regularized classifiers such as Support Vector Machine (SVM).

The first work (Vateekul and Kubat 2009) was developed to handle *the prohibitive computational cost* of decision-tree induction in text-categorization domains which are usually described by ten thousands of features. Specifically, we developed a system called "fast decision-tree induction (FDT)". Our experiments showed that FDT performed better than other traditional classifiers, such as C4.5 and SVM.

Later we found that the performance of SVM, which suffers from the induction on *the imbalanced training set*, can be significantly improved by applying the threshold adjustment strategy. We were motivated to study this approach and developed a new thresholding method called "R-SVM" in the second work (Vateekul et al. 2011). Moreover, the experiments included more domains of multi-label data, i.e., text, biology, music, image, and video.

The chapter is organized as follows. Section 4.1 explains details of multi-label data in our research. The concepts and experimental results of FDT and R-SVM are shown in Section 4.2 and Section 4.3 consecutively. All our works are summarized in Section 4.4.

4.1 Multi-Label Data Sets

This chapter reports six multi-label data sets from real-world applications used in the experiments. Two text-categorization data sets (`EUROVOC` and `RCV1-v2`) are from our own earlier research (Sarinnapakorn and Kubat 2007). The remaining four data sets (`Yeast`, `Emotions`, `Scene`, and `MediaMill`) are available at the website multi-label data sets[8].

The characteristics of the data sets are summarized in Table 4.1. Considering the percentage of the positive class, the class distribution of each data set can be categorized into 3 categories: minority, equal, and majority. For instance, almost all classes of `EUROVOC` have the positive class as a minority. It has the percentage of the positive class between 0% and 55.22% with only 15.01% average. Conversely some classes of `Yeast` and `MediaMill` have the positive class as a majority, more than 70% of total data. More details of all data are shown as follows:

[8] `http://mulan.sourceforge.net/datasets.html`

Table 4.1: The characteristics of data sets in multi-label domains.

Dataset	Domain	Features	Classes	Instances	% Positive Examples Per Class		
					Average	Min	Max
`EUROVOC`	text	4,000	30	10,000	15.01	0	55.22
`RCV1-v2`	text	4,000	101	6,000	2.87	0.02	44.55
`Yeast`	biology	103	14	2417	30.28	1.43	75.13
`Emotions`	music	72	6	593	31.15	24.97	44.55
`Scene`	image	294	6	2407	17.89	15.07	22.11
`Mediamill`	video	120	101	43,907	4.33	0.11	77.11

`EUROVOC`[9] is a multilingual, polythematic thesaurus created by virtue of close cooperation of the European Parliament, the European Commissions Publications Office, and the national organizations of the European Union (EU) member states. The documents in this collection come from such diverse fields as of interest to the activities of the European institutions such as politics, law, economics, trade, and etc.

We were given an access to a part of `EUROVOC` classification system consisting of 78,599 documents, each described by 105,355 numeric features (each specifying the relative frequency of a different word), and known to belong into a subset of more than 5,000 different fields (class labels) in hierarchical form. For this work, we used only the top-level labels composing of 30 classes. Among 78,599 documents, 10,875 documents are not assigned to any classes. Excluding unlabeled documents leaves us with 67,724 documents in 5,452 fields. The file size of unprocessed data where all data with value 0 are omitted is about 3 gigabytes (GB). After filling necessary data values, the total size of data files becomes more than 16GB. We therefore worked

[9]The websites `http://europa.eu/eurovoc/` and `http://langtech.jrc.it/Eurovoc.html` are sources of our version. It is somewhat different from the one available on `http://mulan.sourceforge.net/datasets.html`.

with a simplified database (Sarinnapakorn and Kubat 2007) containing only 10,000 documents described by 4,000 features and labeled with 30 different classes.

`RCV1-v2` is a benchmark collection of news articles made available by Reuters Ltd. The original version is called Reuters Corpus Volume 1 (`RCV1`). Then, (Lewis et al. 2004a) processed it and called their version as `RCV1-v2` with 804,414 documents and 47,236 features. There are several schemes to process the documents, including (*i*) removing stopping words (unimportant words), such as a, an, the, etc., (*ii*) stemming: change words to their root form (stem), such as identifying "cats" as its root "cat", and (*iii*) transforming the documents to vectors with TF-IDF format, and etc. Each instance is cosine normalized. There are three category sets: *Topics*, *Industries*, and *Regions*. In our research, we consider the *Topics* category set. There are 23,149 training and 781,265 testing instances. For labels, 101 appear in the training set and 103 appear in the testing set. For the sake of the study, we chose the subset data[10] which contains 6,000 documents on 5 different subsets. However, our database version (Sarinnapakorn and Kubat 2007) is even more simplified by using only 4,000 features randomly selected from those that have non-zero values in at least more than 5 documents.

`Yeast` is a data set formed by micro-array expression data and phylogenetic profiles from Munich Information Center for Protien Sequences (MIPS). The version on the multi-label website (Elisseeff and Weston 2001) has 2,417 genes described by 103 features. Each gene is associated with a set of hierarchical functional classes with potentially more than 190 functions. However, this data set considers only 14 functional classes on the top level.

[10]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html

`Emotions` is a data set provided by Trohidis et al. (2008) for the automated detection of emotion in music. There are 593 songs with 6 clusters of music emotions comprising of amazed-surprised, happy-pleased, relaxing-calm, quiet-still, sad-lonely, and angry-fearful. The total number of audio features is 72 features falling into 2 categories: 8 rhythmic features and 64 timbre features.

`Scene` is an image data used in the experiment of semantic scene classification. Boutell et al. (2004b) prepared this data with 2,407 images associated with six different semantic scenes (beach, sunset, fall foliage, field, urban, and mountain). Each image can be described by multiple class labels (e.g., a field scene with a mountain in the background). Features are obtained by converting each image into L*U*V space, dividing to 49 blocks, and computing the first and second moments (mean and variance). So, the total number of features is 249 features ($3 \times 49 \times 2$).

`MediaMill` is a generic video indexing data set provided by the MediaMill research group (Snoek et al. 2006). The original data uses 85 hours of video from the 2005 NIST TRECVID benchmark (i.e. the TRECVID 2005 training set), containing news sources, recorded in MPEG-1 during November 2004 by the Linguistic Data Consortium. The objective is to automatically detect semantic concepts for each video shot. There are 43,907 video shots described by 120-dimensional visual feature vectors on 101 semantic concepts, such as people, face, outdoor, sky, and etc.

4.2 Classifier Induction from Decision Trees

This section discusses the details of our first research in text categorization. It was inspired by `EUROVOC`, a large scale text corpus which is described by tens of thousands of features, which, of course, leads to extreme induction costs. To help reduce these costs, the system, called "fast decision-tree induction (FDT)," was developed. It identifies a subset of features likely to contribute to the classifier's accuracy and discards the other features; then, a set of decision trees is induced from subsets of the training examples. In the classification phase, the output class is obtained by combining the results of the induced decision trees.

The idea to apply decision trees to text categorization appeared as early as in 1994 (Yang 1999). As of now, most decision-tree generators rely on memory-resident data, and often are prohibitively expensive when dealing with very large data sets. Perhaps the best-known exception is SLIQ (Mehta et al. 1996) which eliminates a major part of the (sometimes repetitive) calculations by the use of two pre-sorted-data structures: *class list* and *attribute list*—several improvements have been proposed, among them SPRINT (Shafer et al. 1996) that does not use the *class list* and some other techniques that rely on various heuristics such as those searching for optimum splits of numeric attributes (Chandra et al. 2002; Chandra and Varghese 2008). Efforts along these lines improve the scalability of decision-tree induction (Quinlan 1993; Quinlan 1996), but they are impaired by a high complexity in terms of implementation. To avoid this complication, Quinlan's $C4.5$ (Section 2.3.1), the most widely used decision tree, was chosen to use in FDT because of its promising classification accuracy and reproducibility of the experiments since it is easy to obtain from his public-domain software[11].

To create a multi-label classifier on the EUROVOC data set, induction of neither a single multi-label classifier nor a set of binary classifiers can handle it due to the

[11]http://www.rulequest.com/Personal/

very large computational costs; methods to reduce them need to be found. This is why we developed the FDT system. The concept and the implementation of FDT will be explained in Section 4.2.1 and Section 4.2.2, and then its results of extensive experiments in terms of computation time and classification accuracy will be reported in Section 4.2.3. Finally, Section 4.2.4 will provide the comparison to other well-known classification algorithms, such as SVM.

4.2.1 FDT Concepts

FDT handles multi-label data by inducing a binary classifier which is $C4.5$ for each class separately. To reduce the computational costs of tree induction, FDT employs "feature pre-selection" and "data partitioning," the latter seeking to induce *a set of decision trees* from different subsets of the training data. The mechanism to combine the results of these decision trees uses the fact that they are induced from data with imbalanced classes, where the negative examples outnumber the positive ones.

Feature pre-selection. In preliminary experiments, we worked with a simplified version of the EUROVOC collection (Sarinnapakorn and Kubat 2007) with 4,000 features. We found that none of the induced trees used more than 700 features; this indicates that many features are unnecessary, and their removal is thus unlikely to impair the induced classifiers' performance.

When building the decision tree, C4.5 calculates at each node the *gain ratio* for every feature. The complexity of this process is $O(fn)$ where f is the number of features and n is the number of nodes. Thus for 1,000 nodes, we need to compute about 4 million *gain ratios* (4,000 features $\times$ 1,000 nodes). To reduce these computational costs, we remove $r\%$ of the lowest-*gain ratio* features prior to decision-tree induction. The number of chosen $r\%$ will be discussed in the experiments.

Data partitioning. The computation time of decision tree induction grows supralinearly in the number of training examples. Using a smaller training set can thus significantly reduce computational costs—in our preliminary experiments, CPU usage dropped to only about 5% when we reduced the training set from 8,000 to 2,000 examples.

Decision tree algorithms typically follow two steps: tree construction and pruning. Eliminating the pruning phase saves additional costs. Preliminary experiments indicated that the performances of unpruned tree and pruned tree did not differ much on small data samples.

Data Fusion. FDT partitions the training data, and then induces a separate decision tree for each partition. What is needed is a mechanism that these trees use to vote about the final outcome. Traditionally, three data fusion techniques have been used. (1) *Best classifier*: predict the class recommended by the classifier that is (heuristically) deemed to have the highest chance of being correct. (2) *Plain majority voting*: each classifier has the same weight and the system outputs the class that receives the most votes, breaking the ties in favor of classes more frequently represented in the training set. (3) *Weighted majority voting*: confidence scores are used to weigh the voting classifiers.

The fusion method we used in FDT is called *One-Vote* and is summarized by the pseudo code in Table 4.2. Since the number of positive examples is small, the induced trees rarely predict the positive class. To increase the probability of the positive outcome, FDT is biased toward the minority class, using the rule, "the output class is positive if at least one of the subclassifiers said so." The strategy is simple and performs better than other data fusion techniques.

4.2.2 The FDT Program

The FDT technique is summarized by the schema in Figure 4.1 and the pseudocode in Table 4.2. The program consists of three main modules. The first, "data partitioning," divides the data into N non-overlapping, equally-sized subsets, each maintaining the same ratio between positive and negative classes. The second module, "subclassifier construction," uses C4.5 to induce a subclassifier for each subset. To accelerate induction, feature pre-selection removes $r\%$ attributes before the tree induction begins. The last module implements the "data fusion" for the classification stage.

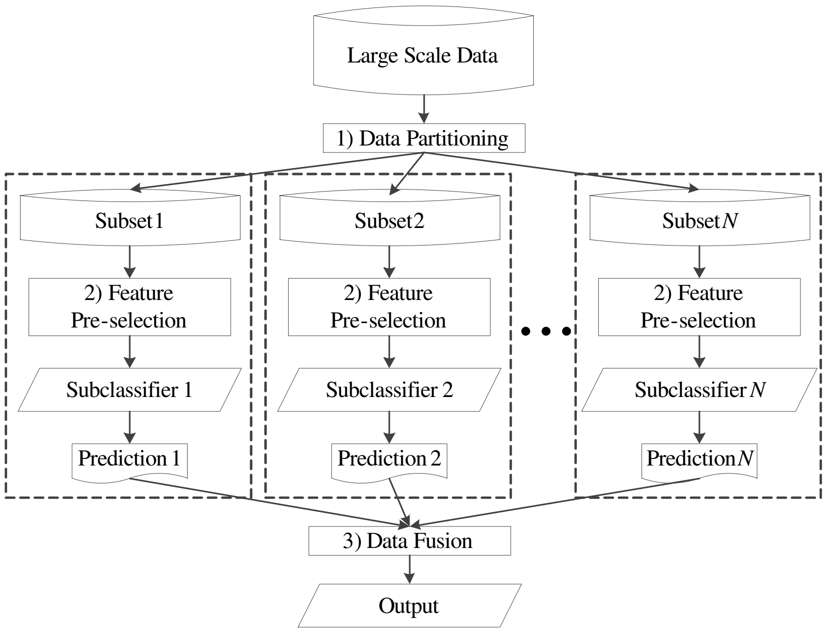


Figure 4.1: An essential framework of FDT.

Table 4.2: The pseudocode of FDT

```
MAIN($train, test, N, r$)
 1  ▷ Training Process
 2  ▷ 1) Data partitioning
 3  $ratio \leftarrow$ GET_RATIO($train$);
 4  $subtrain \leftarrow$ PARITION($train, ratio, N$);
 5
 6  ▷ 2) Subclassifier construction
 7  for $i \leftarrow 1$ to $N$
 8        do $subtrain[i] \leftarrow$ REMOVE_ATT($subtrain[i], r$);
 9           $subtrees[i] \leftarrow$ C4.5($subtrain[i]$);
10
11  ▷ Testing Process
12  ▷ 3) Data fusion from all subclassifiers
13  for $j \leftarrow 1$ to SIZE($test$)
14        do $output[j] \leftarrow$ ONEVOTE($subtrees, test[j]$);

ONEVOTE($subtrees, unknown$)
1  for $i \leftarrow 1$ to SIZE($subtrees$)
2        do $pclass \leftarrow$ EVAL($subtrees[i], unknown$);
3           if $pclass = minor\_class$
4              then break;
5  return $pclass$;
```

4.2.3 Experiments and Discussion

We applied FDT's to a our version of `EUROVOC` database (Sarinnapakorn and Kubat 2007): 10,000 documents described by 4,000 features, each document being labeled by 30 classes. We simplified our work by conducting experiments on the first five class labels to reduce experimental time. All graphs in this section were obtained as averages from 3-fold cross-validation. The statistical significance of the differences in performance has been checked by the paired t-test with confidence level 95%.

The task for the first experiment was to find out how many features should be removed during the feature pre-selection stage. We experimented with four alternatives: 0%, 25%, 50%, and 75% features removed. Figure 4.2 shows the results in terms of CPU time and tree size. The reader can see that the removal of 75% attributes saved more than 50% time. The reader may be surprised that the removal of 25% attributes actually increased the costs. The reason is the overhead time of the calculations of gain rations and on sorting these numbers.

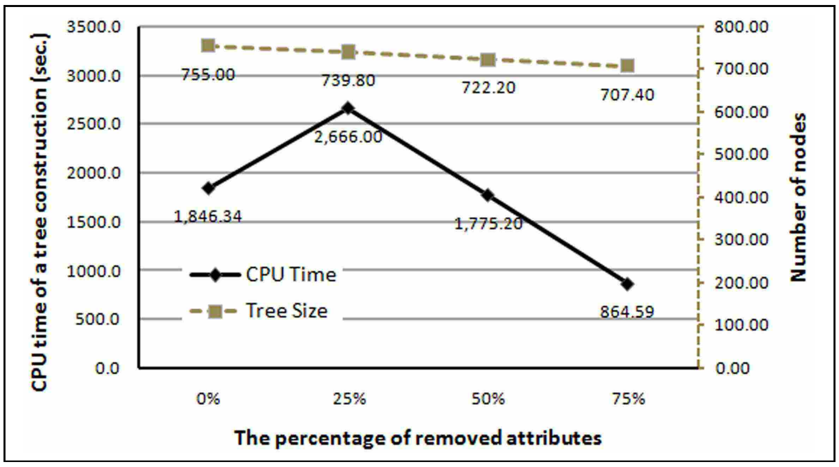


Figure 4.2: CPU time and size of trees with varied percentages of removed attributes.

Figure 4.3 shows that feature pre-selection did not affect classification performance: even with the removal of 75% features prior to induction, the performance of the induced classifier did not drop more than 0.1 in terms of F_1, yet we gained more than 50% in computation costs.

One of the most important aspects of FDT is that it induces, for a given class, a set of decision trees in which each from a different subset of the training set. When used to classify a testing example, these trees combine their outputs. The motivation is simple: the costs of decision tree induction grow supralinearly in the number of examples; therefore, inducing N decision trees, each from one $\frac{1}{N}$ of the data is much

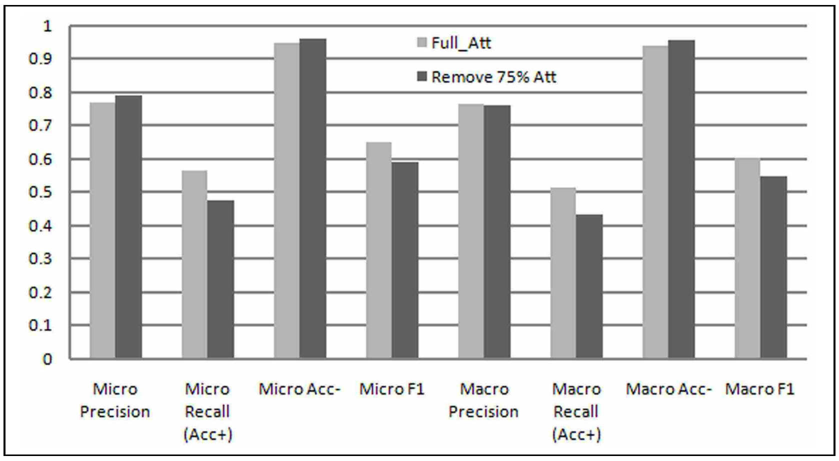


Figure 4.3: A comparison in the performance of varied percentages of removed attributes using the feature pre-selection mechanism.

faster (even if they are induced serially, one after another) than the induction of a single tree from the entire training set.

The task for the second experiment is to put this conjecture to test. We measured the CPU time needed to induce decision trees from training sets of different sizes. In particular, we used 1000, 2000, 4000, 6000, and 8000 examples, and measured the induction time separately for the case with all features and for the case with 25% most promising features. The results are depicted in Figure 4.4. For instance, if 2000 examples are used, induction is one hundred times faster than when the entire training set is used. The graph also shows that the computational costs grow more slowly in the case where only 25% features are used.

Thus encouraged, we proceeded to the next experiment whose task was to establish how much the computational costs depends on the number of subclassifiers. The results are shown in Table 4.3 and Figure 4.5, again separately for the case where all features were used and for the case where 75% features were removed prior to induction. The first row gives the CPU times of the induction of a single decision tree from the entire training set; the second row gives the CPU times for the induction of three decision trees, each from one third of the training examples; the third row gives the CPU time for the induction of five decision trees, each from one fifth of the examples. Note that, in the case of removal of 75% features, only about 1% of the CPU time for the whole single tree induction is needed.

Importantly, we must make sure we do not pay for the impressive time reduction by unacceptably compromised classification performance. This was the task for the final experiment. Figure 4.6 provides detailed results along the *micro* and *macro* averaging performance criteria. In the experiments, we always induced three decision trees from three non-overlapping training subsets. Testing examples were submitted to all trees in parallel, and the results were combined by the fusion mechanisms discussed in Section 4.2.1. Then we induced five decision trees from five non-overlapping training subsets and repeated the experiment. Each graph represents a different performance criterion; in all cases, four different ways to combine the decision tree outputs are plotted: majority voting versus the *"One-Vote"* method, each plotted separately for the case with all features and for the case with 75% features removed.

All in all, the reader can see that the majority-vote mechanism performs better on negative examples; on the positive examples, the *One-Vote* methods is better. For the needs of the text-categorization domain, high rate on the positive examples is of course more important if we want to deal with the problem of imbalanced classes. The *One-Vote* mechanism also statistically outperforms the majority-vote along the F_1-criterion.

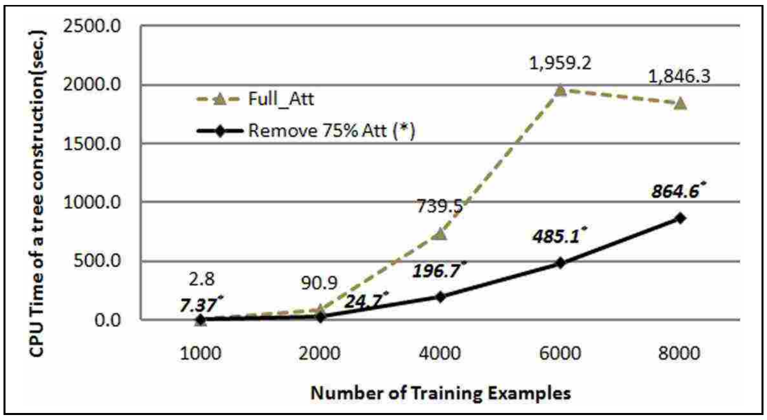


Figure 4.4: CPU time on varied sizes of data.

Table 4.3: CPU time on varied numbers of subclassifiers

	Max. Building Time (sec.)	
#Subsets	Full Att.	Remove 75%
1	1,846.3 ± 48.2	864.6 ± 31.7
3	119.7 ± 8.9	32.7 ± 0.3
5	31.7 ± 1.0	8.8 ± 0.2

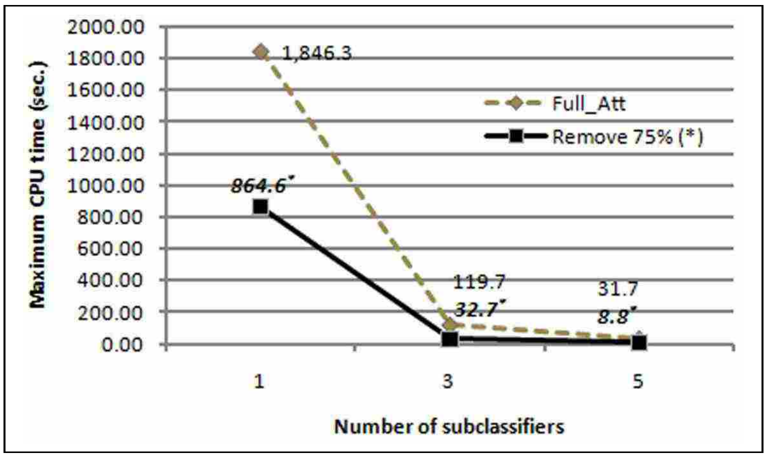


Figure 4.5: CPU time on varied numbers of subsets.

In conclusion, we can say that, although the F_1-results are somewhat better when a single decision tree is built, the multiple-decision-tree solution still deserves attention: the modest loss in classification performance is more than compensated by the large reduction of induction costs. Especially when 75% of features removed and 5 subtrees induced, the induction time was dropped to only 8.8 seconds from 1,849.3 seconds comparing to that of the single large decision tree.

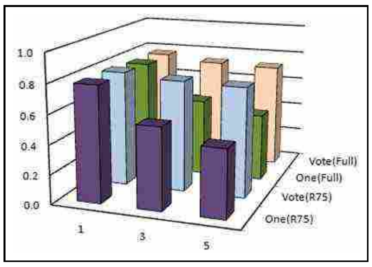
4.2.4 Performance Comparison to SVM

The objective of FDT is to present a way to induce a decision tree on large scale data which require a high computational cost on the traditional decision algorithm. As shown in the results in the previous section, it achieved the goal by significantly reducing induction time spent, but somewhat decreasing in accuracies.

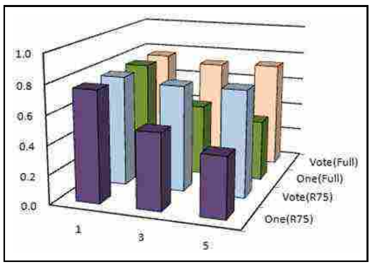
Is FDT superior to other classification techniques, such as SVM? To find the answer, experiments of SVM following the previous section were conducted. Note that we induced the SVM by using the publicly available package *svmlight*[12] (Joachims 1999). Table 4.4 shows the comparison among the single decision tree, FDT, and SVM. The single decision tree outperformed the others in terms of accuracy, but suffered the prohibitive computational cost. FDT showed the most compromised result because its induction time was the most promising although its accuracy was less than that of the single tree, but it was still as good as that of SVM.

However, this comparison also motivated us to develop an alternative solution based on SVM as shown details in Section 4.3. Although the accuracies of SVM were less than those of the single tree, its induction time was still impressive, only about

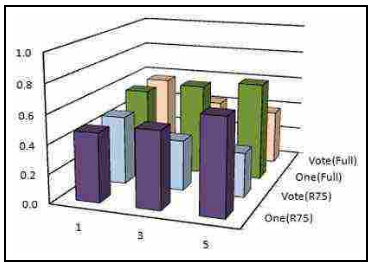
[12]`http://svmlight.joachims.org/`



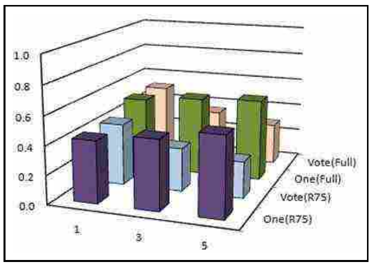
(a) Micro Precision



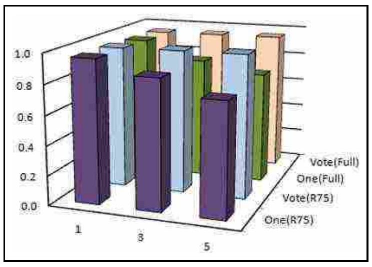
(b) Macro Precision



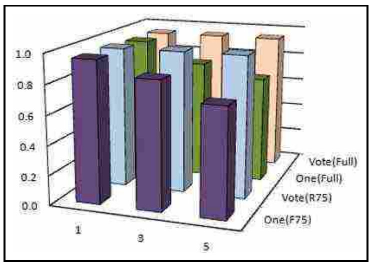
(c) Micro Recall



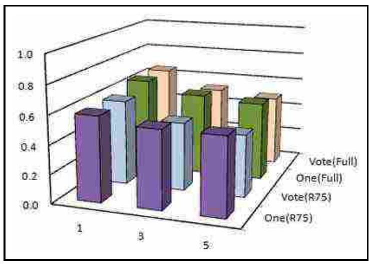
(d) Macro Recall



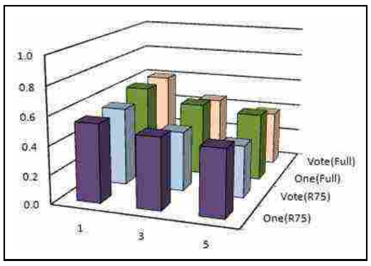
(e) Micro True Negative Rate



(f) Macro True Negative Rate



(g) Micro F1



(h) Macro F1

Figure 4.6: A comparison in the performance of varied numbers of subclassifiers. X-axis represents the number of subclassifiers and Y-axis represents measures in performance criteria.

Table 4.4: Performance comparison among the baseline decision tree, FDT, and SVM.

Algorithms	Micro. F_1	Macro F_1	Induction time for each class in seconds
Baseline Decision Tree	0.65 ± 0.019	0.60 ± 0.003	$1{,}846.3 \pm 48.2$
FDT: 5 subsets with 25% features using *"One-Vote"*	0.54 ± 0.002	0.47 ± 0.002	8.8 ± 0.2
SVM	0.54 ± 0.015	0.46 ± 0.017	137.6 ± 4.8

2 minutes (137.6 seconds). So, we believed that the accuracies of SVM can further be improved by using some modifications at acceptable extra computation costs.

4.3 Classifier Induction from Optimized Support Vector Machine

This section demonstrates our extended research based on SVM in domains of multi-label classification by induction of a separate binary classifier for each class. SVM has been successfully applied in solving the classification problems offering superior performance. However, its success is often impaired by the phenomenon known as *imbalanced training set* – negative examples outnumbering the positive ones or the other way round.

In a typical implementation, SVM first maps the training examples to a space where the classes have a higher chance of being linearly separable, and then finds the parameters ($\vec{w}$ as orientation and b as translation) of a hyperplane to accomplish this separation. When doing so, SVM seeks to maximize a margin separating between two classes while minimize the probability of future training examples being misclassified.

This, however, means to minimize *the error rate* which can be a highly misleading indicator when it comes to classification performance in domains with imbalanced class representation. Consider the case where 1% training examples are positive, and all the remaining 99% are negative. A classifier that labels every single example as negative will exhibit the impressively low error rate of 1%, and yet it is virtually useless on account of being unable to identify a single positive example. This is why many researchers prefer to work here with performance metrics borrowed from the field of information retrieval: *precision*, *recall*, and F_β . It stands to reason that a classifier induced in a way that optimizes one criterion may disappoint when evaluated along another; and indeed, SVM's behavior in multi-label domains is usually good in terms of error rate, but often leaves a lot to be desired when measured by, say, F_β.

To alleviate this problem, three basic strategies have been employed: (*i*) *resampling* (undersampling and oversampling), (*ii*) *weighting* (each class associated with different misclassification costs), and (*iii*) *thresholding* (manipulating the offset of the separating hyperplane). Studies comparing these methods have shown that the thresholding strategy often outperformed the other two (Brank et al. 2003; Sun et al. 2009). The reason is that the resampling strategy changes the distribution of training data that can lead to a declined orientation of the hyperplane and the weighting strategy is, in fact, equivalent to the oversampling approach. While the thresholding strategy directly optimizes a bias of the hyperplane without changing any characteristics of the training data.

But even in the field of threshold adjustment, the results are not ideal because existing methods suffer from high classification costs and suboptimal classification behavior (Brank et al. 2003; Li et al. 2008; Goertzel and Venuto 2006; Lewis et al.

2004b; Peter et al. 1998; Shanahan and Roma 2003; Yan et al. 2009; Imam et al. 2006). Seeking an improvement, we therefore developed a new technique, R-SVM, that in our experiments compared favorably not only with the baseline SVM, but also with such thresholding algorithms as SVM$_{F_1}$ (Brank et al. 2003), SVM$_{CV}$ (Brank et al. 2003), *ScutFBR* (Lewis et al. 2004b), and *BetaGamma* (Peter et al. 1998; Shanahan and Roma 2003).

This section is organized as follows. Section 4.3.1 gives an overview of the thresholding techniques. The concepts and details of the developed framework are presented in Section 4.3.2. The experiments and discussion are reported in Section 4.3.3.

4.3.1 Threshold Adjustment

The thresholding strategy, which is also called "threshold adjustment" or "threshold relaxation", is a post-process to translate the hyperplane, $h(\vec{x}) = f(\vec{w}, b) = \vec{w} \cdot \vec{x} + b = 0$, (without changing the orientation) by the adjustment of the bias b as indicated by Figure 4.7. The assumption behind this strategy is that the orientation of the hyperplane ($\vec{w}$) is already in the best direction and does not needed to be altered. The goal illustrated by Formula 4.1 is to search for the best threshold θ that shows the highest performance function $perf$, such as the F_1-criterion, given the data set S mapped onto the SVM space, $L = \{(s_1, y_1), ..., (s_n, y_n)\}$, and all possible thresholds, Θ. The SVM hyperplane is then updated to $h^*(\vec{x_i}) = h(\vec{x_i}) - \theta$.

$$\{\theta \in \Theta | \theta = \max(perf(L, \Theta))\} \tag{4.1}$$

Brank et al. (2003) and Sun et al. (2009) showed that this technique can improve classification performance at acceptable computational costs. Several methods to find

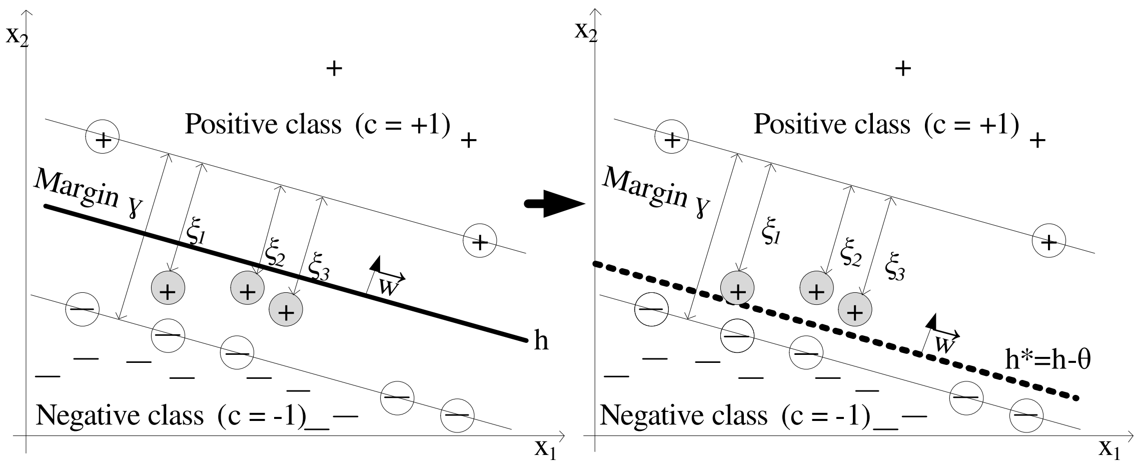


Figure 4.7: SVM hyperplanes before (left) and after (right) threshold adjustment. The classification of three examples is corrected.

an optimum threshold have been proposed. The approaches advocated by Yan et al. (2009) and Li et al. (2008) rely on the Gaussian distribution assumption, with the obvious limitation that the behavior is suboptimal in the case of other distributions. A more general approach is presented by Goertzel and Venuto (2006) who used a simple heuristic to estimate the cut-point between positive and negative distributions, but they did not address the case where the number of cut-points is more than one.

Brank et al. (2003) compared among different thresholding algorithms such as maximizing the values of F_1 over the whole training data (SVM$_{F_1}$) and cross-validation over subsets of training data (SVM$_{cv}$). The latter thresholding method was proposed to reduce the danger of overfitting the result b to the training data. It firstly generates stratified N-fold cross-validation (CV), and then, for each "CV," induces the SVM classifier on 80% of the training data and searches for the threshold with maximum F_1 on the remaining 20%; finally the N thresholds are averaged as an output threshold. The experiments indicated that both methods significantly improved over the baseline SVM, and especially the cross-validation thresholding one showed the best results. ScutFBR (Lewis et al. 2004b) is another thresholding algorithm that relies on heuristics tied to N-fold cross-validation and showed promising results. However, the need of SVM$_{cv}$ and ScutFBR to generate N SVM classifiers adds to computational costs.

Moreover, this kind of the thresholding procedure tied to N-fold cross-validation, such as SVM$_{cv}$, may not be able to give the suitable threshold θ to adjust the hyperplane, $\vec{h} = f(\vec{w}, b)$, which was induced during the training process. Let us recall the concept of the thresholding strategy that aims to find the best translation θ by modifying b at "*a specific orientation* $\vec{w}$". For the N-fold cross-validation thresholding approach, there are N SVM hyperplanes generated for each CV, $(\{h_1, ..., h_N\} = \{f(\vec{w_1}, b_1), ..., f(\vec{w_N}, b_N)\})$. Thus, the adjusted threshold for each CV $(\{\theta_1, ..., \theta_N\})$ maximizes F_1 on the "*different*" orientation $(\{\vec{w_1}, ..., \vec{w_N}\})$ leading to their average as an output threshold $(\theta = average(\theta_1, ..., \theta_N))$ not optimized to the orientation $\vec{w}$ of the target hyperplane.

Some researchers chose to optimize other performance measures than F_1. Thus Imam et al. (2006) describes z-SVM that maximizes the geometric mean (gmean), $gmean = \sqrt{acc_+ \times acc_-}$, where acc_+ and acc_- are the classification rate for positive and negative classes respectively, a metric recommended by Kubat and Matwin (1997). Peter et al. (1998, Shanahan and Roma (2003) developed BetaGamma which applies adaptive filtering based on *linear utility* calculated as $(2 \times TP - FP)$. The experimental results showed that z-SVM and BetaGamma outperform the baseline SVM based on *gmean* and $F_{0.5}$ consecutively. However, no performance evaluation along F_1 has been undertaken.

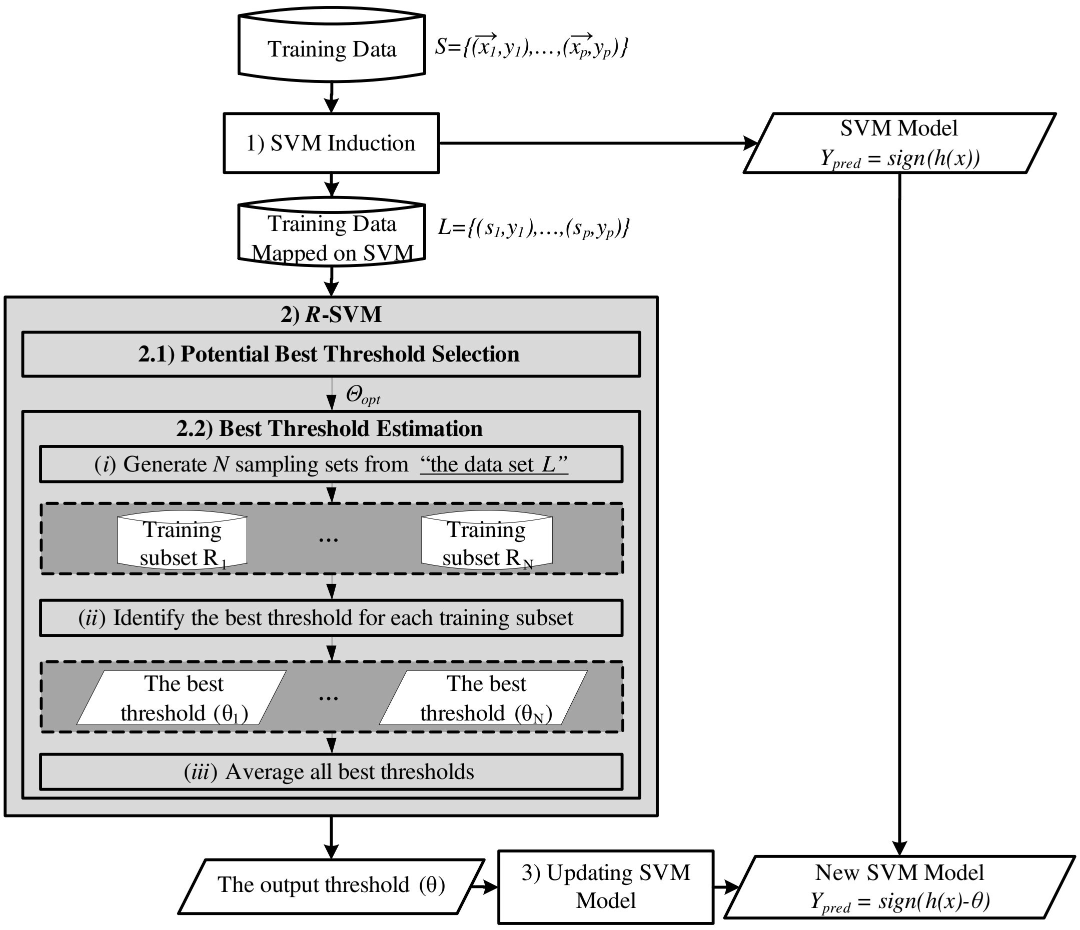
4.3.2 *R*-SVM Concepts

R-SVM, a developed threshold adjustment technique, not only improves SVM performance on imbalanced data, but also overcomes all issues of the previous thresholding works mentioned in the previous section, especially high computational cost and overfitting problems. The flowchart from Figure 4.8 explains its place in the whole scheme: after the induction of SVM, R-SVM employs “potential best threshold selection” to search for a set of important candidate thresholds Θ, and then “best threshold estimation” to calculate the non-overfitting threshold θ to update the SVM model.

Potential Best Threshold Selection

Regarding to Equation 4.1, a set of all possible thresholds Θ must be provided for the thresholding process. The size of this set affects the additional thresholding time; thus, it is important to select only potential thresholds as indicated in the flowchart Box 2.1.

Let us now describe this concept based on an example in Figure 4.9. It illustrates how, as a result of the bias toward the majority class, the mapping function incorrectly labels all examples as negative. Each column represents an example. The second row gives the SVM value of each example as calculated by Equation 2.13, and the examples are ordered according to this value. The third row gives the true class label (positive or negative). The reader can see that, at least as evaluated along the F_1-criterion, the results will be improved if we relocate the threshold to one of the locations called “potential best thresholds.”

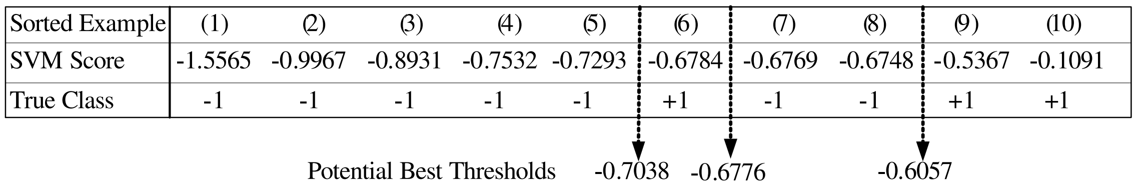


Training Data
$S=\{(\vec{x_1},y_1),\ldots,(\vec{x_p},y_p)\}$
1) SVM Induction
SVM Model
$Y_{pred} = sign(h(x))$
Training Data Mapped on SVM
$L=\{(s_1,y_1),\ldots,(s_p,y_p)\}$
2) *R*-SVM
2.1) Potential Best Threshold Selection
Θ_{opt}
2.2) Best Threshold Estimation
(*i*) Generate *N* sampling sets from "the data set *L*"
Training subset R_1
...
Training subset R_N
(*ii*) Identify the best threshold for each training subset
The best threshold (θ_1)
...
The best threshold (θ_N)
(*iii*) Average all best thresholds
The output threshold (θ)
3) Updating SVM Model
New SVM Model
$Y_{pred} = sign(h(x)-\theta)$

Figure 4.8: The *R*-SVM based framework.

How do we find the best threshold? The simplest approach might consider any *center* between a pair of adjacent sorted examples because it gives the maximum separation in terms of distances. One such location is at $(-1.5565 - 0.9967)/2)$, another at $(-0.9967 - 0.8931)/2)$, and so on, giving us the total of nine candidates. For each, we calculate the F_1 and then pick the one with the highest value.

Since this can be expensive in the case of large training sets, we considered only those pairs of neighboring sorted examples that differed in class labels called "potential best thresholds" , Θ_{opt}, (in Figure 4.9, there are three such locations). For instance, in the large `EUROVOC` dataset, only 50 such locations exist.



Sorted Example	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
SVM Score	-1.5565	-0.9967	-0.8931	-0.7532	-0.7293	-0.6784	-0.6769	-0.6748	-0.5367	-0.1091
True Class	-1	-1	-1	-1	-1	+1	-1	-1	+1	+1

Potential Best Thresholds -0.7038 -0.6776 -0.6057

Figure 4.9: Illustration of the three potential best thresholds on the data set with 10 examples.

Best Threshold Estimation

The next task is to compute the suitable output threshold that is not overfitting to any specific data sets as shown in the flowchart by Box 2.2. This can be accomplished in three steps: (*i*) generating *several training subsets* from the training data on the SVM space, (*ii*) in each training subset, identify the best threshold, and (*iii*) obtain the output threshold as an average value of these best thresholds. For instance, suppose there are three "potentially best thresholds," $\{1.1, 1.7, 2.1\}$, and let the evaluation in five training subsets identified the following "winners," respectively: $\{1.1, 1.1, 1.7, 1.1, 2.1\}$. The final threshold is then the average of these: $\theta_{final} = 1.42$.

Let us compare this procedure in *R*-SVM to the previous thresholding works in Section 4.3.1. *R*-SVM can be applied to more domains than the methods proposed by Yan et al. (2009), Li et al. (2008), and Goertzel and Venuto (2006) because there is no data distribution assumption. Comparing to $\texttt{SVM}_{F1}$ (Brank et al. 2003), *R*-SVM can reduce a risk of overfitting the result to a specific data by using several training subsets.

Comparing to the thresholding methods tied to cross-validation, such as SVM$_{cv}$ (Brank et al. 2003) and ScutFBR (Lewis et al. 2004b), R-SVM constructs N training subsets from "*the training data mapped on the SVM space*" ($\{R_1, R_2, ..., R_N\}$), while those methods generate N-fold cross-validation from "*the original training data*" ($\{CV_1, CV_2, ..., CV_N\}$) and then induce N SVM classifiers just to find the threshold. In terms of the computational cost, R-SVM is much faster than those methods since it does not generates any extra SVMs (induces only "one SVM" from the entire training set). Furthermore, the output threshold of R-SVM is more suitable for the target hyperplane than that of the cross-validation based methods. Table 4.5 shows the proof of our claim, where $\{\theta_1, ..., \theta_N\}$ is a set of all best thresholds for N training subsets, and θ denotes the output threshold. On the one hand, all best thresholds for each training subset in R-SVM unanimously maximizes the performance of the target hyperplane, $f(\vec{w}, b)$; on the other hand, SVM$_{cv}$ gives a set of all best thresholds maximizing different hyperplanes, $f(\vec{w_i}, b_i)$, in performance varied for each CV_i. Thus, the output threshold which is an average of all best thresholds ($\{\theta_1, ..., \theta_N\}$) obtained in R-SVM can actually improve the target hyperplane comparing to an average of those in SVM$_{cv}$.

Table 4.5: A comparison between R-SVM and SVM$_{cv}$ to find the estimate of the output threshold θ which can be described by Equation 4.1.

Description	R-SVM	SVM$_{cv}$
The best threshold θ_1	$\max(perf(R_1, f(\vec{w}, b), \Theta))$	$\max(perf(CV_1, f(\vec{w_1}, b_1), \Theta))$
The best threshold θ_2	$\max(perf(R_2, f(\vec{w}, b), \Theta))$	$\max(perf(CV_2, f(\vec{w_2}, b_2), \Theta))$
...	...	...
The best threshold θ_N	$\max(perf(R_N, f(\vec{w}, b), \Theta))$	$\max(perf(CV_N, f(\vec{w_N}, b_N), \Theta))$
The output threshold θ	$average(\theta_1, \theta_2, ..., \theta_N)$	$average(\theta_1, \theta_2, ..., \theta_N)$

R-SVM has implemented four ways to generate training subsets as follows:

- *Boostrapping (BST)* (Efron 1979; Chernick 2008) relies on a random sampling with replacement, where each training subset has the same size as the original data.

- *Bootstrapping .632 (BST632)* (Efron 1983) uses the sampling with replacement in the same way as BST, but has a different calculation of the overall error, $err = 0.368 err_{train} + 0.632 err_{sample}$, where err_{train} is the error on the original training data and err_{sample} is the error on a training subset.

- *Partitioning (PT)* divides the data into N nonoverlapping subsets and considers each single subset as a training subset.

- *LeaveOnePartOut (LOPO)* divides the data into N nonoverlapping subsets; each union of $(N-1)$ subsets is treated as a training subset.

The experiments in Section 4.3.3 will show the above methods with the suitable number of training subsets giving the best results. We always made sure that the relative representation of the positive and negative examples was in each training subset the same as in the original set ("stratified randomization"). Table 4.6 contains *R*-SVM's pseudocode.

4.3.3 Experiments and Discussion

We experimented with the six real-world databases presented in Section 4.1. For experimental convenience, we pre-processed these databases so as not to get distracted by extremely rare classes and to limit the (impractically high) number of attributes in

Table 4.6: The pseudocode of R-SVM

```
SamplingThresholding($L, \Theta, N$)
▷ $L$ is a set of training examples mapped on SVM space represented by the tuples ($s_i$, $y_i$).
▷ $\Theta$ is a set of potential best thresholds.
▷ $N$ is the number of training subsets.

▷ Step 1: obtain the training subsets
$R \leftarrow$ generate_strtified_training_subset($L, N$);

▷ Step 2: in each training subset, identify the best threshold
for Sample Set $i \leftarrow 1$ to $length[R]$ ▷ loop1: for each training subset
    do for Threshold $j \leftarrow 1$ to $length[\Theta]$ ▷ loop2: for each threshold
        do $Perf[j]$ = performance($R[i], \Theta[j]$);

      ▷ The best threshold for the $i^{th}$ training subset giving the maximum performance
      $T[i] = \Theta$[position(max($Perf$))];

▷ Step 3: obtain the output threshold as an average value of these best thresholds
return $\theta$ = average($T$); ▷ The output threshold
```

some domains. In the case of `EUROVOC` and `RCV1-v2`, we ignored all classes represented by less than six documents following our previous research (Vateekul and Kubat 2009; Dendamrongvit et al. 2011). In the other domains, we ignored classes represented by less than 10% of total examples because the setting in the previous research is too low for any classes to be removed. For attribute selection, we followed the recommendation from Section 4.2 and focused on attributes with the highest gain ratio (Quinlan 1993). Specifically, we used 25% highest-gain-ratio attributes in `EUROVOC` and `RCV1-v2` and 50% highest-gain-ratio attributes for `Emotions`. In all other domains, we used the complete set of attributes. For statistically reliable results, 5-fold cross-validation (CV) was used. The data characteristics after pre-processing are summarized as in Table 4.7.

Table 4.7: The statistics of experimental data sets in *R-SVM* after our pre-processing.

Dataset	Domain	Features	Classes	Instances	% Positive Examples Per Class		
					Average	Min	Max
`EUROVOC`	text	4,000	20	10,000	22.44	4.51	55.22
`RCV1-v2`	text	4,000	53	6,000	5.17	0.88	44.55
`Yeast`	biology	103	12	2417	30.28	1.43	75.13
`Emotions`	music	72	6	593	31.15	24.97	44.55
`Scene`	image	294	6	2407	17.89	15.07	22.11
`Mediamill`	video	120	11	43,907	28.32	10.89	77.11

To facilitate the replicability of our experiments, we induced the SVM by the publicly available package *svmlight*[13] (Joachims 1999), relying on default parameter settings wherever possible. The only exception was parameter γ (the width of the gaussian distribution in the RBF kernel functions): in `emotions` and `scene`, we used $\gamma = 0.1$ and $\gamma = 0.01$, respectively; in all other domains, we used the default setting, where the value of γ is 1.

The experiments address two different aspects. First, we wanted to know which of the four variations of *R*-SVM on their optimal number of training subsets would give the best result. Second, the comparison between the best setting of *R*-SVM and other thresholding algorithms was conducted. All results are expressed by *macro-averaging.*

The best version of *R*-SVM

This experiment aims to find the optimal setting of *R*-SVM before comparing to other thresholding methods. There are four ways to generate training subsets in our developed method: *Bootstrap (BST)*, *Bootstrap .632 (BST632)*, *Partitioning (PT)*,

[13]`http://svmlight.joachims.org/`

and *LeaveOnePartOut (LOPO)*. Each of these methods was needed to search for the suitable number of training subsets. Then we compared each of these methods and chose the one giving its prominent performance in terms of F_1.

The number of training subsets: In the case of BST and *BST632*, we experimented with the following numbers of training subsets for: 5, 10, 30, 50, 100, and 150.[14] In the case of PT and $LOPO$, we used the following numbers of training subsets: 3, 5, 10, and 20 sets (in `MediaMill`, the maximum was 10). If we wanted more disjunctive subsets, they would become impractically small.

Figure 4.10 shows F_1 and computation time of BST. As expected, the computation costs grow linearly in the number of training subsets. As for F_1, the optimum number of subsets seems to be 50, when the performance reached its maximum in `EUROVOC`, `RCV1-v2`, and `Yeast`; in the other domains, higher numbers of training subsets *did* occasionally lead to further improvement, but the difference was not statistically significant. The results for *BST632* were so similar to those from Figure 4.10 that there was no need to detail them here. Again, the optimum number of training subsets was about 50.

Figure 4.11 shows the results for PT. Again, the computation costs grow linearly in the number of training subsets. As for performance, the ideal number of subsets is 10; for this value, F_1 reaches its maximum in `EUROVOC` and `MediaMill` and is a close second in `RCV1-v2`, `Yeast`, and `Scene`. The results of $LOPO$ were so similar to those from Figure 4.11 that we decided not to present them here. The optimum number of training subsets was 5.

[14]In the domain `MediaMill`, we did not go beyond 50 training subsets because this is a large set and the computation is costly.

Comparison among different methods to generate training subsets: Table 4.8 summarizes performance comparison among different sampling methods in R-SVM. When "*ranking*" the individual algorithms along the F_1 criterion, we used the ANOVA rank followed by Bonferroni multiple comparisons (Delwiche and Slaughter 2008) in order to test the statistical significance. The column headed by "ANOVA" gives the ranking, with "A" denoting the solution deemed statistically best, "B" be second, etc. The next column contains an output threshold θ from R-SVM applying different choices of training subsets. This output threshold will be used to update the original SVM hyperplane to $\vec{h^*} = \vec{h} - \theta$. Finally the rightmost column gives the additional computational time consumed by the corresponding sampling methods.

Table 4.8: Performance comparison of R-SVM among different methods of generating training subsets. Within each dataset, the techniques are sorted by their F_1. For the reader's convenience, we also provide the ANOVA ranks.

Dataset	Method ordered by F_1	*Precision*	*Recall*	F_1	ANOVA	Threshold	Time (seconds)
EUROVOC	1. *PT* 10 sets	0.8868	0.6492	0.7408	A	-0.7098	1.11
	2. *BST* 50 sets	0.8969	0.6401	0.7401	A	-0.6866	18.21
	3. *BST632* 50 sets	0.8964	0.6400	0.7391	A	-0.6734	18.64
	4. *LOPO* 5 sets	0.8801	0.6464	0.7313	A	-0.6795	1.94
RCV1-v2	1. *PT* 10 sets	0.6320	0.6974	0.6466	A	-0.5359	0.94
	2. *BST* 50 sets	0.6342	0.6949	0.6463	A	-0.5289	10.81
	3. *BST632* 50 sets	0.6353	0.6927	0.6457	A	-0.5244	12.10
	4. *LOPO* 5 sets	0.6277	0.6988	0.6435	A	-0.5391	1.17
Yeast	1. *PT* 10 sets	0.5231	0.5950	0.5524	A	-0.3941	1.34
	2. *BST* 50 sets	0.5176	0.6025	0.5500	A	-0.3977	13.03
	3. *BST632* 50 sets	0.5178	0.6023	0.5495	A	-0.3956	13.32
	4. *LOPO* 5 sets	0.5154	0.6062	0.5486	A	-0.3963	1.46
Emotions	1. *BST* 50 sets	0.5597	0.7436	0.6318	A	-0.7085	3.73
	2. *LOPO* 5 sets	0.5519	0.7556	0.6291	A	-0.7208	0.39
	3. *BST632* 50 sets	0.5543	0.7474	0.6284	A	-0.7130	3.80
	4. *PT* 10 sets	0.5627	0.7148	0.6203	A	-0.6755	0.56
Scene	1. *BST632* 50 sets	0.7742	0.8184	0.7941	A	-0.4122	5.66
	2. *PT* 10 sets	0.7782	0.8129	0.794	A	-0.4050	0.54
	3. *BST* 50 sets	0.7721	0.8178	0.793	A	-0.4118	5.55
	4. *LOPO* 5 sets	0.7714	0.8189	0.7929	A	-0.4182	0.51
MediaMill	1. *PT* 10 sets	0.5302	0.667	0.5607	A	-0.7219	99.98
	2. *BST* 50 sets	0.5197	0.6973	0.5464	B	-0.7289	3,237.26
	3. *LOPO* 5 sets	0.5177	0.7043	0.5448	B	-0.7322	314.81
	4. *BST632* 50 sets	0.5178	0.7031	0.5447	B	-0.7319	3,006.13

From the table, the *PT* method was chosen due to its promising results in terms of F_1, especially in the data set `MediaMill`. Although the results of all methods were not significantly different, *R*-SVM with *PT*-training subsets gave the highest F_1 on most data sets and always had the fastest thersholding time because it generates the smallest size of training subsets. In the cases of *BST* and *BST632*, their results were slightly worse than those of *PT* because their output thresholds were biased toward duplicated examples in the *same* Boostrap training subset. Furthermore, *BST632* outperformed *BST* only one data set, `Scene`, due to its calculation of the overall error making its results in favor of the thresholds overfitting to the training data. The results also indicated that *LOPO* gave the worst results on four out of six data sets.

Comparing *R*-SVM with other threshold-adjustment techniques

Let us now compare *R*-SVM's performance with that of alternative threshold-adjustment techniques proposed earlier: SVM_{F_1}, SVM_{CV}, *ScutFBR*, and *BetaGamma*. Recall that they all seek an improvement over the original SVM (denoted below as SVM_{ORG}) whose performance can therefore be seen as the lower bound on what can be achieved. The upper bound, by contrast, can be estimated by the threshold giving the highest F_1 on the *testing set* (which, of course, is unavailable at the time of induction, and therefore represents something like "perfect information"). In the experiments reported below, parameter values in *R*-SVM are set in accordance with the recommendation from the experiments in the previous section (i.e, 10 training subsets generated by the *PT* method).

Table 4.9 sorts the performances of diverse threshold-adjustment methods by the F_1-values. The results of *ScutFBR* and SVM_{CV} are always on the same line because their classification performance is indiscernible. The upper and lower bounds are labeled, respectively, as SVM$_{ORG}$ and "Best F_1 on Test." The rightmost column gives the computation time. For each domain, the first row ("SVM$_{ORG}$ (lower)") gives induction time of the plain SVM. Induction time of "Best F_1 on Test" is omitted because this row refers to values obtained only by "cheating"—considering the best result possible on the testing set. Using this table, we wanted to answer the following questions:

- Does R-SVM offer a significant F_1-improvement over the original SVM?
- Do the multiple training subsets in R-SVM lead to an improvement over SVM$_{F_1}$?
- Is the thresholding technique of R-SVM faster than that of SVM$_{CV}$ while still providing comparable F_1?
- Does R-SVM outperform previous threshold-adjustment algorithms such as *ScutFBR* and *BetaGamma*?

Comparing R-*SVM to SVM$_{ORG}$:* The results summarized in Table 4.9 indicate that *R-SVM* always outperformed the original SVM (significantly according to *t*-test). We noted that SVM$_{ORG}$ usually exhibited high *precision* but very low *recall* (F_1 was then low, too). We attribute this phenomenon SVM's tendency to bias the offset of the separating hyperplane toward the majority class (negative examples), which results in a threshold that classifies most examples as negative. The shift of the hyperplane in the opposite direction then corrects some of these misclassifications. Consequently, the technique considerably improved *recall* by only a moderate loss in

Table 4.9: Performance of diverse threshold-adjustment techniques. For each data set, the techniques are sorted by their F_1. The top rows, "SVM" and "Best F1 on Test," define the lower bound and upper bound, respectively. The performance of *ScutFBR* is essentially the same as that of SVM_{CV}.

Dataset	Method ordered by F_1	*Precision*	*Recall*	F_1	Time (sec.)
`EUROVOC`	SVM_{ORG} (lower)	0.9430	0.2885	0.4266	1,505.19
	Best F_1 on Test (upper)	0.8943	0.6624	0.7539	N/A
	1. SVM_{CV} (*ScutFBR*)	0.8748	0.6727	0.7423	4,405.76
	2. *R*-SVM	0.8868	0.6492	0.7408	1.11
	3. SVM_{F_1}	0.8121	0.6731	0.6879	0.52
	4. *BetaGamma*	0.9437	0.2865	0.4241	29.83
`RCV1-v2`	SVM_{ORG} (lower)	0.8707	0.4418	0.5451	55.93
	Best F_1 on Test (upper)	0.7292	0.6748	0.6934	N/A
	1. SVM_{CV} (*ScutFBR*)	0.7183	0.6313	0.6567	522.65
	2. *R*-SVM	0.6320	0.6974	0.6466	0.94
	3. SVM_{F_1}	0.6244	0.6985	0.6400	0.33
	4. *BetaGamma*	0.6532	0.6540	0.5807	20.02
`Yeast`	SVM_{ORG} (lower)	0.7594	0.3893	0.4207	31.38
	Best F_1 on Test (upper)	0.5419	0.6821	0.5918	N/A
	1. SVM_{CV} (*ScutFBR*)	0.5174	0.6675	0.5735	144.65
	2. *R*-SVM	0.5231	0.5950	0.5524	1.34
	3. SVM_{F_1}	0.5129	0.6089	0.5473	0.37
	4. *BetaGamma*	0.7505	0.3416	0.4021	2.51
`Emotions`	SVM_{ORG} (lower)	0.5995	0.4111	0.4436	0.63
	Best F_1 on Test (upper)	0.5926	0.7984	0.6652	N/A
	1. SVM_{F_1}	0.5528	0.7499	0.6268	0.09
	2. *R*-SVM	0.5627	0.7148	0.6203	0.56
	3. SVM_{CV} (*ScutFBR*)	0.3576	0.9733	0.5170	3.70
	4. *BetaGamma*	0.6243	0.4245	0.4537	0.21
`Scene`	SVM_{ORG} (lower)	0.8933	0.6419	0.7383	12.67
	Best F_1 on Test (upper)	0.8116	0.8183	0.8111	N/A
	1. *R*-SVM	0.7782	0.8129	0.7940	0.54
	2. SVM_{CV} (*ScutFBR*)	0.7815	0.8098	0.7936	63.46
	3. SVM_{F_1}	0.7716	0.8191	0.7929	0.16
	4. *BetaGamma*	0.8963	0.6311	0.7323	1.28
`MediaMill`	SVM_{ORG} (lower)	0.5667	0.3169	0.3522	13,271.52
	Best F_1 on Test (upper)	0.5297	0.6752	0.5923	N/A
	1. SVM_{CV} (*ScutFBR*)	0.5320	0.6639	0.5895	9,598.04
	2. *R*-SVM	0.5302	0.667	0.5607	99.98
	3. SVM_{F_1}	0.5155	0.7072	0.5444	81.99
	4. *BetaGamma*	0.6276	0.3163	0.3518	390.86

precision, thus improving F_1. In diverse domains, the improvement of R-SVM over the original SVM varied from 7.54% to 87.26%.

Moreover, the F_1-results of R-SVM almost reach the best possible performance, "Best F_1 on Test." Especially on the data sets `EUROVOC` and `Emotions`, the differences in terms of F_1 are only about 0.01, and, on the remaining data sets, the differences are always less than 0.04. Besides, R-SVM is also acceptable in terms of computational costs. Its threshold adjustment module adds less than 1% to the SVM induction time.

Comparing to SVM_{F_1}: Table 4.10 shows the positive effect of R-SVM's using multiple training subsets by comparing its performance with that of SVM_{F_1}, a technique that uses only a single training set. The reader can see that R-SVM significantly (according to the t-test with 95.5% confidence level) outperformed SVM_{F_1} on three data sets, while SVM_{F_1} never significantly outperformed R-SVM. We explain this by SVM_{F_1}'s tendency to overfit the training set. The added costs of R-SVM's search for the best threshold are only a fraction of a second on all data sets except for `MediaMill`.

Table 4.10: Performance comparison of *R-SVM* and SVM_{F_1}. The boldface font indicates that the improvement is statistically significant. The numbers in parentheses give the performance edge of the given technique over the original SVM.

Data set	Macro-averaging F_1		Thresholding time (seconds)	
	R-SVM	SVM_{F_1}	R-SVM	SVM_{F_1}
EUROVOC	**0.7408 (87.26%)**	0.6879 (73.90%)	1.11	0.52
RCV1-v2	**0.6466 (18.62%)**	0.6400 (17.41%)	0.94	0.33
Yeast	0.5524 (31.30%)	0.5473 (30.10%)	1.34	0.37
Emotions	0.6219 (40.19%)	0.6216 (40.13%)	0.56	0.09
Scene	0.7940 (7.54%)	0.7929 (7.40%)	0.54	0.16
MediaMill	**0.5607 (59.25%)**	0.5444 (54.61%)	99.98	81.99

Comparing to SVM$_{CV}$: As far as the comparison with SVM$_{CV}$ is concerned, Table 4.11 shows that R-SVM's slight decline in F_1 is compensated by considerable savings in terms of the CPU-time needed to find the best threshold. The difference in terms of F_1 is always less than 0.02 (with 8% improvement over the original SVM) in `MediaMill`. As for thresholding costs, SVM$_{CV}$ apparently paid a heavy price for the fact that it has to generate many extra SVM models, something that R-SVM does not do. Note that on the data set `Emotions`, R-SVM is not only superior to SVM$_{CV}$ in terms of CPU-time, but also outperforms it in terms of F_1.

Table 4.11: The comparison of *R-SVM* and SVM$_{CV}$. The boldface font indicates a statistically significant improvement. The numbers in parentheses give the performance edge of the given technique over the original SVM.

Data set	Macro-averaging F_1		Thresholding time (seconds)	
	R-SVM	SVM$_{CV}$	R-SVM	SVM$_{CV}$
EUROVOC	0.7408 (87.26%)	0.7423 (87.63%)	1.11	4,405.76
RCV1-v2	0.6466 (18.62%)	**0.6567 (20.47%)**	0.94	522.65
Yeast	0.5524 (31.30%)	**0.5735 (36.32%)**	1.34	144.65
Emotions	**0.6219 (40.19%)**	0.5170 (16.55%)	0.56	3.70
Scene	0.7940 (7.54%)	0.7936 (7.49%)	0.54	63.46
MediaMill	0.5607 (59.25%)	**0.5895 (67.42%)**	99.98	9,598.04

Comparing to ScutFBR *and* BetaGamma*:* Since the F_1-results of *ScutFBR* and SVM$_{CV}$ are the same as illustrated in Table 4.9, our previous observations regarding R-SVM's superiority over SVM$_{CV}$ apply to *ScutFBR* as well. It should be noted that *ScutFBR* uses cross-validation just like SVM$_{CV}$, but it also uses a heuristic that seeks to set the threshold higher in order to prevent the occurrence of false-positive examples. However, in imbalanced data sets (especially negative examples outnumber the positive ones), the original SVM always constructs a model with a high threshold. Increasing this threshold even further therefore cannot improve the original SVM's

F_1. By consequence, the heuristic in *ScutFBR* has never been used, and this makes its results essentially the same as those of SVM_{CV}.

As for *BetaGamma*, Table 4.9 shows that it falls behind R-SVM both in accuracy and time. Moreover, this technique has always had the worst classification performance among those we experimented with (even worse than the plain SVM on four data sets). This is explained by the simple fact that it does not maximize F_1, but another criterion (*linear utility*).

4.4 Conclusion

In multi-label domains, it is customary to induce multiple binary classifiers, each for a different class. The nature of these domains often means that the binary classifiers usually have to be induced from heavily imbalanced class training sets. This circumstance is known to hurt the classification performance, particulary in terms of F_1. Moreover, data sets from some benchmark domains, such as text categorization, are described by a large number of features, which leads to an extreme induction time. This chapter demonstrates two of our contributed systems that cope with multi-label classification on two issues: scalability and imbalanced training sets.

FDT is an efficient tree-based classification system. The idea is, first, to reduce the number of features using the feature pre-selection mechanism and, second, to induce sets of decision trees, each from a smaller subset of the training examples. In the classification phase, the trees combine their outputs (data fusion) by a simple mechanism that favors the minority class—this is important in domains with multi-label examples where a separate binary classifier is usually induced for each class.

***R*-SVM** is an optimized SVM framework along our thresholding strategy. For the sake of scalability, the feature selection of FDT is continuously applied to this classifier, and SVM is chosen to be used as the baseline classifier instead of the decision tree because its induction is faster. For the imbalanced training issue, the new thresholding strategy is developed whose goal is to translate the hyperplane offset by combining (e.g., by averaging) the results obtained from different training subsets. As for the creation of training subsets, four methods were considered: *Bootstrap (BST)*, *Bootstrap .632 (BST632)*, *Partitioning (PT)*, and *LeaveOnePartOut (LOPO)*. The mechanism to generate the training subsets is costless because no extra SVM models have to be induced. Experiments with six real-world benchmark domains showed that *R*-SVM (with the *PT* mechanism) performed well in both terms of classification performance and computational costs. It outperformed not only the classical SVM, but also other thresholding methods previously proposed by the relevant literature: SVM_{F_1}, SVM_{CV}, *ScutFBR*, and *BetaGamma*. In some experiments, the results almost reached what we regard as the upper bound on the possible performance.

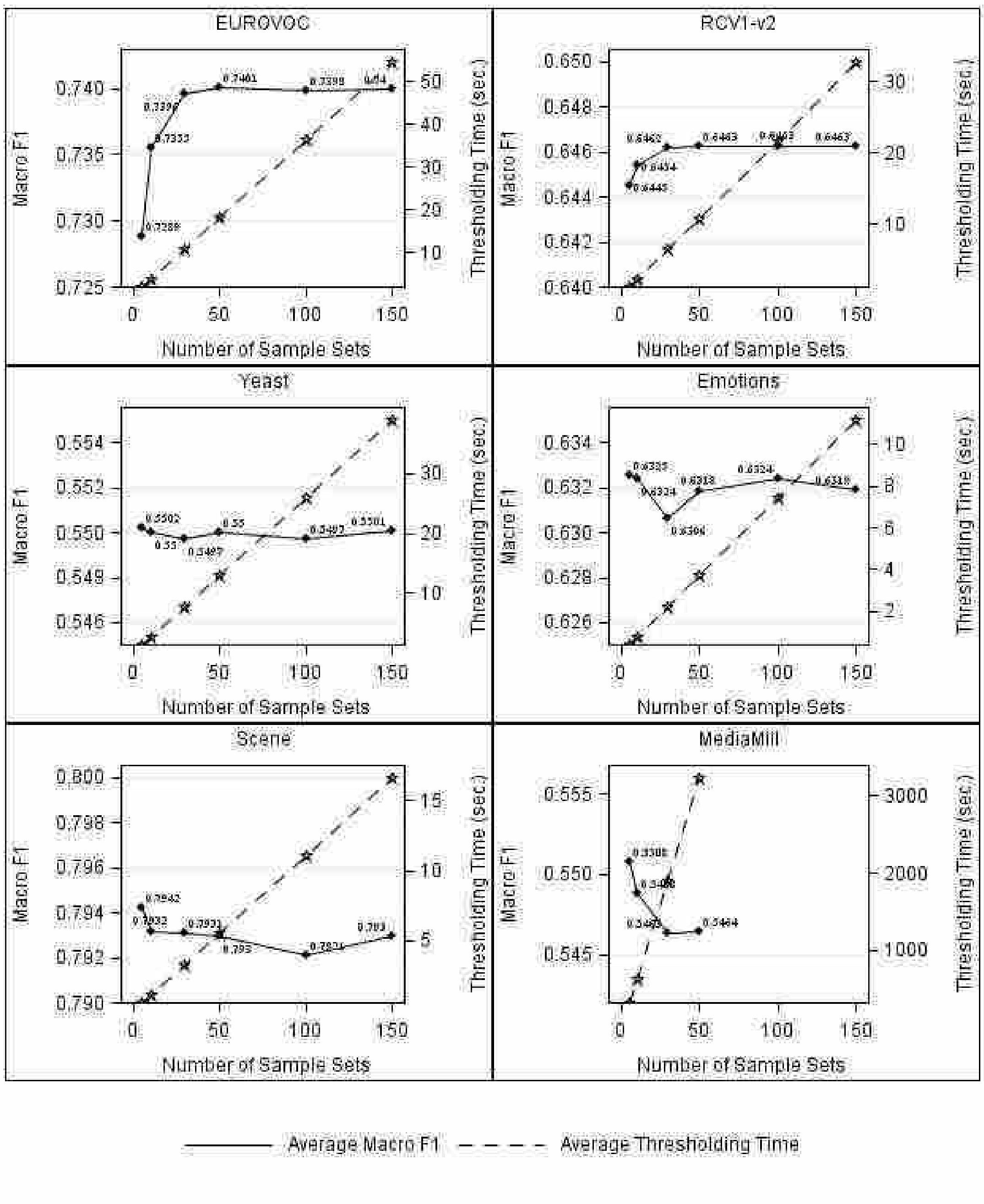


Figure 4.10: Classification performance and the additional computational time incurred by threshold modification of R-SVM when the *BST* method was used to generate training subsets.

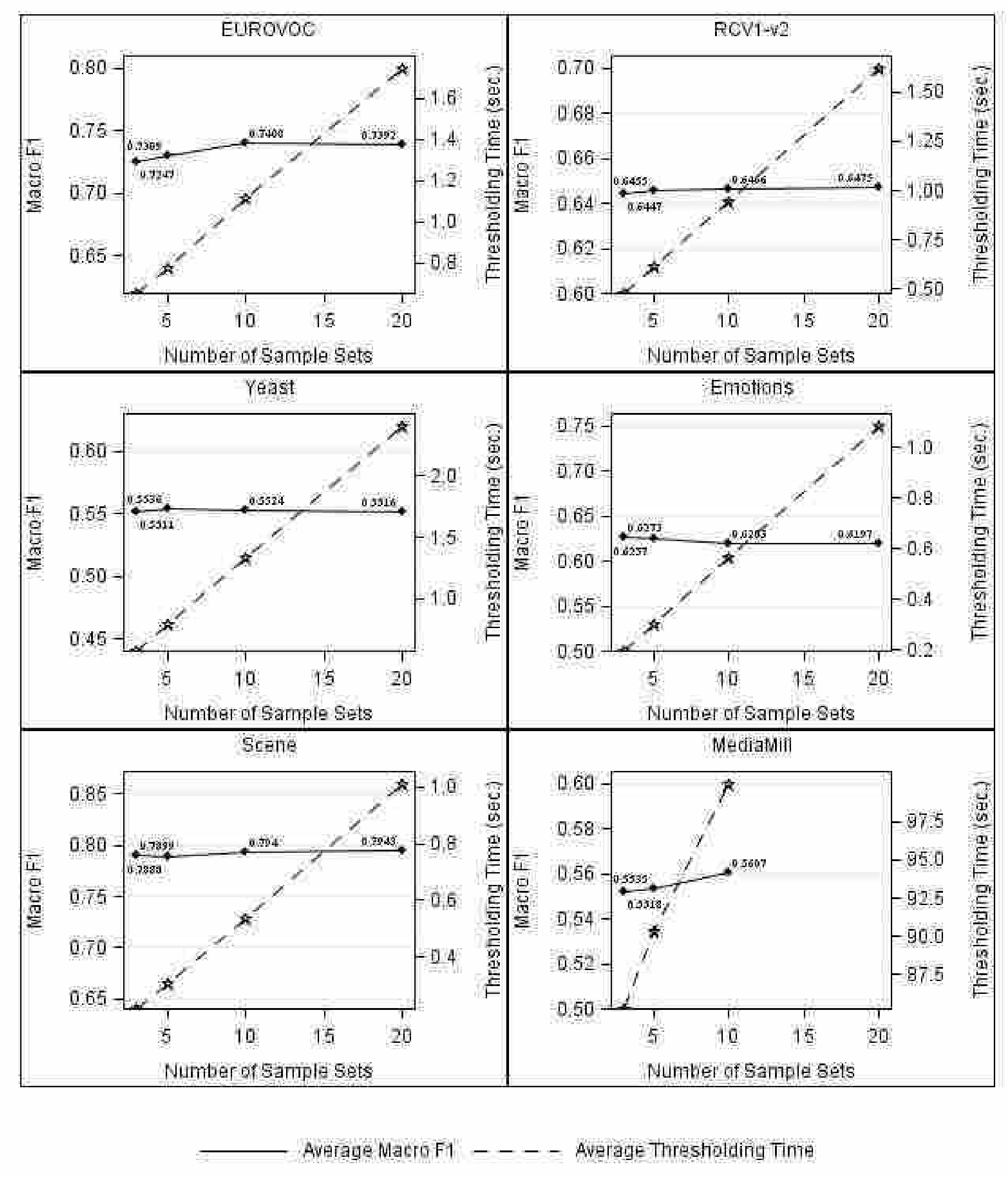


Figure 4.11: Classification performance and the additional computational time incurred by threshold modification of R-SVM when the PT method was used to generate training subsets.

CHAPTER 5

Learning from Hierarchies in Functional Genomics

This dissertation mainly focuses on hierarchical multi-label classification, a variant of classification where examples may belong to multiple classes at the same time and these classes are organized in a hierarchy. In this chapter, we explore two main aspects of this domain. The first aspect is to present a novel learning algorithm called "*HR*-SVM", which is a hierarchical extended version of the proposed multi-label classification technique, *R*-SVM, presented in Section 4.3. Performance evaluation is the second aspect demonstrated in this chapter. We introduce a new hierarchical classification measure called "example-label based macro-averaging measure", which is an extension of the traditional classification metrics, *precision*, *recall*, and F_1.

The particular application targeted by this work is "*gene function prediction*", an important task in bioinformatics. Exploiting the functions of genes is expensive and time-consuming since it involves many experimental lab processes by experts. The automated classification system can offer a set of possible gene functions, which can drive the biological validation and discover of novel functions of genes and gene product, and, thus, lessen the experimental burden.

Here is the task of this application in details: given an unknown gene (example), the task is to classify its functions (classes) referring to a hierarchy of predefined functions, e.g., Gene Ontology (GO) as shown in Figure 5.1. The class hierarchies are no longer constrained to trees, but can be a "directed acyclic graph (DAG)", where each class can be inherited from multiple parents. Each gene may belong to multiple classes at the same time (multi-label) and can be assigned to any classes in the hierarchy (the non-mandatory leaf node problem (NMLNP)).

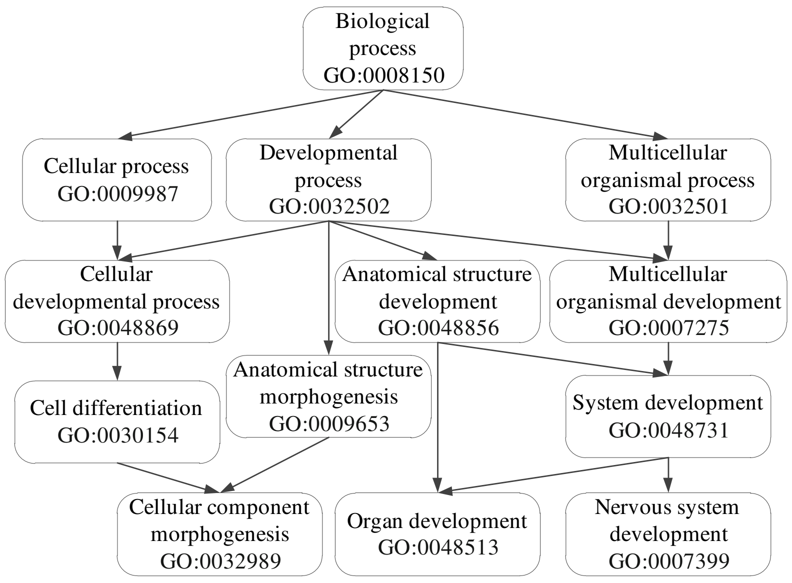

Biological process GO:0008150
Cellular process GO:0009987
Developmental process GO:0032502
Multicellular organismal process GO:0032501
Cellular developmental process GO:0048869
Anatomical structure development GO:0048856
Multicellular organismal development GO:0007275
Anatomical structure morphogenesis GO:0009653
Cell differentiation GO:0030154
System development GO:0048731
Cellular component morphogenesis GO:0032989
Organ development GO:0048513
Nervous system development GO:0007399

Figure 5.1: An example class hierarchy of immune system processes in the field of Gene Ontology (GO).

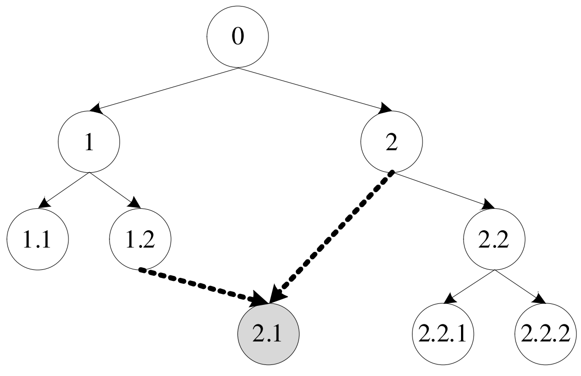
This chapter is organized as follows. The developed hierarchical classification framework, *HR*-SVM, is demonstrated in Section 5.1. Then Section 5.2 gives the detail of the presented performance measure. Section 5.3 and Section 5.4 report the experimental data and results respectively. Conclusion, discussion, and future work are the topics for Section 5.5.

5.1 *HR*-SVM Concepts

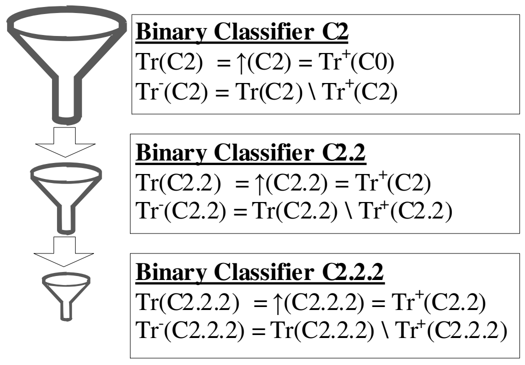
The detail of the proposed hierarchical classification framework, "*HR*-SVM", is explained in this section. Note that all explanation and examples in this section are based on the class hierarchy in Figure 5.2(a). *HR*-SVM is based on the top-down approach by inducing a local classifier *per class node* and classifying unknown examples by employing classifiers from the root to leaves. Unlike other approaches, e.g., global approach, the top-down approach supports the domain of problems we are interested in (DAG, multi-label, and NMLNP) with only some modifications of the traditional machine learning methods. In the framework, *R*-SVM is considered as a local classifier. The number of classifiers is equal to the number classes (nodes). For instance, eight *R*-SVM's classifiers have to be constructed for the class hierarchy in Figure 5.2(a) except that there is no classifier at the root node.

Care is taken to follow the *hierarchical constraint*, the important criterion of hierarchical classification. For instance, if the classifier $C2$ labels an example $\mathbf{x}$ as negative, then $\mathbf{x}$ has to be labeled as negative also by classifiers corresponding to $C2$'s subclasses: $\{C2.1, C2.2, C2.2.1, C2.2.2\}$. Conversely, $\mathbf{x}$ is classified as $C2.2.2$ if the classifiers $\{C2, C2.2, C2.2.2\}$ all issue the positive label for $\mathbf{x}$. Figure 5.2(b) shows a conceptual view of the top-down approach which is like a stack of filters (classifiers).

Positive experience with the top-down principle has been reported by several research groups (Koller and Sahami 1997; Sun and Lim 2001; Nguyen et al. 2005; Secker et al. 2007; Fagni and Sebastiani 2007; Fagni and Sebastiani 2010). It has many advantages in terms of simplicity, efficiency, and suitability to the target problem, the gene function prediction. However, by observation of these papers, there are three



(a) A DAG-structured class hierarchy


Binary Classifier C2
Tr(C2) = ↑(C2) = Tr+(C0)
Tr-(C2) = Tr(C2) \ Tr+(C2)
Binary Classifier C2.2
Tr(C2.2) = ↑(C2.2) = Tr+(C2)
Tr-(C2.2) = Tr(C2.2) \ Tr+(C2.2)
Binary Classifier C2.2.2
Tr(C2.2.2) = ↑(C2.2.2) = Tr+(C2.2)
Tr-(C2.2.2) = Tr(C2.2.2) \ Tr+(C2.2.2)

(b) A conceptual view of the top-down approach classifiers

Figure 5.2: (a) An example of the DAG-structured class hierarchy. The gray node is a class with multiple parents. (b) A conceptual view of top-down classifiers from Classifier $C2$ until Classifier $C2.2.2$.

critical factors which adversely affect performance. First, misclassifications committed at higher levels of the class hierarchy tend to get propagated downward, making it hard to induce accurate classifiers for the lowest-level classes. There are two types of errors generated by superclasses' classifiers: (*i*) false negative (FN), positive examples classified as negative, and (*ii*) false positive (FP), negative examples predicted as positive. Each of these errors causes different problems to the system and should be treated separately. Second, inducing a separate binary classifier for each class often leads to the situation where the training data are induced are heavily imbalanced (negative examples outnumbering positive examples), a circumstance known to impair many machine learning techniques. Third, since different classes are often characterized by different sets of attributes, it is necessary to run attribute-selection techniques separately for each of them.

HR-SVM is a novel top-down algorithm implemented to overcome those above issues: error propagation, imbalanced training data, and irrelevant attributes. As indicated in Figure 5.3, *HR*-SVM consists of four modules. The first three modules are used for data pre-processing, and the last one, *R*-SVM (Section 4.3), is responsible for induction from imbalanced training sets.

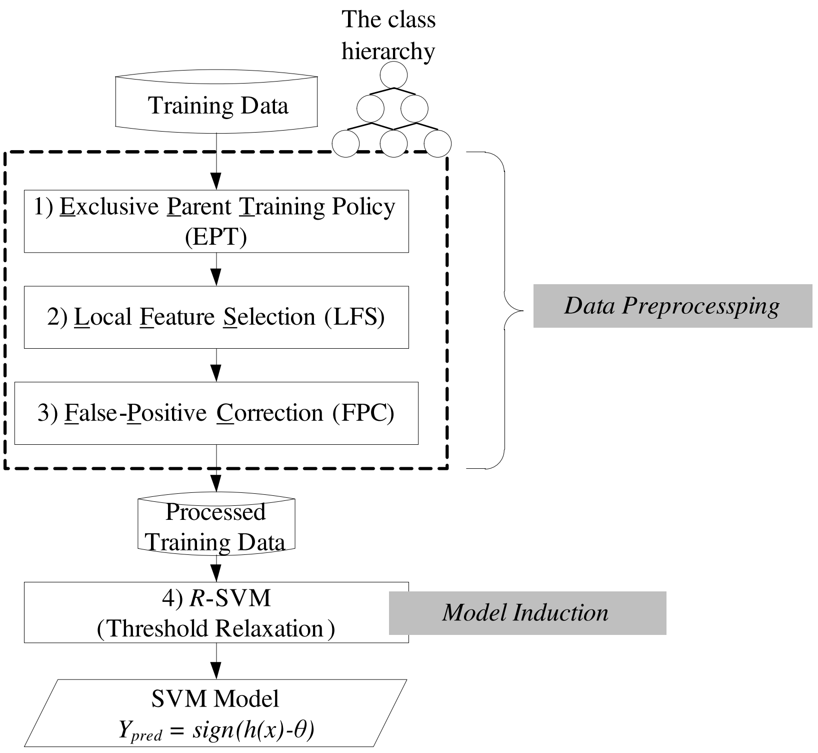

The class hierarchy
Training Data
1) Exclusive Parent Training Policy (EPT)
2) Local Feature Selection (LFS)
3) False-Positive Correction (FPC)
Data Preprocessping
Processed Training Data
4) R-SVM (Threshold Relaxation)
Model Induction
SVM Model
Y_pred = sign(h(x)-θ)

Figure 5.3: The *HR*-SVM's general architecture.

5.1.1 Exclusive Parent Training Policy (EPT)

For individual-node class induction, the first step is the generation of the corresponding (binary) training set. Several methods have been proposed in the literature so far (Silla and Freitas 2010; Fagni and Sebastiani 2007; Eisner et al. 2005). The one we use in *HR*-SVM is referred to as EPT (<u>E</u>xclusive <u>P</u>arent <u>T</u>raining Policy). Note that EPT is similar to the "siblings policy" in Fagni and Sebastiani (2007) which was shown to be the best method to create binary training sets. Let us now describe it in detail as follows.

Let Tr be the set of all training examples, let $Tr(C_i)$ denote the set of training examples used for the induction of class C_i, and let $Tr^+(C_i)$, and $Tr^-(C_i)$ denote the sets of the positive and negative training examples of C_i, respectively. $|C_i|$ denotes the number of examples representing C_i, and $\uparrow C_i$ is the set of the parents of C_i. Finally, "$\backslash$" is the set exclusion operator. *HR*-SVM's way of choosing the training examples for the induction of C_i is defined as follows:

$$\begin{aligned} Tr(C_i) &= Tr^+(\uparrow (C_i)) \\ Tr^-(C_i) &= Tr(C_i) \backslash Tr^+(C_i) \end{aligned} \tag{5.1}$$

The set of C_i's training data includes all positive examples of its *parent class(es)*. In that set, examples are labeled as positive if they belong to C_i, while the remaining examples are labeled as negative. Thus, the number of training examples is equal to the total number of examples belonging to the parent class(es).

Let us now illustrate the process of EPT by two examples.

- Example 1: on a class with one parent, such as $C2.2$:
  - $Tr(C2.2) = Tr(\uparrow (C2.2)) = Tr^+(C2)$
  - $Tr^-(C2.2) = Tr(C2.2) \backslash Tr^+(C2.2)$,

    Note that $Tr^-(C2.2) \neq Tr^+(C2.1)$ because, if this is an NMLNP problem, some examples in $C2$ may belong to neither $C2.1$ nor $C2.2$.

- Example 2: on a class with multiple parents, such as $C2.1$:
  - $Tr(C2.1) = Tr^{+}(\uparrow (C2.1)) = (Tr^{+}(C1.2) \bigcup Tr^{+}(C2))$
  - $Tr^{-}(C2.1) = Tr(C2.1) \backslash Tr^{+}(C2.1)$.

The advantages of EPT are best illustrated by the comparison with another policy that has been used in the past, namely EAT (Exclusive All Training Policy) (Fagni and Sebastiani 2007) (Note that EAT was called "ALL" policy by the original paper), where each class is trained using the entire training set: $Tr(C_i) = Tr$, and $Tr^{-}(C_i) = Tr \backslash Tr^{+}(C_i)$. Assuming that the root node represents $|C0| = |Tr| = 1000$ examples, we have $|C2| = 100$ examples, $|C2.2| = 50$ examples, and $|C2.2.2| = 10$ examples. The sizes of the training sets generated by EPT and EAT, respectively, are given and compared in Table 5.1. The reader can see that, at the lower-level classes, EPT generates smaller and more balanced training sets than EAT. This can benefit the classification system in two aspects:

Table 5.1: Comparing the training sets generated by the EPT and EAT.

Class	The number of positive examples versus the number of all training examples	
	EPT	EAT
$C2$	100/1000	100/1000
$C2.2$	50/100	50/1000
$C2.2.2$	10/50	10/1000

First, the total induction time of all classifiers in EPT is faster that of EAT since EPT generates smaller training sets than EAT. For instance, the number of $C2.2$'s training examples followed by EPT is only 100 examples instead of using the whole data, 1000 examples, in EAT. Furthermore, it is unnecessary to train the classifier

$C2.2$ by $C1$'s examples because it is a responsibility of $C2.2$'s parent classifier, $C2$, for removing those $C1$'s examples.

Second, EPT can help to improve a classification performance of the system because it creates training sets with less degree of imbalance than EAT as shown in Table 5.1. Due to a large number of classes in this domain, a classifier always suffers from the scarcity of positive examples in the training data. For instance, for $C2.2$, the percentage of positive examples is extremely low in EAT ($\frac{50}{1000} = 0.05$), while it is higher ($\frac{50}{100} = 0.5$) in EPT.

5.1.2 Local Feature Selection (LFS)

In our domains, the examples are often described by thousands of attributes, which can lead not only to prohibitive induction costs, but also to performance degradation if many of the attributes are irrelevant. Importantly, relevance of the individual attributes can vary from class to class. Many scientists have studied attribute-selection techniques (see, e.g., (Guyon and Elisseeff 2003)). We need one that is computationally efficient, and we need to apply it *separately* to each class.

In our previous work (Vateekul and Kubat 2009; Dendamrongvit et al. 2011), we made a good experience with first ordering the attributes by their *gain ratio*, and then selecting a certain percentage of the highest ones. To estimate the gain ratio, we used the formulas from Quinlan's C4.5 (Quinlan 1993): they are easy to obtain from his public-domain software, and thus facilitate replicability of the experiments in this work.

In *HR*-SVM, we improved this (rather simplistic) approach by choosing those attributes that satisfy the following two conditions: the minimum accumulated gain ratio ($G\%$ of total gain ratio) and the minimum number of attributes ($P\%$ of the total number of attributes). The numbers of attributes followed the criteria $G\%$ and $P\%$ are denoted by i_G and i_P attributes, respectively.

Figure 5.4 illustrates the flowchart of LFS. The algorithm first gathers the gain ratio from the i_G highest-ranking attributes until the summation of the gain ratio meets the prior condition, $G\%$. This guarantees that the selected attributes do not lose too much information of the total gain ratio. However, the number of selected attributes i_G may sometimes be too small and affects the accuracy of classifiers. Thus, the latter condition on the minimum number of attributes i_P (or $P\%$ of the total number of attributes) helps to avoid this problem. If $i_G < i_P$, this module will collect more top-ranking attributes until the total number of chosen attributes is i_P.

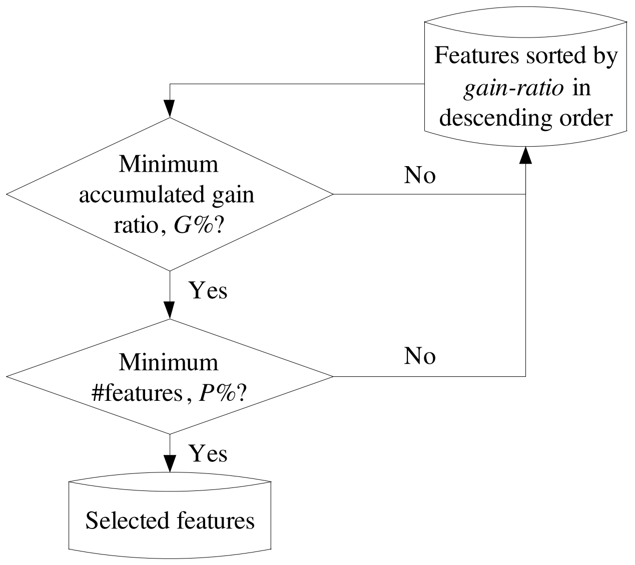

Features sorted by *gain-ratio* in descending order
Minimum accumulated gain ratio, *G%*?
No
Yes
Minimum #features, *P%*?
No
Yes
Selected features

Figure 5.4: The LFS flowchart.

The illustration of LFS is shown by the following examples; suppose the user has set $G = 95\%$ and $P = 25\%$:

- Example 1: let the gain ratios of the first example data set with 10 attributes be $\{0.40, 0.30, 0.10, 0.10, 0.05, 0.01, 0.01, 0.01, 0.01, 0.01\}$ (the summation of the total gain ratios being 1.0). Under these conditions, the first five attributes will be chosen because their sum of gain ratios is 0.95 (which satisfies the requirement of reaching at least 95% of 1.0), and five attributes are more than the required 25% of the total 10 attributes (3 attributes).
- Example 2: let the gain ratios of the second example data set with 10 attributes be $\{0.80, 0.15, 0.01, 0.01, 0.01, 0.01, 0.01, 0.00, 0.00, 0.00\}$, where the summation of the total gain ratios also being 1.0. On this data set, the first criterion on $G\%$ is satisfied by selecting only the first two attributes ($i_G = 2$) or the overall gain ratios equal to 0.95. However, one more attribute has to be included because the second criterion on $P\%$ requires at least $i_P = 3$ chosen attributes.

5.1.3 False-Positive Correction (FPC)

HR-SVM's next module seeks to correct the false positives (FP)— "false positive (FP)", incorrectly classifying data as positive. These FP-examples are also known as "*incorrect path errors*". For instance, the classifier $C2$ wrongly classifies $C1$'s examples, which come from the different path in the class hierarchy, as positive and propagates these errors to its subclassifiers.

Let us explain how this kind of error affect the system by giving an example of the classifier $C2.2$. The EPT policy ensures that classifier $C2.2$ is induced from examples belonging to the $C2.2$'s parent class, $C2$, and these do not include any examples of $C1$. Suppose a testing example of $C1$ is incorrectly labeled by the $C2$-classifier as positive as shown in Figure 5.5. This error is propagated to classifier $C2.2$, which has never been trained using $C1$'s examples, and may therefore fail. This error is then passed to $C2.2$'s subclasses, $C2.2.1$ and $C2.2.2$, which are likely to make the same mistake as $C2.2$. In this sense the FP errors from $C2$ negatively affect the performance of $C2$'s subclasses and the overall system.

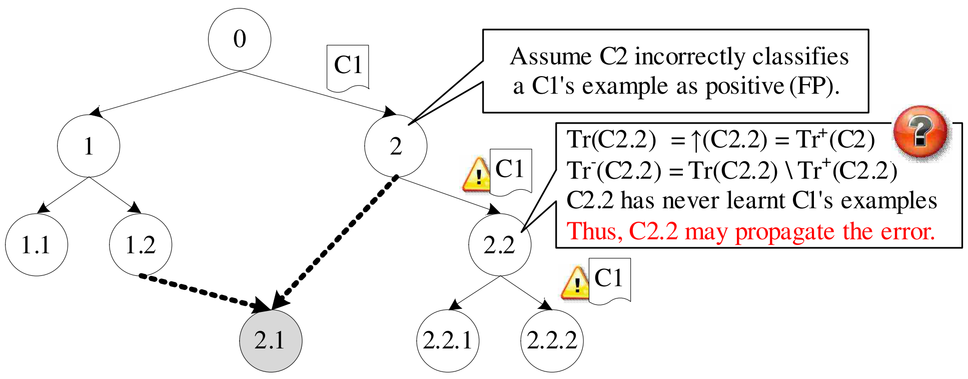

0
C1
Assume C2 incorrectly classifies a C1's example as positive (FP).
1
2
C1
Tr(C2.2) = ↑(C2.2) = Tr⁺(C2)
Tr⁻(C2.2) = Tr(C2.2) \ Tr⁺(C2.2)
C2.2 has never learnt C1's examples
Thus, C2.2 may propagate the error.
1.1
1.2
2.2
C1
2.1
2.2.1
2.2.2

Figure 5.5: A scenario in which $C2$ misclassifies a $C1$'s example as positive (FP).

HR-SVM addresses this issue by our *False-Positive Correction* strategy (FPC). The idea is to add to C_i's negative training examples, $Tr^-(C_i)$, also a set of FP examples at C_i's superclass(es) (denoted in Equation 5.2 by $FP(\uparrow(C_i))$), to give it a chance to learn how to correct these FP's inherited from the parents. Note that FPC cannot be applied to classifiers at the top level; the FPC process begins after the SVM models of all C_i's parents have been induced. These classifiers are then tested on their own training data, and the results are used to identify the FP examples which are finally added to the set of negative training examples at C_i, $Tr^-(C_i)$.

$$\begin{aligned} Tr(C_i) &= Tr^{+}(\uparrow(C_i)) + FP(\uparrow(C_i)) \\ Tr^{-}(C_i) &= Tr(C_i) \backslash Tr^{+}(C_i) + FP(\uparrow(C_i)) \end{aligned} \tag{5.2}$$

Figure 5.6 illustrates how the FPC strategy corrects the propagated FP-errors shown in Figure 5.5. At Classifier $C2.2$, the *potential* errors propagated from its parents, $FP(\uparrow (C2.2)) = FP(C2)$, are added to $C2.2$'s training set; thus, there can be more accuracy of Classifier $C2.2$ in identifying these propagated errors. Another example is Classifier $C2.1$, which has multiple parents. The extra data set added by the FPC strategy is $FP(\uparrow (C2.1)) = FP(C1.2) \bigcup FP(2)$.

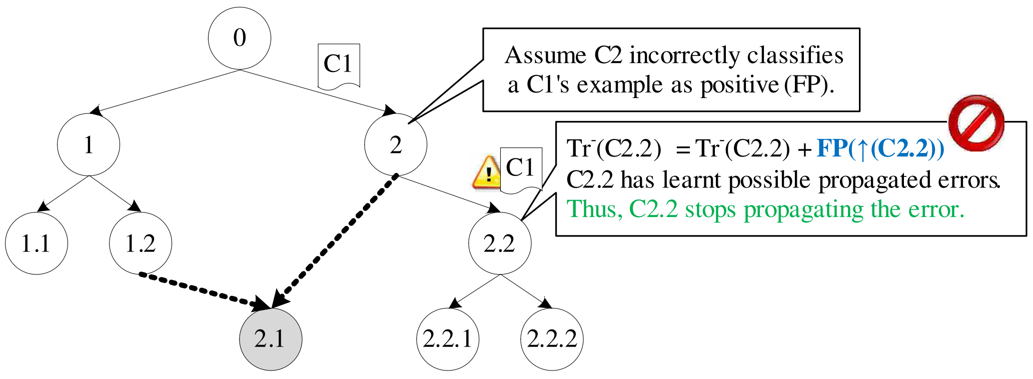

0
C1
Assume C2 incorrectly classifies a C1's example as positive (FP).
1
2
C1
Tr⁻(C2.2) = Tr⁻(C2.2) + FP(↑(C2.2))
C2.2 has learnt possible propagated errors.
Thus, C2.2 stops propagating the error.
1.1
1.2
2.2
2.1
2.2.1
2.2.2

Figure 5.6: The propagated error shown in Figure 5.5 is fixed by the FPC strategy.

We expect that the FPC strategy can potentially decrease the number of false positives, thus improving *precision* as well as F_1.

5.1.4 *R*-SVM

R-SVM is the last module in Box 4 of the *HR*-SVM's diagram. It is responsible for inducing a non-biased SVM's model on a set of training data generated by the first three modules. In addition, this module also aims to correct another type of propagated errors called "false negative (FN)".

In hierarchical classification domains with great many classes, negative examples tend to outnumber positive examples. Traditional classifiers are biased towards the majority (negative) creating the false negatives (FN). This kind of errors at the superclasses causes what is called *blocking* which directly affects accuracies of classifiers at the lower levels. For instance, Figure 5.7 illustrates that those of $C2.2.2$'s examples which have been misclassified by classifier $C2$ as negative will not be recognized as belonging to classes $C2.2$ and $C2.2.2$.

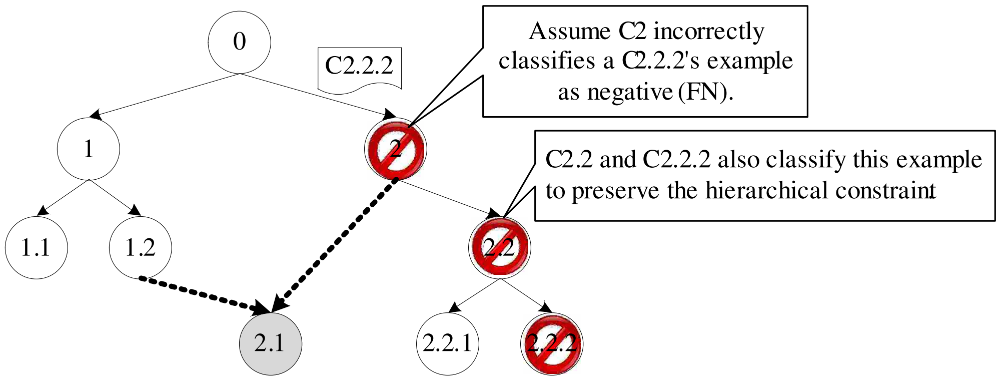

0
C2.2.2
Assume C2 incorrectly classifies a C2.2.2's example as negative (FN).
1
2
C2.2 and C2.2.2 also classify this example to preserve the hierarchical constraint
1.1
1.2
2.2
2.1
2.2.1
2.2.2

Figure 5.7: A scenario in which $C2$ misclassifies a $C2.2.2$'s example as negative (FN).

This issue can be addressed by reducing the bias towards the majority class in the baseline classifier for each class in the hierarchy which commonly induces on the imbalanced training set. The solution as demonstrated in Figure 5.8 is to replace the traditional SVM for each class node by R-SVM, our previous work in Section 4.3. R-SVM is a threshold adjustment algorithm specially proposed to rectify the imbalanced issue found in the traditional SVM; therefore, by employing it for each class node, the system is expected to have a decreased number of FN and to significantly improve the performance in terms of *recall* and also F_1.

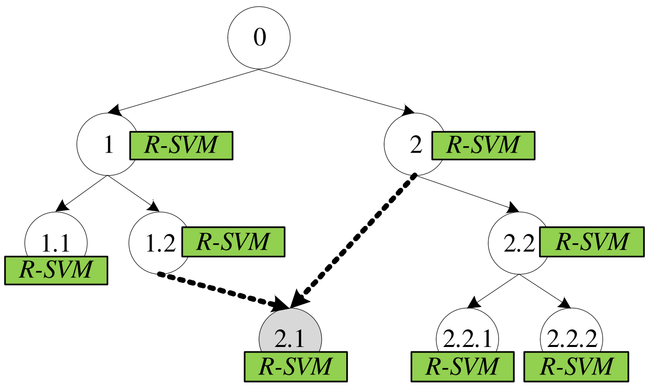


Figure 5.8: A non-biased classifier, *R*-SVM, is induced as a baseline classifier in order to solve the FN issue.

5.1.5 Complexity Analysis

Yang et al. (2003) presented a complexity analysis of an SVM-based algorithm on hierarchical as well as non-hierarchical domains. Since *HR*-SVM uses a threshold-adjusted SVM (*R*-SVM) as its baseline classifier, its complexity can be derived from this same analysis.

Let M be the number of classes, let N be the number of training examples, let V be the number of attributes, and let L_v be the average number of non-zero attributes. On multi-label (non-hierarchical) classification system, the training time of the traditional SVM is $O(MN^c)$ (where $c \approx 1.2 \sim 1.5$ is a domain-specific constant), and its testing time is $O(ML_v)$.

The training time of *R*-SVM is given by Equation 5.3, where c_1 and c_2 are constant times for (i) finding a set of candidate thresholds and (ii) searching the output threshold, respectively. Since the process of threshold adjustment is applied after the model induction, the first term in the equation is the SVM induction time, and the second term is the evaluation time of the SVM model on training data.

$$\begin{aligned} \text{Training Time} &= O(MN^c) + O(ML_v) + c_1 + c_2 \\ &= O(M(N^c + L_v)) + c_1 + c_2 \end{aligned} \tag{5.3}$$

For the hierarchical classification system, the total complexity of the top-down approach, including *HR*-SVM, is given by Equation 5.4, where h is the depth of the hierarchy, b is the number of branches at the leaf nodes, m_i is the number of classes at the i-th level, $i = \{0, .., h\}$ is an index for the hierarchical level, $j = \{1, ..., m_i\}$ is an index for the class at the i-th level, n_{ij} is the number of local training examples, N_i is the total number of training examples at the i-th level, $N_0 \leq N_i$, and π_{ij} is defined as $\frac{n_{ij}}{N_i}$.

$$b \times O(N_0^c) \sum_{h-1}^{i=0} \sum_{m_i}^{j=1} \pi_{ij}^c \tag{5.4}$$

5.2 Proposed Evaluation

How to evaluate performance in the domain of hierarchical classification is not an easy question, and the research community has not reached a consensus about how to proceed. We aim to improve the situation by developing evaluation criteria that we believe are sufficiently objective and robust.

Although there are evaluation criteria available for hierarchical classification as explained in Section 3.4, they are not adequate to assess the goodness of classifiers in the hierarchical classification domain. To see the limitation of those criteria, consider the domain from Table 5.2 and the class hierarchy from Figure 5.2. For five examples, the table lists their true class labels as well as the labels assigned to them by the classifier.

Table 5.2: An example of hierarchical classification results.

Example	True Class	Predicted Class
x_1	C1.1	C1.2
x_2	C2.2.1	C1.1, C2.2.1
x_3	C1, C2.2	C2.2
x_4	C1	C1.1
x_5	C1.1	C1

This information is expressed in the matrix form in Table 5.3 (top) for the true classes, and in Table 5.3 (bottom) for the classes predicted by the classifier. Note that each row represents an example, and each column represents a class. The value of a given field is "1" if the example belongs to the class and "0" if it does not.

The readers can see that the hierarchical measure in Section 3.4 evaluates only the classification performance "*for each example*" across *the row* of the matrix. On the other hand, the classification performance "*for each class*" is calculated along *the column* of the matrix by the multi-label measure in Section 2.5.2. Thus, both performance measures, "*for each example*" and "*for each class*", are suggested.

Incidently, such combination was encouraged by the organizers of a recent competition to develop the best system on "Large Scale Hierarchical Text Classification (LSHTC2)"[15]. In their notation, the multi-label criteria are referred to as "Label-based (Macro)" ($LbMa$), so that $LbPr$, $LbRe$, and LbF_1 refer to multi-label *precision*,

[15]http://lshtc.iit.demokritos.gr/

Table 5.3: The true class matrix T (top) and the predicted class matrix P (bottom) for the examples from Table 5.2.

	C1	C1.1	C1.2	C2	C2.1	C2.2	C2.2.1	C2.2
x_1	1	1	0	0	0	0	0	0
x_2	0	0	0	1	0	1	1	0
x_3	1	0	0	1	0	1	0	0
x_4	1	0	0	0	0	0	0	0
x_5	1	1	0	0	0	0	0	0

	C1	C1.1	C1.2	C2	C2.1	C2.2	C2.2.1	C2.2
x_1	1	0	1	0	0	0	0	0
x_2	1	1	0	1	0	1	1	0
x_3	0	0	0	1	0	1	0	0
x_4	1	1	0	0	0	0	0	0
x_5	1	0	0	0	0	0	0	0

recall, and F_1, respectively. Note that we ignore "Macro (Ma)." from the notations since this work considers only macro-average. Similarly, the hierarchical (Example-based) criteria are denoted by *EbPr*, *EbRe*, and EbF_1, respectively.

Whether to prefer example-based criteria or label-based ones may be a matter of some dispute, but a good classifier should satisfy both. In line with this argument, we propose a simple way to accomplish just that. Denoting the criterion by the acronym *ELb* (Example-Label-based), we define it by Equation 5.6 which is applied to the classical criteria: *precision*, *recall*, and F_1, as shown in Equation 5.6.

$$ELbFunc(Eb, Lb) = \tfrac{2\times Eb\times Lb}{Eb+Lb} \tag{5.5}$$

$$\begin{aligned} ELbPr &= ElbFunc(EbPr, LbPr) \\ ELbRe &= ElbFunc(EbRe, LbRe) \\ ELbF_1 &= ElbFunc(EbF_1, LbF_1) \end{aligned} \tag{5.6}$$

In some domains, classes at the upper levels are more important than those at the lower levels, so it is possible to modify this measure such that they receive more weight. However, we decided not to include this modification yet because the classification criteria without the weighting concept are more standard and suitable for comparing the real performance among different hierarchical classification algorithms.

5.3 Experimental Data

We experimented with real-world databases in the field of functional genomics provided by the DTAI webpage[16] (Schietgat et al. 2010) to predict gene functions of three organisms:

Saccharomyces cerevisiae (S. cerevisiae) is a baker's or brewer's yeast. It is one of biology's classic model organisms, and has been the subject of intensive study for years. The input feature vector for a gene consists of pairwise interaction information, membership to co-localization locale, possession of transcription factor binding sites and results from microarray experiments.

Arabidopsis thaliana (A. thaliana) is a small flowering plant native to Europe, Asia, and northwestern Africa. There are originally six datasets created by Clare et al. (2006), originating from different sources: sequence statistics, expression, predicted SCOP class, predicted secondary structure, InterPro, and homology. How-

[16]http://dtai.cs.kuleuven.be/clus/hmcdatasets/

ever, only annotations for the top four levels are given. Then, Schietgat et al. (2010) annotated the data into more levels.

Mus musculus (M. musculus) is a house mouse. This data is originally provided by MouseFunc challenge[17]. It consists of 21,603 genes of which 1718 are set aside as test genes. The data consists of several sources: gene expression data, protein sequence pattern annotations, protein-protein interactions, phenotype annotations, phylogenetic profile, and disease associations. The version of the dataset we use in our experiments is the preprocessed version provided by (Schietgat et al. 2010).

In this work, we worked with 8 data sets annotated by the functional hierarchy in Gene Ontology (GO) whose structure forms a DAG. Each data set is described by different aspects (attributes) of the genes that originate at diverse sources. The characteristics of all experimental data sets are summarized in Table 5.4.

Table 5.4: Properties of data sets used in experiments: the numbers of examples $|D|$, attributes $|A|$, classes $|C|$, and hierarchical levels $|H|$. "M?" indicates whether a data set includes missing values – yes (Y) or no (N).

Organism	**Id**	**Feature Description**	$\|D\|$	$\|A\|$	$\|C\|$	$\|H\|$	$M?$
S. cerevisiae	D0	Joining all sources	3465	5931	132	7	N
A. thaliana	$D13$	Sequence statistics (seq)	11763	4451	629	6	Y
	$D14$	Affy.'s experiments (exprindiv)	10840	1252	626	6	Y
	$D15$	SCOP superfamily (scop)	9843	2004	571	6	N
	$D16$	Secondary structure (struc)	11763	14805	629	6	N
	$D17$	InterProScan (interpro)	11763	2816	629	6	N
	$D18$	All microarray (expr)	11121	72870	622	6	N
M. musculus	D19	Joining all sources	21153	18748	5620	13	Y

Since the data contained nominal attributes, and many values were missing, some pre-processing was necessary.

[17]http://hugheslab.med.utoronto.ca/supplementary-data/mouseFuncI

1. *Data cleaning*: Assuming that rare classes cannot be reliably induced, we ignored all classes that were represented by less than 1% of total examples, which is about 50 examples on all data sets, except in the very large domain $D19$ where this minimum was set to 200 examples. Then, we deleted examples with none attribute values, and we, finally, removed attributes that were never used in the "surviving" examples.

2. *Missing value imputation*: In the case of nominal attributes, we replaced a missing value with the most common value; in the case of continuous attributes, we used the average (mean). Note that missing data is commonly encountered in real-world data; thus, a new imputation method, "Imputation Tree (ITree)," has been developed in this dissertation to estimate missing values as reported in Appendix A. However, ITree was not used in this work yet since it can handle only the imputation of numerical data, not categorical data.

3. *Nominal-to-numerical conversion*: Some attributes were nominal, acquiring one out of m different values. We converted each of these nominal attributes to a set of m binary attributes. For instance, an attribute with three values, $\{A, B, C\}$, was replaced with the triplet of binary attributes whose possible combinations of values were limited to (0,0,1), (0,1,0), and (1,0,0).

4. *Data transformation*: It is necessary to transform data if a range of feature values is extremely small or large. Among all data sets, we applied $log10$ to the data set $D15$ since the original range of values is small, [3.8E-123, 5.6E-108].

5. *Scaling range of feature values*: The main advantage of scaling is to avoid features in greater numeric ranges dominating those in smaller numeric ranges. Hence, we normalized the values of numeric attributes to the interval $[0, 1]$ in the same way for both training and testing data.

Each domain provided by the DTAI website consists of 3 files that were originally intended for training, validation, and testing, respectively. To facilitate the evaluation of statistical significance of performance comparisons, we merged these three files in one, and then used 5-fold cross-validation.

5.4 Experiments

HR-SVM induces a hierarchical classifier by a mechanism built around the publicly available *svmlight*[18]. As for the kernel function, preliminary experiments revealed that the linear function gave better results (in terms of F_1) than the radial basis function (RBF). This can be expected: the training examples being described by thousands of attributes, the individual classes were easy to separate from each other linearly.

The task of the experiments reported below is twofold. First, to show that our system outperforms earlier attempts known from the literature, namely *H*-SVM and Clus-HMC[19]. Second, to find out how each of its modules (see Figure 5.3) contributes to the overall performance. To this end, we added these modules one by one, obtaining the four variants listed in Table 5.5 where *H*-SVM is a hierarchical version of the traditional SVM (Fagni and Sebastiani 2007; Fagni and Sebastiani 2010), and

[18]http://svmlight.joachims.org/

[19]The package is available at `http://dtai.cs.kuleuven.be/clus/`.

HR-SVM-ALL is the complete system. The only exception were the experiments with the domain $D18$ where only *H*-SVM-ALL could be experimented with—the high number of attributes made it impossible to use SVM-based systems without attribute selection (LFS).

All results are in terms of the example-label based macro-averaging (ELbMa) version of the performance criteria from Section 5.2.

Table 5.5: Evolution of *HR*-SVM's framework.

System	Descriptions
H-SVM	**EPT** + The baseline SVM
*H**R***-SVM	EPT + ***R*-SVM**
HR-SVM-**FPC**	EPT + *R*-SVM + **FPC**
HR-SVM-**ALL**	EPT + *R*-SVM + FPC + **LFS**; (all modules)

5.4.1 The system's performance compared to previous work

A new technique is usually deemed useful if its performance compares favorably with that of other techniques. Therefore, we compare here *HR*-SVM-ALL with *H*-SVM (a hierarchical variant of SVM) and Clus-HMC (a global approach that relies on decision trees).

The results in terms of F_1 are summarized in Figure 5.9, and the computational times are given in Table 5.6. The reader can see that, in terms of F_1, *HR*-SVM-ALL outperforms *H*-SVM in all domains, the results being particularly impressive in $D0$; *H*-SVM achieved only $F_1 = 0.0929$, whereas *HR*-SVM-ALL scored $F_1 = 0.4811$ (an improvement of more than 400%). The new technique induced faster than *H*-SVM on all data sets except $D14$ and $D17$—in these, the efficiency of the employed attribute-

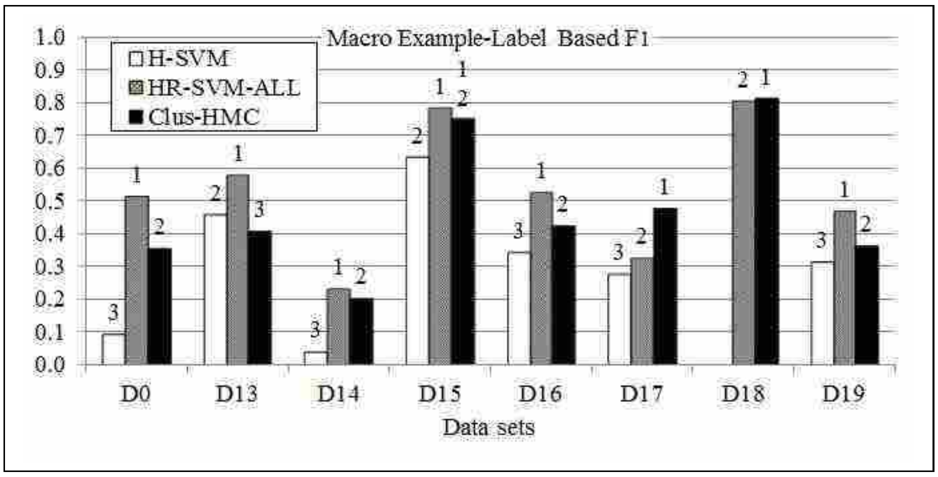


Figure 5.9: Comparing the performance (along F_1) of *H*-SVM, *HR*-SVM-ALL, and Clus-HMC. The integers over each of the vertical bars give the ranks as obtained by the ANOVA methodology followed by Bonferroni multiple comparisons.

selection technique was impaired by the additional time needed to process the FP data sets (FPC). Comparing with Clus-HMC along F_1, *HR*-SVM-ALL was significantly better on six out of eight domains. The induction time is comparable, *HR*-SVM-ALL being faster than Clus-HMC in $D0$, $D13$, $D16$, and $D17$, and slower in $D14$, $D15$, $D18$, and $D19$.

Table 5.6: The total induction time (in seconds) of *HR*-SVM-ALL and two other systems, *H*-SVM and Clus-HMC.

Data set	*HR*-SVM-ALL	*H*-SVM	Clus-HMC
D0	7.68	10.26	31.04
D13	204.68	357.75	999.27
D14	3,615.49	1,900.42	208.90
D15	13,641.90	17,408.56	159.44
D16	406.99	526.25	1,164.25
D17	141.36	100.76	146.05
D18	2,989.98	N/A	604.06
D19	17,691.80	23,334.07	3,396.86

5.4.2 Contributions of ***HR***-SVM's modules

In the next step, we wanted to find out how much each of the modules listed in Figure 5.3 contributed to *HR*-SVM's classification performance (see Table 5.7). The induction times of the individual consecutive steps are summarized in Table 5.8. The following subsections discuss the classification-performance aspects.

Table 5.7: Hierarchical systems (see also Table 5.5) whose performance is to be compared.

Module	System Comparison
1. *R*-SVM	*HR*-SVM vs. *H*-SVM
2. FPC	*HR*-SVM vs. *HR*-SVM-FPC
3. LFS	*HR*-SVM-ALL vs. *HR*-SVM-FPC

Table 5.8: Induction time (in seconds) of the *top-down* hierarchical classification systems from Table 5.5. In each column, the percentages in the parentheses give the time increase over the previous column.

Data set	*H*-SVM	*HR*-SVM	*HR*-SVM-FPC	*HR*-SVM-ALL
D0	10.26	10.33 (+<1%)	10.95 (+6%)	7.68 (-30%)
D13	357.75	358.24 (+<1%)	393.01 (+10%)	204.68 (-48%)
D14	1,900.42	1,901.23 (+<1%)	4,656.21 (+145%)	3,615.49 (-22%)
D15	17,408.56	17,408.77 (+<1%)	12,002.88 (-31%)	13,641.90 (+14%)
D16	526.25	527.11 (+<1%)	623.81 (+18%)	406.99 (-35%)
D17	100.76	101.11 (+<1%)	123.15 (+22%)	141.36 (+15%)
D18	N/A	N/A	N/A	2,989.98
D19	23,334.07	23,336.36 (+<1%)	40,320.39 (+73%)	17,691.80 (-56%)

The effect of ***R***-SVM

In *HR*-SVM, SVM was replaced by *R*-SVM with the intention to reduce SVM's bias to the majority class (recall that our experimental domains were dominated by false negatives). The hypothesis that *R*-SVM helps is verified by comparing *HR*-SVM's performance with that of *H*-SVM.

The results in Figure 5.10 show that *HR*-SVM had better F_1 than *H*-SVM in all domains. In *D*0, *HR*-SVM's F_1-increase over the baseline SVM was more than 400% (from 0.0929 to 0.4693). As for computational costs shown in Table 5.8, both systems needed about the same time. This indicates that the additional thresholding time in *HR*-SVM is very small (only at most 1% of the SVM induction time). Especially in *D*19, we spent only 2.29 seconds to adjust the separation hyperplane in addition to the induction time of the traditional SVM, 23,334.07 seconds.

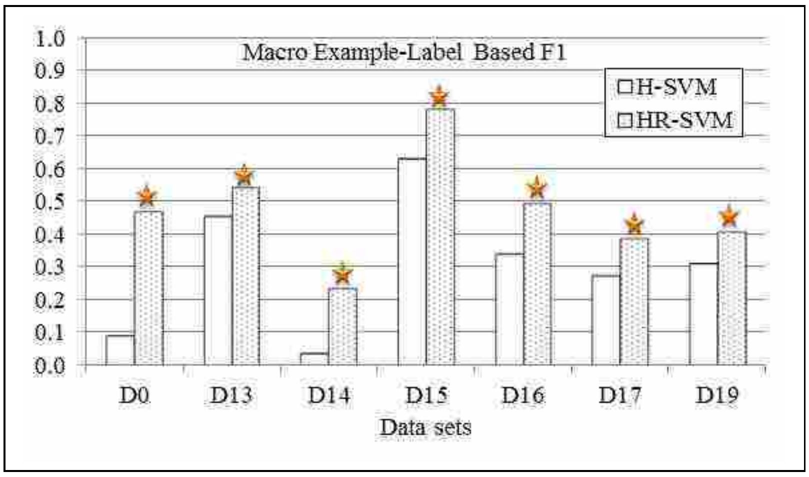


Figure 5.10: Comparing *HR*-SVM with *H*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to *t*-tests (0.05 level).

HR-SVM's classification success was largely due to its significant improvement of *recall* (as illustrated in Figure 5.11), which was achieved at the cost of slightly reduced *precision* on some domains. However, the average of *precision* on all data sets still increased by 25%. It shows that *R*-SVM properly adjusts the class-separation hyperplane, thus alleviating the bias to the majority class, which in turn means that the number of false negative examples propagated downward the class hierarchy is significantly reduced.

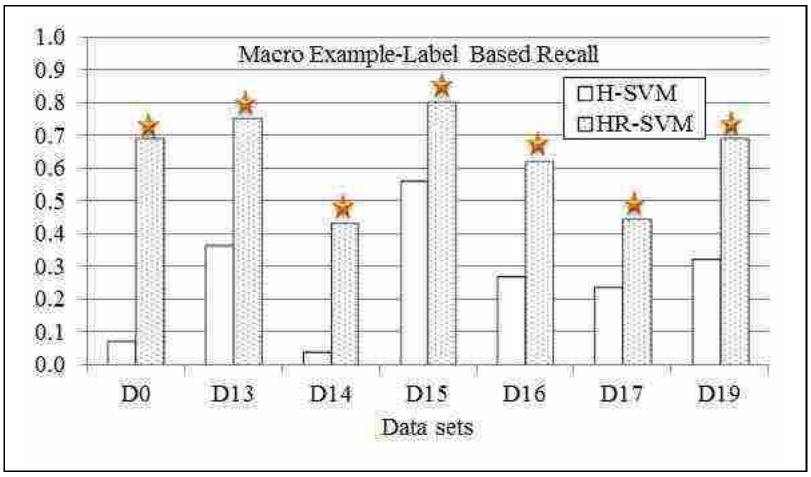


Figure 5.11: Comparing *HR*-SVM with *H*-SVM in terms of *recall*. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

The effect of FPC

Let us now compare *HR*-SVM-FPC with *HR*-SVM, thus establishing how the system's performance benefits from False-Positive Correction. Recall that false positives occuring at higher levels are propagated to lower levels in the class hierarchy. The total number of false positives can be high, leading to decreased *precision* and F_1.

The summary of the experimental results (Figure 5.12) indicates that *HR*-SVM-FPC outperforms *HR*-SVM in almost all domains. In 5 out of 7 domains, the improvement is statistically significant. As indicated by Figure 5.13, this is due to FPC's ability to increase *precision* in all eight domains (13% on average), while *recall* slightly decreases on some domains. This points that the number of FP-errors propagated throughout the system was indeed decreased.

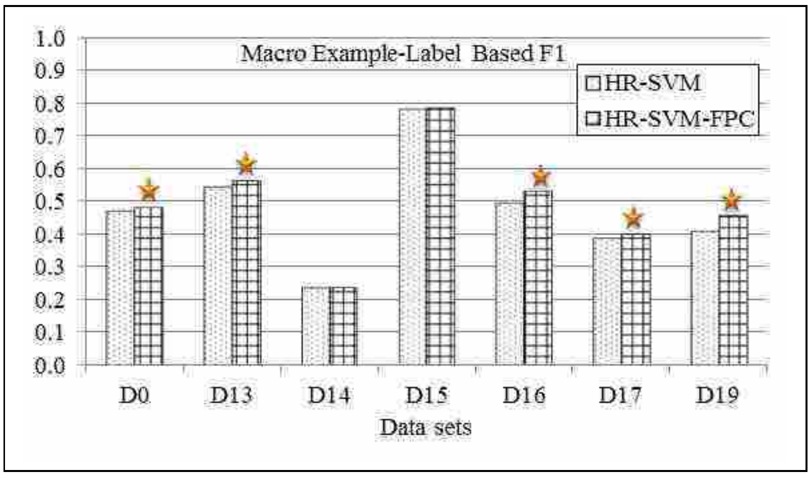


Figure 5.12: Comparing *HR*-SVM-FPC with *HR*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to *t*-tests (0.05 level).

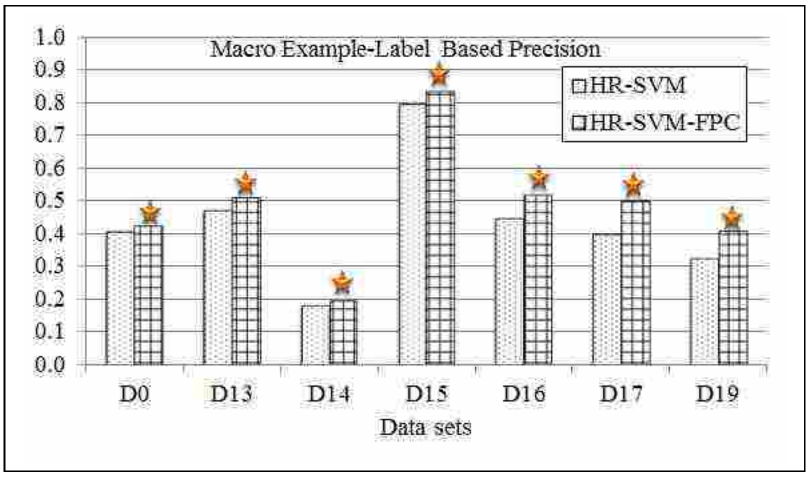


Figure 5.13: Comparing *HR*-SVM-FPC with *HR*-SVM in terms of *precision*. The stars above some of the bars indicate significant improvements according to *t*-tests (0.05 level).

Table 5.8 shows how much FPC increases the computational costs (on account of the extra false positives added to the training data). Note that, in $D15$, FPC led to reduced induction time because the SVM model here converged faster in spite of the training data being larger (a phenomenon explained by (Shwartz and Srebro 2008)).

The effect of LFS

The results of the comparison between *HR*-SVM-ALL and *HR*-SVM-FPC give us an idea of the the contribution of the attribute-selection technique (LFS).

Recall that LFS relies on two user-defined parameters: (*i*) the minimum percentage of accumulated *gain ratio* and (*ii*) the minimum number of attributes. In the experiments reported below, we relied on experience from our earlier research by setting the former at 95% of the overall gain ratio (Dendamrongvit et al. 2011) and the latter at 25% of the total number of attributes (Vateekul and Kubat 2009).

Comparing to *HR*-SVM-FPC in terms of F_1, Figure 5.14 shows that *HR*-SVM-ALL won on $D0$, $D13$, and $D19$, but lost on $D17$. The results of the remaining data sets ($D14$, $D15$, and $D16$) are statistically indistinguishable. Figure 5.15 illustrates that the removal of less relevant attributes indeed improved *precision* on all data sets, besides $D17$.

Table 5.8 showed that LFS reduced the training time in most domains, most remarkably in $D19$ where the training time was reduced by 56% while F_1 also improved by 2%. However, *HR*-SVM-ALL's induction costs increased in $D15$ and $D17$. This is caused by SVM's *inverse convergence* (Shwartz and Srebro 2008). LFS is especially useful in $D18$ where the number of attributes is so high, comparing to other data sets,

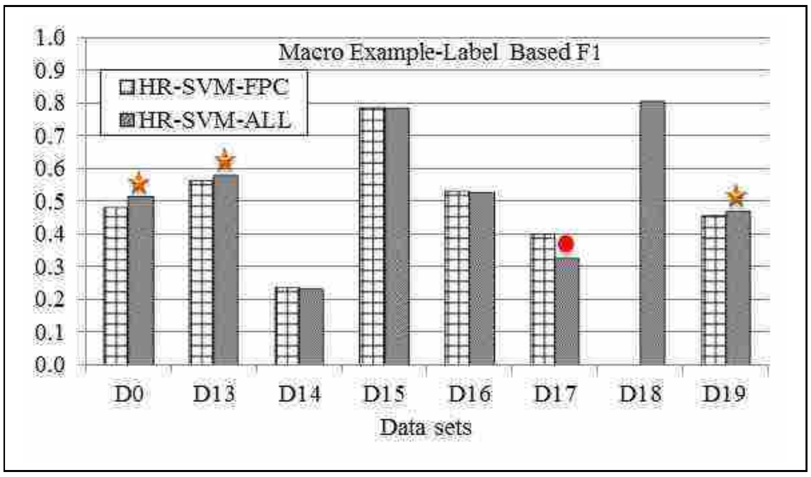


Figure 5.14: Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of F_1. The stars and circles above some of the bars indicate significant improvements and declines according to t-tests (0.05 level).

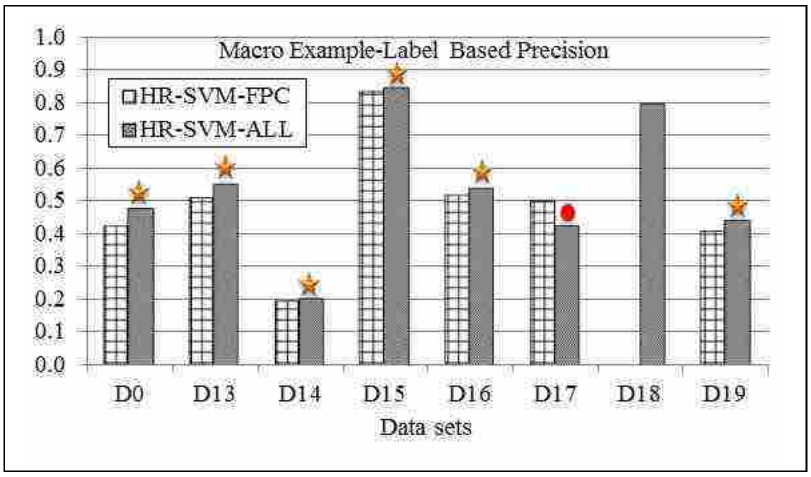


Figure 5.15: Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of *precision*. The stars and circles above some of the bars indicate significant improvements and decline according to t-tests (0.05 level).

that memory-resident algorithms cannot load the whole data set into the computer's memory.[20]

In conclusion, attribute selection is beneficial in our experimental data sets, where the examples are described by great many attributes. LFS increases F_1 by improving *precision*; it also reduces computational costs.

5.4.3 Performance at different hierarchical levels

Let us now try to gain more insight by comparing the classification performance of *HR*-SVM-ALL, *H*-SVM, and Clus-HMC, at different levels of the class hierarchies.

Note that the label-based metrics (*Lb*) are here better suited than the example-label based ones (*ELb*). The *ELb*-metrics evaluate the performance along the *class paths*, such as $C1 \rightarrow C1.1 \rightarrow C1.1.1$, which does not make much sense when evaluating each class-level independently.

Three factors are known to affect an induced classifier's performance: (*i*) the training set size, (*ii*) the number of positive examples, and (*iii*) the data distribution (e.g., the degree of imbalance). Hierarchical domains add one more factor: the "propagated error"—the accuracy of lower-level classifiers is related to that of the parent classifiers.

Table 5.9 summarizes the expected tendencies. Note that the accuracies of higher-level classifiers suffer from highly imbalanced training data, and the accuracies of lower-level classifiers suffer from the scarcity of positive training examples and from

[20]In the Java implementation of *HR*-SVM, each attribute value is treated as "double" (8 bytes). Therefore, at least 6.5 GB of memory in the 32-bit system are needed to handle $D18$. On the other hand, Clus-HMC treats each value as an "integer" (4 bytes), thus needing less memory space (3.2 GB).

the propagated errors. As we go down the hierarchy, the impact of these factors increases and the performance of the lower-level classifiers declines.

Table 5.9: Expected impact of four aspects: (*i*) the training-set size, (*ii*) the number positive examples, (*iii*) the degree of imbalance, and (*iv*) the downward-propagated errors. "$>$" indicates an increase and "$<$" indicates a decrease.

Classes	#Training	#Positive	Imb. Ratio	Prop. Errors
Upper levels	$>$	$>$	$>$	$<$
Lower levels	$<$	$<$	$<$	$>$

Let us first take a closer look at one of the domains: $D0$. Figure 5.16 plots the classification performance at the different levels. We can see that *HR*-SVM-ALL outperforms *H*-SVM and Clus-HMC at all levels, and *H*-SVM always lags behind the other two. As seen in Table 5.10, the performance at the highest level is negatively affected by the highly imbalanced class representation. The percentage of positive examples at this level is only 0.07%.

Even more insight is gained from Table 5.11 which gives the number of classifiers that incorrectly label *all* examples as negative (which means that $F_1 = 0$). Since the traditional SVM suffers from imbalanced class representation, most classifiers in *H*-SVM have $F_1 = 0$. However, *R*-SVM was designed in a way that improves its "reaction" to imbalanced classes, and this is why there are no classifiers with $F_1 = 0$ in the case of *HR*-SVM-ALL. On the other hand, Clus-HMC fails to recognize examples of some classes (resulting in $F_1 = 0$).

In all other domains (apart from $D0$), the F_1-results were similar to those shown in Figure 5.17 for $D16$. As we proceed to lower levels, F_1 gradually decreases. This is caused by the decreasing numbers of training examples (Table 5.12) and by the

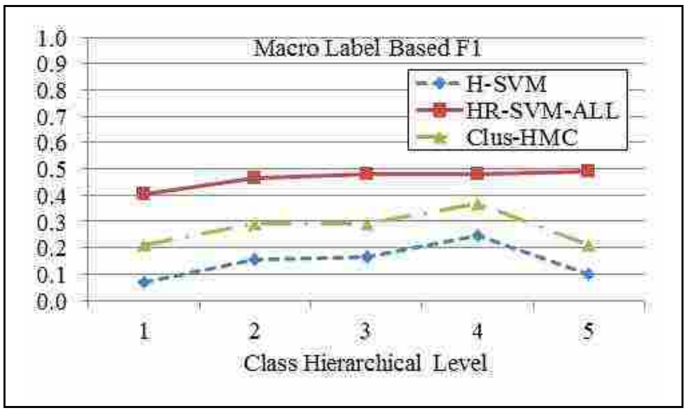


Figure 5.16: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in $D0$.

Table 5.10: Statistics for $D0$'s hierarchical levels.

Level	#Classes	AvgPositive	AvgTotal	%AvgPositive
1	23	188.39	2,834.00	**0.07**
2	18	148.94	333.56	0.52
3	13	111.92	313.46	0.40
4	7	72.57	180.00	0.53
5	3	71.33	108.33	0.69

Table 5.11: The number of classifiers with $F_1 = 0$ in $D0$.

Level	Classes	*H*-SVM	*HR*-SVM-ALL	Clus-HMC
1	23	14	0	4
2	18	9	0	1
3	13	5	0	2
4	7	2	0	0
5	3	2	0	0

errors propagated from higher levels (Table 5.13). F_1 at the fourth hierarchical level is slightly better than that at the third level because its ratio of imbalance is lower. Comparing the three hierarchical classification systems, we see that *HR*-SVM-ALL exhibits the best classification performance, whereas *H*-SVM is the worst.

From the analysis in this section, we can conclude that *HR*-SVM-ALL handles the imbalanced training sets especially at the upper levels of the class hierarchy better than other systems. It also takes care of propagated errors at the lower levels resulting in none (or few) classifiers that cannot correctly predict any positive examples (F_1=0).

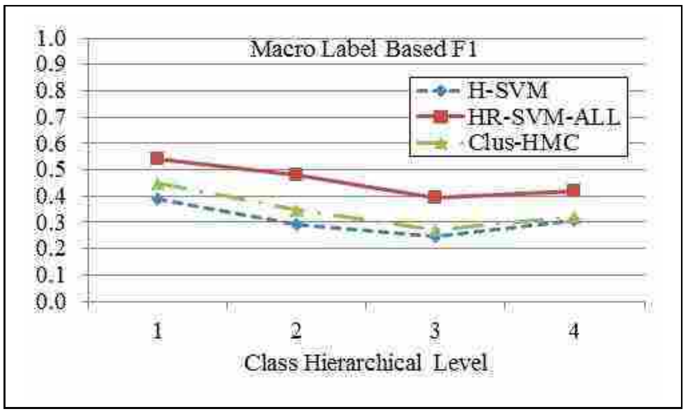


Figure 5.17: Comparing F_1 of *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different class hierarchical levels in *D*16.

Table 5.12: Statistics for *D*16's hierarchical levels.

Level	#Classes	AvgPositive	AvgTotal	%AvgPositive
1	11	1093.27	9548.00	0.01
2	28	391.00	2814.82	0.21
3	41	188.22	795.83	0.34
4	34	162.68	318.71	0.56

Table 5.13: The number of classifiers with $F_1 = 0$ in $D16$.

Level	Classes	*H*-SVM	*HR*-SVM-ALL	Clus-HMC
1	11	5	0	0
2	28	11	2	0
3	41	17	3	9
4	34	9	2	4

5.5 Conclusion

We proposed *HR*-SVM, a new top-down technique for induction of multi-label classifiers in domains with hierarchically organized classes. In designing it, we followed the common strategy to induce a separate binary classifier for each class in the hierarchy, and to employ higher-level classifiers when creating the training sets for the induction of lower-level classifiers. The paper described the system, and then reported experiments illustrating its performance as well as diverse aspects of its overall behavior.

Top-down approaches often suffer from two issues: imbalanced class representation and top-down error propagation. *HR*-SVM addresses them explicitly by 4 proposed concepts: (*i*) exclusive-parent training sets (EPT), (*ii*) attribute-selection module (LFS), (*iii*) a mechanism to correct false positives (FPC), and (*iv*) the correction of the majority-class bias. The first three are data pre-processing techniques that help generate smaller and not-so-imbalanced training sets, “enriched” by examples misclassified by parent classifiers. The last module, *R*-SVM, seeks to induce unbiased SVM-hyperplanes.

Importantly, we argue that, in hierarchical classification, the usual metrics for performance evaluation (*precision, recall*, and F_1) are not ideally suited. By way of rectification, we introduced a new measure, “example-label based macro-averaging,” that uses the harmonic mean of macro-averaging performances in two dimensions, *per example* and *per class.*

We applied our system to eight real-world domains from the the field of gene-function prediction of three organisms: *S. cerevisiae, A. thaliana,* and *M. musculus.* These data are available at the DTAI website. In each domain, the data were annotated by their own functional hierarchy in Gene Ontology (GO), whose structure is a directed acyclic graph (DAG).

Experimenting with these domains, we observed that our system significantly outperformed alternative hierarchical-classification techniques such as *H*-SVM (a top-down hierarchical version of SVM) and Clus-HMC (a "global" approach based on decision trees). Especially in $D0$, the F_1-improvement of *HR*-SVM over *H*-SVM is about 400%, while the induction costs are much lower. The explanation is that the module EPT creates training sets that are less imbalanced, the module *R*-SVM reduces the number of false negatives (which resulted in better *recall*), the module FPC decreases the number of false positives (which leads to higher *precision*), and the module LFS removes irrelevant attributes (which saves a lot of induction time).

CHAPTER 6

Conclusion and Future Work

Over the last decade or so, the need for systems capable of automated categorization has become more and more urgent. The main reason is that, with the advent of the world-wide web, the amount of available data is growing so fast that it is impractical—even impossible!—to do it all manually. In the domain of gene function annotation, the input is an unknown gene product (example), and the task is to classify it into one of a set of predefined functions (e.g., Gene Ontology). The number of these functional classes is large. Moreover, each gene may belong to multiple classes at the same time (multi-label classification), and the classes are hierarchically structured (hierarchical classification). Finally, the representation of the individual classes in the data is heavily unbalanced. Apart from these specific complications, there are also the more traditional ones such as uncertain and incomplete class labels (non-mandatory leaf-node problems (NMLNP)). Seeking to address all these issues in a single system, this dissertation offered practical solutions for "*hierarchical multi-label classification*," focusing on the general case where the generalization-specialization relations among classes are specified in terms of a directed acyclic graph (DAG).

6.1 Summary and Contributions

The first part of this dissertation examines the problem of classifier induction from multi-labeled examples, as exemplified by the field of large-scale text categorization systems. The author followed the most common approach of inducing a separate binary classifier for each class, and then employing them all in parallel. Analysis of the low classification accuracy of these classifiers indicated that the cause of this under performance is in the *imbalanced training sets* (one class outnumbering the other). Fortunately, machine-learning literature offers enough guidance as to the general principles of dealing with this issue. Another difficulty is presented by the impractically high computational costs involved in applications with great many classes and great many features.

The author's original attempt to deal with these problems was implemented in the framework of his "Fast Induction Tree (FDT)"(Vateekul and Kubat 2009). The application domain targeted in this stage was decision-tree induction in large scale text collection,the EUROVOC data. FDT reduces computational costs by "feature preselection" using *the information gain-ratio criteria* and "data partitioning," the latter seeking to induce a set of decision trees from different subsets. In the classification phase, these trees combine their outputs (data fusion) by a mechanism that favors minority classes. Experimental results showed that FDT achieved significant savings in CPU time; however, their classification performance (expressed in the F_1-metric) still left a lot to be desired.

Later experiments showed that although FDT outperformed SVM in terms of F_1, the performance of SVM could be improved by the use of an appropriate threshold adjustment strategy. This led us to the development of a novel multi-label classification algorithm based on SVM. "*R*-SVM" (Vateekul et al. 2011) is another framework proposed here for the needs of multi-label classification. Here is the basis: it is known that the SVM hyperplane can be biased toward the majority class on the imbalanced training data and needed to readjust by an appropriate translation of the SVM hyperplane (threshold). The work reported here uses its own novel strategy to find the best threshold, relying on results from a group of diverse data samples. Having experimented with six real-world benchmark domains, it was observed that *R*-SVM compared favorably with the baseline SVM as well as with other earlier methods: $\texttt{SVM}_{F_1}$, `ScutFBR`, and `BetaGamma`.

However, more important is the second part of the dissertation that seeks to extend the principle of *R*-SVM to the more difficult problem of hierarchical multi-label classification. The resulting system is referred to as *HR*-SVM. In essence, it is based on a top-down approach: inducing a separate classifier for each class, and then proceeding to training and testing from the classifiers at upper levels all the way down to bottom levels. Two ways to further improve the predictive power were identified. First, the negative examples usually outnumber the positive ones, which means that adequate methods to deal with the problem of imbalanced classes are called for. The second issue is known as "propagated errors": errors committed at upper levels are propagated down the hierarchy, thus negatively affecting the overall performance. All these issues are addressed by the four modules of *HR*-SVM. These include (*i*) the exclusive-parent training policy, (*ii*) the local feature selection (LFS), (*iii*) the false-positive correction, and (*iv*) *R*-SVM. The first three rely on data preprocessing to help generate a smaller and less-imbalanced training set "enriched" by misclassified data from parent classifiers (the latter seeks to offer the opportunity to correct previous errors). The last module, *R*-SVM, is responsible for inducing a non-biased SVM hyperplane.

In hierarchical classification, classical performance measures, such as *precision*, *recall*, and F_1, are not adequate. This is why this dissertation introduced a new measure, "example-label based macro-averaging," that uses the harmonic mean of macro-averaging performances in two dimensions, called "*per example*" and"*per class*." The result is a way to measure the performance on a "*per example/class*" basis. As an application domain, the author chose the field of gene function prediction, relying on the real-world databases available at the DTAI website. Here, the task is to predict gene functions of three organisms: S. cerevisiae, A. thaliana, and M. musculus. The experiments were run on 8 data sets annotated by the functional hierarchy in Gene Ontology (GO) whose structure is DAG. Each data set describes different aspects (features) that may be from various sources.

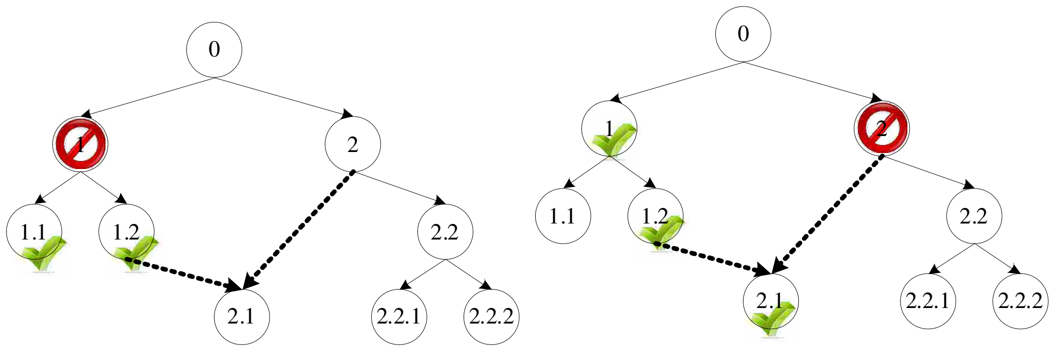
In early experiments, the results proved that each developed concept in*HR*-SVM can really improve the classification performance in terms of F_1. The author then compared *HR*-SVM to other hierarchical-classification algorithms, *H*-SVM (a top-down hierarchical version of SVM) and Clus-HMC (a"big-bang" approach, also available from the DTAI website). The results showed that *HR*-SVM significantly outperformed *H*-SVM on all data sets, while outperforming Clus-HMC on 6 out of 8 data sets.

6.2 Future Research Direction

Although experiments have confirmed the promise of the proposed hierarchical-learning framework, further extensions and improvements are possible.

One is the improvement of the false-positive correction concept (FPC). The idea is two-fold: (*i*) assigning more weight to propagated FP-examples that have been misclassified by classifiers at higher levels, and (*ii*) combining this with the threshold adjustment in R-SVM. Future research should show whether this would lead to the reduction of the number of FP examples, thus resulting in improved *precision* and F_1.

The second possibility would be to propose *a new testing mechanism* of the top-down approach. Although there are existing top-down hierarchical classification techniques, it is surprising that all of them propose algorithms only at *the induction phase*; thus, an invent of the testing technique is still an unexplored area. In the top-down approach, a set of classifiers are induced and collaborated. The motivation is to recover mistaken predictions of classifiers by considering those of their neighbor classifiers during the testing phase. Figure 6.1(a) illustrates the case in which Classifier $C1$ and its subclassifiers have conflicting results, assuming that $C1$ makes a mistake. In the new system, the result of $C1$ should be corrected to positive following that of its subclassifiers, while the traditional testing process does not recover the mistake of $C1$ and even change the results of $C1.1$ and $C1.2$ to negative to preserve the hierarchical constraint. Figure 6.1(b) demonstrates the case of Classifier $C2.1$ which has multiple parents, and its parents give conflicting predictions. Assume that the disagreement at $C2$ is actually a misclassification; this error must be addressed and recovered in the new testing system. Therefore, the collaboration among classifiers at the test phase is expected to recover misclassification and improve the overall accuracy of the system.



(a) The confliction between parent and child classes.

(b) The confliction between two parents.

Figure 6.1: Examples of the conflict at *the testing phase*. The mark marks and the "no" signs denote classifying as *positive* and *negative* respectively.

Third, the author plan to improve the algorithm to handle extremely large scale data. Hierarchies are becoming ever more popular for the organization of text documents, particularly on the web. Very recently (this year), the 2nd edition of the Large Scale Hierarchical Text Classification (LSHTC) Pascal Challenge[21] has been announced. In the challenge, there are three large data sets based on Wikipedia and DMOZ as summarized in Table 6.1. They are much larger than the data sets of the gene function prediction domain as shown in Table 5.4. With these sizes of the data, neither any memory-resident algorithms nor any top-down hierarchical classification systems, including *HR*-SVM, can process them. As for the Wikipedia Large one, the traditional top-down system has to induce at least 325,056 classifiers which requires

[21]http://lshtc.iit.demokritos.gr

Table 6.1: The statistics of the challenging data sets in the 2nd LSHTC. MD refers to the maximum depth of the hierarchy.

Data set	Categories	Stems (Feature)	Train Doc.	Test Doc.	MD
DMOZ	27,875	594,158	394,756	104,263	5
Wikipedia Small	36,504	643,328	456,886	81,262	10
Wikipedia Large	325,056	1,766,163	2,365,436	452,167	14

almost the whole year spent for the total induction time, assuming one minute per classifier. Thus, it is challenging to improve *HR*-SVM to handle these data. One possible approach is to reduce the number of classes by employing the clustering techniques. Another solution is to run the program in the parallel or distributed system.

Another direction for future research is to implement an incremental version of *HR*-SVM. The amount of digital information has been exponentially growing. For instance, due to the wealth of bimolecular technology, several descriptions of genes are being discovered every day, and the number of new webpages is drastically growing. Unfortunately, the reconstruction of the whole classifier seems to be impractical. The only solution left is to partially update the model.

In conclusion, there are four possible research directions of the developed hierarchical system: (*i*) an improvement of the training process on the FPC concept, (*ii*) a self-error-corrected testing process, (*iii*) a solution to handle the extremely large scale data, and (*iv*) the incremental learning concept. All of these directions show that the work in this dissertation and this research area are active and interesting.

APPENDIX A

Missing Value Imputation

The presence of a certain amount of missing data is inevitable in real world data sets, including our experimental data, and it is often regarded as a difficult problem to deal with. There are many previous works on missing data focusing on filling missing values (Twala 2009; Pearson 2006; Nakagawa and Freckleton 2008; Allison 2001; Fielding et al. 2008; Giardina et al. 2005; Winkler 2003). However, most methods require data to follow some assumptions. Thus, one would expect that when assumptions are not satisfied, the results on imputed data should be far different from the source data.

In this chapter, we develop a novel tree-based imputation algorithm called "Imputation Tree" (ITree) (Vateekul and Sarinnapakorn 2009). It first studies the predictability of missingness using all observations by constructing a binary classification tree called "Missing Pattern Tree" (MPT). Then, missing values in each cluster or terminal node are estimated by a regression tree of observations at that node. We present empirical results using both synthetic and real data. Almost all experiments show that ITree is superior to other commonly used methods in estimating missing values. The algorithm not only produces an impressive accuracy of its imputation comparing to the original complete data, but also provides the nature of missingness.

The rest of the chapter is organized as follows: In Section A.1, we briefly review the definitions of missingness types and discuss about some imputation techniques that are used in our study. Section A.2 explains our tree-based imputation method, ITree. Section A.3 gives the details of all experiments and Section A.4 shows the results. Conclusion and future work are in Section A.5.

A.1 Missing Data and Imputation

A.1.1 Types of Missing Values

There is a long history of analysis of missing data in classical statistics literature (Little and Rubin 1986; Rubin 1996) that uses the cause of missingness to classify missing data as one of the three following types.

Missing completely at random (MCAR) happens when missing values are randomly distributed across all observations. For the real example of MCAR (Page 115 (Bryman and Hardy 2009)), CESD is an index of depression and was measured at two time points in the Los Angeles Epidemiological Catchment Area study. There were 3047 individuals for whom CESD was measured at the first time point. In the second interview, 2240 of the 3047 cases had their measured; thus, 807 were missing for the second CESD measurement. For this example, the data would be MCAR if the missing CESDs at the second time point are completely independent of the depression level of individuals at both time points. We can verify whether missing data is MCAR by applying Little's MCAR test (Little 1988), based on a likelihood ratio testing method. The test is available in many statistical softwares.

Missing at random (MAR) is the condition that exists when missing values are randomly distributed within one or more subsamples instead of the whole data set like MCAR. Moreover, when the variable that explains the missingness is included, the result is now MCAR. For example, respondents with lower education may be less likely to complete the entire survey. So, the cause of missing data is due to some other external variables, not the actual missing variable itself.

Missing not at random (MNAR) is the type of missingness that arises when missing values are not randomly distributed across observations. The simplest form of MNAR is that the probability of missing data is systematically related to the values that are missing. For example, people who earn very high salary tend to hide their own information. In contrast to the MAR situation, the cause of missing data is due to the value of the actual missing variable itself. The probability of missing cannot be predicted directly from other variables in the database. This missingness is considered non-ignorable.

A.1.2 Related Works

Before outlining our proposed method, ITree, a short overview of five frequently used missing data techniques in statistics and engineering fields is warranted. Furthermore, these techniques can be categorized by "*the number of trials to estimate missing values*" into two groups: (*i*) single imputation and (*ii*) multiple imputation.

Among five reviewed techniques, two of them are considered as the single imputation. The first technique is the simple arithmetic mean. It replaces missing values with the mean of complete data. Second, a regression tree (REG) (Breiman et al.

1984) can also be used to impute missing values. A tree is built from non-missing data through a process known as binary recursive partitioning until each node reaches a user-specified minimum node size and becomes a terminal node. The mean at each terminal node is taken as imputed value for missing data.

The remaining three missing data techniques in this review are categorized as the multiple imputation. First, multiple imputation (MI) method (Allison 2000; Rubin 1987) has received a great attention during last decades. The concept of MI is to replace each missing value with a set of plausible values that represent the uncertainty about the value to impute. Second, Schneider (2001) proposed a regularized expectation-maximization (EM) algorithm[22] that integrates the conventional EM and iterated linear regression to estimate missing values for Gaussian data. The method is only applicable to data in which missing values are missing at random. Third, a Bayesian Principal Component Analysis (BPCA)[23] (Oba et al. 2003) uses a conventional EM algorithm together with a Bayesian model to approximate the principal axes using the concept of PCA. Experiments showed that it exhibited markedly better estimation ability than the singular value decomposition and k-nearest neighbors algorithm.

Although there are other existing missing data techniques such as “full information maximum likelihood” (FIML) (Allison 1987), only those methods above are selected in this paper for two reasons. First, one objective of this work is to estimate missing values; thus, we selected only the methods that fill all missing values in the data set. FIML is not chosen because it is a method to estimate the parameters and standard

[22]http://www.gps.caltech.edu/∼tapio/imputation/
[23]http://hawaii.sys.i.kyoto-u.ac.jp/∼oba/tools/BPCAFill.html

errors of the model directly from the available data without missing value estimation. Second, those chosen methods are publicly available for the sake of the reproducibility of the experiments. REG and MI are provided in the SAS packages. BPCA and EM are available to download at the author's website.

A.2 The Proposed Tree-Based Approach

A.2.1 Imputation Tree

Imputation Tree (ITree) is a tree-based algorithm for missing values imputation. Table A.1 shows the pseudo-code of ITree which is composed of two main processes. Given a set of data, the first process of ITree is to construct a Missing Pattern Tree (MPT), which is a binary classification tree that attempts to identify the missingness of each observation. This may at first look like many clustering techniques, such as K-means clustering, that are used to impute missing values. However, MPT has an advantage over clustering techniques since it uses information of every single observation whereas clustering techniques ignore any incomplete observations. Considering a particular attribute with missing values, MPT uses missing information of each observation to generate a binary class variable where "0" represents a record having missing value and "1" represents a record with no missing value. As seen from the pseudo-code in Table A.1, we generate an additional class attribute and use a classification tree algorithm to construct the initial MPT. The MPT captures the missing pattern by identifying a set of covariations that help categorize missingness.

Table A.1: The pseudocode for generating missing data.

```
MAIN($data, missing\_att$)
▷ 1) Construct missing pattern tree (MPT)
▷ 1.1 Generate initial MPT
for $record$ in $data$
    do if $(data(missing\_att)) ismissing$
        else $class = 1$;
$mpt \leftarrow$ CLASSIFICATION_TREE($data, class$);

▷ 1.2 Prune MPT at confidence 95%
PRUNING($mpt, missing\_att, 0.05$);

▷ 2) Construct a regression tree
for terminal node ($tnode$) in $mpt$
    do
        ▷ $tnode.data$ are observations belong to $tnode$
        $reg \leftarrow$ REGRESSION_TREE($tnode.data$);

        ▷ impute missing value
        EVAL($reg, tnode.data$);
```

While MPT gives us an insight into the nature of missing data, it tends to overfit the missing value model. For the imputation purpose, we will avoid overfitting of the MPT by reducing *redundant* leaf nodes, a pruning process. Each pair of terminal nodes from the same parent must from two different populations. To ensure that each terminal node in MPT represents a different population. We, therefore, propose a pruning procedure based on a two-sample t-test. The procedure consists of conducting an independent two-sample t-test on the observations of the missing variable at each pair of terminal nodes having the same parent. If the result of a t-test indicates the observations in the two terminal nodes are from the same population (i.e., having the same mean), then we remove the two terminal nodes and declare their parent node a terminal instead. Otherwise, we leave the two terminal nodes alone. We continue checking every pair of terminals until all unnecessary splits are discarded. In our experiments, the significance level of t-test for each pair of terminal nodes is set to be 0.05.

The second process of ITree is to estimate missing values using regression tree analysis, which is an approach suitable for modeling nonlinear relationship in data. At each terminal node in the MPT, we construct a regression tree from all complete observations at that node in order to predict the value of missing variable. The average of observations at the terminal node of the regression tree provides an estimate of missing value for any observations landing at that terminal.

A.2.2 Schema of Missing Data Treatment

A framework based on the proposed ITree for handling research data with missing values is presented in Figure A.1. The system analyzes the results from each process of ITree and provides two optional modules,“Missing Pattern Type Inference” and “Auxiliary Variables Suggestion” engines, as shown in the dashed-boxes.

In the first module, the inference engine determines the type of missingness (MCAR, MAR, or MNAR) by inferring from the process of constructing the MPT whether it can discover a missing pattern. It considers the missing type by checking if an MPT can be built or not. On the one hand, if the system can generate an MPT, the inference engine will confidently conclude that the missing type is MAR because a missing pattern can be identified, and each node in MPT is a cause of missing. On the other hand, the inference engine deduces that the missing type is either MCAR or MNAR when an MPT cannot be constructed, or it can be created but its perfor-

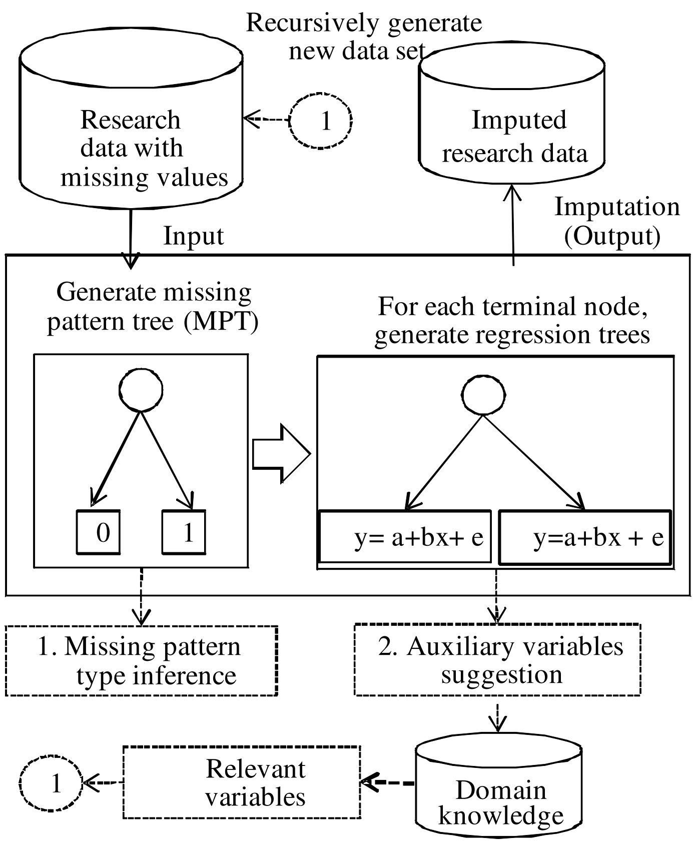

Recursively generate new data set
Research data with missing values
1
Imputed research data
Imputation (Output)
Input
Generate missing pattern tree (MPT)
For each terminal node, generate regression trees
0
1
y= a+bx+ e
y=a+bx + e
1. Missing pattern type inference
2. Auxiliary variables suggestion
1
Relevant variables
Domain knowledge

Figure A.1: The ITree based framework.

mance, e.g., *recall* and F_1, is unacceptably poor. We can distinguish MCAR from MNAR using Little's MCAR test.

When the system can generate regression trees with high prediction accuracy, it means missing values can be well estimated by other variables in the database. Conversely, variables in the database may not provide useful information for estimating missing values when the system cannot generate either MPT or regression trees. An estimate of missing values that we use in this circumstance could be a simple arithmetic mean. In our future work, we will be working on implementing the second additional module, "Auxiliary Variables Suggestion", where auxiliary variables refer to external variables which are in the domain knowledge database, but not originally included in the data set. This module will suggest related auxiliary variables in domain knowledge that are helpful for imputation. With additional auxiliary variables, missing values will be estimated more accurately.

It is important to note that there are two different types of trees in the proposed system: (*i*) the tree to find the *missing pattern* (MPT) and (*ii*) the tree to estimate *missing values.* Thus, when ITree cannot generate an MPT, it only implies the missingness cannot be explained by other variables in the data set. However, it is still possible to obtain a very good estimate of missing values from available data, especially when the missing variable and other variables are highly related.

A.3 Experiments

We conduct experiments on both real and simulated data to evaluate the performance of ITree and compare it to five selected imputation algorithms: MI, regression tree (REG), EM, BPCA, and arithmetic mean. They were chosen because they can estimate missing values in the database, not just the model parameter estimation, and their softwares are publicly available. For the generation process of both MPT and regression trees, we use CART methodology by (Breiman et al. 1984), which is available in Matlab. For MI, SAS provides it via the command "`PROC MI`" where its default setting without the option `MODEL` refers to the -missing value imputation[24].

[24]http://support.sas.com/rnd/app/da/new/dami.html

A.3.1 Simulated Data

We generate data sets from three scenarios: 1) normal distribution, 2) lognormal distribution, and 3) two normal distributions having different means. All data sets have 7 variables, X1 to X7, where X1 to X5 are linearly related while X6 and X7 are independent of each other as well as independent of X1 to X5. For each experimental setup, we repeat the experiment 5 times in order to learn the variation in the performance measure. To investigate the effect of sample size among different imputation techniques, we consider two sample sizes, small and large. Since this work is inspired by research in behavioral medicine where it is rare to have a very large number of patients or interviewees, we limit the sample size to be 1000 observations for large and 200 for small samples.

A.3.2 Pima Indians Diabetes Database (PIMA)

PIMA is a real data set from UCI Machine Learning Repository (Asuncion and Newman 2007) that has been used in many papers since 1990. It comprises observations from 768 female patients of Pima Indian heritage and at least 21 years old who may show signs of diabetes. Each patient is described by one class variable indicating whether she is diabetic and 8 attributes including medical measurements and personal history such as age, body mass index (BMI), number of pregnancies (NP), and triceps skin fold thickness (SF). We discard some observations because their attributes have inconsistent or incorrect values, e.g., some patients have BMI equal 0. After removal of inaccurate records, we end up with 392 complete observations for the experiments.

A.3.3 Generating Missing Data

Table A.2 shows the pseudo code, sim_missing_data function, used for generating missing data of different types in a given data set. We specify the condition of being missing (parameter qt) to be either value smaller than the first quartile or larger than the third quartile of the conditioning attribute (parameter cond_att). This implies we will have roughly no more than 25 percent of data missing. The parameter *prob* is the probability of missing for those observations meeting the quartile criterion. When the missing condition is satisfied under the specified probability, the value of missing attribute (parameter missing_att) is replaced by a pre-defined missing value (NaN).

Table A.2: The pseudocode of ITree

```
SIM_MISSING_DATA(data, type, missing_att, cond_att, prob, qt)
 1  prob_data ← ASSIGN_RAND(data); ▷ range [0,1]
 2  if (type == MNAR)
 3      then cond_att = missint_att;
 4  SORT(data, cond_att);
 5
 6  if ((type == MAR)or(type == MNAR))
 7      then for record in data
 8                 do if (prob_data < prob)
 9                        then data(missing_att) = NaN;
10      else  if (prob_data < prob)
11                then data(missing_att) = NaN;
```

If the type of missingness is MCAR, missing values do not depend on any attributes, and so conditioning parameters cond_att and qt need not be specified. We experiment with three different values for missing probability: 0.05, 0.15, and 0.25. To generate MAR missing data, we select a conditioning attribute and its conditioning value, the first or the third quartile depending on the characteristic of the data. The probability of being missing is varied from 0.6 to 1.0 at 0.1 increment. For MNAR missing data, none of the variables in the data set provide useful information in estimating missing values. We assume, for simplicity, that the missing variable and the conditioning variable are the same one, and vary the probability of missing from 0.6 to 1.0 at 0.1 increment.

We investigate all three types of missing values for every data set. In total there are four sets of experiments with different characteristics as depicted in Table A.3. For normal and lognormal data, we consider two cases: when missing variable (X3) is related to other variables in the data set and when missing variable (X6) is independent of other variables. How well an imputation method performs is measured by the deviation of an imputed value from the actual one (in the complete data set). The comparison among imputation methods will be based on the root mean square error (RMSE) of the imputation averaged over 5-fold cross validation.

Table A.3: Summary of four experiments

Experiments	**Data**	**Miss**	**Types of missing values**
1	2POP	X6	1) mcar 2) marX7 3) mnar
2	NORM	X3	1) mcar 2) marX1 3) mnar
		X6	1) mcar 2) marX7 3) mnar
3	LOGN	X3	1) mcar 2) marX1 3) mnar
		X6	1) mcar 2) marX7 3) mnar
4	PIMA	BMI	1) mcar 2) marSF 3) mnar
		NP	1) mcar 2) marAge 3) mnar

A.4 Results and Discussions

The experimental results are analyzed statistically using ANOVA and followed by Bonferroni multiple comparisons in order to rank the performance of different imputation techniques. Many experiments lead to the same conclusion, so only some selected results are presented.

The first set of experiments deals with data from 2 populations. The results shown in Figure A.2(a), (b), and (c) indicate that ITree performs the best among all methods. It is not surprising that most other methods, MI, EM, BPCA, and mean, give undesirable results since they estimate missing values using information of two different distributions treated as one, thus producing erroneous imputed values. On the contrary, tree-based methods like ITree and REG estimate missing values separately for each population at the terminal nodes.

ITree and REG perform comparably most of the time. The exception is the MNAR case where ITree is much better, particularly when the probability of missing is 1.0 (Figure A.2(c)). In such a case, REG cannot generate any regression trees because one population has too few complete observations. Hence, REG simply produces an arithmetic mean of 2-population data as an estimate of missing values. In contrast, ITree clusters data nicely into 2 populations in the MPT and it does not attempt to estimate the missing value when there is obviously insufficient data.

In the second set of experiments we worked with normal distributed data (NORM). When the missing variable X3 is dependent on other variables, it is possible to use information from other variables to help in imputation. For all types of missing, MI is far better than other methods and arithmetic mean is the worst one. Figure A.3(a) and (b) show examples of results for MAR and MNAR cases. While ITree is not the best algorithm, it is usually one of the top three performances. When the missing variable X6 is independent of other variables, the results are different, however. BPCA is the worst performer here. It is interesting to see in Figure A.3(c) that MI cannot outperform single imputation algorithms including arithmetic mean because the estimation of X6 by MI involves other variables, X1-X5, while X6 is independent variable, so its value cannot be estimated by others. Also, the results show that ITree wins MI in every case, though not significantly.

The third set of experiments is conducted on a lognormal distribution (LOGN) giving some result examples in Figure A.4(a), (b), and (c). ITree and BPCA are not significantly different for almost every case in the study, and both tend to be more accurate in estimating missing values compared to other methods. It should be noted that a sophisticated method like MI is inferior to others when data is not normal distributed, which implies that MI does not tolerate a departure from its distributional assumption. Although there are some improvement of MI for the non-normality assumption, they are not commonly used and provided by SAS yet; we, therefore, decided not to include them in the experiment.

The last set of experiments is carried out on real data, PIMA. One of the experiments uses "number of pregnancies" (NP) as a missing variable. NP has a weak correlation with other variables. We see from Figure A.5(a) that in the MAR case when the conditioning variable is age, ITree almost always gives the most accurate imputed value for NP. It is only when the probability of missing is 1 (which means every observation that meets the conditioning variable criterion has NP missing) that ITree loses to MI and EM algorithms. In another experiment, "body mass index" (BMI) is the missing variable. BMI is linearly related to "skin fold" (SF). Figure A.5(b) shows that MI and EM are the two algorithms that stand out for having low RMSE. ITree comes close to these two; it is not significantly different from them in many cases.

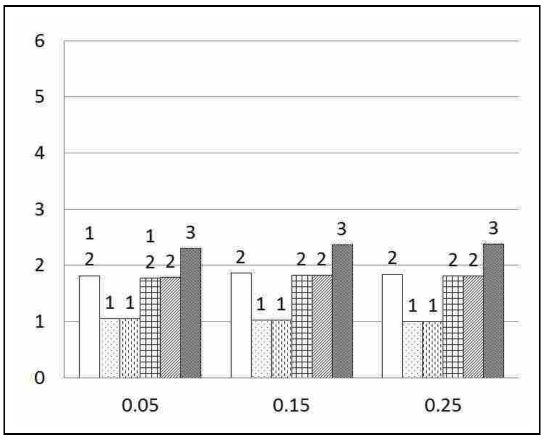
In all, we see that there is no single method that is always the best for imputation. ITree performs reasonably well for all the scenarios in our study, and often it is one of our top choices.

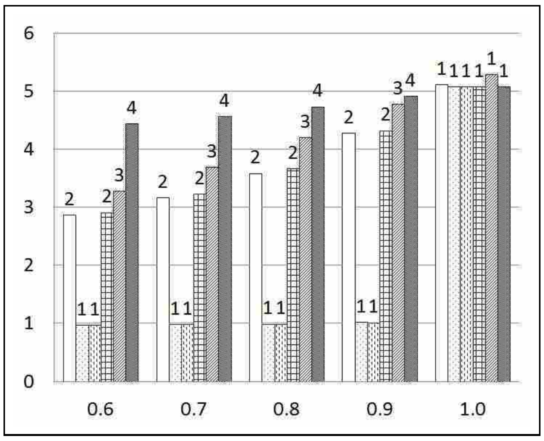
A.5 Conclusion

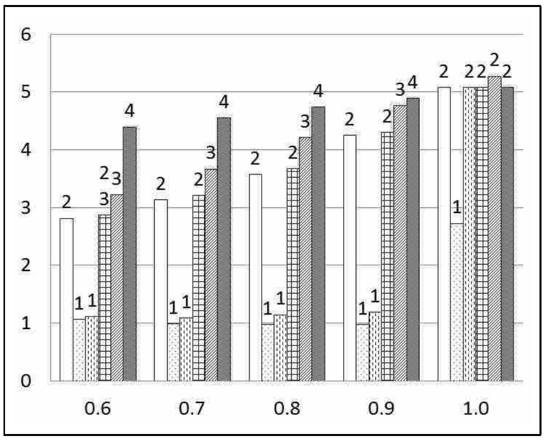
We have proposed a new imputation method, Imputation Tree (ITree), to handle data with missing values. ITree is a tree-based algorithm that initially generates a missing pattern tree (MPT) to help us understand the nature of missingness, and then further provides regression trees that produce estimates for missing values. The advantage of ITree over other imputation methods is that it tells us if the missingness of a variable is influenced by the existence of other variables in the data set. It gives us the knowledge of missingness type by analyzing the data, rather than assuming like other methods do. Thus, the ways to handle missing data can be appropriately provided to a specific type of missing data.

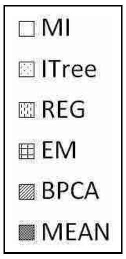
In general, ITree performs satisfactorily in terms of having small imputation error comparing to the original complete data. It usually is one among the most precise imputation algorithms in our experiments, and it works best with 2 populations and lognormal data. The nonparametric nature of ITree makes it flexible and robust to be used without distributional assumption issue. However when the assumptions are met such as in the case of MI with normal data, ITree often cannot compete with MI.

In this work we only discuss the imputation of one missing variable. ITree deals with one missing variable each time, because the causes of missingness of different variables are not the same. Therefore when there are multiple missing variables, each variable will be treated separately. ITree will be applied to each individual missing variable, one after another. The order of missing variables to be imputed could be important as mentioned in (Conversano and Siciliano 2003), and this issue will be covered in our future work. We will also extend the method to handle the imputation of missing categorical data.


(a) 2POP_missingX6_mcar

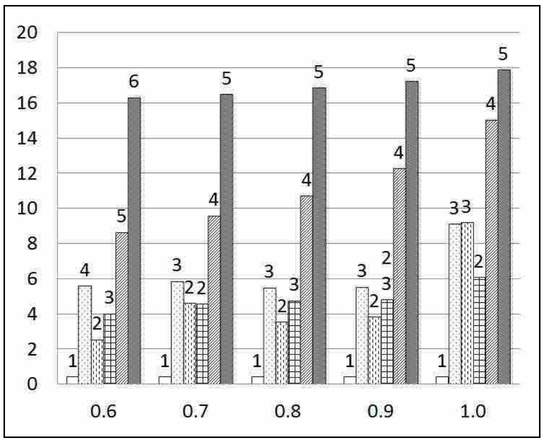

(b) 2POP_missingX6_marX7


(c) 2POP_missingX6_mnar

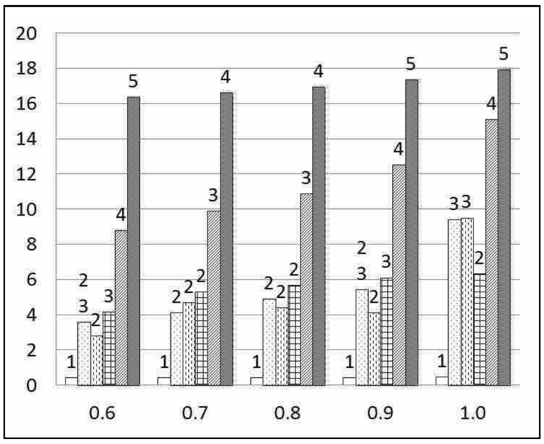


(d) Label

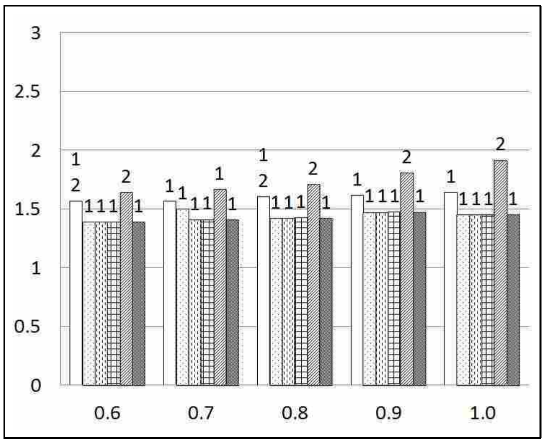
Figure A.2: Average RMSE of six imputation methods at different levels of missing probability on *two population data (2POP)*. X-axis is the probability of missing and Y-axis is the average RMSE. The number above each bar provides ranking of the performance.



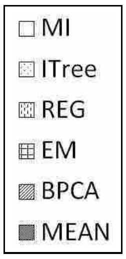
(a) NORM_missingX3_marX1



(b) NORM_missingX3_mnar

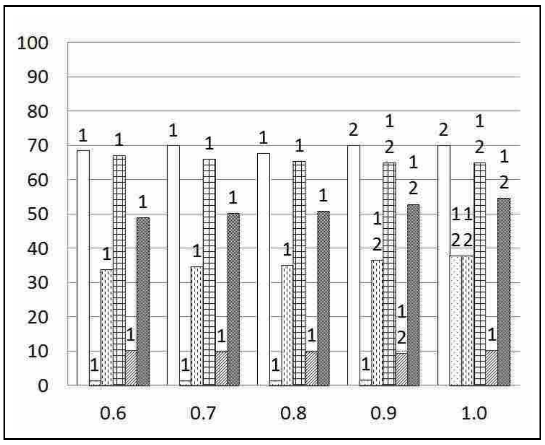


(c) NORM_missingX6_marX7

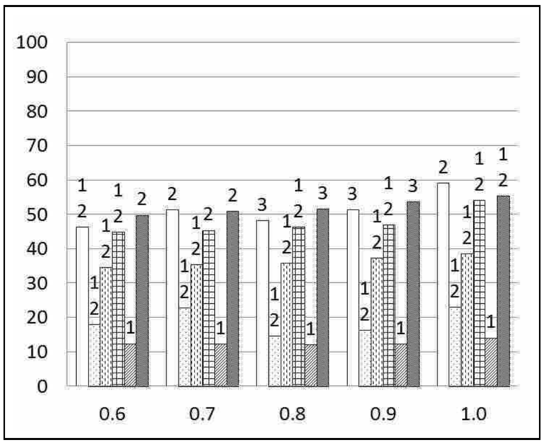


(d) Label

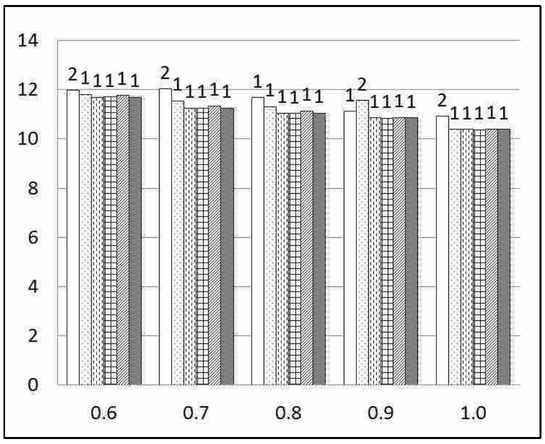
Figure A.3: Average RMSE of six imputation methods at different levels of missing probability on *normal distribution data (NORM).* X-axis is the probability of missing and Y-axis is the average RMSE. The number above each bar provides ranking of the performance.



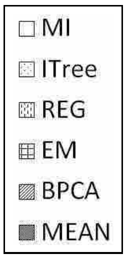
(a) LOGN_missingX3_marX1



(b) LOGN_missingX3_mnar

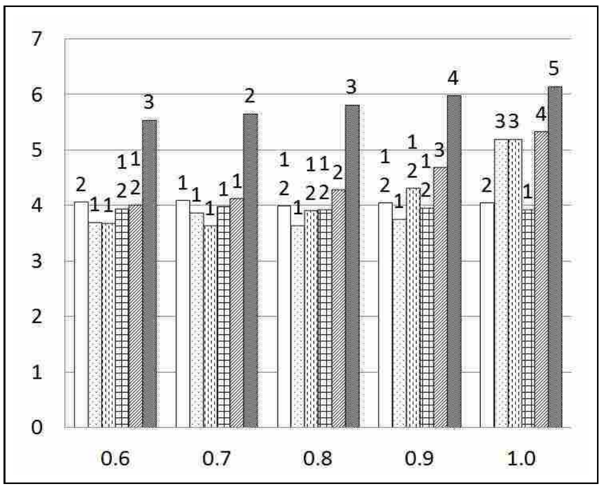


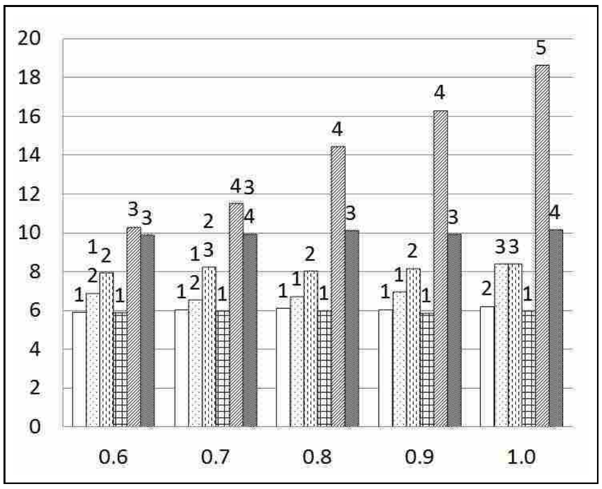
(c) LOGN_missingX6_marX7

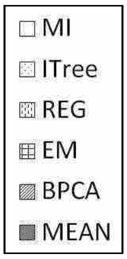


(d) Label

Figure A.4: Average RMSE of six imputation methods at different levels of missing probability on *lognormal distribution data (LOGN).* X-axis is the probability of missing and Y-axis is the average RMSE. The number above each bar provides ranking of the performance.


(a) PIMA_missingNP_marAge


(b) PIMA_missingBMI_marSF



(c) Label

Figure A.5: Average RMSE of six imputation methods at different levels of missing probability on *Pima Indians Diabetes data (PIMA)*. X-axis is the probability of missing and Y-axis is the average RMSE. The number above each bar provides ranking of the performance.

APPENDIX B

More Experimental Results of *HR*-SVM

Previously in Section 5.4, all experiments were evaluated by the proposed criteria, "Example-Label based macro-averaging" (ELb) – with its own version of *precision* ($ELbPr$), *recall* ($ELbRe$), and F_1 ($ELbF_1$). Some may argue that the success of the developed hierarchical classification framework, *HR*-SVM, can be mistaken by the evaluation of the proposed criteria. Thus, this appendix aims to show the experimental results in Section 5.4 evaluated by other existing measures in this domain: (i) "Example based criteria" (Eb) – or the hierarchical criteria in Section 3.4, and (ii) "Label based criteria" (Lb) – or the multi-label classification criteria in Section 2.5.2.

As expected, the evaluations of those existing criteria are similar to the ELb-results in Section 5.4.

- Like in Section 5.4.1, the F_1 of the developed system, *HR*-SVM-ALL, compares favorably with that of other techniques, *H*-SVM and Clus-HMC, as shown in Figure B.1. It surpassed *H*-SVM on all domains, except $D15$ and $D17$ that the Lb-results of both systems are comparable. Comparing to Clus-HMC, it won on most cases out of 8 data sets. It lost only on one data set ($D17$) in terms of the Eb-measure and two data sets ($D17$ and $D18$) in terms of the Lb-measure.

- In the next step, similar to Section 5.4.2, each proposed concept has been investigated on how much it contributes to the system performance. First, the *R*-SVM module positively affected the system performance in terms of F_1 and *recall* as illustrated in Figure B.2 and Figure B.3 respectively. Then, by applying the FPC concept, F_1 and *precision* were significantly improved as demonstrated in Figure B.4 and Figure B.5 respectively. Finally the LFS concept was successful in increasing F_1 though *precision* on most data sets as shown in Figure B.6 and Figure B.7 consecutively.

- In the end, Section 5.4.3 demonstrated performance at different hierarchical levels of *HR*-SVM-ALL, *H*-SVM, and Clus-HMC. Since each class-level was independently evaluated, the label-based metrics (*Lb*) were chosen instead of the *ELb*-metrics in Section 5.4.3. Thus, there is no need to show the results of different criteria here. Previously, only the class-level results of $D0$ and $D16$ were reported; for the sake of completeness, the results of the remaining data sets are shown here in Figure B.8 – Figure B.13.

Moreover, the comparison of those existing criteria (the *Eb*-measure and the *Lb*-measure) are also summarized as follows:

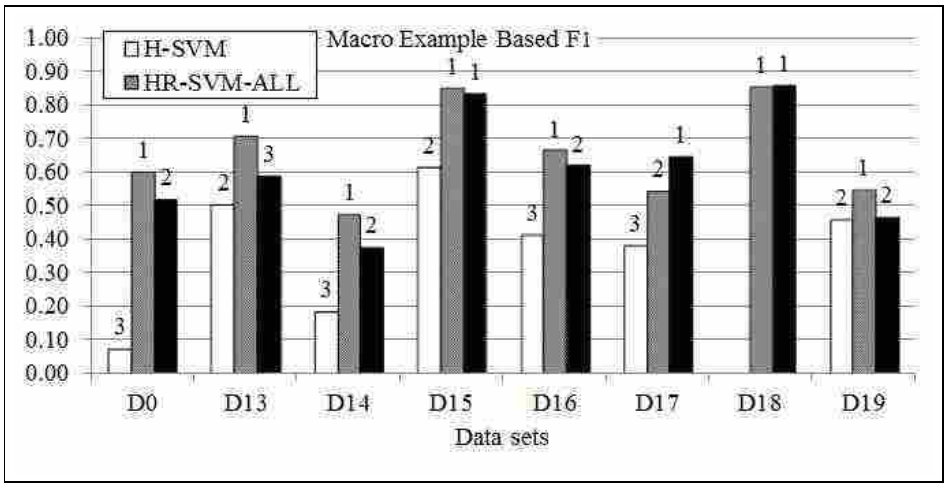
- Although those existing criteria mostly give the same conclusion, the label based results are always less than the example based results. The *Eb*-criteria refers to an average of the classification performance *per each example* along the predicted paths in the class hierarchy, while the *Lb*-criteria represent an average of the performance *per each label (class)* along the predicted examples in the data set. Table B.1 is an example of the prediction in the matrix form where the *Eb*-criteria and the *Lb*-criteria compute across the row and column of the matrix respectively. Since the number of classes is usually smaller than the number of examples, the *Eb*-calculation along "*class paths*" (across the row) is easier to be satisfied and obtains higher values than the *Lb*-calculation along "*all examples*" (across the column).

Table B.1: An example of the prediction in the matrix form. Assume that there are 8 classes in the hierarchy and 10000 examples in the data set.

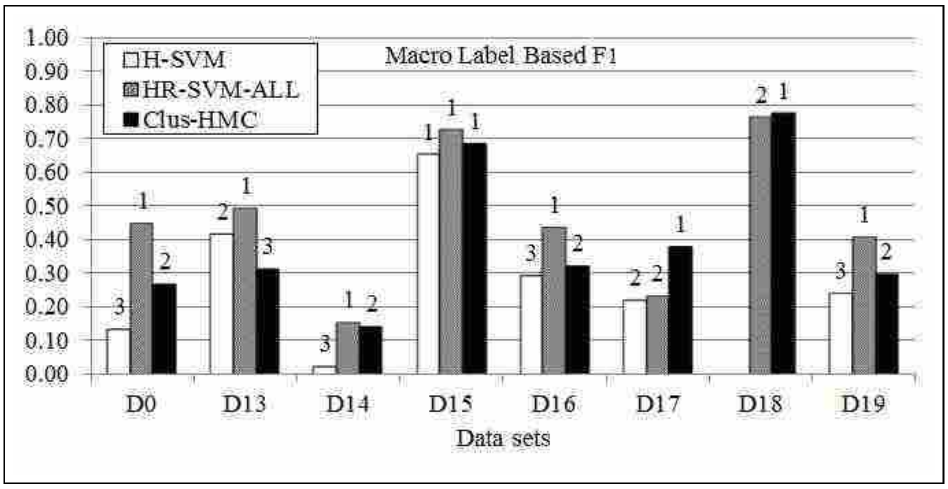
	C1	C2	C3	C4	C5	C6	C7	C8
x_1	1	1	0	0	0	0	0	0
x_2	0	0	0	1	0	1	1	0
...								
x_{10000}	1	1	0	0	0	0	0	0

- The results of those existing criteria can sometimes be opposite as shown in Figure B.4; this, therefore, shows the need of the introduced *ELb*-measure. On $D14$, EbF_1 of *HR*-SVM-FPC was higher than that of *HR*-SVM, while the result was opposite in terms of the LbF_1. In the domain of hierarchical multi-label classification, the classifier showing promising results on both measures is preferred. Thus, the presented evaluation measure, which is a harmonic mean of those measures, can give a compromised judgement that F_1 of *HR*-SVM-FPC was greater than that of *HR*-SVM on $D14$ as shown in Figure 5.12.

In conclusion, the results evaluated by the existing criteria in this appendix agreed to those measured by the proposed criteria in Section 5.4. This can prove the success of the proposed system, *HR*-SVM. Moreover, the need of the presented performance criteria was shown when the contradiction of those existing evaluation appeared.

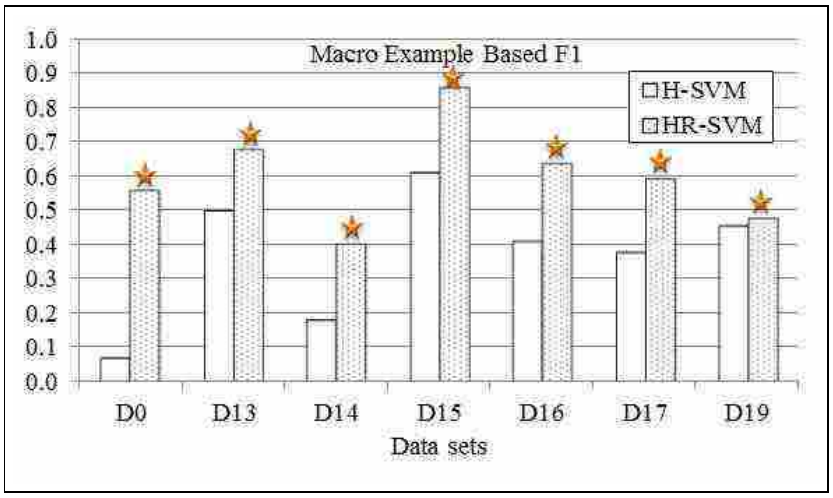


(a) Example based F_1 (EbF_1)

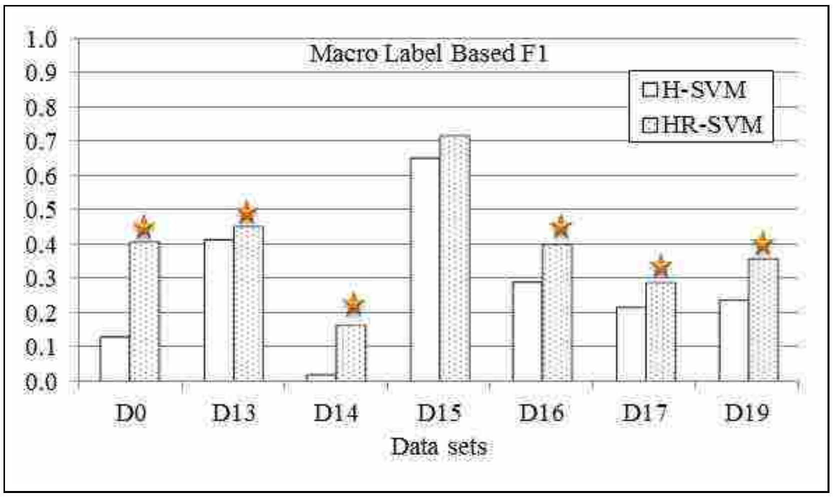


(b) Label based F_1 (LbF_1)

Figure B.1: Comparing the performance (along F_1) of *H*-SVM, *HR*-SVM-ALL, and Clus-HMC. The integers over each of the vertical bars give the ranks as obtained by the ANOVA methodology followed by Bonferroni multiple comparisons.

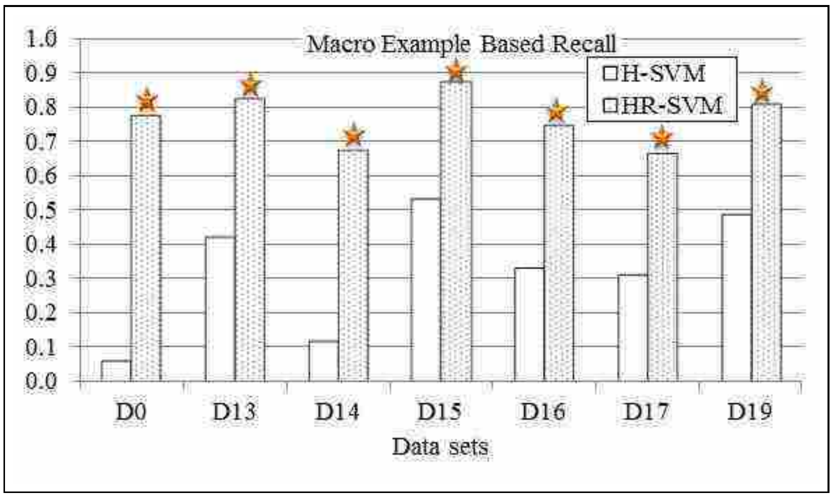


(a) Example based F_1 (EbF_1)

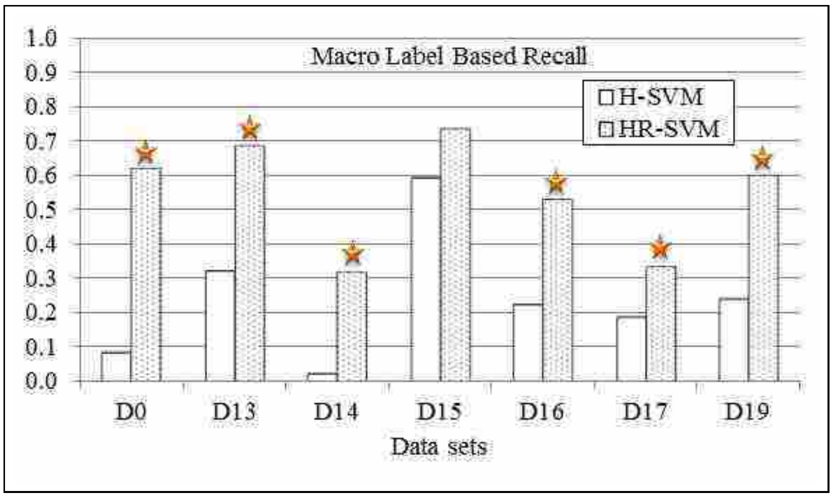


(b) Label based F_1 (LbF_1)

Figure B.2: The effect of *R*-SVM: Comparing *HR*-SVM with *H*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

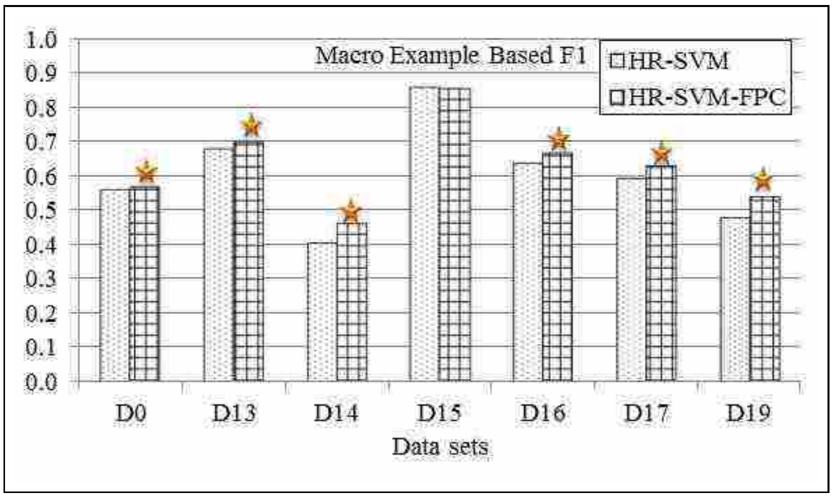


(a) Example based *recall* (*EbRe*)

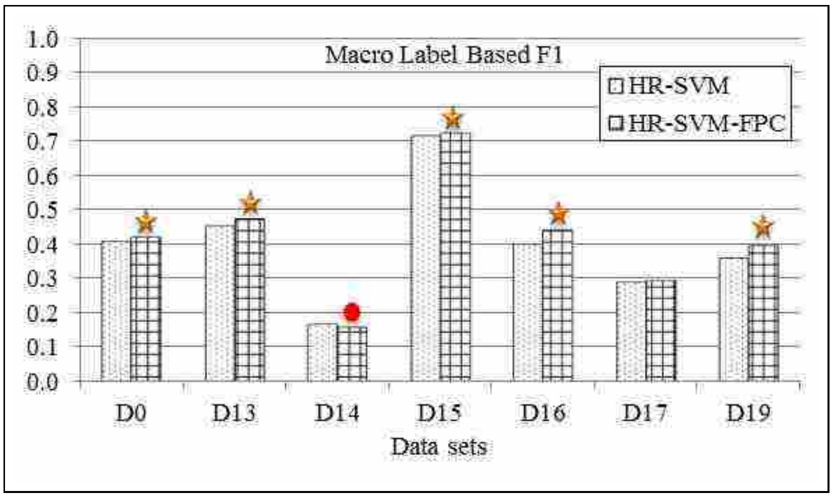


(b) Label based *recall* (*LbRe*)

Figure B.3: The effect of *R*-SVM: Comparing *HR*-SVM with *H*-SVM in terms of *recall*. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

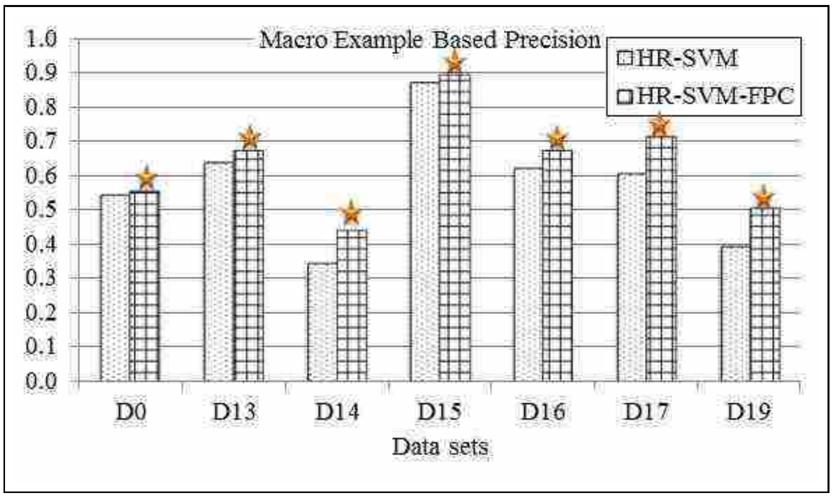


(a) Example based F_1 (EbF_1)

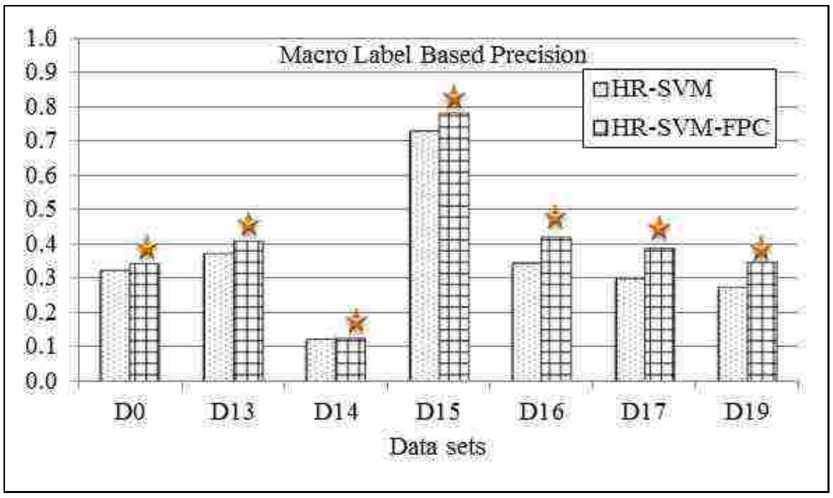


(b) Label based F_1 (LbF_1)

Figure B.4: The effect of FPC: Comparing *HR*-SVM-FPC with *HR*-SVM in terms of F_1. The stars above some of the bars indicate significant improvements according to t-tests (0.05 level).

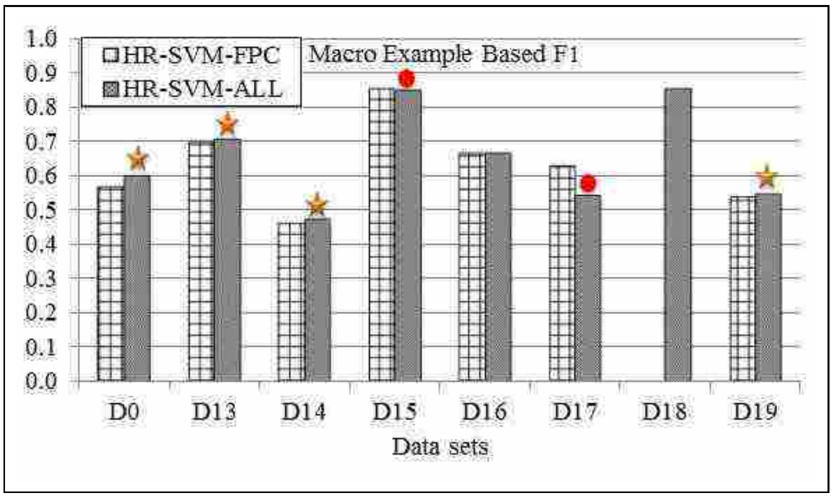


(a) Example based *precision* (*EbPr*)

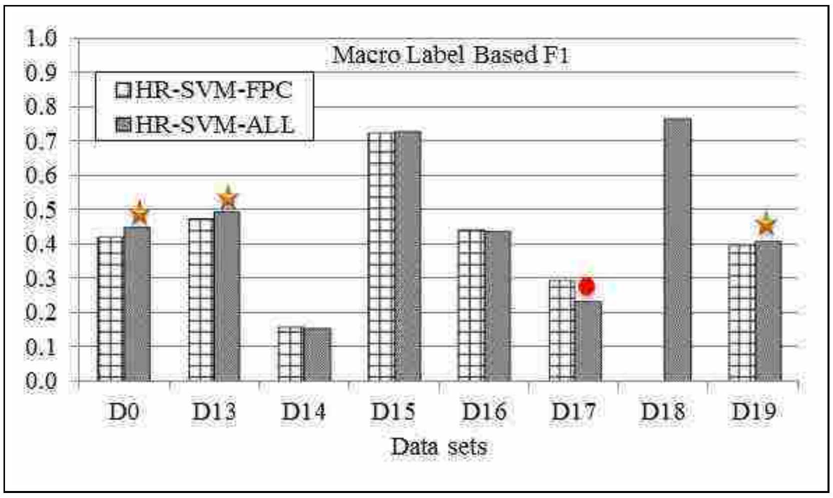


(b) Label based *precision* (*LbPr*)

Figure B.5: The effect of FPC: Comparing *HR*-SVM-FPC with *HR*-SVM in terms of *precision*. The stars above some of the bars indicate significant improvements according to *t*-tests (0.05 level).

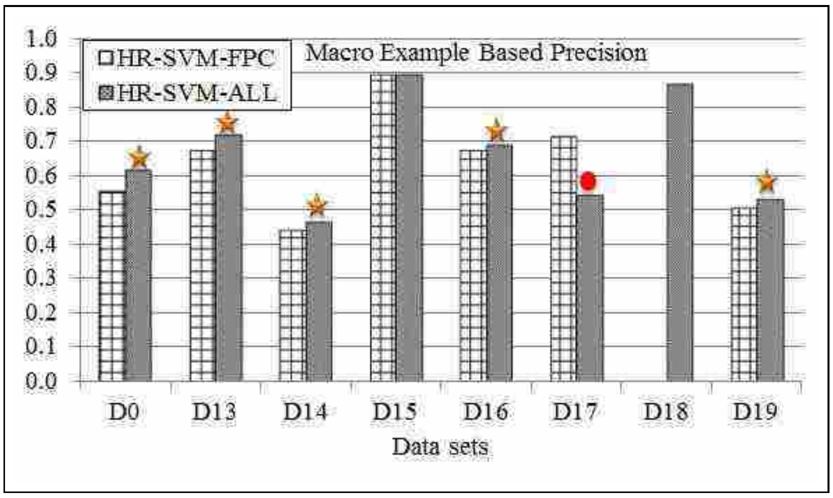


(a) Example based F_1 (EbF_1)

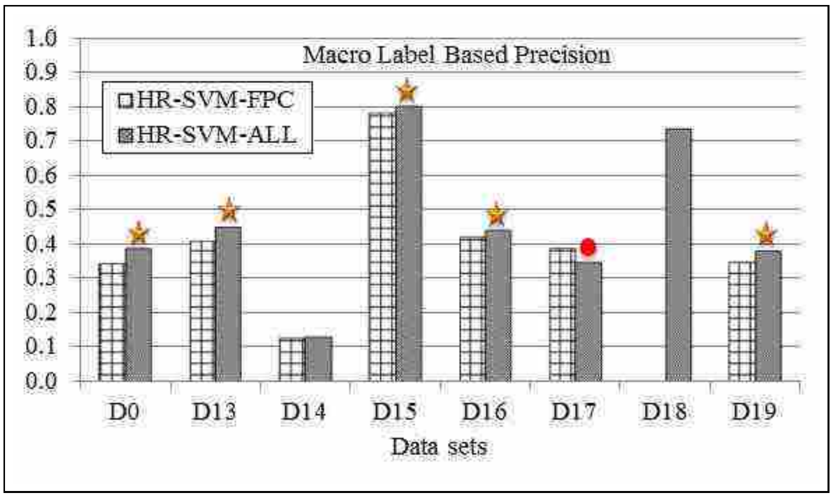


(b) Label based F_1 (EbF_1)

Figure B.6: The effect of LFS: Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of F_1. The stars and circles above some of the bars indicate significant improvements and declines according to *t*-tests (0.05 level).



(a) Example based *precision* (*EbPr*)



(b) Label based *precision* (*EbPr*)

Figure B.7: The effect of LFS: Comparing *HR*-SVM-ALL with *HR*-SVM-FPC in terms of *precision*. The stars and circles above some of the bars indicate significant improvements and decline according to *t*-tests (0.05 level).

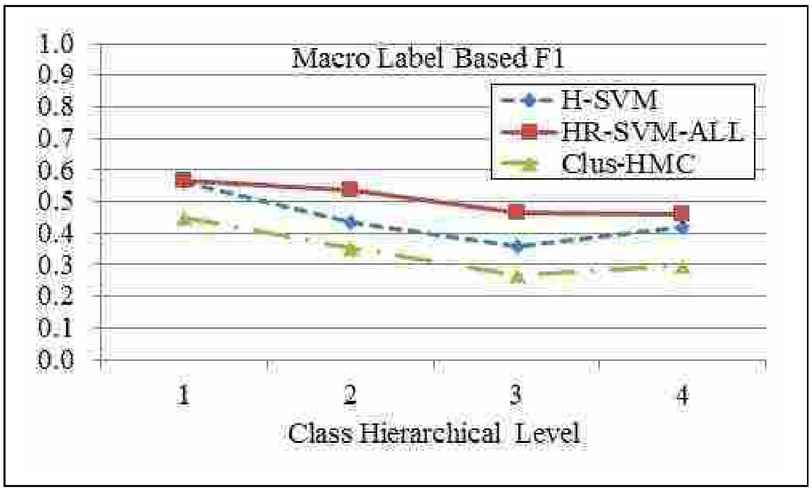


Figure B.8: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in *D*13.

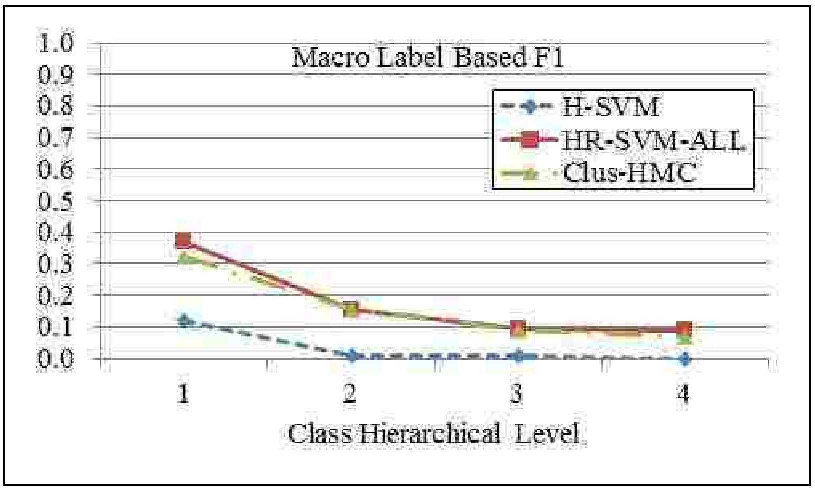


Figure B.9: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in *D*14.

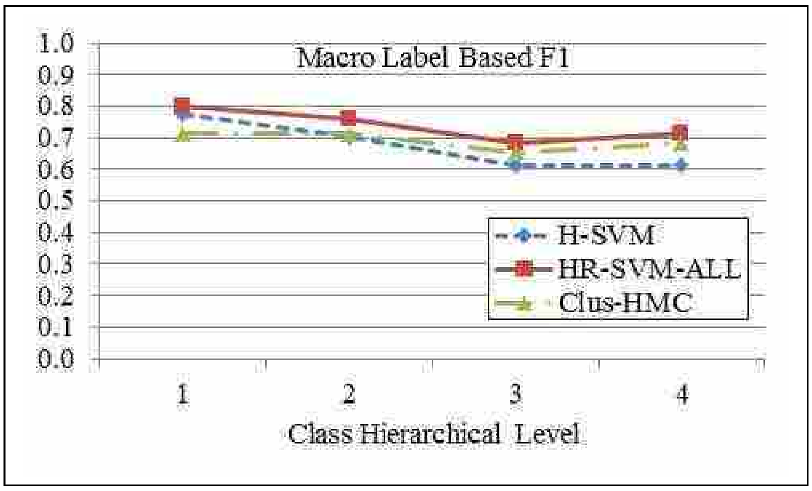


Figure B.10: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in $D15$.

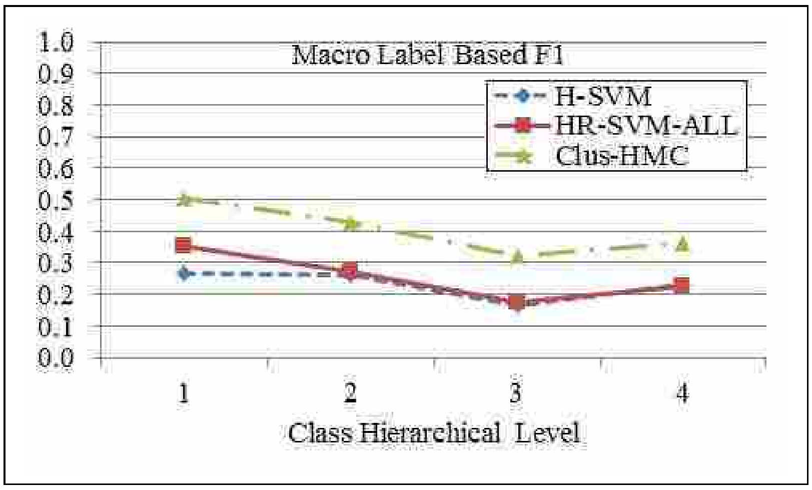


Figure B.11: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in $D17$.

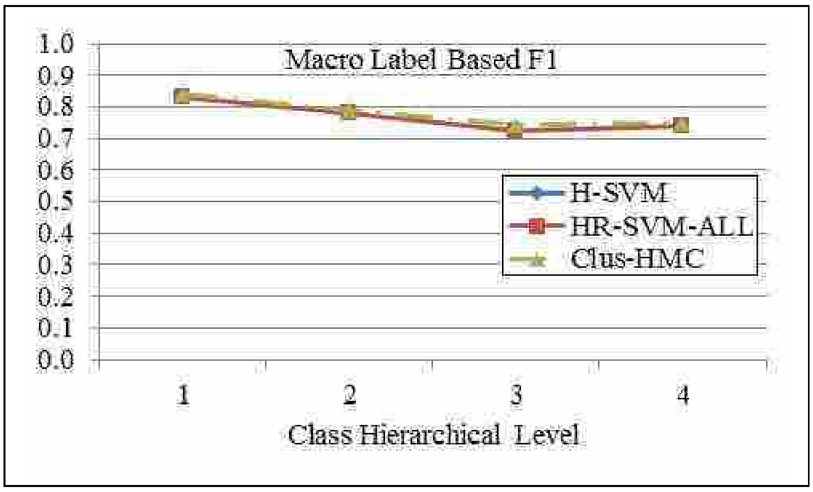


Figure B.12: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in *D*18.

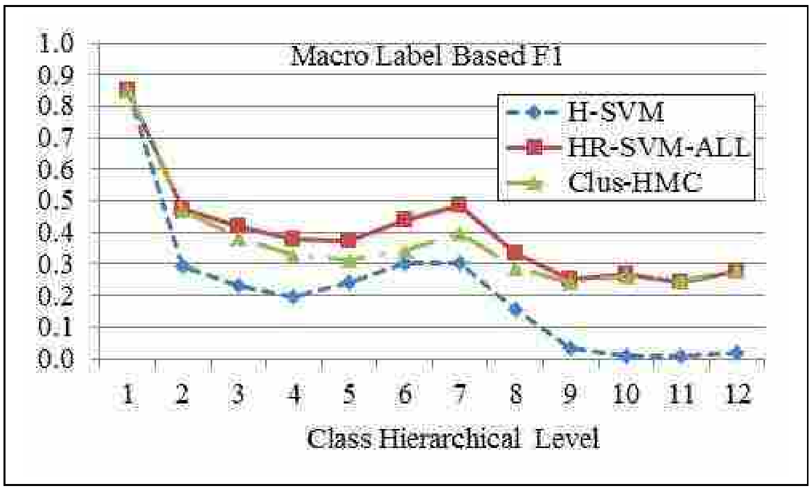


Figure B.13: Comparing *HR*-SVM-ALL, *H*-SVM, and Clus-HMC at different hierarchical levels in *D*19.

Bibliography

Agrawal, R., Imieliski, T., and Swami, A. 1993. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. Vol. 22. ACM, 207–216.

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*. 487–499.

Allison, P. D. 1987. Full information estimation in the presence of incomplete data. *Sociological Methodology 17*, 71–103.

Allison, P. D. 2000. Multiple imputation for missing data: A cautionary tale. *Sociological Methods Research 28,* 3, 301–309.

Allison, P. D. 2001. *Missing Data*. Sage Publications, Thousand Oaks, CA.

Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, M., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., and Sherlock, G. 2000. Gene ontology: tool for the unification of biology. *Nature Genetics 25,* 1, 25–29.

Asuncion, A. and Newman, D. 2007. Uci machine learning repository.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I., and Bourne, P. 2000. The protein data bank. *Nucleic Acids Research 28,* 1, 235–242.

Blockeel, H., De Raedt, L., and Ramon, J. 1998. Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, J. Shavlik, Ed. Morgan Kaufmann, 55–63.

Blockeel, H., Schietgat, L., Struyf, J., Dzeroski, S., and Clare, A. 2006. Decision trees for hierarchical multilabel classification: A case study in functional genomics. *Knowledge Discovery in Databases: PKDD 2006, Proceedings 4213*, 18–29.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. 2004a. Learning multi-label scene classification. *Pattern Recognition 37,* 9, 1757–1771.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. 2004b. Learning multi-label scene classification. *Pattern Recognition 37,* 9, 1757 – 1771.

Brank, J., Grobelnik, M., Milic-Frayling, N., and Mladenic, D. 2003. Training text classifiers with svm on very few positive examples. Tech. rep., Microsoft Research.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. 1984. *Classification and Regression Trees.* Chapman & Hall, New York, NY.

Bryman, A. and Hardy, M. A. 2009. *Handbook of Data Analysis.* SAGE Publications Ltd, London/GB.

Chandra, B., Mazumdar, S., Arena, V., and Parimi, N. 2002. Elegant decision tree algorithm for classification in data mining. In *WISEW '02: Proceedings of the Third International Conference on Web Information Systems Engineering (Workshops) - (WISEw'02).* IEEE Computer Society, Washington, DC, USA, 160.

Chandra, B. and Varghese, P. P. 2008. Fuzzy sliq decision tree algorithm. *Systems, Man, and Cybernetics, Part B, IEEE Transactions on 38,* 5, 1294–1301.

Chang, C.-C. and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chernick, M. R. 2008. *Bootstrap methods : a guide for practitioners and researchers*, 2nd ed. Wiley-Interscience, Hoboken, N.J.

Clare, A., Karwath, A., Ougham, H., and King, R. D. 2006. Functional bioinformatics for arabidopsis thaliana. *Bioinformatics* 22 (9), 1130–1136.

Clare, A. and King, R. D. 2001. Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'01).* Freiburg, Germany.

Clare, A. and King, R. D. 2003. Predicting gene function in saccharomyces cerevisiae. *Bioinformatics 19*, 42–49.

Colas, F. and Brazdil, P. 2006. Comparison of svm and some older classification algorithms in text classification tasks. *Artificial Intelligence in Theory and Practice*, 169–178.

Conversano, C. and Siciliano, R. 2003. Claudio conversano and roberta siciliano. In *Interface 2003: Security and Infrastructure Protection.*

Costa, E. P., Lorena, A. C., Carvalho, and Freitas, A. A. 2007. A review of performance evaluation measures for hierarchical classifiers. In *2007 AAAI Workshop, Vancouver*. AAAI Press.

Delwiche, L. D. and Slaughter, S. J. 2008. *The little SAS book : a primer*, Fourth ed. SAS Press.

Dendamrongvit, S., Vateekul, P., and Kubat, M. 2011. Irrelevant attributes and imbalanced classes in multi-label text-categorization domains. *Intelligent Data Analysis*.

Diplaris, S., Tsoumakas, G., Mitkas, P. A., and Vlahavas, I. P. 2005. Protein classification with multiple algorithms. In *Panhellenic Conference on Informatics*. 448–456.

Dong, G., Zhang, X., Wong, L., and Li, J. 1999. Caep: Classification by aggregating emerging patterns. In *Proceedings of the Second International Conference on Discovery Science (DS'99)*. 737–737.

Dumais, S. and Chen, H. 2000. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 256–263.

Efron, B. 1979. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics 7,* 1, 1–26.

Efron, B. 1983. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association 78,* 382, 316–331.

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 *Proceedings of IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*.

Elisseeff, A. and Weston, J. 2001. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14*. MIT Press, 681–687.

Fagni, T. and Sebastiani, F. 2007. On the selection of negative examples for hierarchical text categorization. In *Proceedings of the 3rd language technology conference*. 24–28.

Fagni, T. and Sebastiani, F. 2010. Selecting negative examples for hierarchical text classification: An experimental comparison. *Journal of the American Society for Information Science and Technology 61,* 11, 2256–2265.

Feldman, R. and Sanger, J. 2007. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, New York, NY, 410.

Fielding, S., Fayers, P. M., McDonald, A., McPherson, G., and Campbell, M. K. 2008. *Simple imputation methods were inadequate for missing not at random (MNAR) quality of life data.* Vol. 6.

Gao, S., Wu, W., Lee, C.-H., and Chua, T.-S. 2004. A MFoM learning approach to robust multiclass multi-label text categorization. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*. 329–336.

Giardina, M., Huo, Y., Azuaje, F., McCullagh, P., and Harper, R. 2005. A missing data estimation analysis in type ii diabetes databases. In *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*. IEEE Computer Society, 347–352.

Goertzel, B. and Venuto, J. 2006. Accurate svm text classification for highly skewed data using threshold tuning and query-expansion-based feature selection. In *IJCNN'06: International Joint Conference on Neural Networks 2006*. 1220–1225.

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

Harris, C. J. 2004. The gene ontology (go) database and informatics resource – gene ontology consortium 32 (supplement 1): 258 – nucleic acids research. *Nucleic Acids Res. 1,* 32, D258–D261.

Hersh, W., Buckley, C., Leone, T. J., and Hickam, D. 1994. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., 192–201.

Imam, T., Ting, K., and Kamruzzaman, J. 2006. z-SVM: An svm for improved classification of imbalanced data. *AI 2006: Advances in Artificial Intelligence*, 264–273.

Jensen, L. J., Gupta, R., Blom, N., Devos, D., Tamames, J., Kesmir, C., Nielsen, H., Staerfeldt, H. H., Rapacki, K., Workman, C., Andersen, C. A., Knudsen, S., Krogh, A., Valencia, A., and Brunak, S. 2002. Prediction of human protein function from post-translational modifications and localization features. *Journal of Molecular Biology (JMB) 319,* 5, 1257–1265.

Joachims, T. 1998. Text categorization with support vector machines: learning with many relevant features. In *The European Conference on Machine Learning (ECML'98)*. Springer Verlag, Heidelberg, DE, Chemnitz, DE, 137–142.

JOACHIMS, T. 1999. Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning.* MIT Press, Cambridge, MA, USA, 169–184.

JOACHIMS, T. 2003. *Learning to Classify Text Using Support Vector Machines.* Kluwer Academic Publishers, Norwell, MA.

JOACHIMS, T. 2006. Training linear svms in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD).* ACM.

KANG, F., JIN, R., AND SUKTHANKAR, R. 2006. Correlated label propagation with application to multi-label learning. In *IEEE Computer Vision and Pattern Recognition (CVPR) 2006.* 1719–1726.

KELL, D. B. AND KING, R. D. 2000. On the optimization of classes for the assignment of unidentified reading frames in functional genomics programmes: the need for machine learning. *Trends Biotechnol 18,* 3, 93–98.

KIRITCHENKO, S., MATWIN, S., AND FAMILI, A. F. 2005. Functional annotation of genes using hierarchical text categorization. In *in Proceedings of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05).*

KOLLER, D. AND SAHAMI, M. 1997. Hierarchically classifying documents using very few words. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, D. Fisher, Ed. Morgan Kaufmann Publishers, San Francisco, US, 170–178.

KUBAT, M. AND MATWIN, S. 1997. Addressing the curse of imbalanced training sets: One-sided selection. In *Proceedings of the Fourteenth International Conference on Machine Learning.* Morgan Kaufmann, 179–186.

KWOK, J. T. 1998. Automated text categorization using support vector machine. In *Proceedings of the Fifth International Conference on Neural Information Processing (ICONIP'98).* Kitakyushu, JP, 347–351.

LAUSER, B. AND HOTHO, A. 2003. Automatic multi-label subject indexing in a multilingual environment. In *ECDL.* 140–151.

LEWIS, D. D., YANG, Y., ROSE, T. G., AND LI, F. 2004a. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research 5*, 361–397.

LEWIS, D. D., YANG, Y., ROSE, T. G., AND LI, F. 2004b. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research 5*, 361–397.

LI, B., MA, L., HU, J., AND HIRASAWA, K. 2008. Gene classification using an improved svm classifier with soft decision boundary. In SICE Annual Conference, 2008. *SICE Annual Conference, 2008*, 2476–2480.

LI, W., HAN, J., AND PEI, J. 2001. Cmar: accurate and efficient classification based on multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. 369–376.

LITTLE, R. 1988. A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association 83,* 404, 1198–1202.

LITTLE, R. J. A. AND RUBIN, D. B. 1986. *Statistical analysis with missing data*. John Wiley & Sons, Inc.

LIU, B., HSU, W., AND MA, Y. 1998. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*. 80–86.

MCCALLUM, A., ROSENFELD, R., MITCHELL, T. M., AND NG, A. Y. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 359–367.

MEHTA, M., AGRAWAL, R., AND RISSANEN, J. 1996. Sliq: A fast scalable classifier for data mining. In *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*. Springer-Verlag, London, UK, 18–32.

MEWES, H. W., ALBERMANN, K., HEUMANN, K., LIEBL, S., AND PFEIFFER, F. 1997. Mips: a database for protein sequences, homology data and yeast genome information. *Nucleic Acids Res 25,* 1, 28–30.

MEWES, H. W., FRISHMAN, D., GILDENER, U., MANNHAUPT, G., MAYER, K., MOKREJS, M., MORGENSTERN, B., MINSTERKTTER, M., RUDD, S., AND WEIL, B. 2002. Mips: a database for genomes and protein sequences. *Nucleic Acids Res 30*, 31–34.

MLADENIE, D. 1998. Machine learning on non-homogeneous, distributed text data. Ph.D. thesis, University of Ljubljana, Faculty of Computer and Information Science.

NAKAGAWA, S. AND FRECKLETON, R. 2008. Missing inaction: the dangers of ignoring missing data. *Trends in Ecology & Evolution 23,* 11, 592–596.

NGUYEN, C., DUNG, T., AND CAO, T. 2005. Text classification for dag-structured categories. In *Advances in Knowledge Discovery and Data Mining,* T. Ho, D. Cheung, and H. Liu, Eds. Lecture Notes in Computer Science, vol. 3518. Springer Berlin / Heidelberg, 97–114.

OBA, S., SATO, M.-A., TAKEMASA, I., MONDEN, M., MATSUBARA, K.-I., AND ISHII, S. 2003. A bayesian missing value estimation method for gene expression profile data. *Bioinformatics 19,* 16, 2088–2096.

PEARSON, R. K. 2006. The problem of disguised missing data. *SIGKDD Explorations 8,* 1, 83–92.

Peter, C. Z., Jansen, P., Stoica, E., Grot, N., and Evans, D. A. 1998. Threshold calibration in clarit adaptive filtering. In *Proceedings of Seventh Text REtrieval Conference (TREC-7).* 149–156.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning 1,* 1 (March), 81–106.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*, 1 ed. Morgan Kaufmann.

Quinlan, J. R. 1996. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research 4*, 77–90.

Rifkin, R. and Klautau, A. 2004. In defense of one-vs-all classification. *Journal of Machine Learning Research 5*, 101–141.

Riley, M. 1993. Functions of the gene products of escherichia coli. *Microbiol. Mol. Biol. Rev. 57,* 4, 862–952.

Rubin, D. 1996. Multiple imputation after 18+ years. *Journal of the American Statistical Association 91,* 434, 473–489.

Rubin, D. B. 1987. *Multiple Imputation for Nonresponse in Surveys.* John Wiley & Sons, New York.

Sarinnapakorn, K. and Kubat, M. 2007. Combining subclassifiers in text categorization: A dst-based solution and a case study. *IEEE Transactions on Knowledge and Data Engineering 19,* 12, 1638–1651.

Schietgat, L., Vens, C., Struyf, J., Blockeel, H., Kocev, D., and Dzeroski, S. 2010. Predicting gene function using hierarchical multi-label decision tree ensembles. *Bmc Bioinformatics 11.*

Schneider, T. 2001. Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of Climate 14*, 853–871.

Schölkopf, B., Burges, C. J. C., and Smola, A. J., Eds. 1999. *Advances in kernel methods: support vector learning.* MIT Press, Cambridge, MA, USA.

Sebastiani, F. 2006. Classification of text, automatic. In *The Encyclopedia of Language and Linguistics*, Second ed., K. Brown, Ed. Vol. 2. Elsevier Science Publishers, Amsterdam, NL, 457–463.

Secker, A., Davies, M. N., Freitas, A. A., Timmis, J., Mendao, M., and Flower, D. R. 2007. An experimental comparison of classification algorithms for hierarchical prediction of protein function. *3rd UK Data mining and Knowledge Discovery Symposium UKKDD 2007 Canterbury*, 1318.

Shafer, J. C., Agrawal, R., and Mehta, M. 1996. SPRINT: A scalable parallel classifier for data mining. *VLDB'96*, 544–555.

SHANAHAN, J. G. AND ROMA, N. 2003. Boosting support vector machines for text classification through parameter-free threshold relaxation. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*. ACM, 247–254.

SHWARTZ, S. AND SREBRO, N. 2008. Svm optimization: inverse dependence on training set size. In *Proceedings of the 25th international conference on Machine learning*. ACM, 928–935.

SILLA, C. AND FREITAS, A. 2010. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*.

SNOEK, C. G. M., WORRING, M., VAN GEMERT, J. C., GEUSEBROEK, J. M., AND SMEULDERS, A. W. M. 2006. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*. ACM, New York, NY, USA, 421–430.

STEIN, L. 2001. Genome annotation: from sequence to biology. *Nat Rev Genet 2,* 7, 493–503.

STENGER, B., THAYANANTHAN, A., TORR, P. H. S., AND CIPOLLA, R. 2007. Estimating 3d hand pose using hierarchical multi-label classification. *Image Vision Comput. 25,* 12, 1885–1894.

SUN, A. AND LIM, E.-P. 2001. Hierarchical text classification and evaluation. In *Proceedings of the 2001 IEEE International Conference on Data Mining*. ICDM '01. IEEE Computer Society, Washington, DC, USA, 521–528.

SUN, A., LIM, E.-P., AND LIU, Y. 2009. On strategies for imbalanced text classification using svm: A comparative study. *Decision Support Systems 48,* 1, 191–201.

TAN, P.-N., STEINBACH, M., AND KUMAR, V. 2005. *Introduction to Data Mining*. Addison Wesley.

THABTAH, F. A., COWLING, P., AND YONGHONG, P. 2004. Mmac: a new multi-class, multi-label associative classification approach. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04)*. 217–224.

TROHIDIS, K., TSOUMAKAS, G., KALLIRIS, G., AND VLAHAVAS, I. 2008. Multilabel classification of music into emotions. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008), Philadelphia, PA, USA, 2008*.

TSOUMAKAS, G. AND KATAKIS, I. 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining 3,* 3, 1–13.

TWALA, B. 2009. An empirical comparison of techniques for handling incomplete data using decision tree. *Appl. Artif. Intell. 23,* 5, 373–405.

van Rijsbergen, C. J. 1979. *Information Retrieval*, 2 ed. Butterworths, London.

Vateekul, P., Dendamrongvit, S., and Kubat, M. 2011. Improving svm performance in multi-label domains: Threshold adjustment. *International Journal on Artificial Intelligence Tools (IJAIT).* (Submitted and In Process).

Vateekul, P. and Kubat, M. 2009. Fast induction of multiple decision trees in text categorization from large scale, imbalanced, and multi-label data. In *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops.* Miami, FL, USA, 320–325.

Vateekul, P., Kubat, M., and Sarinnapakorn, K. 2012. Hierarchical multi-label classification with svms: a case study in gene function prediction. *IEEE Transactions on Knowledge and Data Engineering (TKDE).* (Submitted and In Process).

Vateekul, P. and Sarinnapakorn, K. 2009. Tree-based approach to missing data imputation. In *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops.* IEEE Computer Society, Miami, FL, USA, 70–75.

Vateekul, P. and Shyu, M.-L. 2008. A conflict-based confidence measure for associative classification. In *Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2008*, M. IEEE Systems and C. Society, Eds. 256–261.

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, K., Zhou, S., and He, Y. 2000. Growing decision trees on support-less association rules. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.* 265–269.

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

Williams, N., Zander, S., and Armitage, G. 2006. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review 36,* 5 (Oct.), 5–16.

Winkler, W. E. 2003. Methods for evaluating and creating data quality. *Information Systems 29*, 531–550.

Yan, L., Xei, D., and Du, Z. 2009. A new method of support vector machine for class imbalance problem. In *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization.* 904–907.

Yang, Y. 1999. An evaluation of statistical approaches to text categorization. *Information Retrieval 1,* 1/2, 69–90.

Yang, Y., Zhang, J., and Kisiel, B. 2003. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval.* SIGIR '03. ACM, New York, NY, USA, 96–103.