2016-03-18

A Novel Stacking Method for Multi-label Classification

Abdulaziz Alali
University of Miami, grader318@gmail.com

UNIVERSITY OF MIAMI

A NOVEL STACKING METHOD FOR MULTI-LABEL CLASSIFICATION

By

Abdulaziz Alali

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 2016

UNIVERSITY OF MIAMI

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

A NOVEL STACKING METHOD FOR MULTI-LABEL CLASSIFICATION

Abdulaziz Alali

Approved:

Miroslav Kubat, Ph.D. Associate Professor of Electrical and Computer Engineering	Kamal Premaratne, Ph.D. Victor P. Clarke Professor of Electrical and Computer Engineering
Mei-Ling Shyu, Ph.D. Professor of Electrical and Computer Engineering	Nigel John, Ph.D. Lecturer of Electrical and Computer Engineering
Ubbo Visser, Ph.D. Associate Professor of Computer Science	Guillermo Prado, Ph.D. Dean of the Graduate School

ALALI, ABDULAZIZ (Ph.D., Electrical and Computer Engineering)

A Novel Stacking Method for Multi-label Classification (May 2016)

Abstract of a dissertation at the University of Miami.

Dissertation supervised by Professor Miroslav Kubat.
No. of pages in text. (128)

Classical machine learning algorithms were tailored to automatically classify examples that belong to mutually exclusive classes; each example may belong to *one* class out of a finite set of classes. In realistic applications, however, examples often belong to more than one class *at the same time*. For example, a text document that belongs to `Geography` may also be labeled as `Geology`. Perhaps due to the popularity of its applications, targeting this category of problems has garnered great research interest over the past decade. A widely popular approach, called *Binary Relevance (BR)*, is to induce a separate classifier for each class; to determine whether the class is relevant for an example, or not. Despite showing some success, researchers have pointed out a critical drawback in this method. By targeting each class independently, the learner does not model class correlations: knowing if an example belongs to class X may indicate that it is likely to belong also to class Y. Conversely, this information can make the example less likely to belong to class Z. Research groups sought to incorporate class correlation information into *BR* by using the class labels as additional example features. Since the information about which class an example belongs to is unknown in unseen instances, the missing values are typically filled-in using the outputs of other classifiers, which makes them prone to errors. This dissertation identifies two weaknesses in existing methods: *unnecessary label correlations*, and

error-propagation. To overcome these problems, this dissertation introduces a new multi-label classification method, called *PruDent*. Experiments over a broad range of benchmark datasets indicate that *PruDent* compares rather favorably with existing state-of-the-art methods. Additionally, *PruDent* improves classification accuracy while maintaining a linear complexity in the number of classes.

To my parents

Acknowledgements

I would like to thank my advisor Dr. Miroslav Kubat who supported me in the past few years through the research and completion of my degree. I believe his personality and technical capability was an indispensable factor for me to finish this endeavor.

Abdulaziz Alali

University of Miami

May 2016

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The influx of data produced by venues such as social media, news articles, and blogs is only expected to grow in the upcoming years. In order to access these data in an efficient manner, it is desired to index them by organizing them into known categories. When the data are sizable (e.g. millions of documents), using man-power to categorize them no longer becomes a viable option; not only is this process impractical, but it is also financially costly. To this end, improving automatic classification using the techniques from the Machine Learning discipline has become an attractive research topic. The idea is to expose the data to a learning algorithm, known as a *classifier*, that can relate attributes that define examples to their respective categories. This chapter will give a brief introduction to the traditional classification problem first. After that is an overture to the branch of problems that constitute the focus point of this disseration, which is known as multi-label classification. Towards the end of this chapter, an overview of the current research difficulties is provided, followed by the goals pursued by this research work.

1.1 The Classification Problem

The objective in a classification task is to learn a *concept* based on previously known information about this concept in the form of examples that represent it. In the Machine Learning literature, the term *concept* is interchangeably used with the terms *class* and *category*. As illustrated by Figure 1.1, the learning process is performed by exposing the learning algorithm to examples that are known to represent a concept (positive examples), and those that do not represent it (negative examples). Based on the usually numerical attributes that define examples (also called *features*), a classification model is sought to predict whether a previously unseen example represents this concept or not. This classification model is referred to using the term *classifier*.

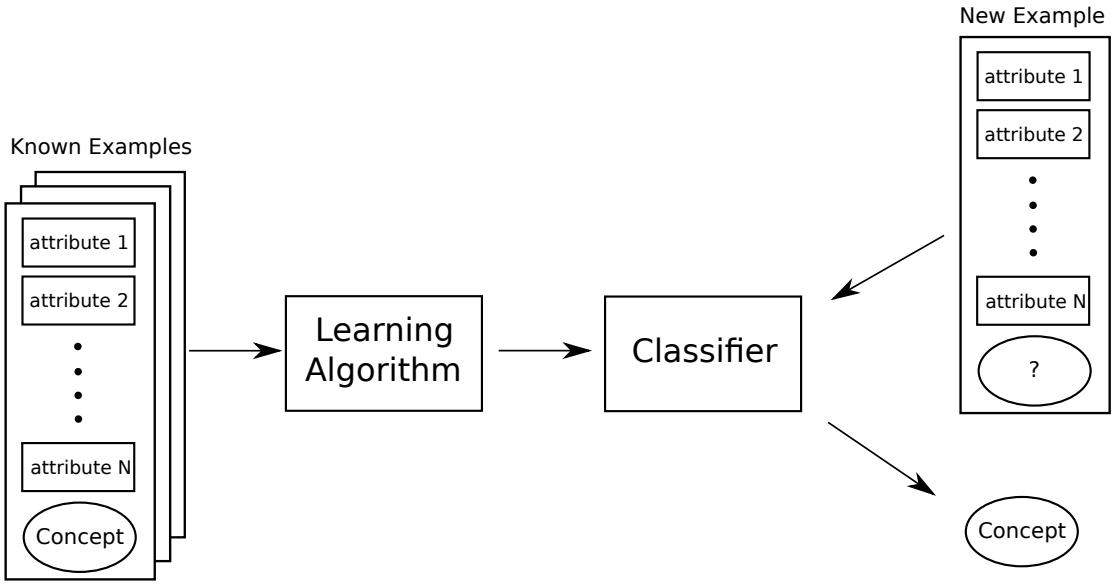


Figure 1.1: Classifier induction process.

Classical machine learning algorithms were tailored to classify objects that may belong to *one* class out of a finite set of classes. An example application of such

is to detect oil spills in satellite images of oceanic surfaces [2]. In this case, the concept (`oil-spill`) is either present in an image or not. Since there are only two possible outcomes, this problem is known as *binary* classification. Also of interest are *multi-class* problems where more than two outcomes are possible. For instance, one might be interested in knowing a suitable commuting method given the day's traffic conditions. Here, the output can perhaps be restricted to `bus`, `car`, or `metro-rail`.

Classification problems that adhere to the rule that limits each example to one class (*single-label*) were targeted for more than three decades. Consequently, several learning algorithms can now achieve somewhat satisfactory classification accuracy in these domains. Examples of the commonly used algorithms today include Decision Trees, Neural Networks, K-Nearest-Neighbors, and Support Vector Machines.

1.2 From Single-label to Multi-label Classification

Early machine learning algorithms sought to classify examples that are categorized using individual classes. However, this is rarely the case in many realistic applications. For instance, as depicted in Figure 1.2, an image that contains '`PalmTree`' may also include '`Sun`', '`Clouds`', and '`Sea`'. By relaxing the limitation imposed on the number of classes an example can belong to, the problem becomes very challenging: in addition to not knowing the number of classes relevant to an unseen example, the number of combinations an output can take is now significantly larger, which in turn introduces more room for error. With the start of the new millennium, problems of this kind garnered much of the research interest becoming what is known today as *multi-label* classification.

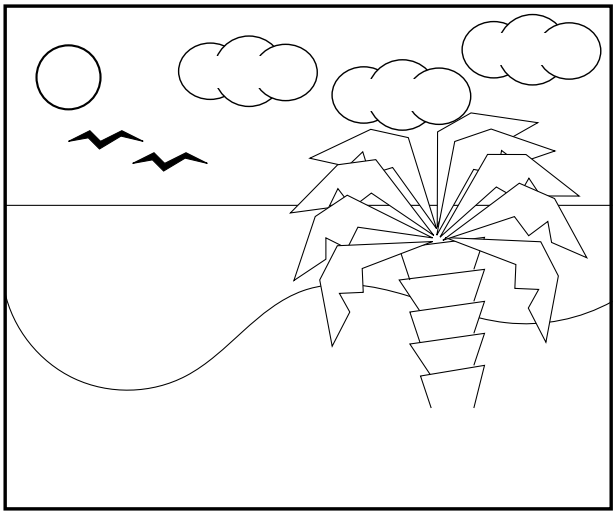


Figure 1.2: An example beach scene image.

Early methods that target multi-label domains resorted to modifying existing algorithms to handle multi-label outputs. In general, the solutions that followed this approach were applicable only to specific domains such as text categorization [3]. Moreover, these methods were restricted to certain classifier types such as Neural Networks [4] and Decision Trees [5].

In the following years, an attractive group of methods were favored among the research community. Instead of adapting current learning algorithms to handle multi-label outputs, they *transform* the multi-label problem into a single-label one. A widely popular approach from this category of solutions is known as the *Binary Relevance (BR)* method. The idea is to independently induce a separate binary classifier for each category; each classifier is thus tasked with predicting whether a category is relevant for a given example or not. A final classification is then obtained by aggregating all the individual classifiers' outputs. This approach soon became popular because of the following advantages: First, it is very simple to implement since it involves transforming the data rather than the algorithms. Additionally, this

method benefits from the flexibility of being applicable to the vast library of existing single-label learning methods. Finally, this approach scales well in large domains since its complexity is linear in the number of classes; for each additional class, only one more classifier is needed.

The above advantages motivated the adoption of the Binary Relevance framework in this dissertation. Despite showing promising classification performance when compared to other existing approaches [6], the Binary Relevance method also has its own drawbacks, which are discussed in the next section.

1.3 Research Difficulties

The state of the current multi-label classification approaches leaves much to be desired. Perhaps due to the infancy of this subdicipline of machine learning, the classification performance of current multi-label solutions is found to be lacking when compared to their single-label counterparts. The next paragraphs will discuss the difficulties typically faced when dealing with multi-label classification.

In multi-label domains, the number of classes a new example belongs to is typically unknown beforehand; it is only limited by the number of possible classes observed previously in the dataset. While the lack of this information may not constitute a major problem in domains with few classes, in larger domains, however, lacking this information can significantly complicate classification. Suppose that there are 100 possible categories in a given classification problem. In this case, the classifier is required to target an output space of 2^{100} possibilities. Here, finding the exact categories a new example belongs to becomes a difficult task.

The next research problems are mainly associated with the *BR* method that is discussed towards the end of the previous section.

Recall that the *BR* approach assigns each class an independent classifier to identify whether the class applies to future examples or not. To induce each of these classifiers, the training sets are generated by choosing the examples that represent the target class as positive examples, and all others as negative examples. As a result, the example representation in the training sets will almost always be imbalanced; there are more negative examples (which may belong to any other class) than positive examples of the target class. This situation can lead classifiers that minimize the general error to perform in an undesired manner [7]. For example, suppose that we are tasked to predict whether a patient has a disease that occurs in only 2% of the population. Here, a classifier that always makes a '`negative`' diagnosis recommendation will be 98% accurate. Obviously, such information is not helpful in this case.

By targeting each class individually, the *BR* approach does not account for the possible class correlations in multi-label domains. For example, in the beach image illustrated previously by Figure 1.2, detecting the concept `ocean` is a strong indicator for the possible occurrence of the concepts `water` and `beach`. Conversely, belonging to these concepts makes the image less likely to represent `downtown`. Research indicates that leveraging this information can potentially increase classification accuracy [8–12]. However, correlation information is not explicitly provided in a typical multi-label problem. Leveraging this information is a challenging task as classes may, or may not, be correlated. Moreover, as will be detailed in the next chapter, some correlations may exist in only a subset of the data making them conditional to the example at hand [13].

The following is a summary of the challenges mentioned above:

- The number of classes assigned to each unseen example is unknown. As the number of classes increase, this problem is further complicated.
- The example representation of classes tend to be imbalanced. Targeting each class individually may bias the classifier towards choosing the class with the majority of examples.
- Classes may be correlated, and this information is not explicitly provided in the data. Capturing this information is difficult as not all classes are necessarily related.

1.4 Research Objectives and Contributions

Despite the ubiquitous nature of multi-label applications, methods which target this category of problems leave ample room for improvement. This research aims to develop a multi-label solution which targets the limitations mentioned in the previous section by adhering to the following guidelines:

- The solution must scale well in domains that are characterized by a large number of labels.
- Classification must consider that multi-label classes are imbalanced (some classes outnumber others). Consequently, the proposed method must maintain a good prediction balance for each class.
- The algorithm must be able to automatically detect important class dependencies that are relevant to the problem at hand.

- When incorporating class dependencies, it must be done in a manner that improves classification more often than not.

This research resulted in the development of a new multi-label classification solution [14] which targets all of the difficulties mentioned in the previous section. Originally, the goal was to incorporate all possible class correlation information in the *BR* framework. However, as research progressed, it was evident that not all correlations can improve classification accuracy. Moreover, the proposed work shows that if handled incorrectly, incorporating class dependencies can harm classification performance. As a result, this work was focused on automatically identifying important class dependencies and incorporating them in a manner than improves classification accuracy.

The newly developed multi-label classification solution, which is named *PruDent* [14], improves upon a previously known framework known as *Stacking* by mitigating two of its inherit problems. As demonstrated by the experiments in Section 4.4, a clear improvement in classification accuracy was achieved. Moreover, the proposed solution was able to improve classification performance while lowering the computational cost of the original framework. When compared to other existing state-of-the-art approaches, the proposed solution performed rather well across a broad range of multi-label domains.

CHAPTER 2

Multi-label Classification

Multi-label classification is a branch of machine learning which targets examples that may belong to more than one class at the same time. This situation has been encountered in many realistic applications such as text categorization [3,9,15–18], image and video annotation [19–22], and gene function prediction [5]. In the text classification domain, a '`Geography`' document may also be labeled as '`History`'. Similarly, in the medical field, a '`Cough`' diagnosis may also be associated with '`Asthma`'. Also, in scene classification applications, a picture tagged as '`Sunset`' may also be tagged as '`Ocean`'. Thus, unlike single-label problems, classes in the multi-label domain can co-occur over the same set of examples.

This chapter starts with a formal definition of the multi-label classification framework. Then, the reader is presented with the general types of class dependencies associated with multi-label domains. Following that is a section that details the criteria used to evaluate multi-label classifiers. This chapter ends with a description of the employed multi-label benchmark datasets, followed by their background information.

2.1 The Multi-label Framework

Let us first define the problem at hand. In multi-label classification, it is assumed that classes are represented by a finite label space $\mathcal{L} = \{l_1, l_2, ..., l_L\}$, and that examples are defined using a d-dimensional instance space $\mathcal{X} \in \mathbb{R}^d$. It is also assumed that each instance, $\mathbf{x} \in \mathcal{X}$, is associated with a subset of the labels, $Y \subseteq \mathcal{L}$. The associated label-set can be represented as a vector, $\mathbf{y} = (y_1, y_2, ..., y_L)$, such that $y_i = 1$ indicates that label l_i is relevant for a given example, and $y_i = 0$ indicates that it is not. Using an independently and identically distributed training set, $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, a classifier $\mathbf{h}$ is sought to carry out a mapping $\mathbf{h} : \mathcal{X} \rightarrow \mathcal{L}$ that generalizes beyond the training set. When the classifier system is composed of a separate classifier per class, the notation follows the form $\mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x}))$, where each binary classifier $h_i(\mathbf{x})$ assigns to example $\mathbf{x}$ a prediction $\hat{y}_i \in \{0, 1\}$. To distinguish *predicted* label vectors from the *actual* groundtruth labels, the former will be denoted as $\hat{\mathbf{y}}$ and the latter as $\mathbf{y}$. Unless specified otherwise, the term N will be used to denote the number of examples, and L will be used to represent the number of classes.

2.2 Types of Class Dependencies

Multi-label domains are characterized by having examples that may belong to more than one class simultaneously. As a result, some classes may exhibit a correlation pattern with others; they either co-occur frequently, or rarely occur at the same time. Thus, it is sensible to develop an understanding of the nature of these dependencies, which is the task of this section.

The authors of [12] identify two types of dependencies that can exist between classes: *global dependencies*, and *local dependencies*. They are also commonly referred to as *unconditional* and *conditional* dependencies respectively. The subsections below explain their differences and illustrate examples of each type. Note that the following explanation will closely resemble that given in [12].

2.2.1 Global Dependencies

Global dependencies are characterized as being generalizable to the majority of examples in a given dataset. In other words, they are *unconditional* to any particular subset of the data. Due to this property, the literature also refers to these correlations using the term *unconditional dependencies*.

Given the prevalence of this category of dependencies in the data, they tend to be easily identifiable. For instance, suppose that a text document is about the topic '`Diet`'. This document can arguably be attributed also to '`Cooking`'. However, the same document can not represent '`Mining-Industry`'. Note that we can infer this information from the topics alone, and without any knowledge of the examples' attributes.

In [12], the authors offer a probabilistic interpretation of global dependencies; they assert that when the classes are independent, the following equality must hold:

$$P(\mathbf{y}) = \prod_{i=1}^{L} P(y_i) \tag{2.1}$$

Thus, when there are underlying global class dependencies, the product of the independent (marginal) class probabilities should not equal the joint mode of the class distribution.

2.2.2 Local Dependencies

In contrast to global dependencies, local dependencies are associated with only a *subset* of examples that usually represents a smaller portion of the data [13]. As such, these dependencies are conditional on the examples' features. For this reason, they are also commonly referred to as *conditional dependencies*. Perhaps a good example of this dependency type can be brought from the medical field; an '`Asthma`' diagnosis may be associated with '`Flu`' only if `Coughing` is one of the symptoms. In this case, `Coughing` defines the subset in which this relationship applies.

From a probabilistic perspective, local dependencies differ from global ones by incorporating the example, $\mathbf{x}$, as a condition in the mathematical formulation. Thus, when two classes are conditionally independent the following equation must hold:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{L} P(y_i|\mathbf{x}) \tag{2.2}$$

When considering the product rule of probability, the conditional joint mode of a random vector can be expressed as follows:

$$P(\mathbf{y}|\mathbf{x}) = P(y_1|\mathbf{x}) \prod_{i=2}^{L} P(y_i|y_1, ..., y_{i-1}, \mathbf{x}) \tag{2.3}$$

Here, the reader can see that when the classes are conditionally independent, Equation 2.3 should simplify to Equation 2.2.

Having pointed out the difference between global and local dependencies, it is imperative to also show how they are connected. This is accomplished by considering the following term:

$$P(\mathbf{y}) = \int_{\mathcal{X}} P(\mathbf{y}|\mathbf{x})\, dx \tag{2.4}$$

The above equation tells us that global dependencies can be thought of as expected dependencies which are averaged over all instances.

2.3 Performance Criteria

This section starts by explaining the classical performance criteria that are used to assess binary and multi-class domains. Building on that knowledge, this section will then show how some of these criteria can be extended to handle multi-label problems. Also, this section will provide some of the criteria that were created specifically to evaluate multi-label classifiers.

2.3.1 Classical Performance Criteria

Traditionally, in order to measure the classification performance of (single-label) classifiers, one could simply use the `accuracy` function as defined below:

$$accuracy = \frac{\#\ correctly\ classified\ examples}{\#\ examples} \tag{2.5}$$

This function may be suitable for problems where classes are equally represented by examples. However, when class representations are imbalanced, using this method will not be sufficient; a classifier can always choose the prevailing class and attain good accuracy (see example in Section 1.3). Instead, we need a way to evaluate positive and negative examples separately. This is where metrics such as `precision (Pr)` and `recall (Re)`, which are borrowed from the *information retrieval* field, come into play. These evaluation measures are computed by considering the number of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) examples. Typically, a contingency table such as Table 2.1 is constructed. For

example, false positive (FP) examples are those which were marked by a classifier as positive but are in fact negative examples.

		Actual Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Table 2.1: An example contingency table

Based on the frequency counts of a contingency table, the metrics are defined as follows:

$$\texttt{Pr} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}} \qquad \texttt{Re} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}} \tag{2.6}$$

On the one hand, `precision` measures the number of correctly classified positive examples over the number of examples that are *predicted* as positive. On the other hand, `recall` measures the number of correctly classified positive examples over the number of examples that are *actually* positive. By looking at the denominator of the `precision` equation, the reader can see that it is sensitive to the purity of the predictions; the denominator represents the number of examples *predicted* as positive. Conversely, `recall` quantifies how many positive examples were retrieved from the data. In general, optimizing an algorithm towards one measure will sacrifice the other. For instance, if a classifier is biased towards positive label predictions, it may incur a higher number of false positives. In this scenario, `recall` will increase and `precision` will decrease.

To establish a method of quantifying the trade-off between the two measures, [23] suggested the use of the the F-measure as follows:

$$\mathtt{F}_\beta = \frac{(\beta^2 + 1) \times \mathtt{pr} \times \mathtt{re}}{\beta^2 \times \mathtt{pr} + \mathtt{re}} \tag{2.7}$$

The β term in the above equation is a user specified parameter in the range $[0, \infty)$. Using a value of $\beta > 1$ assigns more weight to `recall`, and when $\beta < 1$, more weight is given to `precision`. When an equal balance between the two criteria is desired, the value $\beta = 1$ is used, which represents the harmonic mean of `recall` and `precision`. As it became widely adopted in the literature, this criterion became known as the $\mathtt{F}_1$ metric, and it is calculated as illustrated by Equation 2.8 below:

$$\mathtt{F}_1 = \frac{2 \times \mathtt{pr} \times \mathtt{re}}{\mathtt{pr} + \mathtt{re}} \tag{2.8}$$

Although the above performance criteria were designed to evaluate single-label classifiers, these performance measures can be extended to handle multi-label classifiers as well. Their multi-label extensions will be provided in the next section.

2.3.2 Multi-label Evaluation Criteria

To quantify diverse behavioral aspects of multi-label classifiers, several performance criteria were proposed. This section will use the notations from Section 2.1 to define the evaluation metrics. Recall that N and L denote the numbers of examples and classes, respectively.

- `0/1 loss`:

  The `0/1 loss` measures the *exactness* of the multi-label prediction by issuing a penalty each time the *predicted* class vector ($\hat{\mathbf{y}}$) of an example differs from the vector of *actual* labels ($\mathbf{y}$). This function is thus a generalization of the

traditional error function to multi-label domains. For each example, a '0' loss is awarded for correct *label set* classification and '1' otherwise (hence the name). A final loss is then obtained by averaging the losses over all examples.

$$\begin{aligned} \texttt{0/1 loss} &= 1 - \frac{1}{N}\sum_{i=1}^{N} k_i \\ \mathtt{k_i} &= \begin{cases} 1, & \hat{\mathbf{y}}_\mathbf{i} = \mathbf{y_i} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2.9}$$

The authors of [12] discovered that `0/1 loss` can be minimized by algorithms that are designed to find the joint mode of the label distribution, $P(\mathbf{y})$, because they target classes in tandem. This will not be the case in the `hamming loss` measure, which is explained next.

- `Hamming loss`:

  This criterion punishes a multi-label classification on a *per-label* basis. As such, a partially correct classification is penalized less severely than a completely incorrect one. It was found in [12] that algorithms which target classes independently are sufficient for reducing this error.

$$\begin{aligned} \texttt{hamming loss} &= 1 - \frac{1}{L}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{L} k_i^j \\ \mathtt{k_i^j} &= \begin{cases} 1, & \hat{y}_i^j = y_i^j \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2.10}$$

Thus far, we have covered two *loss* functions in which a *lower* score signifies *better* performance. The following criteria have *higher* scores when classifications are *better*.

- `Accuracy`:

  In [9], the authors introduced `accuracy` which is calculated by the following formula:

$$\texttt{accuracy} = \frac{1}{\texttt{N}} \sum_{\texttt{i}=1}^{\texttt{N}} \frac{|\hat{\mathbf{y}}_\texttt{i} \cap \mathbf{y}_\texttt{i}|}{|\hat{\mathbf{y}}_\texttt{i} \cup \mathbf{y}_\texttt{i}|} \tag{2.11}$$

  The purpose of `accuracy` is to quantify the "closeness" of a binary class prediction vector ($\hat{\mathbf{y}}$) to the actual label vector ($\mathbf{y}$). In the definition, the $|.|$ symbol denotes the cardinality of positive labels in the given binary vector.

- `Precision, recall,` and `f1`

  Let us now consider the information retrieval metrics defined in the previous section (Equations 2.6 and 2.8). Recall that they are employed to evaluate classifiers which target single-label domains. The goal here is to generalize them for application in multi-label domains. To accomplish this task, there are two commonly used approaches: the *macro*, and *micro* averaging introduced by [24].

  In the *macro* averaging method, the measures are first calculated for each class separately as defined in the previous section. To fuse the calculated measures of each class together, they are then averaged over the total number of classes as shown in Equation 2.12 below. In essence, each class is thus *equally weighted*.

$$\texttt{Pr}^{\texttt{M}} = \frac{1}{L}\sum_{i=1}^{L}\texttt{Pr}_{\texttt{i}}, \tag{2.12a}$$

$$\texttt{Re}^{\texttt{M}} = \frac{1}{L}\sum_{i=1}^{L}\texttt{Re}_{\texttt{i}}, \tag{2.12b}$$

$$\texttt{F1}^{\texttt{M}} = \frac{1}{L}\sum_{i=1}^{L}\texttt{F1}_{\texttt{i}} \tag{2.12c}$$

When it is preferable to focus on classes which occur more frequently than others, the *micro* averaging method is used instead. This averaging approach weighs the `precision, recall`, and `f1` measures by the "popularity" of the individual classes. As such, the metrics are averaged over all examples as if they belonged to one class. More populous classes are thus marked by higher weights than less populous ones; because they contain more representative (positive) examples.

$$\texttt{Pr}^{\mu} = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FP_i}, \tag{2.13a}$$

$$\texttt{Re}^{\mu} = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FN_i}, \tag{2.13b}$$

$$\texttt{F1}^{\mu} = \frac{2 \times Pr^{\mu} \times Re^{\mu}}{Pr^{\mu} + Re^{\mu}} \tag{2.13c}$$

As presented in this section, there are many aspects to consider when evaluating multi-label classifiers. Choosing which of these metrics to optimize is perhaps determined by the end-user's needs. For completeness, this dissertation will document the experiments using all of the above performance criteria.

2.4 Multi-label Benchmark Data

The circumstance in which classes overlap over the same set of examples occurs in many real world applications. For the sake of comparing algorithms that target this kind of data, several benchmark testbeds were brought from the biology; image, audio, and video annotation; and text document categorization domains. This dissertation employs a broad range of such datasets that are presented in Table 2.2[1]. The choice of testbeds was motivated by their frequent appearance in recent papers. The datasets are sorted in the table by their complexity ($L \times N \times D$), where L, N, and D stand for the number of classes, examples and features respectively. In the number of features column (D), features that are continuous are denoted using the term 'n', and binary features are denoted using 'b'.

The dataset statistics table also includes two metrics that specifically characterize multi-label datasets: *Label Cardinality (LC)*, and *Label Density (LD)*. They were introduced by [10] in order to quantify the distribution of classes in a given dataset. LC is the average number of classes that appear per example, $LC = \frac{1}{N}\sum_{i=1}^{N}|Y_i|$. The value of LC is higher if the examples frequently belong to several categories. It is also important to know the proportion of the total number of labels that LC constitutes. For example, having an average co-occurrence of 2 classes out of 100 indicates a sparser distribution than 2 out of 6. To quantify this circumstance is the task for the other measure. More specifically, LD is LC divided by the total number of

[1]The datasets can be obtained from `http://meka.sourceforge.net/#datasets` and `http://mulan.sourceforge.net/datasets-mlc.html`. For `rcv1`, *subset1* was used. For `nus-wide`, the cVLAD+ features version was used.

name	domain	features (D)	classes (L)	examples (N)	LC	LD
emotion	music	$72n$	6	593	1.87	0.31
yeast	biology	$103n$	14	2417	4.24	0.3
scene	image	$294n$	6	2407	1.07	0.18
cal500	music	$68n$	174	502	26	0.15
genbase	biology	$1186b$	27	662	1.25	0.05
medical	text	$1449b$	45	978	1.25	0.03
slashdot	text	$1079b$	22	3782	1.18	0.05
enron	text	$1001b$	53	1702	3.38	0.06
langlog	text	$1004b$	75	1460	1.18	0.02
mediamill	video	$120n$	101	43907	4.4	0.04
bibtex	text	$1836b$	159	7395	2.4	0.02
nus-wide	image	$128n$	81	269648	1.87	0.02
imdb	text	$1001b$	28	120919	2.0	0.07
rcv1	text	$47236n$	101	6000	2.88	0.03
tmc2007	text	$500b$	22	28596	2.16	0.1

Table 2.2: Data Sets Statistics

possible labels, $LD = \frac{LC}{L}$. While LC provides a measure of the "multi-labelledness," LD shows the sparsity of label occurrences.

Throughout the remainder of this chapter, the reader will be presented with the necessary background information about each of the employed datasets in the order of appearance in Table 2.2:

- Emotion

  This dataset represents the problem of extracting human emotions and mood that are experienced when listening to music [25, 26]. The data was created by extracting a 30 second sound clip that occurs after an initial 30 seconds have elapsed from a full sound track. Based on this short clip, a set of 72 features are extracted. Of which, 8 are rhythmic and 64 are timbre features. To quantify emotions, the Tellegen-Watson-Clark model was used giving a total of

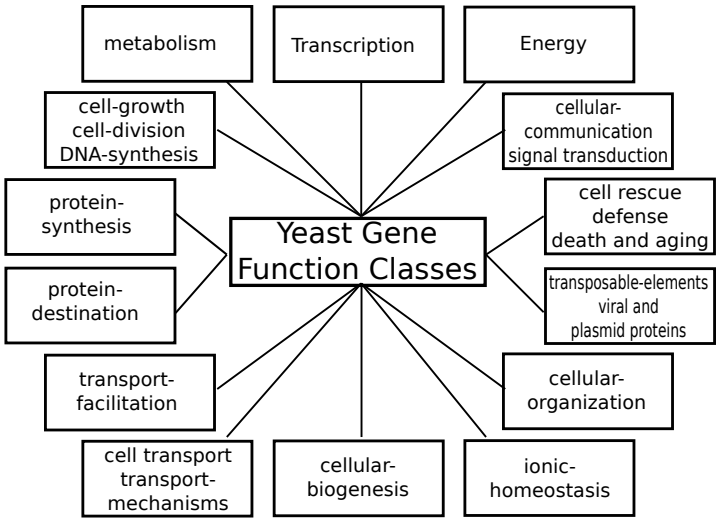


Figure 2.1: Gene functional groups of the first level in the hierarchy of yeast gene functions [1].

6 emotional classes: *amazed-surprised*, *happy-pleased*, *relaxing-calm*, *quiet-still*, *sad-lonely*, and *angry-fearful*. Finding the appropriate labels associated with each song was done with the aid of three experts from the School of Music Studies in the authors' University. Perhaps to reduce the noise in the data, only the songs with identical chosen labels from the three experts were allowed in the database. Ultimately, 593 clips remained.

- Yeast

  This testbed represents a collection of gene micro-array expressions and phylogenetic data of the Yeast Saccharomyces cerevisiae organism [1,27]. Micro-array expression data is obtained by subjecting yeast to specific test criteria after a target gene is modified. Experiments include diauxic shift, the mitotic cell division cycle, sporulation and temperature reduction shocks. The task here is to predict the functional class of genes using their micro-array expression

data. However, each gene can be associated with more than 190 functional classes. As such, the authors of [1] utilized the hierarchical structure of the functional classes such that only the classes of the first level of the hierarchy were included. This reduced the set of classes to the 14 categories depicted in Figure 2.1. Examples of these classes include *metabolism*, *protein synthesis*, and *cellular biogenesis*. The 2417 genes of this data set are described by 103 numerical features.

- Scene

  The scene database consists of a set of scenic images which are obtained from the Corel Stock Photo and some personal images of the authors in [19]. Each image is first transformed to the CIELUV color space. Then, images are converted to a low resolution 7×7 bins. For each bin, the mean, variance, and some other computationally inexpensive texture-based features are extracted forming a total of 294 numerical features. The 2407 images of this domain are allowed to belong to 6 possible classes and their combinations. These classes are *Beach*, *Sunset*, *Fall-Foliage*, *Field*, *Mountain*, and *Urban*. To assign the groundtruth labels to each image, they used the annotations of three human observers.

- Cal500

  Targeting the music domain, this dataset represents a compilation of 500 "Western Popular" songs [28]. The task is to annotate new songs using meaningful keywords such as instrument names (e.g. Guitar, Piano, and Saxophone) and music genres (e.g. Hip-hop, Jazz, and Rock). The objective is to provide a *query-by-text* system for music lookup. The features of each song consist of

time series MFCC-Delta feature vectors, which are obtained by sliding a half-overlapping short-time window [28]. In order to acquire the training set classes, the authors conducted a survey over a group of 66 undergraduate students. At least 3 semantic annotations were provided for each song. A total of 1708 annotations were acquired for all songs.

- Genbase

  This dataset belongs to the biology domain where the task is to predict the function of a protein based on its structure [29]. The idea comes from the foundation that proteins that belong to the same family of functions are found to be structurally similar. Protein patterns are defined by the specific sequence of their amino-acid chains. The authors refer to such sequences in combination with computational representations of sequence alignments as *motifs*. The features used in this testbed are binary representations of whether a motif exists in a given protein or not. The vocabulary used in this dataset is composed of 1186 motifs. In total, 662 proteins were exported, and 27 class labels composed of the 10 most common families of protein functions were selected. Examples of class families include *cytokines-and-growth-factors*, *receptors*, *transferases*, and *DNA-or-RNA-associated-proteins*.

- Medical

  The medical dataset is concerned with classifying clinical free text of radiology reports (chest x-ray and renal procedures) using 45 ICD-9-CDM diagnosis codes [30]. As described by the authors, radiology reports are composed of two essential parts. The first of which is the *clinical history*, and the second is the

impression. The former is provided by the ordering physician before the radiological procedure is conducted. The latter is given by the specialized radiologist after the procedure. To create this dataset, radiology reports were collected over a period of one-year from the Cincinnati Childrens Hospital Medical Centers (CCHMC) Department of Radiology. This resulted in 20,275 documents. Data was then subjected to anonymization techniques, followed by manual inspection and removal of records that contain protected health information to meet the United States HIPAA standards. Since some annotations can be subjective, the authors employed a majority voting strategy using the annotations of the CCHMC staff and two independent coding companies. Whenever the three entities produced disjoint label sets, the corresponding medical record was removed. As a result of the above procedures, only 978 documents remained.

- Slashdot

  The target of this database is to automatically tag articles submitted to a web blog known as Slashdot[2]. As introduced by [11], each article's text was converted to a bag of words feature vector using the *StringToWordVector* function from the Weka machine learning software package [31]. There are 22 tags (labels) in this dataset. Examples of these tags include topics such as `Entertainment`, `News`, `Science`, `Mobile`, and `Games`.

- Enron

  During the Enron Corporation scandal investigation in 2001, the Federal Energy Regulatory Commission (FERC) posted the company's email library on the

[2]http://slashdot.org

Coarse Genre	Forwarded Information	Primary Topics	Emotional Tone
Company Business	Government report(s)	Regulations	Humor
Purely Personal	News Articles	Internal Projects	Admiration
Employment	Press Releases	Legal Advise	Sarcasm
Document Editing	Business Letters	Trip Reports	Worry/Anxiety

Table 2.3: Enron Sample Topics

web and made it public. Soon after, research groups leveraged this library to improve classification of emails into users' folders [32]. The raw corpus consists of 619,446 messages which belonged to 158 users. However, a cleaner version of the database was later made available[3] by a joint effort between Massachusetts Institute of Technology (MIT), Carnegie Mellon University (CMU), Stanford Research Institute (SRI), and University of California Berkeley. Cleaning the dataset involved removing personal emails and fixing integrity problems that existed in the original data. The 1702 emails were labeled by students of the Applied Natural Language Processing (ANLP) course in UC Berkeley. Each message was labeled by two students. However, the web source does not claim consistency, comprehensiveness, nor generality about the provided labelings. There are a total of 53 possible classes that are placed into four groups. Table 2.3 shows a sample of 16 topics and their corresponding groups.

- Langlog

  The same authors [11] of the *slashdot* dataset mentioned above also mined blog posts from a website that is about languages[4]. Here, the classification task is concerned with tagging each post with the correct topic(s). This dataset contains 75 tags which include topics such as `Humor, Punctuation,`

[3]The dataset is available at `http://bailando.sims.berkeley.edu/enron_email.html`
[4]http://languagelog.ldc.upenn.edu/nll/

`Language_and_culture, Semantics, and Idioms`. To describe each post, textual content was converted into a bag-of-words (boolean) feature vector using the *StringToWordVector* function available in the Weka machine learning software package [31].

- Mediamill

  The American National Institute of Standards and Technology (NIST) initiated the TRECVID Video Retrieval Evaluation in an effort to push research in the areas of automatic segmentation, indexing, and content-based retrieval of digital video. In the year 2006, the authors of [20] provided a processed version of the 85 hours video content from the TRECVID data of the previous year. Examples in this dataset are composed of key-frames taken from the corpus videos. The authors provide a 101 concept lexicon which were chosen at random from the videos. Concepts such as `Tree, Desert, Tony Blair, Smoke, Prisoner, Waterfall,` and `Football` were among the concepts they considered. The chosen features were taken from the visual content by choosing color-invariant texture features from specific regions within each key frame. Then, the obtained features were translated into similarity scores of 15 low-level concepts such as `Road, Waterbody`, and `Sky`. The authors then varied the regions and concatenated the similarity scores producing a total of 120 numerical features.

- Bibtex

  The authors in [33] provided this dataset to target the automatic tagging of Bookmarks and Bibtex entries in the Bibsonomy[5] social bookmarking website. The website enables users to store, share, and organize (using tags) Bibtex entries as well as website bookmarks. However, this dataset is only concerned with the problem of tagging Bibtex entries. The desired classification system for this dataset should recommend in an efficient manner a set of relevant tags when a user submits a new item to the Bibsonomy web page. The authors kept only the `journal, booktitle, bibtexAbstract,` and `title` fields of the bibtex entries. Textual content was converted to binary bag-of-words features. To reduce the dataset, the authors considered only tags assigned to at least 50 Bibtex entries and words appearing in a minimum of 100 entries. Examples of the remaining tags include `Programming, Physics, Science, Imaging, Language,` and `Architecture`.

- Nus-wide

  Image sharing websites such as Flickr[6] allow users to tag their images using keywords to keep them organized. Taking advantage of Flickr's publicly accessible database, the National University of Singapore affiliated authors [21] randomly collected a sample of $300,000$ images along with their $425,059$ unique user assigned tags. Among all tags, the authors selected only 81 concept based on the following: (i) they appear frequently in the database.(ii) correspond to general and specific categories and (iii) are consistent with previous works in

[5]http://www.bibsonomy.org
[6]http://www.flickr.com

Activities	Program	Scene	People	Objects	Graphics
Swimming	Sports	Airport	Police	Animal	Map
Earthquake	-	Temple	Military	Flags	-
Surfing	-	Ocean	Person	Boats	-
Wedding	-	Grass	Person	Tiger	-

Table 2.4: Nus-wide Sample Topics

the literature. A sample of the selected tags are presented in Table 2.4 along with their top categories. The elimination process left $269,648$ examples. To reduce the tagging errors (noise), the images were given to high-school and University of Singapore students to verify present tags and complete missing ones. In terms of features, the paper represented the images using a vector composed of a color histogram (64 features), color correlogram (144 features), edge direction histogram (73 features), wavelet texture (128 features), color moments selected from a 5×5 grid (255 features), and bag-of-words SIFT descriptions (500 features). Later, these features were deemed unnecessary and prohibitive for large scale image retrieval problems by [22]. Instead, the authors extracted features using the VLAD (Vector of Locally Aggregated Descriptors) feature representation and then applied PCA plus whitening to reduce the vector size to 128 features. Their experiments indicate an improvement in classification accuracy as well as efficiency.

- IMDb

  The goal of this data set is to assign movie genres to films using only the text content of their synopsis. Data was collected by [11] from the famous Interna-

tional Movie Database website[7]. There are 28 movie genres in the database. Examples of these genres include `Thriller, Action, Drama, Comedy, Horror`, and `Sci-Fi`. The movie synopsis text was modeled using the bag-of-words method with the help of the *StringToWordVector* function in the Weka machine learning software package [31]. The repository contains a total of $120,919$ movies.

- Rcv1

  Reuters, one of the biggest international news agency, made available a corpus of newswire articles collected over a year between 2005 and 2006. At that time, the newspaper agency was generating $11,000$ articles per day. To facilitate the retrieval of articles, the agency imposed a hierarchical code book to be used by editors for labeling each news article. The codebook contains three main categories: *Topic*, *Industry*, and *Region*. Coders (i.e. editors) were asked to assign the codes such that each article must contain at least one Topic code and one Region code. Also, they were required to choose codes that were most specific in the hierarchy, such that parents of the code are automatically assigned. However, the authors in [34] found the corpus to have violations and errors to the coding scheme. Consequently, they proposed a revised version in which they fixed the observed errors. The dataset listed in Table 2.2 is a subset of the revised version which contains 6000 documents that are labeled with only the *Topic* category of codes (101 classes). This subset (subset 1) was chosen because it frequently appears in the literature. Among the topic codes are

[7]http://www.imdb.com

`Religion, Sports, Elections`, and `Weather`. Feature representation in this corpus is done by concatenating the headline of the article with the main text and then doing the following: (a) reducing the letters to lower case, (b) applying tokenization, (c) punctuation removal and stemming, (d) term weighting, (e) feature selection, and (f) length normalization.

- Tmc2007

  Aviation safety reports in free text format were provided as part of the SIAM Text Mining Competition during the year 2007. These reports document problems that occurred in certain flights for the purpose of being able to identify recurring anomalies. The classification system is required to label the documents according to the types of problems they describe. Feature representation of the free text reports follow the boolean bag-of-words model (500 features). There are $285,596$ reports and 22 possible problem categories. According to [35], the defined categories include `Alignment, Contamination`, and `Design`.

CHAPTER 3

Related Work

The previous chapter defined the problem of multi-label classification, along with the means to evaluate the classifier which target this field of machine learning. Building on this knowledge, this chapter provides a survey of what has already been done to target the multi-label classification field.

Multi-label classification extends the traditional multi-class problem such that examples may belong to multiple classes at the same time. As surveyed by the authors of [10], and more recently [36], the approaches targeting the multi-label problem are divided into two main categories: *algorithm adaptation* and *problem transformation* methods. The approaches from the first group *adapt* single-label algorithms to make them return multi-label outputs. In the other category, the multi-label problem is *transformed* into a single-label domain where a traditional (binary or multi-class) classifier is then employed. This chapter will review solutions from both categories, and then focus on the family of approaches that are closest to this dissertation's contribution.

3.1 Algorithm Adaptation Methods

The idea of *adapting* a traditional classification algorithm to produce multi-label outputs was perhaps one of the earliest in the multi-label field. In general, solutions of this type of approaches pertain to specific types of classifiers. It is thus sensible to give introductory knowledge on the types of classifiers before discussing their multi-label versions. This will be the followed scheme in this section.

3.1.1 Boosting

The authors of [37] discovered that after inducing a classifier, it is possible to improve its accuracy by inducing an additional classifier which focuses on its incorrectly classified examples. The original algorithm was called *AdaBoost* and the underlying idea became widely known today as *boosting*. *AdaBoost* was implemented per the following iterative steps:

1. Assign training set examples equal numerical weights.

2. Choose a sample $\mathcal{S}$ of the training set according to the assigned example weights. Thus examples marked with higher weights are more likely to be selected.

3. Using the sample $\mathcal{S}$, induce a so called 'weak learner' $h(\mathbf{x})$. The original algorithm's weak learner function predicted the output class of an example based on one of its attributes.

4. Test the weak learner from the previous step by applying it to the entire training set.

5. Increase the weights of examples that were incorrectly classified in the previous step, and decrease the weights of others. Increasing the weights of misclassified examples will ensure that they receive more attention in the next iteration.

6. Store the induced weak learner.

7. Repeat steps 2 to 6 until a termination criterion is met, such as a low error rate.

Once learning is concluded, classification of new examples is done by taking a weighted average of the predictions of all induced 'weak' learners. The weights in this situation can be assigned based on the individual recorded accuracy of each weak learner.

The boosting idea lead to the invention of one of the earliest approaches in multi-label classification. Specifically, the authors of *AdaBoost* extended their work in [38] to handle multi-label outputs in a new algorithm which they called *BoosTexter* [15]. Their intention was to target the problem of text categorization, hence the name. *BoosTexter* handles multi-label domains by assigning weights to each document-class pair. Thus, the total number of weights is $N \times L$, where N and L are the numbers of examples and classes respectively. Two versions were created for the multi-label domains, *AdaBoost.MH* and *AdaBoost.MR*. The first version considers the `hamming-loss` metric (see Section 2.3) to account for classification errors. In the latter version, they consider instead `ranking-loss` which is based on the *order* of relevancy of predicted classes for each example.

Boostexter was criticized by [1] for its rather simple decision function at its core (i.e. the weak learner). Moreover, [4] found that the algorithm performed poorly

in complex domains. Finally, classifier induction was computationally prohibitive in domains with many labels and attributes—see the discussion in [39] and [40].

In [41], the authors modified *AdaBoost.MH* such that multiple decision criteria are selected in each boosting iteration (one per class). Also, they added to the features of examples the outputs of all the weak learners of previous iterations. The authors argue that by taking into account the outputs of previous weak learners (one per class), they essentially incorporate class correlations into the learning system.

3.1.2 Decision Trees

A decision tree learner belongs to the *divide-and-conquer* classification methods. As such, training data is recursively divided into partitions based on features that describe examples. The ultimate goal is to create final data partitions that are each represented predominantly by a single concept.

Figure 3.1 illustrates an example decision tree structure. Here, the learned concept is whether the insurance price for a given driver and car pair is going to be `High` or `Low`. To find the insurance price for a car that is new to the database, classification is done by comparing the features of the car to the decision nodes' criteria (i.e. oval shaped) starting from the root node. Eventually, a leaf node is reached where no further paths can be followed. Determining the output class (i.e. price) is then achieved based on the distribution of the training examples that are represented in this leaf node; the dominant class is typically chosen as the final outcome.

In order to build an efficient decision tree, we would like to minimize classification error using as few decision nodes as possible. One reasonable consideration is to choose a splitting criterion that will create the most homogeneous data partitions

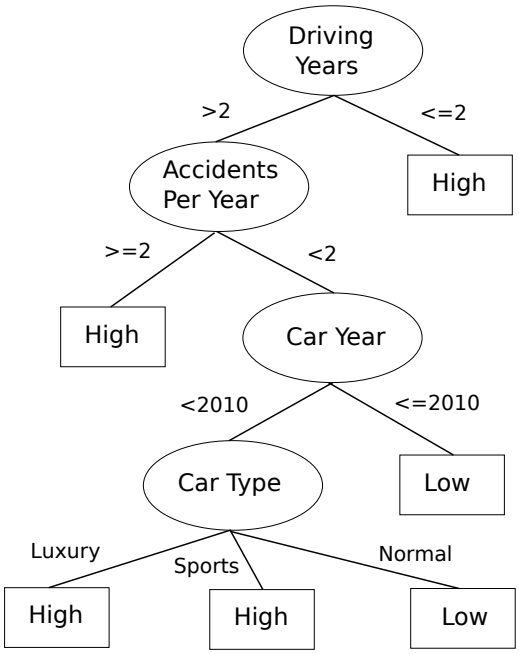


Figure 3.1: A decision tree example. The objective is to determine the price of insuring a given vehicle and driver. Decision nodes are represented using ovals, and boxes represent leaf nodes.

after the split; certain classes will be more dominant than others. This was the motivation behind a famous decision tree algorithm known as *C4.5* [42], which is an improved version of the author's previous algorithm [43]. To choose a suitable splitting criterion, this technique measures the decrease in Entropy (randomness) when a feature value is assumed to be known. Entropy is defined as in Equation 3.1. Here, for a given partition $\mathcal{S}$, the portion of positive examples for class l_i are denoted using P_i^+.

$$\mathtt{H}(\mathcal{S}) = \sum_{\mathtt{i}=1}^{\mathtt{L}} -\mathtt{P}^+_{\mathtt{l_i}} \log(\mathtt{P}^+_{\mathtt{l_i}}) \tag{3.1}$$

In order to choose the best split, the *C4.5* algorithm iterates over all features finding the one that will maximize the *difference* in entropy before and after the split. This is known as *Information Gain*, and it is calculated as follows: Suppose a split using

attribute z generates T partitions. The information gained from using this feature is calculated as in Equation 3.2. Note that $P(\mathcal{S}_i)$ denotes the portion of examples that are represented in the partition $\mathcal{S}_i$ with respect to the original partition $\mathcal{S}$ (before the split).

$$\mathtt{IG}(\mathcal{S}, \mathtt{z}) = \mathtt{H}(\mathcal{S}) - \sum_{\mathtt{i}=1}^{\mathtt{T}} \mathtt{P}(\mathcal{S}_\mathtt{i})\mathtt{H}(\mathcal{S}_\mathtt{i}) \tag{3.2}$$

One issue inherit to using information gain as described above is that it prefers features that have several values even if each split generates a partition that has one example. In the Car Insurance scenario, if we also consider the license plate number of a given car as a feature, the algorithm would ultimately choose this feature yielding a partition for every car in the training set. This happens because a high information gain will be observed in each partition; they will each contain a single car, and therefore a single class. Obviously, using the license plate number is not suitable to determine the insurance price for future cars. To overcome this drawback, it was suggested to use Information Gain Ratio instead [44], which takes into account the number of examples in each generated partition as follows:

$$\mathtt{IGR}(\mathtt{S}, \mathtt{z}) = \mathtt{H}(\mathcal{S}) - \frac{\sum_{\mathtt{i}=1}^{\mathtt{T}} \mathtt{P}(\mathcal{S}_\mathtt{i})\mathtt{H}(\mathcal{S}_\mathtt{i})}{\sum_{\mathtt{i}=1}^{\mathtt{T}} -\mathtt{P}(\mathcal{S}_\mathtt{i}) \log \mathtt{P}(\mathcal{S}_\mathtt{i})} \tag{3.3}$$

The denominator in the fraction part of the equation is known as the Intrinsic Value of the split. After some thought, the reader can see that this method will no longer favor nominal features that have a high number of possible values. Note that continuous variables do not fall under this category; they are typically separated by imposing a threshold value which generates a binary split [44] as illustrated by Figure 3.1 (e.g. 'Accidents Per Year').

In addition to classifying unseen examples, decision trees are able to associate a confidence value with each class prediction by looking at the distribution of training examples that fall into each leaf node. To illustrate how confidence values are calculated, suppose a test example is classified into a leaf node in which 4 positive and 2 negative training examples were observed. Here, the classifier will label the test example with the positive class; since positive examples constitute the major portion in that leaf node. Additionally, the classifier can associate a prediction confidence value of $\frac{2}{3}$ based on the distribution of this leaf's training examples ($\frac{4}{(2+4)}$).

While the estimated probabilities are expected to be accurate when there is a substantial number of examples in each leaf node, this will not be the case when the number of examples is small. According to [45], basing the prediction confidence on a small sample size (especially when the dataset is noisy) can provide misleading probability estimates. To overcome this small-sample estimation hurdle, a technique known as *Laplace* smoothing is typically applied [46]. Calculating the *Laplace* confidence values is done as in Equation 3.4 below. Here, N denotes the number of training examples observed in the leaf node, and N_p refers to the quantity of these examples that belong to the positive class. When applied to the previously mentioned example, the smoothed confidence value will now become $\frac{(4+1)}{(6+2)} = \frac{5}{8}$. As will be seen in the next chapter, this function can be modified to better fit special class distributions, such as that of the imbalanced classes representation mentioned in Section 1.3.

$$P(\oplus) = \frac{N_p + 1}{N + 2} \tag{3.4}$$

Another benefit that is gained when using decision trees is their ability to be interpreted as human readable rules. For instance, in [5] the authors were interested

in interpreting what factors (features) are responsible for determining a class label more than achieving complete classification. Intuitively, each path (root to leaf node) in the tree can be interpreted as a rule which consists of predicate conjunctions. By looking at the previous example in Figure 3.1, the following rules can be inferred:

- if (`DrivingYears` < 2) then `InsurancePrice` is `High`.
- if (`DrivingYears` $>= 2$) and (`AccidentsPerYear` $>= 2$), then InsurancePrice is `High`.

Let us now discuss the computation complexity of decision trees. The following derivation will closely resemble that discussed in [47]. In order to simplify the derivation, a few assumptions have to be made. First, the tree is assumed to have have a depth in the order of $O(\log(N))$ where N is the number of training instances. This assumption follows from the desired goal of having a "bushy" decision tree such that it does not have any long individual branches. It is also assumed that examples are going to be different from each other (no duplicate examples), and are separable by the given number of example-attributes D.

Based on the assumptions from the previous paragraph, the induction complexity of a decision tree is in the order of $O(DN\log(N))$, where D and N are the number of attributes and examples respectively. To elaborate, each node requires D iterations (one per attribute) and each test will be carried out (at the worst case) over all instances, N. This process is carried out at every depth level ($\log(N)$). Note that when the attributes are nominal, they are not going to be used more than once in a given tree-branch since an example can only have one value of the nominal attribute. However, when the attributes are numeric, they are reusable since multiple binary

splits can be formed from the same numeric attribute. The above derivation assumes the worst case scenario in which all attributes as numeric. To classify previously unseen examples, a test operation must be made for every node along a given path in the tree. Thus, classification complexity is in the order of $O(\log(N))$.

Decision trees were designed to handle classes that are mutually exclusive; a restriction that does not apply in multi-label domains. Thus adapting decision trees to produce multi-label outputs was the target of the study in [5]. The authors modified the well-known *C4.5* algorithm by extending the entropy function to handle multiple labels as demonstrated by Equation 3.5 below. Unlike the previous definition (Equation 3.1), for every class, negative examples (or examples that may *also* belong to other classes) are counted. Therefore, an example that belongs to multiple classes is counted more than once. In addition to modifying the entropy equation, they also had to allow the leaves of the tree to be associated with *subsets* of classes rather than individual ones.

$$\mathtt{H}(\mathcal{S}) = \sum_{\mathtt{i}=1}^{\mathtt{L}} -\mathtt{P}^{+}_{\mathtt{l_i}} \log(\mathtt{P}^{+}_{\mathtt{l_i}}) - \mathtt{P}^{-}_{\mathtt{l_i}} \log(\mathtt{P}^{-}_{\mathtt{l_i}}) \tag{3.5}$$

Another multi-label algorithm that relies on the concept of decision trees is that by the authors of [48]. Motivated by the *Alternating Decision Tree* algorithm [49], the authors implemented a multi-label version where the induction of the decision tree is done using the *AdaBoost.MH* [38] algorithm from Section 3.1.1. As a gained advantage of using their method over *AdaBoost.MH*, the authors argue that their approach can produce human readable rules since they construct decision trees.

3.1.3 Neural Networks

The collaborative work between a logician and a neurophysiologist lead to the invention of the Neural Networks classifier [50]. As indicated by its name, the goal of this induction method is to simulate the biological brain's neural activity in an attempt to automatically recognize patterns. Neural Networks consist of several interconnected single neurons that are also known as Perceptrons [51]. Each neuron is mathematically represented using a function of several numerical inputs and a bias as illustrated by Figure 3.2. When the output is binary, as in dichotomies, the sign of the weighted sum of inputs is often selected as the function of the neuron:

$$f(\mathbf{x}) = sign(w_0.x_0 + w_1.x_1 + w_2.x_2) \tag{3.6}$$

In essence, when the the product of the vectors and weights is $\mathbf{w}.\mathbf{x} > 0$ the predicted class is *positive*. Otherwise, a *negative* class is predicted.

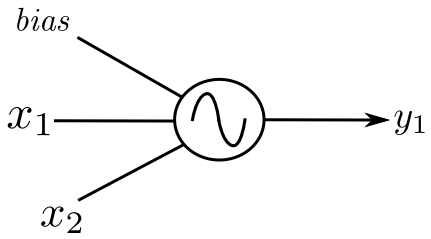


Figure 3.2: An example neuron with two inputs and a single output.

After initializing the internal weights of a perceptron with arbitrary values, training the perceptron is done by presenting it with each example from the training set and observing any errors that appear at the outputs. The errors are then used to update the input weights such that it produces an output closer to that of the expected one. This process, called an "epoch", is repeated until a certain criterion is met,

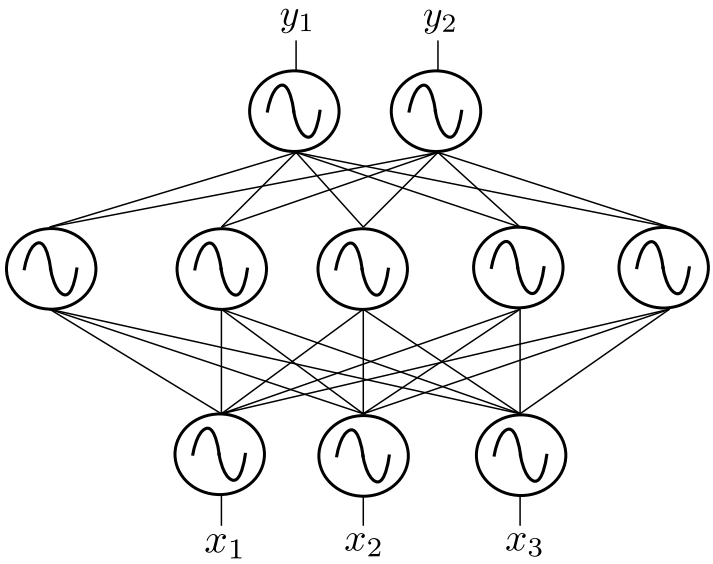


Figure 3.3: An example neural network with 3 neuron layers. The middle (hidden) layer consists of 5 neurons. There are 3 input neurons and 2 output neurons respectively.

such as observing a low error rate. The original update method [51] used the sign of the predicted output to calculate the output error. Later, an alternative method was introduced [52] which instead uses the product of the weights and inputs directly; without using the sign function. The latter approach is known as the *Widrow-Hoff* algorithm.

It was found that perceptrons are suitable for linearly separable problems. However, when problems are more complex (i.e. non-linear), a *network* of neurons becomes necessary; the outputs of some neurons are used as inputs in others. An example of such network is presented in Figure 3.3. Unlike the single-neuron case, neural networks can produce multiple outputs and even contain *hidden* layers of neurons that are located between the input and output layers. Having many neurons requires additional weight variables to realize the inter-neuron connections, which means additional memory and computation requirements. Perhaps one of the most famous implemen-

tations of these methods is the Back-Propagation of errors method from [53]. Showing great success, their approach updates the weights of a multi-layered neural network based on the contributions of each neuron to the error.

Although Neural Networks can handle the multi-label classification problem by implementing a group of outputs at the output layer, it was found that modifying the error function to account for the errors between class *pairs* produced promising results in multi-label domains [4]. In their paper, the authors implemented two versions of neural networks. The first of which uses the regular error function, *BASICBP*. The other variant uses the difference between class pairs as the error function, which they called *BP-MLL*. Their paper shows that *BP-MLL* outperforms *BASICBP* over a broad range of metrics. However, the authors admit that training *BP-MLL* is computationally expensive due to its complex error function. A similar idea was proposed by the authors of [18], where a perceptron is trained for each pair of classes. Their intention was to decouple the optimization process from any specific ranking criteria.

3.1.4 Support Vector Machines

Instead of carrying out several "epochs" to realize the optimal input weights as done in the Perceptron from the previous section, an algorithm called *Support Vector Machines (SVM)* was later invented which finds the optimal values using only a subset of the training examples [54, 55]. To illustrate this process, let us refer to Figure 3.4. Here, 2-dimensional examples are represented using circles which are color coded based on their class affinity; positive examples are white colored, negative examples are gray. Instead of considering all examples in the training set, *SVM* focuses only

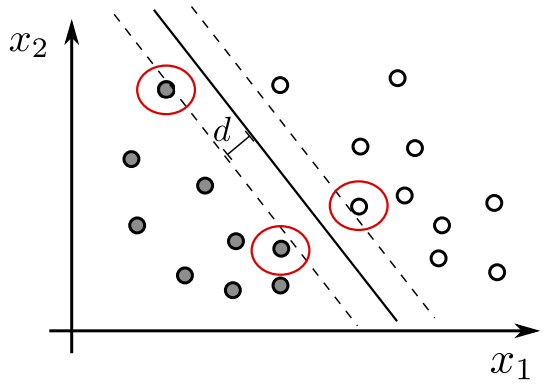


Figure 3.4: A 2-dimensional linearly separable example problem. Positive and negative examples are indicated using white and gray circles respectively. The chosen support vectors are marked with red circles. The marginal distance is denoted using the letter, d.

on examples that neighbor the boundary between the two classes. Intuitively, one can realize a separating hyperplane using only the information from these examples. Marked with red circles in Figure 3.4, these examples are referred to as *support vectors*.

One can see that many orientations of the separating hyperplane can achieve perfect separation of the two depicted classes. The optimization algorithm in *SVM* is driven by the idea that a better generalization can be achieved when the separator is positioned with a maximum distance (d) to the closest support vectors. Thus, future examples are less likely to be affected by the orientation of the separating hyperplane. Note that when a perfect separation is not possible, the optimization method reduces the overall error by incorporating a penalty function for misclassified examples.

To put the *SVM* classifier in mathematical terms, let us suppose the data is comprised of N examples that are each defined by a feature vector $\mathbf{x}$ and class label $y_i = \{-1, 1\}$, $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$. The *SVM* classifier determines the class label of an unseen instance by first calculating the dot product of the features $\mathbf{x}$ and the set of weights $\mathbf{w}$ which define the orientation of the hyperplane. A scalar

term b (known as *bias*) is then added to the dot product which leads to the formula in Equation 3.7.

$$f(\mathbf{x}) = \mathbf{x}.\mathbf{w} + b \tag{3.7}$$

Using the sign of the function's output, a test example is considered positive when $f(\mathbf{x}) > 0$, or negative otherwise.

Finding the optimal weights $\mathbf{w}$ involves solving the following minimization problem:

$$\begin{aligned} \texttt{Minimize} &: \frac{1}{2}\|\mathbf{w}\|^2 \\ \texttt{subject to} &: y(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0, \forall \mathbf{x}_i \in \mathcal{S} \end{aligned} \tag{3.8}$$

Here, minimizing the term $\frac{1}{2}\|\mathbf{w}\|^2$ follows from the width of the margin being $\frac{2}{\|\mathbf{w}\|^2}$. Note that the equality in Equation 3.8 will hold (i.e. $= 0$) when the examples lie exactly at a distance $d = 1$ from the hyperplane. These examples are defined as the *support vectors* which are marked by the red circles in Figure 3.4. Solving the above minimization problem is done with the help of a Quadratic Programming (QP) solver. However, as pointed out by [56], when datasets are large, many off-the-shelf solvers are not practically feasible; since the programs must handle a matrix that has a number of elements equal to the square of the number of training examples. Thus, the same author suggested a method which breaks the problem into smaller subsets. In turn, this reduces the complexity to that between linear and quadratic time in the number of examples. This technique is known as Sequential Minimization Optimization (SMO) [56], and it is currently being used by recent versions of machine learning software tools such as WEKA [31] and LIBSVM [96].

To handle non-linearly separable problems, *SVM* can be extended using what are known as kernel functions, $\Phi(\mathbf{x})$. The process involves mapping the examples' features

into a high dimensional feature space in order to make them easier to separate. Thus the training data will now take the form, $\mathcal{S} = \{(\Phi(\mathbf{x}_1), y_1), (\Phi(\mathbf{x}_2), y_2), ..., (\Phi(\mathbf{x}_n), y_n)\}$. Typical kernel functions used in tandem with *SVM* include Polynomial kernels, Gaussian Radial Basis Functions, and Sigmoids.

SVM is designed to separate dichotomous datasets. In order to handle multi-label domains, a separate *SVM* is typically induced for each class in a one-vs-rest scheme: for each class, examples that belong to the target class are labeled as positive examples and all other examples are regarded as negative instances. In typical multi-label domains, this makes each binary training set imbalanced; negative examples (which may belong to any of the other classes) will outnumber positive ones. Since the target of *SVM*'s optimization problem is to reduce the overall error, it does not differentiate between the positive and negative examples in finding the optimal separating hyperplane. Consequently, when classes are imbalanced, the boundary tends to be closer to positive examples causing many of them to be falsely classified as negative. To balance the predictions of negative and positive examples in *SVM*, the boundary can be shifted towards the negative examples after induction. This was the motivation behind the work in [7].

Inducing a separate *SVM* classifier per class has another associated problem; it assumes that classes are independent of each others. This may not necessarily be the case in multi-label domains as discussed in Section 2.2. To overcome this problem, the authors of [1,27] created an algorithm called *RankSVM* which considers the difference between the hyperplanes of each class in the optimization problem. They argue that class co-occurrence patterns may affect the final hyperplanes' orientations. Their

approach yielded improvements in classification accuracy and outperformed the once famous boosting algorithm *AdaBoost.MH* (see Section 3.1.1).

3.1.5 K-Nearest-Neighbors

In contrast to the previous classifier types, the *K-Nearest-Neighbors (KNN)* algorithm [57, 58] does not undergo a learning process prior to testing new examples. Instead, this classifier delays the learning process until it is presented with a test example [59]. To classify a new example , the algorithm first finds the K training examples that are most similar in the feature-space to the test example. Quantifying the similarity between examples is done using a distance measure such as the *euclidean distance*. After identifying the K closest examples, called *Nearest Neighbors*, the classifier chooses the output label to be that which applies to most of the K selected examples.

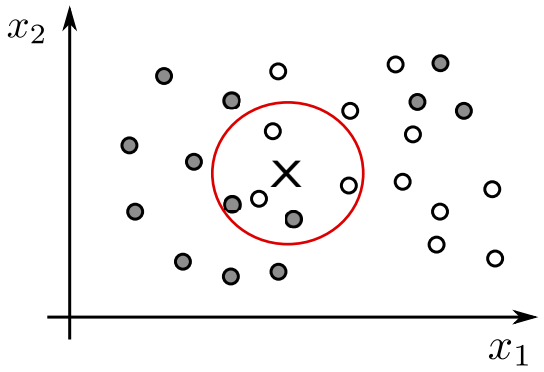


Figure 3.5: A 2-dimensional *K-Nearest-Neighbor* classification problem. Here, positive and negative examples are indicated using white and gray circles respectively. The test example is marked with a cross-mark (X). The closest $K = 5$ training examples are identified by the red circle.

To illustrate this process, let us refer to Figure 3.5. Here, examples are defined using a two-dimensional feature vector $\mathbf{x} = (x_1, x_2)$. Suppose we are tasked to predict

the class of the example identified with a cross-mark (X). If we consider the closest 5 examples (i.e. $K = 5$), then the chosen nearest neighbors are those enclosed by the red circle. As can be seen in the figure, the majority of the enclosed examples are negative instances. Thus, the chosen classification for the test example is negative as well. It is imperative to choose an odd number of nearest neighbors to guarantee a valid voting outcome; where no ties are possible.

One advantage in using the *KNN* classifier is that it can be directly applied to non-linearly separable problems as well. As can be seen in Figure 3.5, if a test example is located in the top right corner in this 2-dimensional field, it will be likely be classified correctly as positive. The same thing cannot be said about a linear classifier such as that previously shown in Figure 3.4 from the previous section.

The drawback of using the *KNN* classifier lies in its classification time. Since each test example must first be compared to the training set examples, the process is extremely sensitive to the dimension and size of the training set. As a result, the algorithm's computational time can be prohibitive in large domains. There are known methods to speed up the look-up of the nearest neighbors such as the *KD-Tree* approach from [60].

Applying the *KNN* classifier to multi-label problems was first done by the authors of [61], which they called *ML-KNN* for *Multi-label K-Nearest-Neighbors*. Their algorithm produces multi-label predictions by applying the same voting scheme described above to each class separately. Although they show that their algorithm outperformed the *SVM* based *RankSVM* algorithm (see Section 3.1.4) and the boosting based *AdaBoost.MH* (see Section 3.1.1), they do not however provide the computational time results.

To end this section, it is important to note that most algorithm adaptation attempts were tailored for specific domains such as biology [5] and text categorization [3]. Moreover, these methods are generally restricted to specific types of classifiers such as *SVM*, Decision Trees and Neural Networks. The next category of methods will be applicable to a much broader variety of classifiers.

3.2 Problem Transformation Methods

Instead of *adapting* the algorithms to make them return multi-label outputs, this category of methods *transform the data* to a single-label domain that traditional classifiers can handle. To obtain a better understanding of these methods, let us use the sample data set from Table 3.1. Note that there are 8 examples(rows) and 4 classes (columns). When a class is relevant for an example, it is indicated using the cross-mark, X. Note that some classes may overlap over the same example, as in the case of Example 2 which belongs to classes A and B. Let us now discuss the transformation methods.

Classes	A	B	C	D
Ex. 1	X	-	-	-
Ex. 2	X	X	-	-
Ex. 3	-	X	X	X
Ex. 4	-	-	-	X
Ex. 5	X	X	-	-
Ex. 6	X	-	X	-
Ex. 7	-	-	X	-
Ex. 8	-	X	X	-

Table 3.1: A Sample Data Set with $L = 4$

	Class A
Ex. 1	X
Ex. 2	X
Ex. 3	-
Ex. 4	-
Ex. 5	X
Ex. 6	X
Ex. 7	-
Ex. 8	-

	Class B
Ex. 1	-
Ex. 2	X
Ex. 3	X
Ex. 4	-
Ex. 5	X
Ex. 6	-
Ex. 7	-
Ex. 8	X

	Class C
Ex. 1	-
Ex. 2	-
Ex. 3	X
Ex. 4	-
Ex. 5	-
Ex. 6	X
Ex. 7	X
Ex. 8	X

	Class D
Ex. 1	-
Ex. 2	-
Ex. 3	X
Ex. 4	X
Ex. 5	-
Ex. 6	-
Ex. 7	-
Ex. 8	-

Table 3.2: Binary Relevance Transformation

3.2.1 Binary Relevance Transformation

The most intuitive approach to transform multi-label problems, which is borrowed from multi-class domains [62], is to target each class *independently*. This is done by dividing the data into multiple single-label (binary) datasets as in Table 3.2. On account of the problem being now represented by multiple dichotomies, a binary classifier is thus induced for each class. The class vector of a new example can then be obtained by aggregating the outputs of all binary classifiers. This technique is known in the literature as the *Binary Relevance (BR)* method. Figure 3.6 illustrates the overall structure of this method.

The *BR* approach has been shown to be very competitive with other multi-label approaches [11]. Despite its successes [6], however, *BR* does not account for class correlations because it targets each class independently. Arguably, a `beach` scene may be strongly correlated with `water`. The same fact cannot be said about `beach` and `mall`. This criticism was the source of concern for many studies [1,8,11,12,17,63–65]. *BR* is also known to suffer from an imbalanced representation of classes in the binary

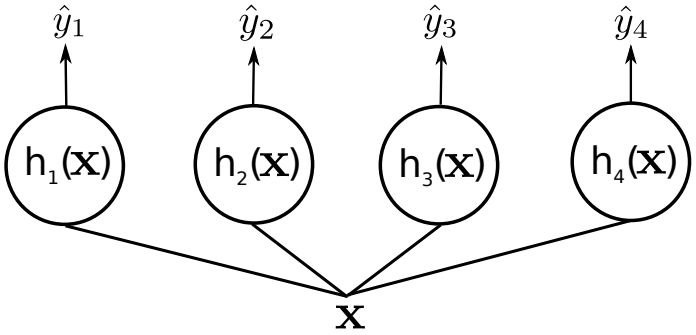


Figure 3.6: An example structure of the *BR* method with $L = 4$. Here, $h_j(\mathbf{x})$ is the binary classifier assigned to the jth class. Each example $\mathbf{x}$ is passed to the binary classifiers in parallel.

training sets. For example, in Table 3.2, one can see that class D has only 2 positive examples out of 8. This issue was discussed at great length in [7] and [66].

3.2.2 Pairwise Transformation

	ClassPair (A,B)
Ex. 1	X
Ex. 3	-
Ex. 6	X
Ex. 8	-

	ClassPair (A,C)
Ex. 1	X
Ex. 2	X
Ex. 3	-
Ex. 5	X
Ex. 7	-
Ex. 8	-

	ClassPair (A,D)
Ex. 1	X
Ex. 2	X
Ex. 3	-
Ex. 4	-
Ex. 5	X
Ex. 6	X

	ClassPair (B,C)
Ex. 2	X
Ex. 5	X
Ex. 6	-
Ex. 7	-

	ClassPair (B,D)
Ex. 2	X
Ex. 4	-
Ex. 5	X
Ex. 8	X

	ClassPair (C,D)
Ex. 4	-
Ex. 6	X
Ex. 7	X
Ex. 8	X

Table 3.3: Pairwise Transformation

Instead of targeting each class independently, the second transformation method creates a dichotomy for each class *pair*. The goal here is to capture pairwise dependencies between classes. Hence, the technique is known as *Pairwise Transformation*

(PW). To train the classifier assigned to each class pair, the approach considers only examples that belong either to the first or to the second class in the pair, but not both. This creates as many as $L(L-1)/2$ binary subsets, where L is the number of classes. Applying this transformation method to our sample dataset is illustrated in Table 3.3. A few remarks can be made: first, not all examples are present in each subset. Secondly, when a pair of classes co-occur frequently, or if either of the classes are rare, this pair will have very few examples in its subset. Consider for example the class pair (C, D); not only is the pair represented by half the examples, it is also highly imbalanced (only one negative example exists in the subset). Conversely, when members of a class pair are abundant and they do not exhibit a direct co-occurrence pattern, the class pair will be represented by a highly populated subset of examples. This is the case in class pair (A,D).

To predict the labels of previously unseen examples, each pairwise classifier casts a vote for one of its class members. Positive example predictions equate to a vote for the first class in the pair, and negative predictions vote for the second class. Following that, classes are ranked according to the vote of all pairwise classifiers. With the help of a predefined number-of-votes threshold, the relevant classes are then separated from the irrelevant ones. A novel approach from this group of transformation methods is that of [67]. Instead of using a prespecified threshold for all classes, the authors created an artificial category which lies between the relevant and irrelevant classes. This means an additional L number of classifiers are added to the system. These classifiers are then trained to produce a positive label prediction for irrelevant classes and negative prediction otherwise. Although the pairwise approach models 2nd degree dependencies, it is however computationally inefficient in domains with great many

Classes	A	C	D	$A+B$	$A+C$	$B+C$	$B+C+D$
Ex. 1	X	-	-	-	-	-	-
Ex. 2	-	-	-	X	-	-	-
Ex. 3	-	-	-	-	-	-	X
Ex. 4	-	-	X	-	-	-	-
Ex. 5	-	-	-	X	-	-	-
Ex. 6	-	-	-	-	X	-	-
Ex. 7	-	X	-	-	-	-	-
Ex. 1	-	-	-	-	-	X	-

Table 3.4: Label Powerset Transformation

classes due to its quadratic running time with respect to the number of classes ($L(L-1)/2$). There are attempts to overcome this limitation such as those in [68] and [69].

3.2.3 Label Powerset Transformation

An alternative transformation technique converts each class *combination* into a single class. Thus, a maximum of 2^L classes may be present, with L being the number of individual classes in the original dataset. This approach, called the *Label Powerset* method (LP) makes it possible to use a single-label (though multi-class) classifier directly on the converted set. Unlike BR (see section 3.2.1), the LP method explicitly models label correlations giving it an advantage in that regard. However, this conversion method also has its trade-offs.

To see the issues that are associated with LP, let us consider applying it to our sample dataset as done in Table 3.4. The reader can observe the following: first of all, not all combinations exist in the data. For instance, class B never occurs on its own and therefore it no longer exists as a singular class. Consequently, the classifier may *overfit* the training data and will never predict this class alone. Also, the dataset has become remarkably sparse; for each combination, there exist only a few positive

examples. Finally, although not shown in our example, when classes frequently co-occur over the examples, this may lead to an upper bound of 2^L classes which renders this approach impractical for large datasets. This is why some authors attempted to reduce the method's complexity by considering only some label combinations [70], [71]. Unfortunately, these algorithms are still costly in large multi-label domains [11].

In a more recent attempt, a hybrid method was proposed, *LPBR*, that combines *BR* with *LP* [72, 73]. The idea is to partition the label space into dependent and independent groups using a heuristic such as Information Gain which was discussed in Section 3.1.2. After partitioning the classes, the authors apply *LP* to the dependent group, and *BR* to the independent classes. While showing an improvement in terms of classification accuracy, the technique was still computationally very costly.

3.3 Label Dependent Binary Relevance Methods

Thus far, this chapter has reviewed methods which *adapt* single-label algorithms to output multi-label predictions, and *transformation* methods which rather convert the data into a single-label domain. For a better insight of proposed idea in this work, this section will focus on the family of methods that are closest to the proposed solution; the group of methods that are based on the *BR* framework discussed previously in Section 3.2.1. Note that for the brevity of this review, the notations from Section 2.1 will be adopted in this section.

Recall that the *BR* technique targets the classes independently from each other. As was demonstrated by so many studies [1,11,12,17,63–65], class dependencies may prove beneficial for classification purposes. Attempts to rectify the class independence

.	inputs							.
	x_1	x_2	x_3	x_4	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	output
$h_1(\mathbf{x})$	0.23	-0.4	0.55	0.1				1
$h_2(\mathbf{x})$	0.23	-0.4	0.55	0.1	1			0
$h_3(\mathbf{x})$	0.23	-0.4	0.55	0.1	1	0		1
$h_4(\mathbf{x})$	0.23	-0.4	0.55	0.1	1	0	1	1

Table 3.5: Prediction steps of the *Classifier Chains*. Here, the number of classes is $L = 4$. The example's original feature vector is $\mathbf{x} = [0.23, -0.4, 0.55, 0.1]$, and the predicted labels are $\hat{\mathbf{y}} = [1, 0, 1, 1]$.

drawback of *BR* have resulted in the development of state-of-the-art multi-label classification methods, which are discussed next.

3.3.1 Classifier Chains

A "straightforward" approach to incorporate label dependencies into *BR* is the *Classifier Chain (CC)* learning system from [11, 74]. The idea (illustrated by Figure 3.7) is to create a *random* sequence of chained classifiers (one per class) in a such a way that each binary learner accepts the classifications of the previous classifiers in the chain as additional features, $h_z(\mathbf{x}) = h(\mathbf{x}, \hat{y}_1, \hat{y}_2, ..., \hat{y}_{z-1})$. Prediction happens following the steps shown in Table 3.5 (from top to bottom). Note that the order of the classifiers in the chain was kept intact to eliminate any confusion.

A probabilistic interpretation by [12] and [75] show that *CC* approximates in a greedy manner the conditional joint mode of the label distribution, $P(\mathbf{y}|\mathbf{x})$. This conclusion stems from the product rule of probability explained in Section 2.2. The same performance criteria from Section 2.3 were used by the authors of [11] to show that *CC* compares favorably with other recent algorithms. However, *CC* also has some related shortcomings.

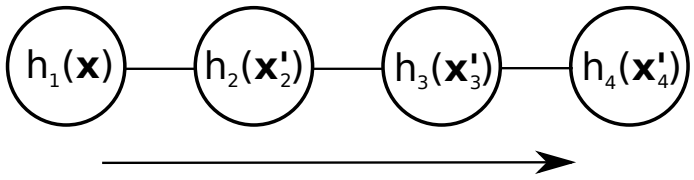


Figure 3.7: A structural representation of the Classifier Chain method with $L = 4$. Here, $h_j(\mathbf{x})$ is the binary classifier assigned to the jth class, and $\mathbf{x}'_\mathbf{z} = \mathbf{x} \cup (\hat{y}_1, \hat{y}_2, ..., \hat{y}_{z-1})$.

The *CC* approach was analyzed by [6, 12, 75–77] and found to underperform in large data sets mainly due to *error-propagation*, and also because its classification depends on the order of the classifiers in the classifier chain. To overcome these problems, the same authors proposed an ensemble version, *Ensemble Classifier Chain ECC*, where they create multiple classifier chains each with a different order of classifiers. Prediction is then carried out by a voting mechanism such as choosing the majority vote for each class. While somewhat mitigating the previous drawbacks, *ECC* incurs additional computational costs because of the additional classifiers. In an effort to show the importance of label ordering, the authors of [78] proposed a Bayesian optimal classifier chain, but the price for their success was that the system was computationally tractable only in data sets with no more than 12-15 labels. In addition to the label-ordering issue, the *CC* algorithm follows the assumption that all classes are related; this makes the system somewhat domain-dependent. Improving *CC* is currently an active research area. In [79], the authors proposed a method which optimizes the selected ensemble of classifier chains according to the *f1* measure from Section 2.3. Another interesting improvement is that introduced in [80] which uses a different sequence of classifiers for each test example.

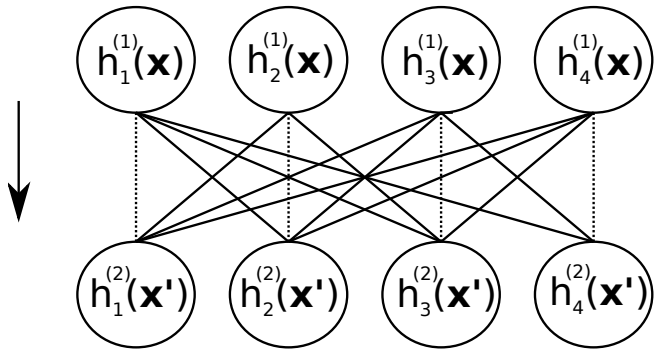


Figure 3.8: A *Stacking* structure example with $L = 4$. Here, $h_j^{(i)}(\mathbf{x})$ denotes the binary classifier for the jth class in the ith layer; $\mathbf{x}$ denotes the regular feature vector; and $\mathbf{x}' = \mathbf{x} \cup \hat{\mathbf{y}}^{(1)}$ is the features-plus-labels vector. The dotted line indicates which features are ignored when training using actual *labels*.

3.3.2 Stacking Classifiers

As discussed in the previous section, the classification performance of *CC* was found to be sensitive to the order of the classes in the classifier chain. One way to deal with the label-ordering problem is to include classifications of all other classes as features to be used by every classifier. Closely resembling the proposed method, this approach is typically implemented by stacking two layers of *BR* classifiers as in Figure 3.8. During classification, *primary* predictions are first obtained using the independent first-layer classifiers as in *BR* (see Section 3.2.1), $\mathbf{h}^{(1)}$. The idea is then to capture label dependence by augmenting the regular feature set, $\mathbf{x}$, with the *primary* predictions, $\hat{\mathbf{y}}^{(1)}$, forming the so-called *meta-examples*, $\mathbf{x}'$. The newly-formed examples are then forwarded to the now-dependent second-layer models to produce *final* classifications. Table 3.6 illustrates the prediction steps as was done for *CC* in the previous section. The reader may note that, the only prediction that was changed in this example is that of class 3 where the original prediction was switched from 1 (relevant) to 0 (irrelevant). This two-step method was first introduced in [9].

STEP 1	inputs				.
	x_1	x_2	x_3	x_4	output $\hat{\mathbf{y}}^{(1)}$
$h_1^{(1)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1
$h_2^{(1)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	0
$h_3^{(1)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1
$h_4^{(1)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1

STEP 2	inputs								.
	x_1	x_2	x_3	x_4	$\hat{y}_1^{(1)}$	$\hat{y}_2^{(1)}$	$\hat{y}_3^{(1)}$	$\hat{y}_4^{(1)}$	output $\hat{\mathbf{y}}^{(2)}$
$h_1^{(2)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	.	0	1	1	1
$h_2^{(2)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1	.	1	1	0
$h_3^{(2)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1	0	.	1	0
$h_4^{(2)}(\mathbf{x})$	0.23	-0.4	0.55	0.1	1	0	1	.	1

Table 3.6: Prediction steps for *Stacking*, The number of labels $L = 4$. The example's original feature vector $\mathbf{x} = [0.23, -0.4, 0.55, 0.1]$, and the primary predictions are $\hat{\mathbf{y}^{(1)}} = [1, 0, 1, 1]$. The secondary (final) predictions are $\hat{\mathbf{y}^{(2)}} = [1, 0, 0, 1]$

Throughout the rest of this dissertation, this technique will be referred to as as *Stacking*.

There are two ways to train the second layer of the *stacking* approach, $\mathbf{h}^{(2)}$. Originally, *meta-examples* are created using (possibly continuous) predictions of the first-layer [9]. This training approach is perhaps motivated by the ability to capture and correct estimation errors of the first layer's classifiers. Nevertheless, using *training* classifications may overfit the data by biasing the *meta-examples*. For instance, in each iteration of the boosting algorithm from Section 3.1.1, a specific classifier is induced to target the selected subset of examples corresponding to that iteration. This induced classifier is *specialized* for that subset, which makes it *biased* towards it. Similarly in *Stacking*, the predictions assigned by the classifiers for the *training* set are not guaranteed to be the same for previously unseen instances. It is possible to

avoid biasing by cross-validation (e.g. [7,81]), the price being increased computational costs.

Alternatively, one can augment the regular feature vector with class *labels* directly obtained from the training set. This method was used recently by [65] who call it *Dependent BR (DBR)*. When creating meta-examples, this method does not include the labels of the target class for each binary classifier to prevent the learner from picking it as the only criterion; since the target class prediction will be equal to the value of that label-feature. This principal is reflected in the example illustrated in Table 3.6, where those label-features are left blank. Additionally, the dotted lines in Figure 3.8 show which label-features are excluded in this mode of training.

3.3.3 Computation Complexity

Let us now consider how much computational complexity the *CC* and *Stacking* approaches add to *BR*. In this analysis, the terms D, N, and L will refer to the number of features, examples, and classes respectively. Since the *CC* and *Stacking* approaches rely on a base (binary) learning algorithm at their core, the terms $\mathcal{F}_{\mathcal{B}}(N, D)$ and $\mathcal{F}'_{\mathcal{B}}(D)$ will refer to the training and testing complexities of the employed base learning algorithm. For instance, when the decision tree classifier is used as the base learner, the term $\mathcal{F}_{\mathcal{B}}(N, D)$ will be refer to $O(DN\log(N))$ during training and $\mathcal{F}'_{\mathcal{B}}(D)$ will refer to $O(\log(N))$ during testing (see Section 3.1.2). Note that this terminology follows that recently used by [36].

The *BR* method constructs one classifier for each category in the dataset. As such, the number of classifiers induced is going to be L. Furthermore, each classifier will be induced on the same number of features D, which yields a training complexity

	BR	*CC*	*Stacking*
Training	$O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D))$	$O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D+L))$	$O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D+L))$
Testing	$O(L \cdot \mathcal{F}'_{\mathcal{B}}(D))$	$O(L \cdot \mathcal{F}'_{\mathcal{B}}(D+L))$	$O(L \cdot \mathcal{F}'_{\mathcal{B}}(D+L))$

Table 3.7: Computational complexities of *BR*, *CC*, and *Stacking*. N, D, and L represent the number of examples, features, and classes respectively. $\mathcal{F}_{\mathcal{B}}$ and $\mathcal{F}'_{\mathcal{B}}$ refer to the induction and testing complexities of the employed base classifier.

in the order of $O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D))$. The *CC* approach is equivalent to *BR* in that the total number of induced classifiers is equal to the number of labels, L. However, each classifier will take into consideration the class labels of the previous classifiers in the chain for a total of $D+L$ features. Thus, the training complexity of *CC* is in the order of $O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D+L))$. In the ensemble version *(ECC)*, for an ensemble of Q classifier chains, the number of induced classifiers is increased to $Q \times L$. Thus, the complexity becomes $O(Q \cdot L \cdot \mathcal{F}_{\mathcal{B}}(N, D+L))$. Due to the two-layer formation of Stacking, a total of $2 \times L$ classifiers are induced. Also, the classifiers of the second layer will take the input of the first layer as additional features. This makes the complexity of stacking in the order of $O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D+L))$.

During training, when *meta-examples* are constructed using the class *labels* (directly from the training set), the *Stacking* and *CC* methods can both be parallelized since the information is available before hand. The same cannot be said when class *predictions* are used to train the dependent classifiers. Here, each classifier needs to be induced in sequence for the *CC* method. In *stacking*, induction of the first-layer can be parallelized, and then the second-layer of classifiers is constructed in the same way.

When testing previously unseen examples, the *BR* method will have a complexity that is $O(L \cdot \mathcal{F}'_{\mathcal{B}}(D))$. Since the *CC* and *Stacking* approaches involve additional class-

features, their testing complexities will be in the order of $O(L \cdot \mathcal{F}'_{\mathcal{B}}(D+L))$. However, in a parallel execution environment, the advantage of *Stacking* is that the first-layer can be run in parallel, followed by the second-layer in the same fashion. This cannot be achieved by the *CC* method because each classifier in the chain depends on the preceding classifiers. When the algorithms are run serially, the *Stacking* approach can take twice the execution time of *CC*. In this case, *Stacking* loses its parallel computational advantage. A summary of the algorithmic complexities deducted in this section is provided in Table 3.7.

CHAPTER 4

The PruDent Algorithm

The popularity of multi-label classification applications has motivated the development of the many existing approaches discussed in Chapter 3. The *BR* method from Section 3.2.1 is arguably the most intuitive way to target multi-label classification; which assigns a separate binary classifier for each class. Perhaps due to its simplicity, and compatibility with the existing library of traditional single-label methods, *BR* quickly became popular among the research community.

Despite showing competitive classification performance [6, 11], some researchers pointed out a possible weakness. By inducing a binary classifier for each class independently from other classes, the learner ignores *class dependencies* (see Section 2.2). And yet, research groups demonstrated significant classification improvement when class correlations are incorporated in the learner [1, 8, 11, 12, 17, 63–65].

One way to incorporate class correlations into *BR* is to extend the feature vector of examples by information about the classes to which the given example has already been shown to belong. In fact, the state-of-the-art *Classifier Chains* and *Stacking* methods from Section 3.3 follow this arrangement specifically. Although they incorporate class correlations in *BR*, however, such methods fail to pay adequate attention

to two critical issues: error-propagation, and unnecessary class dependencies. Let us take a look.

Error-Propagation: When adding to the attribute vector the information about the example's belonging to some other classes, it is often assumed that these added class labels are correct. In real-world applications, however, this is not necessarily the case. Consequently, an error made by one classifier can affect the other (dependent) classifiers. This problem will be referred to as *error-propagation*, a term also used by [76].

Unnecessary Dependencies: Not all class relations can improve classification. Intuitively, if a text document belongs to `Cooking`, it is likely to belong also to `Diet`. Nevertheless, the same document will not likely be classified as `Mining-Industry`. Suppose we extend this problem by adding the topic `Economy`. Here, the class `Economy` does not correlate with `Diet`: a document's belonging to one class does not tell us anything about its belonging to the other class. An attempt to use this particular class relationship, such as between `Economy` and `Diet`, is thus unlikely to improve performance; it may rather lead to an unnecessary increase in computational costs. Even worse, *error-propagation* could become more problematic when there are more error-prone features at hand.

The technique *PruDent* [14], introduced in this chapter, seeks to address both of these issues. Here, the outputs of independent classifiers (using the *BR* framework) are treated as new attributes that are added to the original attribute vector. Unnecessary label dependencies are pruned out, and confidence values calculated for the classifications are employed in a manner that reduces error propagation.

Experiments with popular testbeds indicate that *PruDent*'s classification performance compares favorably to that of other state-of-the-art algorithms. Moreover, the technique scales-up well for large domains: its computational costs grow linearly in the number of labels. Apart from experimental evaluation of the technique's performance, this chapter reports extensive experiments exploring its behavior.

The organization of this chapter is as follows. The motivations driving the design of the proposed technique are presented in Section 4.1. Following the implementation details of *PruDent* in Section 4.2, the experiments and their results are summarized in Sections 4.3 and 4.4, respectively. This chapter ends with some specific aspects of the approach, which are discussed in Section 4.5.

4.1 Goals for The Proposed Technique

The previous chapter described two popular algorithms that use class predictions as additional features: the *Classifier Chains* and *Stacking* methods. Unfortunately, these approaches include *unnecessary* relationships and do not deal with the *error-propagation* associated with these additional features. The goal of the proposed method is to address these issues by targeting the two relevant questions; namely, *which class dependencies are relevant?* and *how to diminish error-propagation?* Let us take a closer look.

4.1.1 Which Class Dependencies Are Relevant?

As discussed in Section 3.3, the *CC* [11] and *Stacking* [9, 65] approaches incorporate label dependence into *BR* by appending the set of class labels to the regular features. Generally, these methods assume dependence between all classes, an as-

sumption that is often violated in realistic applications. As shown by [81], some label dependencies are unnecessary. Including them can not only incur additional computational overhead, but also reduce classification accuracy because of their associated errors.

Which means that some pruning may be necessary.

Many heuristics can be used here. One reasonable consideration is to keep only the most relevant dependencies. Thus [81] and [73] measured dependencies between pairs of classes by a special version of the *Pearson product moment Correlation Coefficient (PCC)* [82] for dichotomous variables; then they eliminated class pairs with values below a certain threshold. The *PruDent* technique follows a similar approach which relies on *Information Gain (IG)* that is often used in feature selection [66, 82, 83]. Although both criteria, *PCC* and *IG*, can identify independence, the latter was used because the learning algorithm used in the experiments (see Sections 4.3.1 and 3.1.2) employs the same heuristic[1]. Note that the novelty of the proposed approach is not specific to this pruning method. This serves as an example to illustrate the idea of the proposed algorithm. The method may perform equally well with other dependency measures.

While simple methods to identify independence like *IG* and *PCC* are commonly used in the literature, one could also employ more sophisticated techniques such as those from the association rule mining field [84–86]. Nevertheless, these approaches typically seek to discover only positively related classes; classes that co-occur *frequently* rather than those that *rarely* co-occur together. Recent studies tell us that

[1]The similar *Information Gain Ratio* was not used since the drawback of *IG* in preferring multi-valued attributes does not apply here; labels can only take binary values.

the latter type of dependencies can be more than valuable for classification purposes, even when they are used exclusively [87].

The proposed algorithm quantifies the strength of a correlation between pairs of classes using *IG*. To calculate *IG*, *entropy* is used per the definition in Equation 3.1 (Chapter 3). However, since entropy is now being used to identify class correlations, it makes sense to replace the symbol for a "subset of examples" ($\mathcal{S}$) with that of the reference class (y_i) entropy is calculated for. Equation 4.1 shows this new formulation. To simplify the expression, the *prior* signifying the portion of positive examples that belong to class y_i, $P(y_i = 1)$, is substituted with P_i^+, and similarly $P(y_i = 0)$ (for negative examples) is replaced with P_i^-. To calculate the information gained by incorporating labels of class 'B' as an additional feature when targeting class 'A', the entropy function is used as illustrated by Equation 4.2. Note that *IG* values are always in the interval $[0, 1]$.

$$\mathtt{H}(\mathtt{y_i}) = -\mathtt{P_i^+}\log(\mathtt{P_i^+}) - \mathtt{P_i^-}\log(\mathtt{P_i^-}) \tag{4.1}$$

$$\mathtt{IG}(\mathtt{y_A}, \mathtt{y_B}) = \mathtt{H}(\mathtt{y_A}) - \mathtt{H}(\mathtt{y_A}|\mathtt{y_B}) \tag{4.2}$$

After calculating *IG* for all label-pair combinations, dependencies with values below a certain threshold, ϕ, are omitted. An optimal value of ϕ can be found by a validation-set procedure; a subset of the training data is used as test examples to find which ϕ values are suitable for the given classification problem.

An important difference between the proposed approach and the one used by [81] has to be pointed out. In their system, they discard the original example features when forming the *meta-examples* discussed in Section 3.3.2. Instead, the only features

they use are the outputs of the first-layer classifiers. The distinction between the two approaches lies in the type of dependency being modeled. On the one hand, using the classification vector alone targets dependencies that are global with respect to the entire training set. On the other hand, concatenating the original features with the classification vector *also* captures dependencies *conditional* on a subset of examples. The reader is referred to Section 2.2 for a detailed explanation.

4.1.2 How to Diminish Error-Propagation?

The previous step related a class to other classes by way of *Information Gain*. The removal of unnecessary dependencies also reduces *error-propagation* thanks to the reduction of error-prone features used in building *meta-examples*. However, a more drastic solution is needed to prevent the remaining classifier-dependent features from propagating errors to other classifiers.

When a feature depends on a previous classification, errors in this classification will affect all classifiers that use this feature. One way to mitigate *error-propagation* is to assign confidence values to the classifications. This makes it possible to compare the confidences of independent and dependent class predictions per unseen instance, and then choose the classification where the confidence is higher. Intuitively, when class dependencies apply to an example, the dependent classifiers should have greater confidence in their classifications.

This intuition is visualized by Figure 4.1. Suppose that classifier $h_3^{(2)}$ (second-layer for class 3) uses the output of $h_1^{(1)}$ as an additional feature. If $h_1^{(1)}$ incorrectly classifies an example, the error is propagated to $h_3^{(2)}$. This error could have been avoided if we had used the class prediction of the independent classifier $h_3^{(1)}$ as a final classification.

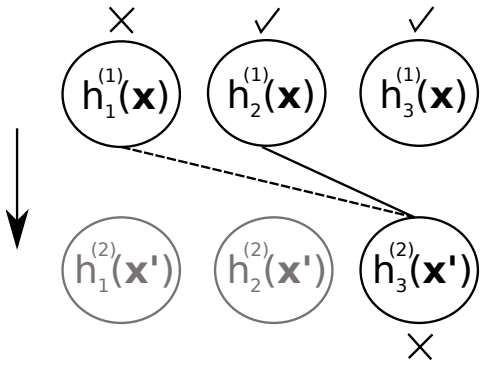


Figure 4.1: Error propagation. The tick-marks denote correct classifications; cross-marks denote incorrect classifications. The dotted line indicates error propagation.

The underlying premise here is that classification confidences can offer a guideline for the decision which classifier (dependent or independent) is safer for a particular example.

4.2 The PruDent Algorithm

The proposed algorithm is based on the *stacking* idea from Section 3.3.2. The reason why *stacking* is preferred over *CC* is to eliminate the effect of the order in which the class labels enter the feature vector; also, stacking can easily re-use independent *BR* classifiers when the classes are deemed mutually independent. In the next section, a formal definition of the *PruDent* algorithm is provided. Whenever needed, the notation from Section 2.1 will be used.

4.2.1 Training Phase

Training is carried out in a fashion similar to that of the *stacking* approach where two sets of binary classifiers are induced (Figure 3.8). Prior to induction, the proposed

algorithm prunes unnecessary dependencies so as to construct specialized feature vectors for each class.

	1	2	3	4
1	0	0.01	0.3	0.02
2	0.01	0	0.4	0.5
3	0.3	0.4	0	0.02
4	0.02	0.5	0.02	0

Figure 4.2: An example **IG** matrix with the number of classes $L = 4$.

Identification of Unnecessary Dependencies:

To identify weak label dependencies, *PruDent* first calculates IG (Equation 4.2) for all label pairs. This results in an $L \times L$ matrix **IG** where entry $\mathbf{IG}_{i,j}$ corresponds to the IG value of the label-pair l_i and l_j. Figure 4.2 illustrates an example matrix when the number of classes $L = 4$. Since **IG** is symmetrical and the IG of a class and itself is '0', only $L(L-1)/2$ calculations are needed.

Pruning Unnecessary Dependencies:

The removal of unnecessary dependencies is carried out by comparing each entry in **IG** to a pre-specified threshold, ϕ. Whenever $\mathbf{IG}_{i,j} < \phi$, the class-pair dependency of l_i and l_j is tagged as unnecessary. Eventually, **IG** is converted to a binary matrix where '0' valued entries correspond to unnecessary dependencies and '1' otherwise. Figure 4.3 illustrates the generated binary matrix when a threshold ($\phi = 0.1$) is applied to the example matrix from the previous step.

Note that higher threshold values may result in entire rows containing nothing but '0.' Labels corresponding to such rows are deemed independent from all others. The value of ϕ can be determined using a validation-set process.

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	0
4	0	1	0	0

Figure 4.3: The remaining dependencies in the **IG** matrix from Figure 4.2 when a threshold $\phi = 0.1$ is applied.

Classifier Induction:

Using the stacking method from Section 3.3.2, the first-layer of classifiers ($\mathbf{h}^{(1)}$ in Figure 3.8) is induced as in *BR* where the regular feature vector $\mathbf{x}$ is used. Recall that the second-layer classifiers ($\mathbf{h}^{(2)}$) are trained using *meta-examples* created by augmenting the original feature set with the vector of class labels. Unlike other stacking techniques, the proposed algorithm uses the previously created **IG** matrix to form specialized feature vectors: for each class, l_i, only training-set labels of classes that correspond to non-zero $\mathbf{IG}_{i,j}$ entries are used. Thus, the feature vector used to train a classifier for l_i becomes $\hat{\mathbf{x}}_i = \mathbf{x} \cup \{y_j\} : \mathbf{IG}_{i,j} \neq 0$. Note that when a class has no dependencies (the row contains only '0's), no second-layer classifier is constructed. This makes the proposed approach faster than previous stacking methods (not all classes require the induction of a second-layer classifier).

4.2.2 Prediction Phase

Classification begins by obtaining *primary* class predictions using the independent first-layer as in *stacking*; then, the predictions are combined with the original features to construct *meta-examples*; after that, they are forwarded to the second-layer to

deliver the final classifications. Here, the proposed approach has an additional step in which the *primary* predictions could be re-used.

First-layer classification:

During testing, each unseen example is submitted to the first-layer of classifiers ($\mathbf{h}^{(1)}$ in Figure 3.8) to obtain *primary* class predictions. In addition to their binary values, the proposed approach also records their associated prediction confidences. The confidence values depend on the technique chosen on the *base classifier*—for instance, they can be determined by the distance to the hyperplane in SVMs [88]. The experimental setup of the proposed method is detailed in Section 4.3.1.

Meta-example construction:

Recall that to train the second-layer classifiers, ($\mathbf{h}^{(\mathbf{2})}$), *meta-examples* are used. To build the *meta-examples*, the class labels from the training set were used to fill in the class-features. Obviously, this information is not available during prediction; labels of unseen instances are unknown. To overcome this hurdle, the missing values are filled using the *primary* predictions of the first-layer classifiers. Consequently, the newly-formed examples for class l_i are now $\hat{\mathbf{x}}_i = \mathbf{x} \cup \{\hat{y}_j\} : \mathbf{IG}_{i,j} \neq 0$. Note that label predictions, $\hat{\mathbf{y}}$, are used here instead of the *actual* labels, $\mathbf{y}$.

Second-layer classification:

After creating the *meta-examples* in the previous step, they are forwarded to the second-layer classifiers. As a result, *secondary* predictions are obtained along with their associated confidence values. In other *stacking* techniques, these class predictions become final, but the proposed algorithm follows a more sophisticated approach.

Choosing the final classification:

Thus far, each test instance has a *primary* and *secondary* prediction from the first-

and second-layer classifiers respectively. Moreover, each of the predictions is assigned a confidence value. Recall (Section 4.1.2) that *error-propagation* can be reduced with the help of confidence values. Here, *PruDent re-uses* the *primary* predictions as final classifications whenever they have higher confidence values than their *secondary* counterparts. Thus, for each class-example pair, the confidences of both class predictions are compared, and the more confident classification is chosen.

4.2.3 Computational Complexity

Having explained the implementation details of *PruDent*, let us now consider its computational complexity. The format of this deduction will follow that from Section 3.3.3. Specifically, the terms N, D, and L will refer to the number of examples, features, and classes respectively. To denote the learning and testing upper bounds of the employed base learning algorithm, the terms $\mathcal{F}_{\mathcal{B}}(N, D)$ and $\mathcal{F}'_{\mathcal{B}}(D)$ will be used.

In terms of training, as in the original *Stacking* approach, *PruDent* requires the induction of a classifier for each class in the first layer. For the second layer, only classes that are deemed dependent will require building an additional classifier (to capture the label dependencies). To establish whether a class is dependent on others or not, the proposed algorithm computes the *Information Gain* for each pair of classes as described by the previous section. Thus, $\frac{L(L-1)}{2}$ operations are required. This makes the complexity in the order of $O(N \cdot L^2)$ since the operation requires checking the class affinities of all examples. Given the above explanation, at its worst case (i.e. when all classes are deemed dependent), *PruDent*'s computational complexity during training becomes $O(L \cdot \mathcal{F}_{\mathcal{B}}(N, D + L) + N \cdot L^2)$. Although checking for class dependencies has a quadratic complexity, in practice, calculating entropy between

pairs of classes is a fast operation and requires a small computational overhead when compared to the induction time of the base learning algorithms. Furthermore, in a parallel computation environment, identifying label dependencies between each pair of classes can be efficiently run on separate processing threads.

During testing, *PruDent* requires all examples to be classified first by the independent layer of classifiers. After that, the class predictions are appended to the test example's features and they are passed to a second layer of classifiers whenever classes are deemed dependent on others. In the worst case scenario, for every test example, the algorithm will require two classification operations for each class. This makes the testing computational bound (for each test instance) in the order of $O(L \cdot \mathcal{F}'_{\mathcal{B}}(D+L))$, which resembles that of the original *Stacking* method from Section 3.3.2.

4.3 Experiments

Two major aspects are of interest: the filtering of unnecessary dependencies, and the use of classification confidence for the reduction of *error-propagation*. The experiments will first demonstrate the effect of each of the two aspects on classification performance. After this, *PruDent* will be compared with other existing state-of-the-art techniques.

4.3.1 Methods and Setup

To conduct the experiments, the publicly available J48[2] algorithm was used to induce the base classifier for binary subproblems. This learning algorithm was chosen not only because it is so well-known, but also because its performance is shown to be

[2]A java implementation of the C4.5 decision-tree induction technique; J48 is part of the WEKA software package [31]

competitive to the likes of SVM and NB [70, 81]. To smooth-out the prediction confidences of the J48 classifier, the built-in Laplace smoothing option for classification confidence (the flag '-A') was used initially. Later, this method was replaced with the m-estimate equation described in Section 4.4.3.

Classification performance was assessed by the criteria from Section 2.3. To evaluate statistical significance of the results, the 5×2 CV recommended by [89] was coupled with the two-tailed t-test (95% confidence level).

The following two versions of *PruDent* were considered:

1. Confident Stacking (Conf-ST)

2. Pruned and Confident Stacking (PruDent)

In *Conf-ST*, only the confidence-comparison modification from Section 4.1.2 is applied. The second version incorporates also the pruning of unnecessary dependencies using information gain; the acronym, *PruDent*, stands for Pruned and confiDent.

To demonstrate the performance of *PruDent*, it has to be compared with competing techniques. To this end, the *CC* and *DBR* approaches described in Section 3.3 were implemented. The latter seems to be the most recent *stacking* approach known in the literature. The decision to use the *CC* and *DBR* methods was motivated by their comparable complexity. The ensemble version of *CC* was not used, in the comparisons, because of its higher computational costs.

Since the proposed idea considers pruning, the *2BR* pruning algorithm from [81] was added to the comparison, which was described earlier in Section 4.1.1. Their approach trains the second-layer classifiers using the prediction *confidences* of the first-layer. To avoid biasing, they apply stratified CV on the training set. Thus,

following the notations from the previous section, the induction complexity of their algorithm is in the order of $O(L[F \cdot \mathcal{F}_{\mathcal{B}}(N, D) + \mathcal{F}_{\mathcal{B}}(N, L)])$, where F is the number of folds. Although their default number of folds is 10, only 3 folds were used to make the computational cost manageable.

The threshold parameters of the pruning algorithms were determined with the help of a validation set that represents $\frac{1}{3}$ of the training set. The tested values were $[0.01, 0.05$, and $0.1]$ for *PruDent* and $[0.1, 0.2$, and $0.3]$ for *2BR*. The target of the optimization was to maximize ***micro-f1***.

To show the best classification performance *stacking* is capable of (i.e. with no error-propagation), another version of stacking was added to the comparison which uses the testing set labels (during testing) to build the *meta-examples*. This version is called *NE-ST* for No-Error Stacking.

4.3.2 Data Characteristics and Data Preprocessing

The experiments relied on real-world benchmark testbeds from the biology, music, text, and image domains, which are presented in Section 2.4. During data preprocessing, the classes represented by less than 10 examples were eliminated (they do not make much sense when the 5x2CV methodology is used). Also, all unlabeled examples were removed. Finally, the features with singular values (only one value is encountered) were eliminated. Surprisingly, this reduced the feature set of the dataset *genbase* from 1185 to only 112—the eliminated features contained the value "NO." For larger data sets (`mediamill, tmc2007, bibtex, rcv1, nus-wide`, and `imdb`), classes that constitute less than 2% of the examples were removed (so as to reduce data complexity). Table 4.1 shows the characteristics of the preprocessed datasets.

The format of the presented table follows that of the original table of datasets (Table 2.2 in Section 2.4).

name	domain	features	labels	examples	LC	LD
emotion	music	72n	6	593	1.87	0.31
genbase	biology	112b	16	662	1.19	0.07
yeast	biology	103n	14	2417	4.24	0.3
scene	image	294n	6	2407	1.07	0.18
cal500	music	68n	141	502	25.5	0.18
medical	text	1421b	20	949	1.2	0.06
langlog	text	916b	38	1197	1.3	0.03
enron	text	1001b	42	1702	3.34	0.08
slashdot	text	1079b	18	3776	1.18	0.07
mediamill	video	120n	29	41962	4.20	0.14
tmc2007	text	499b	15	28098	2.14	0.14
bibtex	text	1836b	26	4804	1.44	0.06
rcv1	text	944n	42	6000	2.46	0.06
nus-wide	image	128n	21	195834	2.1	0.1
imdb	text	1001b	20	118716	1.97	0.1

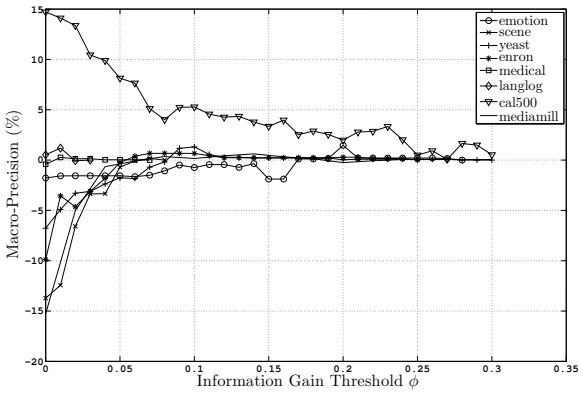
Table 4.1: Preprocessed Data Sets Statistics

4.4 Results

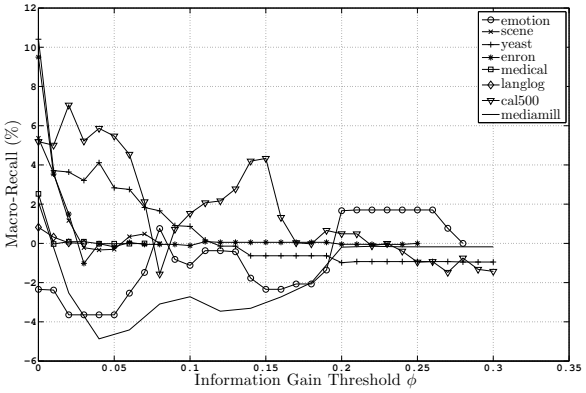
4.4.1 Pruning Unnecessary Dependencies

Earlier, we stipulated that using all class labels as additional features is unnecessary, and this may even impair classification performance due to their associated errors. The task for the first experiment is therefore to investigate the practical impact of pruning unnecessary dependencies.

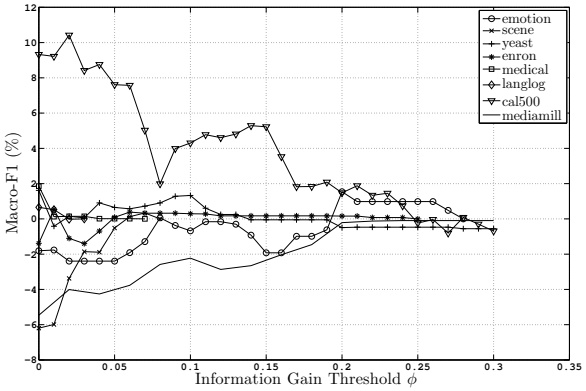
Section 4.1.1 suggested that the choice of the relevant labels be based on their mutual *Information Gain*. Following this suggestion, the proposed method eliminates any pairwise class dependencies whose IG values do not reach a pre-specified



(a) macro-precision



(b) macro-recall

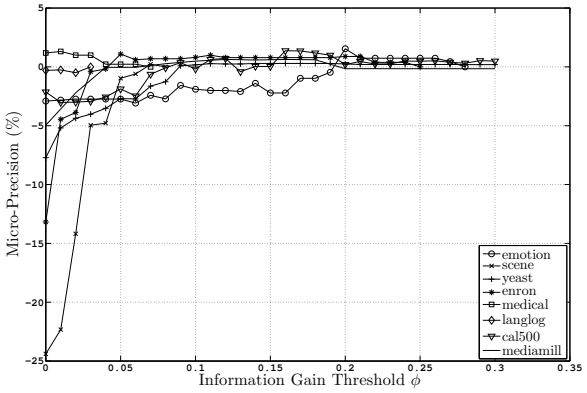


(c) macro-f1

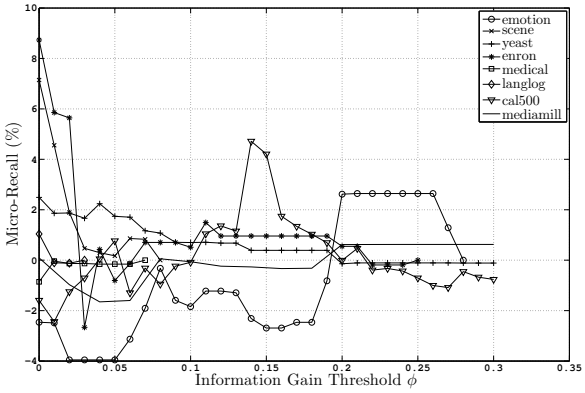
Figure 4.4: The percent difference with respect to *BR* when applying information gain threshold ϕ. The results are in terms of the information retrieval metrics (`macro-precision, macro-recall,` and `macro-f1`).

threshold, ϕ. To learn which concrete values should be used, the next experiment varies the threshold value from the range $\phi \in [0-0.3]$. The results of this experiment are illustrated in Figures 4.4 to 4.6.

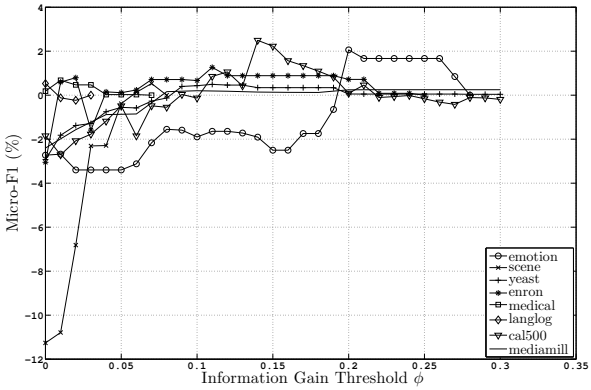
The plotted results in the in the figures show by how many percentage points each metric increases or decreases with respect to *BR* when changing the value of the *IG* threshold, ϕ. To improve the chart's readability, some data sets whose curves were



(a) micro-precision



(b) micro-recall

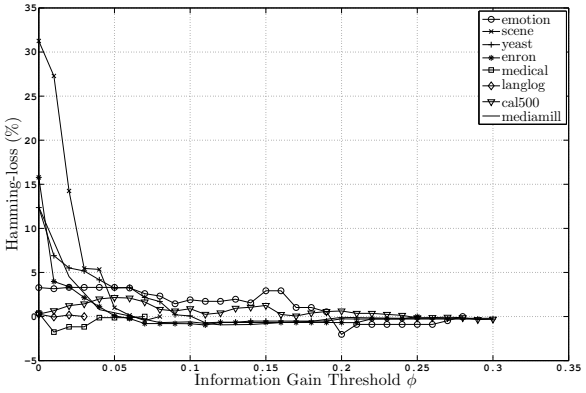


(c) micro-f1

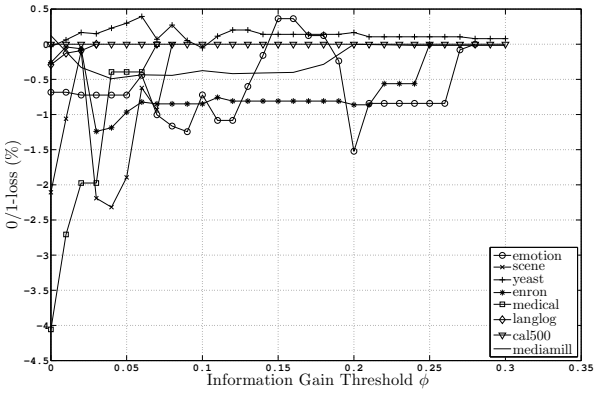
Figure 4.5: The percent difference with respect to *BR* when applying information gain threshold ϕ. The results are in terms of the information retrieval metrics (`micro-precision`, `micro-recall`, and `micro-f1`).

too similar to tell us anything new were eliminated. Let us now take a look at the results.

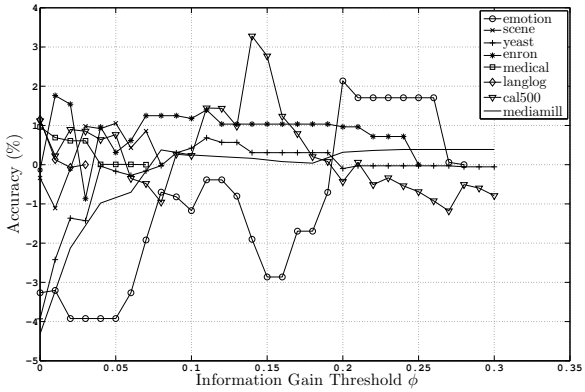
By looking at the information retrieval metrics charts in Figure 4.4 and 4.5, the following observations are evident. When no pruning is applied (i.e. $\phi = 0$), `precision` (Figures 4.4a and 4.5a) decreases indicating that errors may be reducing the classifier's ability to identify the usually dominant negative examples. After applying pruning however, `precision` improves although by a small margin. An exception to



(a) hamming-loss



(b) 0/1-loss



(c) accuracy

Figure 4.6: The percent difference with respect to *BR* when applying information gain threshold ϕ. The results are in terms of `hamming-loss`, `0/1-loss`, and `accuracy`.

this phenomenon is the dataset `cal500` which experienced a decrease in `precision`. This is perhaps due to the high number of co-occurring classes in `cal500` ($LC = 25.5$ in Table 4.1), which in turn lowers the false positive rate.

Figures 4.4b and 4.5b tells us that most datasets experienced a decrease in `recall` when pruning is applied. This means that limiting the number of correlations that are modeled in the learner reduces its positive class predictions. Another observation that can be made is the instability of the results when varying the value of ϕ. This may be attributed to the noisy nature of the class-attributes that are added to the

feature vector. It is worthy to note that some datasets experienced an increase in `micro-recall` (Figure 4.5b) at higher information gain thresholds such as `emotion` and `cal500`. This indicates that major classes may be more susceptible to the presence of unnecessary class correlations in the feature vector.

In terms of the `f1` measures, while some data sets suffered a decline in performance, others gained, even if only by a small margin. A closer inspection reveals that classification on *denser* data sets exhibited stronger improvement than that on sparser data. Notably, the `emotion` data set ($LD = 0.3$) experienced improvement upon BR across a wide range of thresholds. Other data sets showed a minor increase only with certain threshold ranges; this was the case of `yeast`, `scene`, and `enron`. In the case of `mediamill`, no noticeable performance improvement was observed. The domain `genbase` (not shown in the graph) experienced no change in performance across the entire range of threshold values.

Figure 4.6 presents the classification performance results in terms of `hamming-loss`, `0/1-loss`, and `accuracy`. Note that an improvement in the first two measures is indicated by a percentage *decrease* since that they are *loss* measures. Recall that `hamming-loss` punishes a prediction based on the number of classes misclassified per example. As expected, Figure 4.6a shows that most datasets suffer from a higher (worse) `hamming-loss` rate when no pruning is done. As the threshold is set higher, `hamming-loss` is reduced considerably. However, no significant improvements are observed when compared to BR even when high values of ϕ are used. The same cannot be said about `0/1-loss` which punishes a class prediction vector even if it is partially correct. Depicted in Figure 4.6b, the results show that there is an improvement when compared to BR across a wide range of thresholds. Also of note, some datasets re-

quire no pruning to minimize this loss metric. This improvement can be attributed to the higher number of predicted positive examples, observed by the high `recall`, in combination of the sparse nature of most datasets.

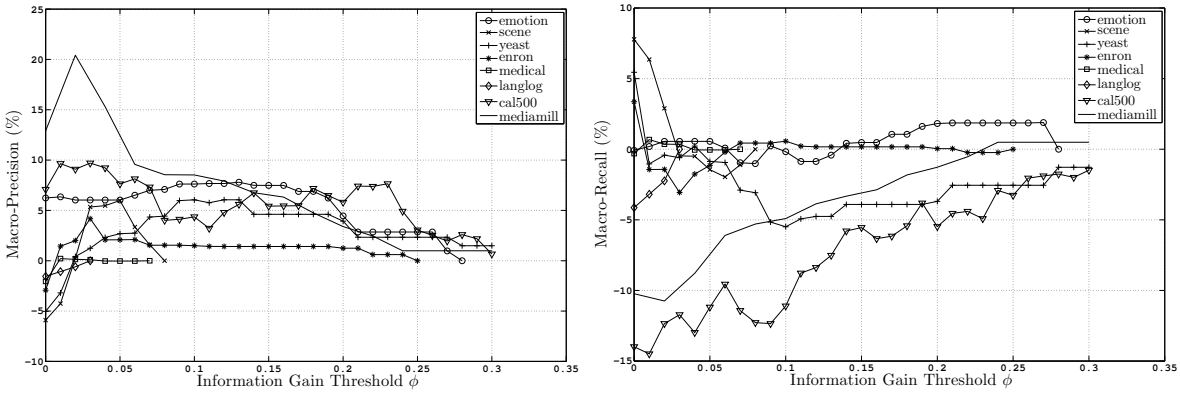
The results of the `accuracy` measure are similar to the `f1` results analyzed above. Specifically, when no pruning is applied, the accuracy of the predictions is lower than that of *BR*. However, as the less relevant correlations are pruned-out, the classifier's accuracy increases.

4.4.2 Incorporating Classification Confidence

The next experiment explores the effect of using *classification confidence* (Section 4.1.2). Recall that, when classifying an example, *PruDent* calculates the classification confidences of the first- and second-layer classifiers, compares them, and then chooses the layer with the higher confidence. The motivation is to reduce *error-propagation*. Whether this really happens has to be established experimentally.

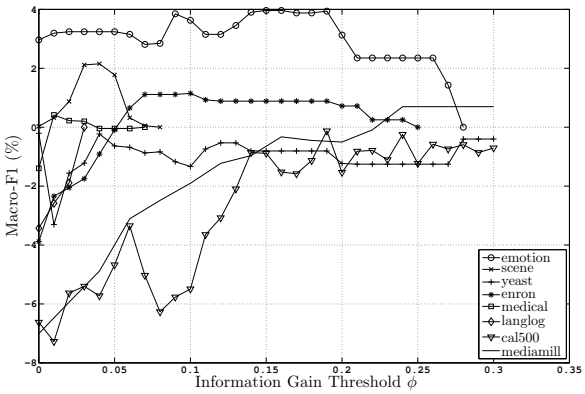
Figures 4.7 to 4.9 present the results of running the same experiments from the previous section with the inclusion of the prediction confidence comparison. When the results are compared with those from the previous experiment, the reader can see that comparing confidences prior to making predictions does affect performance. Let us take a closer look.

Looking at the `precision` results shown in Figures 4.7a and 4.8a, there is a clear improvement with respect to *BR*. This was not the case in the previous experiment (see Figures 4.4a and 4.5a). In terms of `recall`, an improvement over *BR* is apparent in `micro-recall` across a wide range of threshold values. In contrast, `macro-recall` experienced a degradation in performance in most datasets. Thus, classes that are



(a) macro-precision

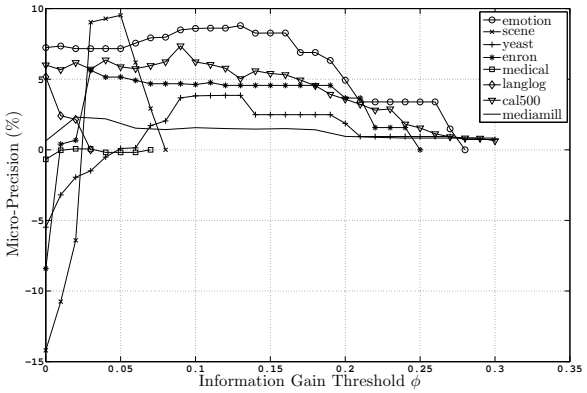
(b) macro-recall



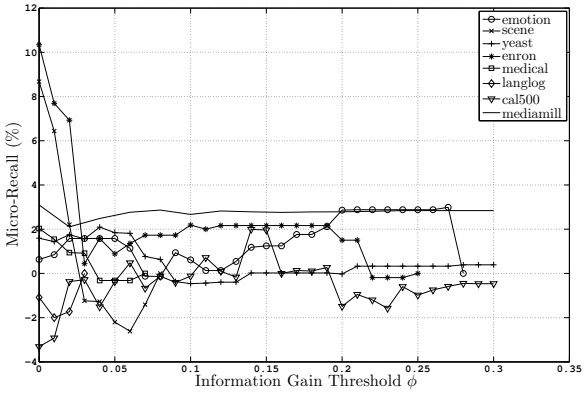
(c) macro-f1

Figure 4.7: The percent difference with respect to *BR* when applying information gain threshold ϕ and prediction-confidence comparison. The results are in terms of the information retrieval metrics (`macro-precision, macro-recall,` and `macro-f1`).

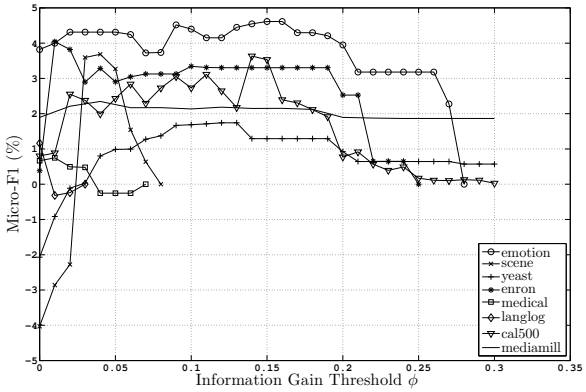
marked by a higher number of positive examples were recalled more successfully than rarely occurring classes. Despite the decrease in recall, the *macro* and *micro* variants of `f1` experienced an overall improvement in most datasets. The improvement in the *micro* variant was again more visible as a result of the more accurate classifications of major classes (those represented by a higher portion of examples). The `macro-f1` results of `cal500` and `yeast`, however, did not experience improvements (with respect



(a) micro-precision



(b) micro-recall

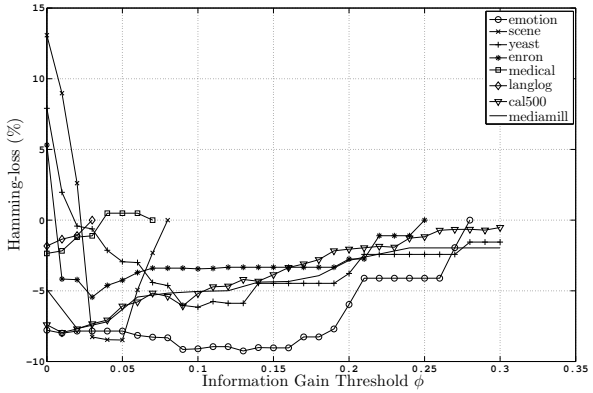


(c) micro-f1

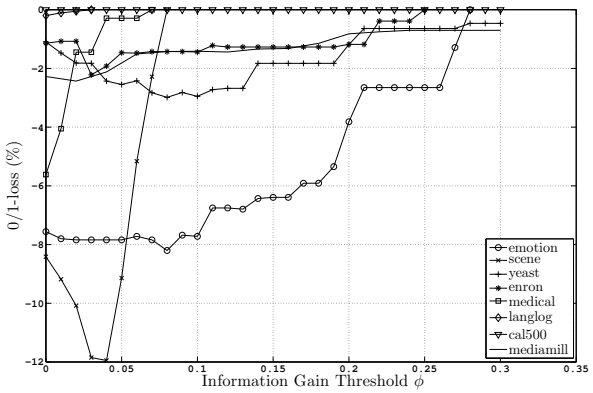
Figure 4.8: The percent difference with respect to *BR* when applying information gain threshold ϕ and prediction-confidence comparison. The results are in terms of the information retrieval metrics (`micro-precision`, `micro-recall`, and `micro-f1`).

to *BR*) throughout the entire range of thresholds. This is somewhat expected given the low `macro-recall` observed for the two datasets.

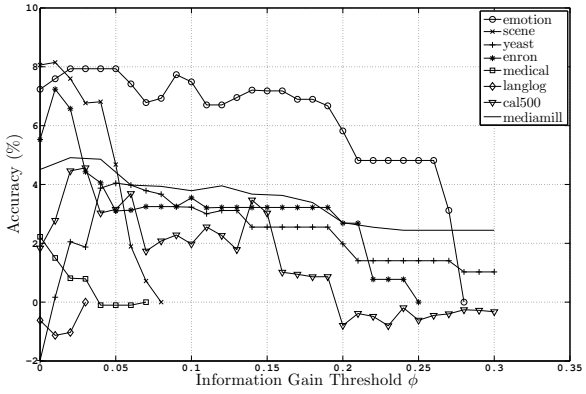
Incorporating classification confidence yields a significant improvement when considering the `hamming-loss` and `0/1-loss` metrics. As illustrated by Figures 4.9a and 4.9b, both of these *loss* metrics experienced a decrease indicating an improvement in the quality of predictions. For example, `hamming-loss` in Figure 4.9a almost always experienced improvements which was not the case in the experiment from



(a) hamming-loss



(b) 0/1-loss



(c) accuracy

Figure 4.9: The percent difference with respect to *BR* when applying information gain threshold ϕ and prediction-confidence comparison. The results are in terms of `hamming-loss`, `0/1-loss`, and `accuracy`.

the previous section (Figure 4.6a). The same can also be said about `0/1-loss`. It is also important to note that in half of the datasets, `hamming-loss` improves only when some of the unnecessary class correlations are pruned. Thus, in some domains, pruning weak class correlations is a vital step to achieve classification improvements.

Perhaps further adding evidence to the importance of incorporating classification confidences is the result of the `accuracy` measure in Figure 4.9c. Here, a remarkable

improvement is achieved when compared to the version which does not account for classification confidence (Figure 4.6c in the previous section).

In general, incorporating classification confidence in stacking resulted in classification improvements across most of the multi-label performance criteria. The only weakness that has not been addressed is the low recall rates when compared to their precision counterparts. The next section proposes a solution for that problem.

4.4.3 Addressing the Low Recall Rates

The previous two experiments demonstrated the effect of pruning unnecessary dependencies and incorporating prediction confidence in the stacking framework. Applying both techniques lead to a clear improvement in classification accuracy. However, as witnessed by the low recall rates after applying the confidence-comparison modification, negative predictions were favored more often than positive ones. This result is perhaps attributed to the imbalanced representation of classes in each binary training set; negative examples usually occupy a higher portion of examples than positive ones. The intuition here is that negative predictions must have higher confidence values than their positive counterparts. To verify this statement, a closer look at how prediction confidences are produced is necessary.

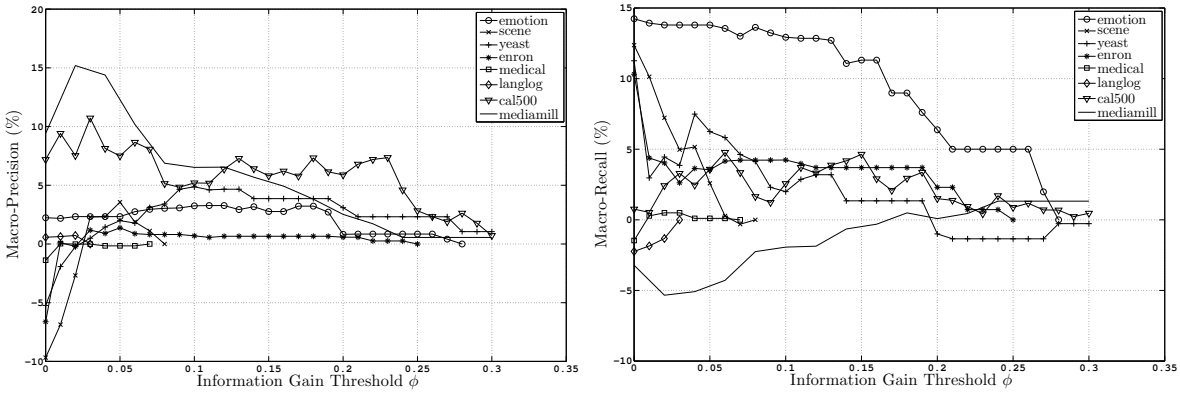
Recall from Section 4.3.1 that the *C4.5* decision tree algorithm (J48) was used as a base learner for each binary subproblem in the experiments. It follows from Section 3.1.2 that prediction confidences in that classifier are calculated using the distribution of examples in each leaf node. Also recall that the *Laplace* method, described in Section 3.1.2, is typically used to smooth-out the prediction confidence values. This has been the method followed thus far in the experiments (see Sec-

tion 4.3.1). The *Laplace* technique is a special case of the *m-estimate* approximation method illustrated by Equation 4.3. As in the *Laplace* equation from Section 3.1.2 (Equation 3.4), the term N denotes the total number of examples, and N_p refers to the number of positive training examples observed in a given leaf node. Here, two additional parameters (p and m) are used to bias examples towards a given *prior* class distribution. The term p refers to the *ratio* of positive examples believed to exist in the dataset, and m is used to control the influence of the chosen p parameter when assigning prediction confidence values.

$$P(\oplus) = \frac{N_p + p.m}{N + m} \tag{4.3}$$

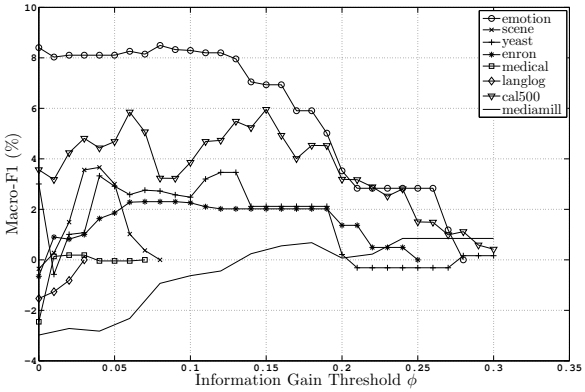
The *Laplace* method uses the value 0.5 for the p parameter and 2 for the m parameter. This implies that the dataset is believed to contain an equal number of positive and negative examples, hence $p = 0.5$. However, in the *BR* setting, each binary classifier is trained using a dichotomous training set which uses the examples of the target class as positive examples and all others as negative. As a result, each binary training set will often have imbalanced class representation; negative classes will outnumber positive examples. This is where a modification becomes necessary.

Since the experiments revealed that negative example predictions are favored more often than positive predictions (low `recall` and high `precision`), it is thus desired to bias the prediction confidence towards positive examples. To this end, the built-in *Laplace* method of the used classifier was replaced with the m-estimate approximation technique described above. To favor positive class predictions, the parameter m was set to 3 and p was set to $\frac{2}{3}$; this means that the prior will now assume that $\frac{2}{3}$ of



(a) macro-precision

(b) macro-recall

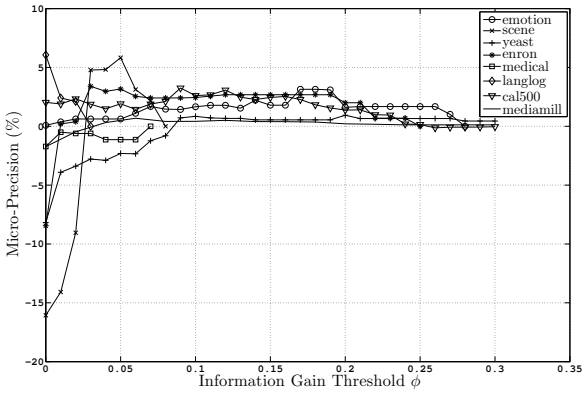


(c) macro-f1

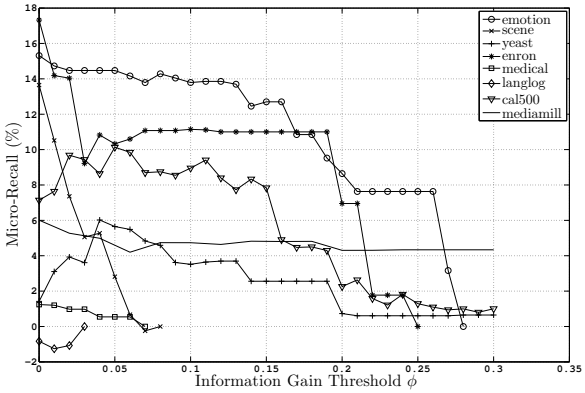
Figure 4.10: The percent difference with respect to *BR* when applying information gain threshold ϕ and the m-estimate prediction-confidence comparison. The results are in terms of the information retrieval metrics (`macro-precision, macro-recall,` and `macro-f1`).

examples are positive in each binary training set. Note that the p and m parameters need not be exact. Auxiliary experiments indicate a similar outcome when p is varied in the range $[\frac{5}{9} - \frac{4}{5}]$.

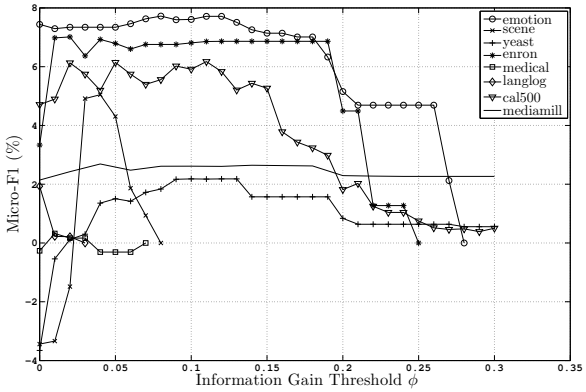
To verify whether biasing the prior towards positive examples affects classification performance, the previous set of experiments was repeated; this time, the m-estimate approximation with the chosen parameters ($m = 3$ and $p = \frac{2}{3}$) was used to estimate the prediction confidence values instead of the original *Laplace* method. Figures 4.10



(a) micro-precision



(b) micro-recall

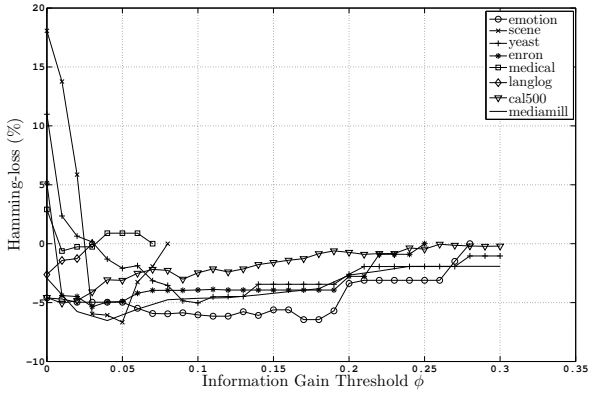


(c) micro-f1

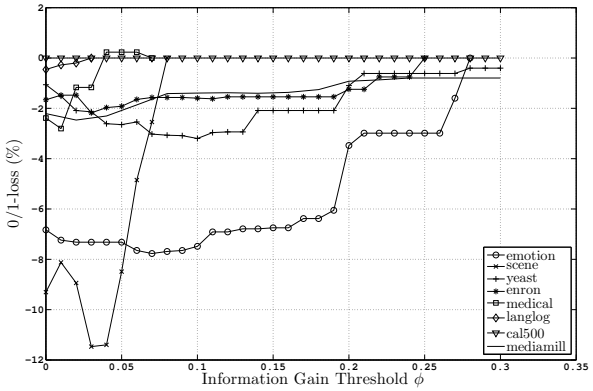
Figure 4.11: The percent difference with respect to *BR* when applying information gain threshold ϕ and the m-estimate prediction-confidence comparison. The results are in terms of the information retrieval metrics (`micro-precision`, `micro-recall`, and `micro-f1`).

to 4.12 plot the acquired results in a similar fashion to that of the previous experiments.

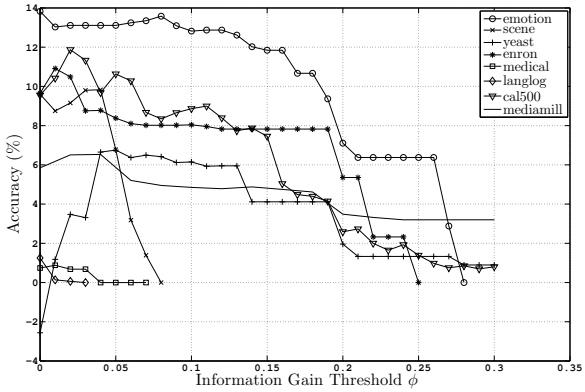
Figure 4.10a and 4.11a tell us that precision decreased when compared to the previous experiment; a result that is expected since the classifier is now biased towards predicting positive labels. Despite this decrease, however, there is still an improvement when compared to *BR*. Moreover, the improvement is more substantial in `macro-precision` indicating that precision of the popular classes was less affected.



(a) hamming-loss



(b) 0/1-loss



(c) accuracy

Figure 4.12: The percent difference with respect to *BR* when applying information gain threshold ϕ and m-estimate prediction-confidence comparison. The results are in terms of `hamming-loss`, `0/1-loss`, and `accuracy`.

The `recall` results (Figures 4.10b and 4.11b) indicate a significant improvement in classifying positive examples when compared to the previous experiment. The proposed algorithm was able to outperform *BR* in `macro-` and `micro-recall`, which was not the case when the *Laplace* confidence estimation was used. The increase in `recall` is also reflected in `macro-` and `micro-f1`. Since `f1` is the harmonic mean between `precision` and `recall`, its increase indicates that `recall` was improved while maintaining good `precision`; leading to class predictions that are more evenly

distributed. Note that in the `emotion` dataset, the highest increase in `micro-f1` (8%) is four times that of the original experiment which uses no confidence comparison in Figure 4.5c (2%).

In terms of the *loss* metrics, the results in Figure 4.12a shows an improvement in `hamming-loss`, but not as much as when the *Laplace* estimation was used. This is perhaps due to the sparsity of the multi-label data; since negative examples in each dataset are more frequent, biasing the predictor towards positive examples makes it more prone to partial errors. Unlike `hamming-loss`, the difference in `0/1-loss` is not very clear; an improvement over *BR* is still evident despite changing the confidence estimation method.

Looking at the `accuracy` results in Figure 4.12c, the reader can see that using the *m-estimate* confidence approximation yielded a better outcome than when using *Laplace* (Figure 4.9c). Moreover, the results indicate an increase in performance across the majority of the datasets.

We learn from this experiment that using prediction confidences in stacking is essential for all data sets. Notably, most data set experienced improvements when we compare Figures 4.4 to 4.6 with Figures 4.10 to 4.12 respectively. However, using confidences alone is not enough to improve classification upon *BR* in some data sets (e.g. `yeast` and `scene` in Figure 4.11c). Here, pruning unnecessary dependencies becomes essential.

It is important to comment on the bell shape of the performance curves. When the pruning threshold ϕ is first increased, some unnecessary dependencies are excluded from the learning model, and this results in better classification, but a further increase of the threshold eliminates also dependencies that might have been helpful. In fact,

the proposed algorithm converges to *BR* when $\phi = 1$. In this case, all label pairs are deemed *independent* and no second-layer classifiers are constructed.

4.4.4 PruDent and Conf-ST Versus Other Algorithms

Thus far, we have seen the effect of enhancing the *stacking* approach with the two proposed modifications: the first prunes out unnecessary dependencies, the second limits *error-propagation*. When used in combination, they should result in a clear improvement in classification performance.

To verify this statement, the next experiments compare the performance of *PruDent* with that of state-of-the-art *CC*, *DBR* and *2BR* techniques. To these comparisons, a variation of the proposed algorithm is added: in this, only the limiting of *error-propagation* is incorporated, leaving all dependencies intact. Recall from Section 4.3.1 that this version is called *Conf-ST* (*Confident Stacking*). It makes sure to also test for the case of no class-dependency pruning.

Tables 4.2 through 4.10 compare the performance of these algorithms along the metrics from Section 2.3. The result of the best algorithm in each domain is boldfaced. When there is more than one winner, they are all boldfaced unless all techniques are tied. Significance tests relied on the paired t-test with 95% confidence. In the tables, the symbols $^{\uparrow}$ and $^{\downarrow}$ indicate that the performance of a technique is significantly higher or lower, respectively, than that of *PruDent*. Similar formalism is used in the case of *Conf-ST*, using symbols $_{\Uparrow}$ and $_{\Downarrow}$, respectively.

Before starting to discuss the experimental results, an important difference between the two variants of the proposed technique, and *CC* and *DBR* must be pointed out. As explained, *Conf-ST* and *PruDent* degenerate to independent *BR* models

macro-precision (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.580 \pm 0.01^{\downarrow}$	0.589 ± 0.01	$0.557 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.550 \pm 0.05^{\downarrow}_{\Downarrow}$	0.595 ± 0.02	$\mathbf{0.602 \pm 0.03}$	0.700 ± 0.02
genbase	0.990 ± 0.003	0.990 ± 0.003	0.990 ± 0.003	$0.864 \pm 0.10^{\downarrow}_{\Downarrow}$	0.990 ± 0.003	0.990 ± 0.003	0.990 ± 0.003
yeast	$0.402 \pm 0.01^{\downarrow}_{\Uparrow}$	$0.384 \pm 0.01^{\downarrow}$	$0.368 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.374 \pm 0.04^{\downarrow}$	$0.379 \pm 0.008^{\downarrow}$	$\mathbf{0.425 \pm 0.02}_{\Uparrow}$	0.841 ± 0.005
scene	$0.626 \pm 0.01_{\Uparrow}$	$0.612 \pm 0.02^{\downarrow}_{\Uparrow}$	$0.542 \pm 0.03^{\downarrow}_{\Downarrow}$	$\mathbf{0.689 \pm 0.05}^{\uparrow}_{\Uparrow}$	$0.575 \pm 0.009^{\downarrow}$	$0.638 \pm 0.03_{\Uparrow}$	0.863 ± 0.01
cal500	$0.180 \pm 0.001^{\downarrow}_{\Downarrow}$	0.194 ± 0.008	$\mathbf{0.206 \pm 0.001}^{\uparrow}_{\Uparrow}$	$0.112 \pm 0.01^{\downarrow}_{\Downarrow}$	0.193 ± 0.005	0.193 ± 0.005	0.466 ± 0.02
medical	0.593 ± 0.05	0.600 ± 0.05	$\mathbf{0.608 \pm 0.05}$	$0.466 \pm 0.02^{\downarrow}_{\Downarrow}$	0.595 ± 0.05	0.593 ± 0.05	0.659 ± 0.05
langlog	0.144 ± 0.02	$\mathbf{0.145 \pm 0.03}$	$\mathbf{0.145 \pm 0.02}$	$0.021 \pm 0.01^{\downarrow}_{\Downarrow}$	$\mathbf{0.145 \pm 0.02}$	$\mathbf{0.145 \pm 0.02}$	0.149 ± 0.03
enron	$0.227 \pm 0.02_{\Uparrow}$	$0.227 \pm 0.01_{\Uparrow}$	$0.220 \pm 0.02^{\downarrow}$	$0.115 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.213 \pm 0.01^{\downarrow}$	$\mathbf{0.229 \pm 0.02}_{\Uparrow}$	0.306 ± 0.02
slashdot	$0.447 \pm 0.05^{\downarrow}_{\Uparrow}$	$0.430 \pm 0.03_{\Uparrow}$	$0.221 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.225 \pm 0.03^{\downarrow}$	$0.249 \pm 0.04^{\downarrow}$	$\mathbf{0.450 \pm 0.05}_{\Uparrow}$	0.651 ± 0.04
mediamill	$0.406 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.360 \pm 0.009^{\downarrow}_{\Downarrow}$	$0.346 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.280 \pm 0.04^{\downarrow}_{\Downarrow}$	$\mathbf{0.441 \pm 0.02}$	$\mathbf{0.441 \pm 0.02}$	0.670 ± 0.003
tmc2007	$0.679 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.675 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.665 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.660 \pm 0.04^{\downarrow}$	$0.683 \pm 0.005^{\downarrow}$	$\mathbf{0.688 \pm 0.005}_{\Uparrow}$	0.707 ± 0.006
bibtex	$0.549 \pm 0.008_{\Downarrow}$	$0.549 \pm 0.005_{\Downarrow}$	$0.548 \pm 0.01_{\Downarrow}$	$0.393 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.555 \pm 0.010}^{\uparrow}$	$0.550 \pm 0.009_{\Downarrow}$	0.565 ± 0.009
rcv1	$0.351 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.337 \pm 0.009^{\downarrow}_{\Downarrow}$	$0.325 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.183 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.384 \pm 0.007}^{\uparrow}$	$0.372 \pm 0.007_{\Downarrow}$	0.706 ± 0.01
nus-wide	$0.240 \pm 0.002^{\downarrow}_{\Uparrow}$	$0.224 \pm 0.003^{\downarrow}_{\Uparrow}$	$0.138 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.224 \pm 0.01^{\downarrow}_{\Uparrow}$	$0.166 \pm 0.004^{\downarrow}$	$\mathbf{0.318 \pm 0.004}_{\Uparrow}$	0.474 ± 0.002
imdb	$0.250 \pm 0.01^{\downarrow}_{\Uparrow}$	$0.213 \pm 0.03^{\downarrow}_{\Uparrow}$	$0.123 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.072 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.140 \pm 0.01^{\downarrow}$	$\mathbf{0.276 \pm 0.02}_{\Uparrow}$	0.525 ± 0.02

Table 4.2: macro-precision. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

when their classification confidences are higher than their dependent counterparts. In *DBR*, classifications of the dependent models are *always* used, regardless of their confidence values. Thus, *Conf-ST* and *PruDent* are expected to be more conservative in using label dependencies, behaving like a hybrid between *BR* and *DBR*. When comparing *PruDent* to *Conf-ST*, the former is expected to be more conservative because it prunes unnecessary dependencies, (which makes it even closer to *BR*).

Due to their conservative nature, the two variants of the proposed algorithm excelled in such performance metrics as `precision` and `accuracy`. Tables 4.2 and 4.8 informs us that *PruDent* had the highest `macro-precision` in more domains than the other methods, and in `micro-precision`, the algorithm was very competitive to the others. Also, the two versions had the highest *accuracies* in 11 out of the 15 data sets.

In terms of the `macro` and `micro` versions of the performance metrics, it is clear that the proposed algorithms scored better in the `micro` variants—especially in the

macro-recall (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.562 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.565 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.555 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.496 \pm 0.05^{\downarrow}_{\Downarrow}$	0.618 ± 0.02	$\mathbf{0.624 \pm 0.03}$	0.684 ± 0.02
genbase	0.962 ± 0.02	0.963 ± 0.02	$\mathbf{0.965 \pm 0.02}$	$0.848 \pm 0.08^{\downarrow}_{\Downarrow}$	0.962 ± 0.02	0.962 ± 0.02	0.965 ± 0.02
yeast	$0.376 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.376 \pm 0.01_{\Downarrow}$	$\mathbf{0.411 \pm 0.01}^{\uparrow}$	$0.257 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.411 \pm 0.01}^{\uparrow}$	$0.387 \pm 0.01_{\Downarrow}$	0.836 ± 0.004
scene	$0.615 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.618 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.649 \pm 0.02_{\Downarrow}$	$0.460 \pm 0.05^{\downarrow}_{\Downarrow}$	$\mathbf{0.688 \pm 0.02}^{\uparrow}$	$0.635 \pm 0.02_{\Downarrow}$	0.843 ± 0.02
cal500	$0.153 \pm 0.003^{\downarrow}_{\Downarrow}$	$\mathbf{0.173 \pm 0.006}^{\uparrow}_{\Uparrow}$	$0.161 \pm 0.002^{\uparrow}_{\Uparrow}$	$0.118 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.154 \pm 0.002^{\downarrow}$	$0.156 \pm 0.003_{\Uparrow}$	0.407 ± 0.004
medical	0.515 ± 0.03	0.521 ± 0.03	$\mathbf{0.527 \pm 0.02}$	$0.460 \pm 0.03^{\downarrow}_{\Downarrow}$	0.521 ± 0.02	0.517 ± 0.03	0.581 ± 0.03
langlog	$0.104 \pm 0.02^{\uparrow}$	$\mathbf{0.105 \pm 0.02}^{\uparrow}$	$\mathbf{0.105 \pm 0.02}_{\Uparrow}$	$0.015 \pm 0.02^{\downarrow}_{\Downarrow}$	0.103 ± 0.02	0.103 ± 0.02	0.109 ± 0.02
enron	$0.163 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.173 \pm 0.006_{\Downarrow}$	$\mathbf{0.187 \pm 0.005}^{\uparrow}$	$0.116 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.185 \pm 0.005^{\uparrow}$	$0.171 \pm 0.007_{\Downarrow}$	0.230 ± 0.008
slashdot	$0.204 \pm 0.010_{\Downarrow}$	$0.237 \pm 0.01^{\uparrow}_{\Downarrow}$	$\mathbf{0.510 \pm 0.03}^{\uparrow}_{\Uparrow}$	$0.177 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.459 \pm 0.03^{\uparrow}$	$0.203 \pm 0.010_{\Downarrow}$	0.463 ± 0.03
mediamill	$0.276 \pm 0.005^{\uparrow}_{\Uparrow}$	$\mathbf{0.285 \pm 0.008}^{\uparrow}_{\Uparrow}$	$0.281 \pm 0.002^{\uparrow}_{\Uparrow}$	$0.165 \pm 0.005^{\downarrow}_{\Downarrow}$	0.269 ± 0.004	0.269 ± 0.005	0.619 ± 0.003
tmc2007	$0.578 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.575 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.575 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.491 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.602 \pm 0.005}^{\uparrow}$	$0.594 \pm 0.005_{\Downarrow}$	0.609 ± 0.004
bibtex	$0.453 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.456 \pm 0.010_{\Downarrow}$	$0.457 \pm 0.01_{\Downarrow}$	$0.342 \pm 0.03^{\downarrow}_{\Downarrow}$	$\mathbf{0.469 \pm 0.005}^{\uparrow}$	$0.455 \pm 0.009_{\Downarrow}$	0.471 ± 0.01
rcv1	$\mathbf{0.280 \pm 0.004}^{\uparrow}_{\Uparrow}$	$0.277 \pm 0.004^{\uparrow}_{\Uparrow}$	$0.261 \pm 0.004^{\downarrow}_{\Uparrow}$	$0.133 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.245 \pm 0.004^{\downarrow}$	$0.270 \pm 0.01_{\Uparrow}$	0.628 ± 0.01
nus-wide	$0.195 \pm 0.001^{\uparrow}_{\Downarrow}$	$0.195 \pm 0.003^{\uparrow}_{\Downarrow}$	$\mathbf{0.282 \pm 0.003}^{\uparrow}_{\Uparrow}$	$0.076 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.252 \pm 0.003^{\uparrow}$	$0.177 \pm 0.001_{\Downarrow}$	0.428 ± 0.001
imdb	$0.061 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.086 \pm 0.002^{\uparrow}_{\Downarrow}$	$\mathbf{0.314 \pm 0.02}^{\uparrow}_{\Uparrow}$	$0.013 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.307 \pm 0.02^{\uparrow}$	$0.076 \pm 0.001_{\Downarrow}$	0.298 ± 0.004

Table 4.3: macro-recall. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

case of the `recall` and `f1` criteria. The acquired `micro-recall` and `micro-f1` scores were the highest among all algorithms, which was not the case of `macro-recall` and `macro-f1`. This leads us to believe that major classes (those represented by more examples) were classified with greater success than minor ones. Perhaps this is due to the less defined co-occurrence patterns, given the quantity of positive examples in minor classes. This hypothesis is also supported by the algorithm's success along the `macro-f1` criterion in the dense `emotion`, `yeast` and `scene` domains (see Table 4.1).

In terms of loss metrics, the results show that *PruDent* did particularly well along the `hamming-loss` metric. This was to be expected because *PruDent* is very close to *BR*, which is generally believed to be good in terms of `hamming-loss` [6, 12]. The *2BR* method shares this success to some degree: while it outperformed *PruDent* in some domains, in other domains it was not as successful. The success of *2BR* can be attributed to its ability to correct errors because it trains the second-layer classifiers

macro-f1 (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.567 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.574 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.553 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.497 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.604 \pm 0.01^{\downarrow}$	$\mathbf{0.610 \pm 0.02}_{\Uparrow}$	0.690 ± 0.01
genbase	0.973 ± 0.01	0.973 ± 0.01	$\mathbf{0.974 \pm 0.01}$	$0.850 \pm 0.09^{\downarrow}_{\Downarrow}$	0.973 ± 0.01	0.973 ± 0.01	0.974 ± 0.01
yeast	$0.381 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.376 \pm 0.01^{\downarrow}$	$0.382 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.268 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.387 \pm 0.006^{\downarrow}$	$\mathbf{0.392 \pm 0.004}_{\Uparrow}$	0.839 ± 0.003
scene	$0.619 \pm 0.008^{\downarrow}$	$0.614 \pm 0.01^{\downarrow}$	$0.583 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.532 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.623 \pm 0.009^{\downarrow}$	$\mathbf{0.634 \pm 0.01}_{\Uparrow}$	0.853 ± 0.01
cal500	$0.156 \pm 0.001^{\downarrow}_{\Downarrow}$	$\mathbf{0.176 \pm 0.006}^{\uparrow}_{\Uparrow}$	$0.170 \pm 0.003^{\uparrow}_{\Uparrow}$	$0.104 \pm 0.01^{\downarrow}_{\Downarrow}$	0.161 ± 0.001	0.161 ± 0.002	0.419 ± 0.007
medical	0.532 ± 0.03	0.538 ± 0.03	$\mathbf{0.544 \pm 0.02}_{\Uparrow}$	$0.450 \pm 0.02^{\downarrow}_{\Downarrow}$	0.534 ± 0.02	0.533 ± 0.03	0.597 ± 0.03
langlog	$\mathbf{0.114 \pm 0.02}$	$\mathbf{0.114 \pm 0.02}$	$\mathbf{0.114 \pm 0.02}$	$0.013 \pm 0.01^{\downarrow}_{\Downarrow}$	0.112 ± 0.02	0.112 ± 0.02	0.118 ± 0.02
enron	$0.181 \pm 0.006^{\downarrow}$	0.187 ± 0.006	$\mathbf{0.188 \pm 0.006}^{\uparrow}_{\Uparrow}$	$0.109 \pm 0.01^{\downarrow}_{\Downarrow}$	0.184 ± 0.005	0.184 ± 0.008	0.251 ± 0.007
slashdot	0.239 ± 0.01	$\mathbf{0.249 \pm 0.02}_{\Uparrow}$	$0.229 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.182 \pm 0.02^{\downarrow}_{\Downarrow}$	0.236 ± 0.02	0.240 ± 0.01	0.527 ± 0.03
mediamill	$\mathbf{0.318 \pm 0.005}_{\Uparrow}$	0.313 ± 0.006	$0.300 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.185 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.310 \pm 0.005^{\downarrow}$	$0.316 \pm 0.005_{\Uparrow}$	0.640 ± 0.002
tmc2007	$0.621 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.617 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.614 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.542 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.634 \pm 0.004}^{\uparrow}$	$0.632 \pm 0.003_{\Downarrow}$	0.652 ± 0.003
bibtex	$0.491 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.492 \pm 0.007_{\Downarrow}$	$0.493 \pm 0.009_{\Downarrow}$	$0.348 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.500 \pm 0.004}^{\uparrow}$	$0.492 \pm 0.006_{\Downarrow}$	0.509 ± 0.010
rcv1	$\mathbf{0.308 \pm 0.003}_{\Uparrow}$	$0.301 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.284 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.140 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.287 \pm 0.004^{\downarrow}$	$\mathbf{0.308 \pm 0.005}_{\Uparrow}$	0.661 ± 0.01
nus-wide	$\mathbf{0.213 \pm 0.001}^{\uparrow}_{\Uparrow}$	$0.207 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.176 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.094 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.192 \pm 0.001^{\downarrow}$	$0.210 \pm 0.001_{\Uparrow}$	0.449 ± 0.001
imdb	$0.080 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.096 \pm 0.002^{\uparrow}_{\Downarrow}$	$\mathbf{0.137 \pm 0.004}^{\uparrow}_{\Uparrow}$	$0.013 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.133 \pm 0.005^{\uparrow}$	$0.087 \pm 0.001_{\Downarrow}$	0.363 ± 0.007

Table 4.4: macro-f1. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

using the first-layer predictions. When considering the `0/1-loss` criterion, the *CC* method outperformed the other algorithms. This is due to its greedy approximation of the *joint* conditional mode (see Section 2.2) , which was shown to minimize this metric [12]. Note that minimizing this metric in `cal500` is difficult due to the large number of labels in this data set (see Table 4.1); predicting all 141 labels correctly for each example is hard to achieve considering also the high *LC*-value in this domain.

In general, the performance of both versions of the proposed technique is better in *dense* data sets (see Section 4.3.2). For example, in `emotion`, *PruDent* achieved the best results along all performance criteria. In domains with lower density, the algorithms were still able to perform well in aspects such as `hamming-loss`, `accuracy` and all the `micro` variants. A remarkable example of this is the `micro-f1` score of the sparse `slashdot` domain. The similar *DBR* method underperformed even *BR* in this domain, while *PruDent* was the best performer.

hamming-loss (lower is better)	Previous Techniques				Proposed Techniques		Minimum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.258\pm0.009^{\uparrow}_{\Uparrow}$	$0.255\pm0.01^{\uparrow}_{\Uparrow}$	$0.274\pm0.02^{\uparrow}_{\Uparrow}$	$0.261\pm0.02^{\uparrow}_{\Uparrow}$	0.245 ± 0.01	$\mathbf{0.241\pm0.01}$	0.187 ± 0.008
genbase	$\mathbf{0.002\pm0.001}$	$\mathbf{0.002\pm0.001}$	$\mathbf{0.002\pm0.001}$	$0.007\pm0.004^{\uparrow}_{\Uparrow}$	$\mathbf{0.002\pm0.001}$	$\mathbf{0.002\pm0.001}$	0.002 ± 0.001
yeast	$0.255\pm0.004^{\uparrow}_{\Downarrow}$	$0.273\pm0.005^{\uparrow}$	$0.289\pm0.005^{\uparrow}_{\Uparrow}$	$\mathbf{0.221\pm0.004}^{\downarrow}_{\Downarrow}$	$0.278\pm0.005^{\uparrow}$	$0.243\pm0.003_{\Downarrow}$	0.044 ± 0.001
scene	$0.139\pm0.004_{\Downarrow}$	$0.144\pm0.005^{\uparrow}_{\Downarrow}$	$0.181\pm0.01^{\uparrow}_{\Uparrow}$	$\mathbf{0.129\pm0.004}_{\Downarrow}$	$0.161\pm0.005^{\uparrow}$	$0.134\pm0.009_{\Downarrow}$	0.053 ± 0.005
cal500	$0.205\pm0.002^{\uparrow}_{\Uparrow}$	$0.219\pm0.004^{\uparrow}_{\Uparrow}$	$0.206\pm0.003^{\uparrow}_{\Uparrow}$	$\mathbf{0.189\pm0.01}^{\downarrow}_{\Downarrow}$	$0.196\pm0.003^{\downarrow}$	$0.199\pm0.003_{\Uparrow}$	0.145 ± 0.001
medical	$0.025\pm0.001^{\uparrow}_{\Uparrow}$	$0.025\pm0.002_{\Uparrow}$	$\mathbf{0.024\pm0.001}$	0.031 ± 0.01	$\mathbf{0.024\pm0.001}$	$\mathbf{0.024\pm0.001}$	0.018 ± 0.001
langlog	$0.044\pm0.002^{\uparrow}_{\Uparrow}$	$0.043\pm0.001_{\Uparrow}$	$0.044\pm0.001^{\uparrow}_{\Uparrow}$	$\mathbf{0.037\pm0.006}^{\downarrow}_{\Downarrow}$	$0.043\pm0.001^{\downarrow}$	$0.043\pm0.002_{\Uparrow}$	0.043 ± 0.001
enron	$0.067\pm0.002^{\uparrow}_{\Uparrow}$	$0.069\pm0.001^{\uparrow}_{\Uparrow}$	$0.078\pm0.002^{\uparrow}_{\Uparrow}$	$0.065\pm0.003_{\Downarrow}$	$0.071\pm0.002^{\uparrow}$	$\mathbf{0.064\pm0.001}_{\Downarrow}$	0.052 ± 0.002
slashdot	$0.056\pm0.001^{\uparrow}_{\Downarrow}$	$0.071\pm0.008^{\uparrow}_{\Downarrow}$	$0.266\pm0.03^{\uparrow}_{\Uparrow}$	$0.063\pm0.02_{\Downarrow}$	$0.228\pm0.03^{\uparrow}$	$\mathbf{0.055\pm0.001}_{\Downarrow}$	0.027 ± 0.002
mediamill	$0.108\pm0.001^{\uparrow}_{\Uparrow}$	$0.119\pm0.002^{\uparrow}_{\Uparrow}$	$0.122\pm0.001^{\uparrow}_{\Uparrow}$	$\mathbf{0.098\pm0.001}^{\downarrow}_{\Downarrow}$	$0.105\pm0.001^{\uparrow}$	$0.103\pm0.001_{\Downarrow}$	0.062 ± 0.001
tmc2007	$0.086\pm0.001^{\uparrow}_{\Uparrow}$	$0.087\pm0.001^{\uparrow}_{\Uparrow}$	$0.089\pm0.001^{\uparrow}_{\Uparrow}$	$0.088\pm0.002^{\uparrow}_{\Uparrow}$	$0.082\pm0.001^{\uparrow}$	$\mathbf{0.081\pm0.001}_{\Downarrow}$	0.076 ± 0.001
bibtex	$0.040\pm0.001^{\uparrow}_{\Uparrow}$	$0.040\pm0.001^{\uparrow}_{\Uparrow}$	$0.040\pm0.001_{\Uparrow}$	0.040 ± 0.009	$\mathbf{0.039\pm0.001}^{\downarrow}$	$0.040\pm0.001_{\Uparrow}$	0.039 ± 0.001
rcv1	$0.066\pm0.001^{\uparrow}_{\Uparrow}$	$0.068\pm0.001^{\uparrow}_{\Uparrow}$	$0.070\pm0.001^{\uparrow}_{\Uparrow}$	0.064 ± 0.005	$\mathbf{0.063\pm0.001}^{\downarrow}$	$0.064\pm0.001_{\Uparrow}$	0.026 ± 0.001
nus-wide	$0.119\pm0.001^{\uparrow}_{\Downarrow}$	$0.125\pm0.001^{\uparrow}_{\Downarrow}$	$0.234\pm0.004^{\uparrow}_{\Uparrow}$	$\mathbf{0.093\pm0.001}^{\downarrow}_{\Downarrow}$	$0.183\pm0.003^{\uparrow}$	$0.103\pm0.001_{\Downarrow}$	0.079 ± 0.001
imdb	$0.110\pm0.001^{\downarrow}_{\Downarrow}$	$0.120\pm0.003^{\uparrow}_{\Downarrow}$	$0.300\pm0.02^{\uparrow}_{\Uparrow}$	$\mathbf{0.104\pm0.02}_{\Downarrow}$	$0.295\pm0.02^{\uparrow}$	$0.113\pm0.001_{\Downarrow}$	0.078 ± 0.001

Table 4.5: hamming-loss. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

When comparing *PruDent* with *Conf-ST*, we can see that *Conf-ST* generally performs better in the `recall` metrics. This may be due to its covering a greater range of label dependencies and making a greater number of positive label predictions. However, this advantage comes at the cost of a serious decrease in `precision`, with the inevitable consequences for the `f1` measures. The pattern can be clearly seen in the *slashdot* data, where *PruDent* did not suffer from a decline in `f1`. This further confirms the claim that unnecessary label dependencies can negatively affect classifiers. We are lead to believe that *Conf-ST* is preferable in domains where false positives are more easily tolerated.

To conclude this section, note that *PruDent* was quite able to reach the performance ceiling of stacking in some occasions. For example, the performance in `bibtex` was close to that of the No-Error Stacking, *NE-ST*. Surprisingly, *PruDent* and *Conf-ST* surpassed even *NE-ST* in the `micro-recall` of `tmc2007`. Since *NE-ST*

0/1-loss (lower is better)	Previous Techniques				Proposed Techniques		Minimum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.832 \pm 0.01^{\uparrow}_{\Uparrow}$	0.773 ± 0.02	$0.836 \pm 0.02^{\uparrow}_{\Uparrow}$	$0.819 \pm 0.02^{\uparrow}_{\Uparrow}$	0.779 ± 0.01	$\mathbf{0.770 \pm 0.02}$	0.705 ± 0.02
genbase	0.023 ± 0.010	0.023 ± 0.010	$\mathbf{0.022 \pm 0.010}$	$0.087 \pm 0.06^{\uparrow}_{\Uparrow}$	0.023 ± 0.010	0.023 ± 0.010	0.022 ± 0.010
yeast	$0.946 \pm 0.006^{\uparrow}_{\Uparrow}$	$\mathbf{0.873 \pm 0.01}^{\downarrow}_{\Downarrow}$	$0.946 \pm 0.006^{\uparrow}_{\Uparrow}$	$0.934 \pm 0.009^{\uparrow}$	$0.938 \pm 0.009^{\uparrow}$	$0.916 \pm 0.008_{\Downarrow}$	0.387 ± 0.02
scene	$0.585 \pm 0.01^{\uparrow}_{\Uparrow}$	$\mathbf{0.488 \pm 0.02}^{\downarrow}_{\Downarrow}$	$0.575 \pm 0.01^{\uparrow}_{\Uparrow}$	$0.618 \pm 0.03^{\uparrow}_{\Uparrow}$	0.539 ± 0.01	0.542 ± 0.02	0.254 ± 0.02
cal500	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0
medical	$0.364 \pm 0.02^{\uparrow}_{\Uparrow}$	$0.348 \pm 0.02^{\downarrow}$	$0.356 \pm 0.02_{\Uparrow}$	0.445 ± 0.2	$\mathbf{0.345 \pm 0.01}^{\downarrow}$	$0.354 \pm 0.02_{\Uparrow}$	0.298 ± 0.02
langlog	$0.918 \pm 0.01_{\Uparrow}$	$0.915 \pm 0.01_{\Uparrow}$	$0.914 \pm 0.01_{\Uparrow}$	$0.967 \pm 0.03^{\uparrow}_{\Uparrow}$	$\mathbf{0.911 \pm 0.01}^{\downarrow}$	$0.917 \pm 0.01_{\Uparrow}$	0.913 ± 0.01
enron	$0.909 \pm 0.02^{\uparrow}_{\Uparrow}$	$\mathbf{0.874 \pm 0.02}^{\downarrow}_{\Downarrow}$	$0.907 \pm 0.02^{\uparrow}_{\Uparrow}$	$0.929 \pm 0.03^{\uparrow}_{\Uparrow}$	0.898 ± 0.02	0.891 ± 0.02	0.823 ± 0.02
slashdot	$0.737 \pm 0.009^{\uparrow}_{\Uparrow}$	$\mathbf{0.681 \pm 0.03}^{\downarrow}_{\Downarrow}$	$0.734 \pm 0.009^{\uparrow}_{\Uparrow}$	0.776 ± 0.08	$0.731 \pm 0.008^{\uparrow}$	$0.726 \pm 0.008_{\Downarrow}$	0.304 ± 0.03
mediamill	$0.927 \pm 0.001^{\uparrow}_{\Uparrow}$	$\mathbf{0.887 \pm 0.003}^{\downarrow}_{\Downarrow}$	$0.928 \pm 0.001^{\uparrow}_{\Uparrow}$	$0.913 \pm 0.003_{\Uparrow}$	$0.906 \pm 0.002^{\downarrow}$	$0.911 \pm 0.005_{\Uparrow}$	0.785 ± 0.002
tmc2007	$0.691 \pm 0.004^{\uparrow}_{\Uparrow}$	$0.670 \pm 0.005^{\uparrow}_{\Uparrow}$	$0.692 \pm 0.005^{\uparrow}_{\Uparrow}$	$0.723 \pm 0.01^{\uparrow}_{\Uparrow}$	$\mathbf{0.663 \pm 0.006}$	0.664 ± 0.003	0.638 ± 0.004
bibtex	$0.626 \pm 0.007_{\Uparrow}$	$0.621 \pm 0.008^{\downarrow}_{\Uparrow}$	$0.621 \pm 0.007^{\downarrow}_{\Uparrow}$	0.660 ± 0.1	$\mathbf{0.614 \pm 0.007}^{\downarrow}$	$0.627 \pm 0.007_{\Uparrow}$	0.615 ± 0.007
rcv1	$0.939 \pm 0.004^{\uparrow}_{\Uparrow}$	$\mathbf{0.859 \pm 0.007}^{\downarrow}_{\Downarrow}$	$0.931 \pm 0.003_{\Uparrow}$	$0.974 \pm 0.007^{\uparrow}_{\Uparrow}$	$0.913 \pm 0.003^{\downarrow}$	$0.931 \pm 0.007_{\Uparrow}$	0.623 ± 0.02
nus-wide	$0.924 \pm 0.001^{\uparrow}_{\Uparrow}$	$\mathbf{0.867 \pm 0.002}^{\downarrow}_{\Downarrow}$	$0.922 \pm 0.001^{\uparrow}_{\Uparrow}$	$0.924 \pm 0.006^{\uparrow}_{\Uparrow}$	$0.892 \pm 0.001^{\uparrow}$	$0.873 \pm 0.001_{\Downarrow}$	0.674 ± 0.004
imdb	$0.948 \pm 0.001^{\uparrow}_{\Uparrow}$	$\mathbf{0.906 \pm 0.006}^{\downarrow}_{\Downarrow}$	$0.954 \pm 0.001^{\uparrow}_{\Uparrow}$	$0.997 \pm 0.003^{\uparrow}_{\Uparrow}$	$0.945 \pm 0.001^{\uparrow}$	$0.924 \pm 0.001_{\Downarrow}$	0.704 ± 0.002

Table 4.6: 0/1-loss. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

is unpruned, its high performance implies the problem lies more in the errors of the class-features than in overfitting class dependencies.

The following is a summary of the acquired findings:

- *PruDent* improves upon *BR* in all measures except `macro-recall`.
- *Conf-ST* outperforms *BR* along `recall, 0/1-loss`, and `accuracy`.
- *PruDent* is better-suited for majority classes.
- In most domains, *PruDent* outperformed all other algorithms in terms of `macro-precision` and `micro-f1`.
- *CC* was the winner in most domains along `0/1-loss`; *DBR* was the best along `macro-recall`.
- *PruDent* and *2BR* were superior along `hamming-loss`.
- *Conf-ST* is better than *PruDent* in `recall`.

accuracy (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.438 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.463 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.416 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.419 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.481 \pm 0.01^{\downarrow}$	$\mathbf{0.487 \pm 0.01}_{\Uparrow}$	0.575 ± 0.01
genbase	$\mathbf{0.986 \pm 0.005}$	$\mathbf{0.986 \pm 0.006}$	$\mathbf{0.986 \pm 0.005}$	$0.942 \pm 0.05^{\downarrow}_{\Downarrow}$	$\mathbf{0.986 \pm 0.005}$	$\mathbf{0.986 \pm 0.005}$	0.986 ± 0.005
yeast	$0.423 \pm 0.01^{\downarrow}_{\Uparrow}$	$0.416 \pm 0.004^{\downarrow}$	$0.403 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.429 \pm 0.01^{\downarrow}$	$0.417 \pm 0.01^{\downarrow}$	$\mathbf{0.449 \pm 0.01}_{\Uparrow}$	0.880 ± 0.004
scene	$0.520 \pm 0.01^{\downarrow}_{\Downarrow}$	$\mathbf{0.574 \pm 0.02}^{\uparrow}$	$0.518 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.425 \pm 0.05^{\downarrow}_{\Downarrow}$	$0.567 \pm 0.01^{\uparrow}$	$0.552 \pm 0.02_{\Downarrow}$	0.817 ± 0.02
cal500	$0.217 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.218 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.220 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.221 \pm 0.01^{\downarrow}_{\Downarrow}$	$\mathbf{0.238 \pm 0.005}$	$\mathbf{0.238 \pm 0.006}$	0.412 ± 0.002
medical	$0.708 \pm 0.02^{\downarrow}_{\Downarrow}$	0.716 ± 0.02	$0.715 \pm 0.01_{\Downarrow}$	0.659 ± 0.09	$\mathbf{0.724 \pm 0.01}^{\uparrow}$	$0.714 \pm 0.02_{\Downarrow}$	0.772 ± 0.02
langlog	0.137 ± 0.02	0.139 ± 0.02	0.139 ± 0.02	$0.043 \pm 0.05^{\downarrow}_{\Downarrow}$	$\mathbf{0.141 \pm 0.02}$	0.137 ± 0.02	0.143 ± 0.02
enron	$0.387 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.409 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.391 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.388 \pm 0.03^{\downarrow}_{\Downarrow}$	0.424 ± 0.01	$\mathbf{0.430 \pm 0.01}$	0.529 ± 0.01
slashdot	$0.303 \pm 0.008^{\downarrow}_{\Downarrow}$	$\mathbf{0.364 \pm 0.03}^{\uparrow}$	$0.347 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.267 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.353 \pm 0.008^{\uparrow}$	$0.308 \pm 0.009_{\Downarrow}$	0.729 ± 0.03
mediamill	$0.443 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.440 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.424 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.450 \pm 0.004^{\downarrow}_{\Downarrow}$	$\mathbf{0.469 \pm 0.001}^{\uparrow}$	$0.466 \pm 0.004_{\Downarrow}$	0.661 ± 0.001
tmc2007	$0.576 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.581 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.572 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.551 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.610 \pm 0.004}^{\uparrow}$	$0.605 \pm 0.002_{\Downarrow}$	0.623 ± 0.003
bibtex	$0.522 \pm 0.006_{\Downarrow}$	$0.526 \pm 0.007^{\uparrow}_{\Downarrow}$	$0.528 \pm 0.007^{\uparrow}_{\Downarrow}$	$0.469 \pm 0.05^{\downarrow}_{\Downarrow}$	$\mathbf{0.536 \pm 0.005}^{\uparrow}$	$0.523 \pm 0.007_{\Downarrow}$	0.536 ± 0.007
rcv1	$0.255 \pm 0.004_{\Uparrow}$	$\mathbf{0.272 \pm 0.007}^{\uparrow}_{\Uparrow}$	$0.225 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.172 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.242 \pm 0.004^{\downarrow}$	$0.256 \pm 0.005_{\Uparrow}$	0.619 ± 0.01
nus-wide	$0.237 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.270 \pm 0.003^{\downarrow}_{\Uparrow}$	$0.222 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.177 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.261 \pm 0.001^{\downarrow}$	$\mathbf{0.284 \pm 0.001}_{\Uparrow}$	0.551 ± 0.002
imdb	$0.137 \pm 0.001^{\downarrow}_{\Downarrow}$	$\mathbf{0.197 \pm 0.01}_{\Uparrow}$	$0.171 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.009 \pm 0.009^{\downarrow}_{\Downarrow}$	$0.177 \pm 0.002^{\downarrow}$	$0.188 \pm 0.002_{\Uparrow}$	0.479 ± 0.002

Table 4.7: accuracy. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

4.5 Beyond Performance Criteria

Having explored the performance of the techniques, let us turn our attention to some other behavioral aspects.

4.5.1 Computational Time

Section 4.2.1 tell us that *PruDent* constructs a secondary classifier only if a class is related to others. As an added bonus, this also reduces the computational complexity of the algorithm: when a class is deemed independent from all others, no dependent models are constructed for it; instead, classifications of its respective independent classifier are used. To illustrate the amount of complexity reduction, the graph in Figure 4.13 shows the induction and testing wall times of the algorithms on a sample of the datasets. The experiments were run using 10 threads of the Intel(R) Xeon(R) E5-2670 2.60GHz processor. Whenever possible, the algorithms were parallelized

micro-precision (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.598 \pm 0.02^{\downarrow}$	$0.603 \pm 0.02^{\downarrow}$	$0.571 \pm 0.03^{\downarrow}_{\Downarrow}$	0.589 ± 0.04	0.612 ± 0.02	$\mathbf{0.620 \pm 0.03}$	0.711 ± 0.02
genbase	$\mathbf{0.996 \pm 0.001}$	$\mathbf{0.996 \pm 0.001}$	$\mathbf{0.996 \pm 0.001}$	0.985 ± 0.02	$\mathbf{0.996 \pm 0.001}$	$\mathbf{0.996 \pm 0.001}$	0.996 ± 0.001
yeast	$0.669 \pm 0.01_{\Uparrow}$	$0.639 \pm 0.006^{\downarrow}_{\Uparrow}$	$0.615 \pm 0.01^{\downarrow}_{\Downarrow}$	$\mathbf{0.724 \pm 0.02}^{\uparrow}_{\Uparrow}$	$0.625 \pm 0.01^{\downarrow}$	$0.676 \pm 0.01_{\Uparrow}$	0.952 ± 0.003
scene	$0.603 \pm 0.02_{\Uparrow}$	$0.576 \pm 0.02^{\downarrow}_{\Uparrow}$	$0.462 \pm 0.03^{\downarrow}_{\Downarrow}$	$\mathbf{0.742 \pm 0.04}^{\uparrow}_{\Uparrow}$	$0.506 \pm 0.02^{\downarrow}$	$0.619 \pm 0.03_{\Uparrow}$	0.858 ± 0.01
cal500	$0.539 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.493 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.528 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.529 \pm 0.04^{\downarrow}$	$0.550 \pm 0.008^{\downarrow}$	$\mathbf{0.554 \pm 0.01}_{\Uparrow}$	0.680 ± 0.007
medical	0.848 ± 0.02	0.847 ± 0.02	$\mathbf{0.854 \pm 0.02}_{\Uparrow}$	0.756 ± 0.2	$0.849 \pm 0.02^{\downarrow}$	$0.851 \pm 0.02_{\Uparrow}$	0.909 ± 0.01
langlog	$0.317 \pm 0.04_{\Downarrow}$	$0.323 \pm 0.04_{\Downarrow}$	$0.317 \pm 0.04_{\Downarrow}$	$\mathbf{0.371 \pm 0.3}$	$0.343 \pm 0.04^{\uparrow}$	$0.328 \pm 0.04_{\Downarrow}$	0.330 ± 0.04
enron	$0.663 \pm 0.01_{\Uparrow}$	$0.644 \pm 0.02_{\Uparrow}$	$0.575 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.640 \pm 0.04_{\Uparrow}$	$0.605 \pm 0.01^{\downarrow}$	$\mathbf{0.668 \pm 0.02}_{\Uparrow}$	0.793 ± 0.01
slashdot	$0.616 \pm 0.02^{\downarrow}_{\Uparrow}$	$0.338 \pm 0.1^{\downarrow}_{\Uparrow}$	$0.145 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.565 \pm 0.2_{\Uparrow}$	$0.154 \pm 0.009^{\downarrow}$	$\mathbf{0.647 \pm 0.02}_{\Uparrow}$	0.915 ± 0.03
mediamill	$0.793 \pm 0.003^{\downarrow}_{\Uparrow}$	$0.766 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.752 \pm 0.003^{\downarrow}_{\Downarrow}$	$\mathbf{0.808 \pm 0.008}^{\uparrow}_{\Uparrow}$	$0.777 \pm 0.002^{\downarrow}$	$0.796 \pm 0.004_{\Uparrow}$	0.881 ± 0.002
tmc2007	$\mathbf{0.738 \pm 0.003}^{\uparrow}_{\Uparrow}$	$0.734 \pm 0.002_{\Uparrow}$	$0.723 \pm 0.005^{\downarrow}$	$0.715 \pm 0.02^{\downarrow}$	$0.725 \pm 0.005^{\downarrow}$	$0.735 \pm 0.002_{\Uparrow}$	0.774 ± 0.004
bibtex	$\mathbf{0.829 \pm 0.007}_{\Uparrow}$	0.828 ± 0.007	$0.826 \pm 0.008_{\Uparrow}$	0.788 ± 0.2	$0.822 \pm 0.010^{\downarrow}$	$0.828 \pm 0.007_{\Uparrow}$	0.836 ± 0.007
rcv1	$0.451 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.437 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.410 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.369 \pm 0.1^{\downarrow}_{\Downarrow}$	0.466 ± 0.008	$\mathbf{0.472 \pm 0.01}$	0.870 ± 0.007
nus-wide	$0.481 \pm 0.002^{\downarrow}_{\Uparrow}$	$0.458 \pm 0.003^{\downarrow}_{\Uparrow}$	$0.261 \pm 0.004^{\downarrow}_{\Downarrow}$	$\mathbf{0.616 \pm 0.01}^{\uparrow}_{\Uparrow}$	$0.344 \pm 0.004^{\downarrow}$	$0.544 \pm 0.001_{\Uparrow}$	0.693 ± 0.002
imdb	$0.372 \pm 0.002^{\downarrow}_{\Uparrow}$	$0.342 \pm 0.02^{\downarrow}_{\Uparrow}$	$0.182 \pm 0.008^{\downarrow}_{\Downarrow}$	$\mathbf{0.414 \pm 0.2}_{\Uparrow}$	$0.182 \pm 0.008^{\downarrow}$	$0.373 \pm 0.002_{\Uparrow}$	0.693 ± 0.002

Table 4.8: micro-precision. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

by inducing each binary classifier on a separate thread and submitting all testing examples to classifiers simultaneously.

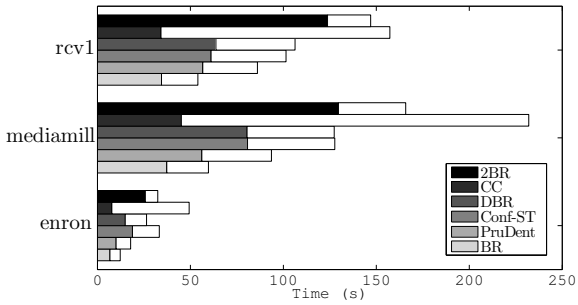


Figure 4.13: Induction times (shaded) and testing times (white) of the competing algorithms.

In terms of training, *CC* and *BR* take the least amount of time because they only train one classifier per class. The *DBR* and *Conf-ST* algorithms train two classifiers per class—they take twice as much time. As expected, *PruDent* takes less time than the other *stacking* methods because it prunes unnecessary dependencies. Since *2BR*

micro-recall (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.591 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.586 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.574 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.554 \pm 0.05^{\downarrow}_{\Downarrow}$	0.645 ± 0.02	$\mathbf{0.651 \pm 0.03}$	0.698 ± 0.02
genbase	$\mathbf{0.994 \pm 0.004}$	$\mathbf{0.994 \pm 0.004}$	$\mathbf{0.994 \pm 0.004}$	$0.975 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.994 \pm 0.004}$	$\mathbf{0.994 \pm 0.004}$	0.994 ± 0.004
yeast	$0.686 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.643 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.701 \pm 0.03^{\downarrow}_{\Downarrow}$	0.686 ± 0.03	0.707 ± 0.03	$\mathbf{0.708 \pm 0.03}$	0.951 ± 0.002
scene	$0.591 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.596 \pm 0.02_{\Downarrow}$	$0.633 \pm 0.02^{\uparrow}_{\Downarrow}$	$0.424 \pm 0.06^{\downarrow}_{\Downarrow}$	$\mathbf{0.669 \pm 0.02}^{\uparrow}$	$0.611 \pm 0.02_{\Downarrow}$	0.841 ± 0.02
cal500	$0.467 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.464 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.460 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.473 \pm 0.04^{\downarrow}$	$0.500 \pm 0.01^{\downarrow}$	$\mathbf{0.510 \pm 0.01}_{\Uparrow}$	0.666 ± 0.005
medical	$0.864 \pm 0.01_{\Downarrow}$	$0.864 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.871 \pm 0.02_{\Downarrow}$	0.876 ± 0.02	$\mathbf{0.883 \pm 0.02}^{\uparrow}$	$0.872 \pm 0.02_{\Downarrow}$	0.925 ± 0.008
langlog	$0.232 \pm 0.04^{\uparrow}$	0.234 ± 0.03	0.234 ± 0.03	$0.083 \pm 0.1^{\downarrow}_{\Downarrow}$	$\mathbf{0.235 \pm 0.03}$	0.229 ± 0.03	0.241 ± 0.03
enron	$0.619 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.627 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.675 \pm 0.02^{\downarrow}_{\Downarrow}$	0.704 ± 0.05	$\mathbf{0.730 \pm 0.01}^{\uparrow}$	$0.702 \pm 0.02_{\Downarrow}$	0.752 ± 0.01
slashdot	$0.369 \pm 0.010_{\Downarrow}$	$0.438 \pm 0.04^{\uparrow}_{\Downarrow}$	$\mathbf{0.761 \pm 0.03}^{\uparrow}_{\Uparrow}$	$0.343 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.726 \pm 0.03^{\uparrow}$	$0.366 \pm 0.01_{\Downarrow}$	0.732 ± 0.01
mediamill	$0.735 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.731 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.739 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.748 \pm 0.009^{\downarrow}_{\Downarrow}$	$\mathbf{0.781 \pm 0.002}^{\uparrow}$	$0.770 \pm 0.003_{\Downarrow}$	0.873 ± 0.001
tmc2007	$0.726 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.723 \pm 0.006^{\downarrow}_{\Downarrow}$	$0.722 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.743 \pm 0.04^{\downarrow}_{\Downarrow}$	$\mathbf{0.796 \pm 0.005}^{\uparrow}$	$0.788 \pm 0.003_{\Downarrow}$	0.757 ± 0.003
bibtex	$0.727 \pm 0.004^{\downarrow}_{\Downarrow}$	$0.730 \pm 0.004_{\Downarrow}$	$0.732 \pm 0.006^{\uparrow}_{\Downarrow}$	$0.676 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.740 \pm 0.004}^{\uparrow}$	$0.728 \pm 0.005_{\Downarrow}$	0.739 ± 0.006
rcv1	$\mathbf{0.390 \pm 0.005}^{\uparrow}_{\Uparrow}$	$0.375 \pm 0.01_{\Uparrow}$	$0.313 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.291 \pm 0.05^{\downarrow}_{\Downarrow}$	$0.331 \pm 0.008^{\downarrow}$	$0.379 \pm 0.009_{\Uparrow}$	0.804 ± 0.006
nus-wide	$0.438 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.436 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.515 \pm 0.003^{\uparrow}_{\Downarrow}$	$0.356 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.548 \pm 0.003}^{\uparrow}$	$0.481 \pm 0.002_{\Downarrow}$	0.667 ± 0.001
imdb	$0.232 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.296 \pm 0.03^{\downarrow}_{\Downarrow}$	$\mathbf{0.587 \pm 0.005}^{\uparrow}_{\Uparrow}$	$0.008 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.581 \pm 0.007^{\uparrow}$	$0.325 \pm 0.004_{\Downarrow}$	0.544 ± 0.002

Table 4.9: micro-recall. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

has a more sophisticated training process (see Section 4.3.1), its induction time is longer than that of the others.

During testing, the stacking methods take roughly twice as much time as *BR*. However, the *CC* method exhibits longer testing times because, unlike the stacking methods, it cannot be parallelized during testing (see Section 3.3.3).

4.5.2 Remaining Dependencies

To help us understand the impact of the remaining dependencies, Figure 4.14 shows the remaining relationships in the `emotion` domain, using the best *IG* threshold from Figure 4.11c (since `micro-f1` was the target criterion when choosing ϕ in the experiments). The solid lines indicate direct concept correlations, and the dashed lines indicate inverse correlations. Classes that are not connected by any lines are deemed unnecessary. Note that the same pairs of dependent classes were reproduced using PCC (Section 4.1.1) with $\phi = 0.25$ in order to reveal the signs of the rela-

micro-f1 (higher is better)	Previous Techniques				Proposed Techniques		Maximum
	BR	*CC*	*DBR*	*2BR*	*Conf-ST*	*PruDent*	*NE-ST*
emotion	$0.594 \pm 0.03^{\downarrow}_{\Downarrow}$	$0.594 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.571 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.568 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.628 \pm 0.01^{\downarrow}$	$\mathbf{0.635 \pm 0.02}_{\Uparrow}$	0.704 ± 0.009
genbase	$\mathbf{0.995 \pm 0.002}$	$\mathbf{0.995 \pm 0.002}$	$\mathbf{0.995 \pm 0.002}$	$0.980 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.995 \pm 0.002}$	$\mathbf{0.995 \pm 0.002}$	0.995 ± 0.002
yeast	$0.677 \pm 0.02^{\downarrow}_{\Uparrow}$	$0.641 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.655 \pm 0.02^{\downarrow}_{\Downarrow}$	$\mathbf{0.703 \pm 0.008}^{\uparrow}_{\Uparrow}$	$0.663 \pm 0.02^{\downarrow}$	$0.691 \pm 0.02_{\Uparrow}$	0.952 ± 0.002
scene	$0.597 \pm 0.01^{\downarrow}_{\Uparrow}$	$0.585 \pm 0.01^{\downarrow}$	$0.534 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.536 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.576 \pm 0.01^{\downarrow}$	$\mathbf{0.614 \pm 0.01}_{\Uparrow}$	0.849 ± 0.01
cal500	$0.500 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.478 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.491 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.498 \pm 0.02^{\downarrow}_{\Downarrow}$	$0.524 \pm 0.009^{\downarrow}$	$\mathbf{0.531 \pm 0.01}_{\Uparrow}$	0.673 ± 0.006
medical	$0.856 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.855 \pm 0.010_{\Downarrow}$	0.863 ± 0.010	0.791 ± 0.2	$\mathbf{0.865 \pm 0.007}^{\uparrow}$	$0.861 \pm 0.008_{\Downarrow}$	0.917 ± 0.005
langlog	$0.268 \pm 0.04_{\Downarrow}$	0.271 ± 0.03	$0.269 \pm 0.03_{\Downarrow}$	$0.096 \pm 0.10^{\downarrow}_{\Downarrow}$	$\mathbf{0.278 \pm 0.03}$	0.269 ± 0.04	0.278 ± 0.03
enron	$0.640 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.635 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.621 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.667 \pm 0.01^{\downarrow}$	$0.661 \pm 0.010^{\downarrow}$	$\mathbf{0.684 \pm 0.01}_{\Uparrow}$	0.772 ± 0.007
slashdot	$0.462 \pm 0.01_{\Uparrow}$	$0.365 \pm 0.06^{\downarrow}_{\Uparrow}$	$0.243 \pm 0.007^{\downarrow}_{\Downarrow}$	$0.406 \pm 0.1_{\Uparrow}$	$0.254 \pm 0.01^{\downarrow}$	$\mathbf{0.467 \pm 0.01}_{\Uparrow}$	0.813 ± 0.02
mediamill	$0.763 \pm 0.001^{\downarrow}_{\Downarrow}$	$0.748 \pm 0.005^{\downarrow}_{\Downarrow}$	$0.746 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.777 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.779 \pm 0.001^{\downarrow}$	$\mathbf{0.783 \pm 0.001}_{\Uparrow}$	0.877 ± 0.001
tmc2007	$0.732 \pm 0.002^{\downarrow}_{\Downarrow}$	$0.728 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.722 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.728 \pm 0.01^{\downarrow}_{\Downarrow}$	0.759 ± 0.003	$\mathbf{0.760 \pm 0.002}$	0.766 ± 0.003
bibtex	$0.775 \pm 0.003_{\Downarrow}$	$0.776 \pm 0.004_{\Downarrow}$	$0.776 \pm 0.003_{\Downarrow}$	0.700 ± 0.2	$\mathbf{0.779 \pm 0.003}^{\uparrow}$	$0.775 \pm 0.003_{\Downarrow}$	0.785 ± 0.004
rcv1	$0.418 \pm 0.003_{\Uparrow}$	$0.404 \pm 0.010^{\downarrow}_{\Uparrow}$	$0.355 \pm 0.008^{\downarrow}_{\Downarrow}$	$0.310 \pm 0.04^{\downarrow}_{\Downarrow}$	$0.387 \pm 0.006^{\downarrow}$	$\mathbf{0.420 \pm 0.005}_{\Uparrow}$	0.835 ± 0.005
nus-wide	$0.458 \pm 0.001^{\downarrow}_{\Uparrow}$	$0.446 \pm 0.004^{\downarrow}_{\Uparrow}$	$0.347 \pm 0.003^{\downarrow}_{\Downarrow}$	$0.451 \pm 0.009^{\downarrow}_{\Uparrow}$	$0.423 \pm 0.003^{\downarrow}$	$\mathbf{0.511 \pm 0.002}_{\Uparrow}$	0.680 ± 0.001
imdb	$0.286 \pm 0.002^{\downarrow}_{\Uparrow}$	$0.317 \pm 0.03^{\downarrow}_{\Uparrow}$	$0.277 \pm 0.009^{\downarrow}$	$0.015 \pm 0.01^{\downarrow}_{\Downarrow}$	$0.277 \pm 0.009^{\downarrow}$	$\mathbf{0.347 \pm 0.003}_{\Uparrow}$	0.610 ± 0.002

Table 4.10: micro-f1. $^{\uparrow}$ or $^{\downarrow}$ indicate significantly higher or lower than *PruDent* respectively. $_{\Uparrow}$ and $_{\Downarrow}$ indicate significantly higher or lower than *Conf-ST* respectively.

tionships. As demonstrated by the diagram, the remaining relations conform with common expectations; the *'SadLonely'* class rarely co-occurs with *'HappyPleased'*, the *'RelaxingCalm'* class is arguably related to *'QuietStill'*, and not all occurrences of *'AmazedSurprised'* are also considered *'HappyPleased'*.

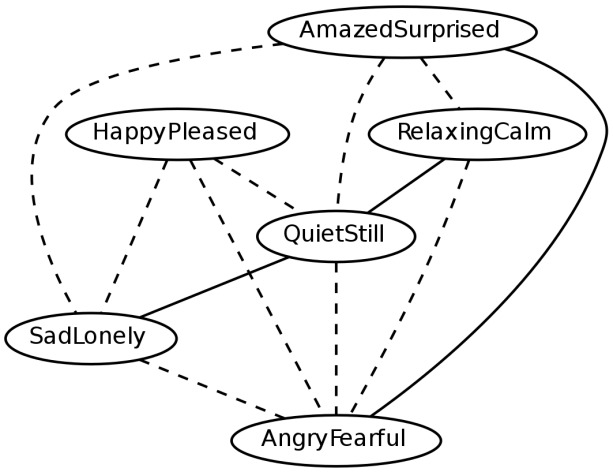


Figure 4.14: Remaining `emotion` relations with $\phi = 0.05$

	PruDent				*Conf-ST*			
	% Classified	% Unchanged	% Corrected	% Harmed	% Classified	% Unchanged	% Corrected	% Harmed
emotion	57.4 ± 6.6	41.7 ± 7.6	8.7 ± 0.8	6.9 ± 1.2	58.3 ± 6.2	42.3 ± 6.9	8.6 ± 0.8	7.4 ± 1.02
yeast	76.4 ± 1.5	59.2 ± 2.6	9.2 ± 0.8	8.0 ± 0.7	90.7 ± 3.2	67.0 ± 2.9	10.7 ± 0.7	13.0 ± 0.9
scene	62.8 ± 23.3	56.0 ± 21.0	3.6 ± 1.4	3.1 ± 1.8	71.1 ± 7.4	57.6 ± 7.7	5.6 ± 0.4	7.9 ± 0.7
cal500	52.3 ± 9.9	43.4 ± 8.7	4.8 ± 0.5	4.2 ± 0.6	75.2 ± 0.05	64.0 ± 0.3	6.1 ± 0.1	5.2 ± 0.2
medical	57.6 ± 13.4	57.2 ± 13.3	0.2 ± 0.1	0.2 ± 0.1	97.6 ± 1.4	97.1 ± 1.4	0.3 ± 0.1	0.2 ± 0.1
slashdot	43.1 ± 15.3	42.9 ± 15.2	0.2 ± 0.1	0.1 ± 0.1	92.9 ± 4.4	70.4 ± 3.9	2.6 ± 0.3	19.8 ± 3.4
mediamill	52.1 ± 13.4	47.9 ± 12.5	2.3 ± 0.5	1.8 ± 0.4	81.9 ± 0.8	75.4 ± 0.8	3.4 ± 0.1	3.1 ± 0.1
bibtex	40.7 ± 42.9	40.5 ± 42.7	0.07 ± 0.08	0.06 ± 0.07	93.8 ± 3.0	93.1 ± 3.0	0.4 ± 0.1	0.3 ± 0.04
nus-wide	57.9 ± 1.1	51.5 ± 1.2	4.0 ± 0.03	2.4 ± 0.03	68.7 ± 0.6	53.2 ± 0.5	4.6 ± 0.03	11.0 ± 0.3

Table 4.11: Percent of *test* examples classified using dependent models.

4.5.3 Percent of Dependently Classified Examples

Section 4.1.2 explained how the proposed technique limits *error-propagation* by re-using independent models whenever they have higher classification confidence than their dependent counterparts. Let us now take a closer look. Specifically, we would like to know the percentage of testing examples classified using dependent models, and to know also the percentage of classifications that were corrected. To this end, another experiment was run which measures the number of dependently-classified testing examples, and noted their independent, and dependent classifiers' predictions. Table 4.11 shows a sample of the averaged values obtained by 5x2 CV.

The results indicate that *PruDent* used the dependent classifiers more than 50% of the time in 10 out of the 15 data sets. However, the algorithm looks rather conservative when compared to *Conf-ST*. Although *Conf-ST* tends to use the dependent classifiers more liberally, the algorithm can produce more harmful classifications than correct ones; this happened more conspicuously in `slashdot` and `nus-wide`. Unlike *Conf-ST*, the *PruDent* version always corrected more incorrect classifications than vice versa. To end this section, it is worth noting that although *PruDent* was successful more often than not in employing the second layer of classifiers, it still left

room for improvement; harmed classifications occupy a considerable portion of test examples in some datasets (e.g. `yeast` and `emotion`). The reason why errors still occur in this paradigm is a topic worth investigating in future research.

CHAPTER 5

Alternative "PruDent" Paradigms

The previous chapter introduced the stacking-based algorithm *PruDent* and demonstrated its classification performance when the *C4.5* decision tree learner is applied to the binary subproblems. This chapter is composed of two parts that address the following questions: first, can *PruDent* be improved further by modifying the *m-estimate* parameters (p and m) that are used to estimate prediction confidence? Furthermore, how can *PruDent* be applied to a base classifier that does not provide probabilistic confidence values, such as *Support Vector Machines*? In an attempt to answer these questions, this chapter will suggest some possible solutions.

5.1 Optimizing the M-estimate Parameters

Recall from Section 4.1.2 that *PruDent* compares the prediction confidences of the first- and second-layer classifiers in stacking to select the class label that is less likely to be error-prone. To estimate these confidence values, the first experiments employed the *Laplace* smoothing method per Section 3.1.2. However, despite showing improved classification accuracy, some datasets were improved only when the *Laplace* method was later replaced with the *m-estimate* technique (see Section 4.4.3). Specifically,

the latter approach can bias the classifier such that it favors predicting the minority classes, which attained a better balance between the `precision` and `recall` criteria from Section 2.3. For the reader's convenience, the mathematical formulation of the *m-estimate* method is repeated below. Here, N denotes the number of examples in the leaf node in which the confidence score is estimated from, and N_p refers to the number of these examples that are instances of the positive class.

$$P(\oplus) = \frac{N_p + p.m}{N + m} \tag{5.1}$$

In the experiments of the previous chapter, the parameters p and m were fixed to the values of $\frac{2}{3}$ and 3 respectively. This means that minority classes are going to be given a higher preference by setting their *prior* to $\frac{2}{3}$ despite typically occurring in a small proportion of the data. In addition, setting the m parameter to the value of 3 gives a very small precedence to the selected prior; it will only affect the probability estimate if the sample size it is drawn from (in the leaf node of the tree) is very small. While these parameter values gave good classification performance, choosing the same fixed values may not necessarily be optimal for all domains. For instance, when a class is extremely rare, over-biasing the classifier towards predicting the minor class may reduce `precision` more than in a less imbalanced distribution. Moreover, the decrease in `precision` can outweigh the gain in `recall` which ultimately decreases their harmonic mean, `f1`. Thus, choosing the *m-estimate* parameter values can perhaps be done by looking at the characteristics of the target domain. The next sections discuss two possible heuristics which set the values of the parameters p and m based on the dataset at hand.

5.1.1 Choosing the Value of M

There are several ways to select the p and m parameters to cater to a specific dataset. Let us first consider the m parameter. As introduced in [45], the value of the m parameter is said to be domain dependent. Specifically, the author suggested that this parameter should be based on the level of noise in the dataset. Thus, when the data are noisy, choosing a higher value of m will yield a more conservative confidence estimate. However, the information about the noise in a given domain is not provided beforehand. In fact, determining the amount of noise in a dataset is a difficult problem [90]. To quantify the noise level, a commonly accepted heuristic is to train a classifier on the dataset and then observe the rate of incorrectly classified examples [91, 92]. The assumption is that when the dataset is perfectly separable, the noise level is expected to be low and the accuracy of the classifier is expected to be high. The author of the employed decision tree algorithm (*C4.5*) illustrated a similar scenario in [43]; as the level of noise increases, the classification accuracy is expected to decrease. Based on the evidence provided by the above literary works, it makes sense to choose the value of m in accordance with the observed error rate of the classifier. However, the error rate by itself provides us with a mere fraction of the entire dataset and does not give us any concrete values. This means that we need to establish an upper limit for the m parameter value.

The m parameter is concerned with the sample size in which a probability estimate is drawn from. On the one hand, choosing an m value that is equal to the number of all examples in the dataset will affect all leaf nodes in the decision tree. On the other hand, if we assume that examples are distributed evenly in the leaf nodes, choosing a

very small value of m will have little to no effect when a dataset is very large. Note that, this will be the case even when a class is rare in a large data set. For example, suppose a dataset contains $10,000$ training examples, of which 200 (i.e. 2%) belong to a given class. Here, selecting a value of $m = 3$ will not have a significant effect on the probability estimates especially if examples are distributed evenly among the leaves. Thus, one should consider the number of examples in the training set when choosing the value of m.

Another important aspect that is worth considering is how the training examples are distributed in the decision tree. If a tree contains only a few leaf nodes, we expect the representation of examples to be large in the leaf nodes. This is the case when a tree can easily separate the data. Conversely, in a more difficult problem, we expect a higher number of leaf nodes to be present in the tree, which leads to a smaller number of training instances in each leaf node.

Based on the above remarks, it is sensible to select an upper value of m that is going to be dependent on: (i) the number of examples in the dataset and, (ii) the number of leaf nodes in the tree. Following these guidelines, the upper limit of m can be set to the average number of examples in the leaf nodes of the decision tree as follows:

$$\texttt{Maximum m} = \frac{\texttt{number of examples}}{\texttt{number of leaf nodes}} \tag{5.2}$$

The above suggested limit provides us with the maximum value to be used for m under the assumption that examples are distributed evenly in the tree. Choosing the final concrete values will be discussed by the experiments setup in Section 5.1.3.

5.1.2 Choosing the Value of P

In terms of the parameter p (the *prior* belief), the introducing author of the *m-estimate* method [45] suggested the choice of p to be equal to the proportion of positive examples in the dataset. However, the results of the previous chapter (see Section 4.4.3) tell us that this approach will not necessarily guarantee a balanced distribution of the predicted classes, especially when the class representations are imbalanced. Therefore, a different method is needed to boost the prediction confidence of the rare classes whenever they are encountered.

When choosing the value of p, a trade-off is commonly faced between the `recall` and `precision` rates (see Section 2.3). One the one hand, choosing a small p will make the classifier more reluctant to label examples as positive instances. Doing so may increase `precision` at the cost of a possible decrease in `recall`. On the other hand, choosing a high p value will bias the classifier towards preferring positive class labels. This leads to an increase in `recall` with the drawback being a possible decrease in `precision`. When the classes have an imbalanced representation in the dataset, which is the circumstance typically faced in multi-label domains, most classifiers are biased towards predicting the class that constitutes a majority of examples. This happens because the goal of most classifiers is to reduce the expected error (e.g. [7]). To counter this behavior, we would like the p value to give a higher preference to the classes that constitute a minority of examples.

One may be inclined to set the value of p based on the proportion of positive examples for a given class in the training set, $P^+ = \frac{N_p}{N}$, where N_p is the number of positive examples and N is the total number of examples in the training set. Thus,

a lower ratio of positive examples P^+ will correspond to a higher p value. One such solution would be to assign the value of p to $1 - P^+$. However, using this scheme can lead to a serious consequence when a class is extremely rare (i.e. $< 5\%$) in the dataset; a situation that is not uncommon in multi-label domains. Consider the case where a class is represented by 2% of the instances in a dataset. Accordingly, using $p = 1 - P^+$ will set the value of p to 0.98. While having a high p value may guarantee the highest possible `recall` the classifier is able to attain, choosing such a high value when a class is extremely rare may dramatically reduce `precision` due to the higher possibility of false positive predictions.

The above realization leads us to the following important question: *how high can the p value be safely set?* Ideally, we would like the classifier to have a stronger bias towards positive instances when they have a reasonable presence in the dataset. To achieve a safe setting for p, the following approach is considered:

$$p = \min\{1,\ \frac{1}{2} + \frac{Np}{N}\} \tag{5.3}$$

In the above formula, when a class is extremely rare (e.g 2%), the p value will only have a very small bias towards the positive class (e.g. $p = 0.52$). Also, when a class has a higher presence in the dataset (e.g. 30%), the p value will correspond to a higher bias (e.g. $p = 0.8$). Note that when the positive examples are close to 50%, the algorithm will almost always prefer the positive class labels. This should not affect the `f1` criteria since the negative class labels will have a smaller influence on the outcome; `precision` will suffer less from false positives as the number of negative examples decreases. Whether using the above heuristic for the parameter p leads to classification improvements will be established experimentally by the next section.

5.1.3 Experiments

The previous sections provided a guideline to choosing the p and m parameters of the *m-estimate* technique employed by *PruDent*. In the next round of experiments, we would like to know whether using these heuristics does improve its classification performance. To this end, the original *PruDent* (with $p = \frac{2}{3}$ and $m = 3$) is compared with two other configurations as follows. In the first version (*PruDent-A*), the m parameter is set to half of the upper limit that was established in Section 5.1.1, and p is set per Equation 5.3 from the preceding section.

Additionally, we would like to know whether basing the m parameter on the error rate of the classifiers (as suggested by [45]) does improve the classifier's outcome. To test this hypothesis, the second compared version (*PruDent-B*) will set the m parameter as follows. First, the error rate of each classifier is recorded using a validation set process that represents 1/3 of the training set. These error rates are then used to calculate m for each class (in each layer) using the following formula: $m = 2 \times error \times \frac{number\ of\ examples}{number\ of\ leaves}$. Thus, a higher error rate will correspond to a higher m value. Note that the error rate is multiplied by 2 since the maximum error a classifier can have is 50%, or 0.5.

The conducted experiments followed the same setup from Section 4.3.1 where the datasets from Table 4.1 are used, and the pruning thrshold ϕ is optimized for **micro-f1**. Since the goal of the experiments is to show whether changing the p and m parameters achieves a better balance between `recall` and `precision`, the results of the experiments are reported using the relevant `macro-f1` and `micro-f1` criteria from Section 2.3.

macro-f1 (higher is better)	Original *PruDent*	Non-Error Based m *PruDent-A*	Error-based m *PruDent-B*
emotion	0.62 ± 0.02	0.62 ± 0.02	0.62 ± 0.02
genbase	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
yeast	0.391 ± 0.008	$\mathbf{0.400 \pm 0.006}^{\uparrow}$	$\mathbf{0.400 \pm 0.006}^{\uparrow}$
scene	0.63 ± 0.01	$\mathbf{0.641 \pm 0.009}^{\uparrow}$	0.64 ± 0.01
cal500	0.161 ± 0.002	$0.173 \pm 0.001^{\uparrow}$	$\mathbf{0.177 \pm 0.001}^{\uparrow}$
medical	0.54 ± 0.03	0.54 ± 0.03	0.54 ± 0.03
langlog	0.11 ± 0.01	0.11 ± 0.01	0.11 ± 0.01
enron	$\mathbf{0.195 \pm 0.008}$	$0.189 \pm 0.007^{\downarrow}$	$0.190 \pm 0.007^{\downarrow}$
slashdot	0.24 ± 0.01	0.24 ± 0.01	0.24 ± 0.01
mediamill	$\mathbf{0.319 \pm 0.005}$	0.314 ± 0.008	0.314 ± 0.006
tmc2007	$\mathbf{0.634 \pm 0.004}$	$\mathbf{0.634 \pm 0.004}$	$0.632 \pm 0.004^{\downarrow}$
bibtex	0.494 ± 0.009	$\mathbf{0.495 \pm 0.009}$	$\mathbf{0.495 \pm 0.010}$
rcv1	$\mathbf{0.304 \pm 0.006}$	0.30 ± 0.01	$0.300 \pm 0.009^{\downarrow}$
nus-wide	$\mathbf{0.209 \pm 0.001}$	$0.207 \pm 0.001^{\downarrow}$	$0.203 \pm 0.006^{\downarrow}$
imdb	0.086 ± 0.001	$\mathbf{0.094 \pm 0.001}^{\uparrow}$	$0.092 \pm 0.002^{\uparrow}$

Table 5.1: The `macro-f1` results when using the new formulas for the p and m parameters. The $\uparrow$ and $\downarrow$ indicate that the result of the respective algorithm is significantly higher, or lower than the original *PruDent* respectively. The result of the highest performing algorithm for a given dataset is boldfaced.

micro-f1 (higher is better)	Original *PruDent*	Non-Error Based m *PruDent-A*	Error-based m *PruDent-B*
emotion	0.63 ± 0.02	$\mathbf{0.64 \pm 0.02}^{\uparrow}$	0.63 ± 0.02
genbase	0.994 ± 0.002	0.994 ± 0.002	0.994 ± 0.002
yeast	0.69 ± 0.02	$\mathbf{0.71 \pm 0.01}^{\uparrow}$	$\mathbf{0.71 \pm 0.01}^{\uparrow}$
scene	0.60 ± 0.02	$\mathbf{0.612 \pm 0.009}^{\uparrow}$	$0.61 \pm 0.01^{\uparrow}$
cal500	0.53 ± 0.01	$0.547 \pm 0.006^{\uparrow}$	$\mathbf{0.552 \pm 0.006}^{\uparrow}$
medical	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01
langlog	0.27 ± 0.02	0.27 ± 0.02	0.27 ± 0.02
enron	0.690 ± 0.005	$\mathbf{0.691 \pm 0.004}$	$\mathbf{0.691 \pm 0.003}$
slashdot	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01
mediamill	0.784 ± 0.001	$\mathbf{0.787 \pm 0.001}^{\uparrow}$	$\mathbf{0.787 \pm 0.001}^{\uparrow}$
tmc2007	0.760 ± 0.002	$\mathbf{0.763 \pm 0.002}^{\uparrow}$	$\mathbf{0.763 \pm 0.002}^{\uparrow}$
bibtex	0.779 ± 0.003	0.779 ± 0.004	0.779 ± 0.003
rcv1	0.420 ± 0.009	$0.427 \pm 0.008^{\uparrow}$	$\mathbf{0.428 \pm 0.010}^{\uparrow}$
nus-wide	0.511 ± 0.001	$\mathbf{0.528 \pm 0.001}^{\uparrow}$	$0.527 \pm 0.001^{\uparrow}$
imdb	0.344 ± 0.003	$0.379 \pm 0.003^{\uparrow}$	$\mathbf{0.387 \pm 0.003}^{\uparrow}$

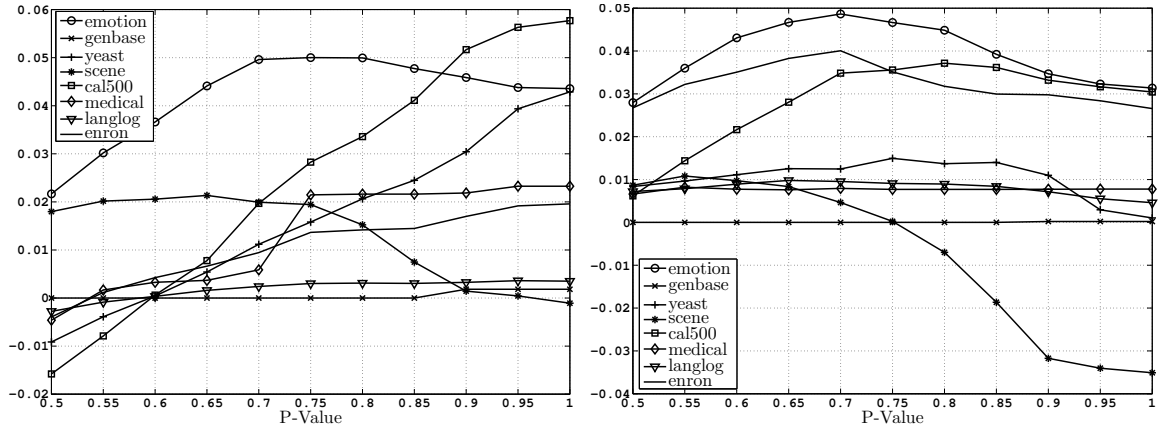
Table 5.2: The `micro-f1` results when using the new formulas for the p and m parameters. The $\uparrow$ and $\downarrow$ indicate that the result of the respective algorithm is significantly higher, or lower than the original *PruDent* respectively. The result of the highest performing algorithm for a given dataset is boldfaced.

5.1.4 Results

The obtained `macro-f1` and `micro-f1` results are provided in Tables 5.1 and 5.2 respectively. Let us begin by comparing the results of the error based m version with the non-error based one. In this regard, the algorithm seems to show no actual benefit from basing the m value on errors; since it was able to achieve improved results regardless of whether m is based on the error, or not. This leads us to believe that the confidence estimates are more sensitive to the p parameter than they are to the m parameter, which conforms with the results previously reported by [93] and [94].

When comparing the new heuristic-based versions of *PruDent* with the original one, a clear performance increase is achieved especially in terms of the targeted `micro-f1` criterion. Thus, classes with a large presence in the data experienced most of the improvements. However, the same cannot be said about `macro-f1` where the results are not as clear-cut; the original *PruDent* version was able to keep its advantage in some of the datasets. This means that `macro-f1` suffers less from over-biasing the p value (since p is always set to 0.66 in original version) than `micro-f1`, which can perhaps be attributed to the following. On the one hand, when classes constitute a majority, their `recall` rates contribute more to `micro-recall` than to `macro-recall`. This is because positive examples have a greater presence in major classes. On the other hand, when classes rarely occur, their `precision` contributes more to `micro-precision` than to `macro-precision`; since minor classes contain more negative examples in their respective training and testing sets. Thus, in rare classes, over-biasing the p value towards the positive class can lead to a decrease in `micro-precision` with an eventual decrease in `micro-f1` as well.

To verify whether over-biasing the classifier with a high p value indeed reduces `micro-f1`, an additional experiment was conducted which records the `macro-` and `micro-f1` criteria with respect to varying the p value in the range $(0.5 - 1.0)$. Here, the m value was set in the same way as the non-error based *PruDent-A* and all class-correlations were left intact (no pruning). The results of this experiment, using a representative subset of the datasets, are illustrated by Figures 5.1a and 5.1b. It becomes clear from the obtained results that the earlier suspicion holds true; having a high p value may increase `macro-f1` at the cost of a decrease in `micro-f1`. Note that the `scene` dataset experienced a decrease in both metrics when a high p value is



(a) macro-f1 (b) micro-f1

Figure 5.1: The difference in `macro-f1` and `micro-f1` with respect to *BR* when varying the p value in *PruDent*.

used, because the classes in this dataset never occupy more than 22% of the examples. Also, this dataset has the lowest average number of classes per example (see *LC* in Table 4.1).

To conclude this section, although the performance improvements in terms of the targeted `micro-f1` criterion were statistically significant more often than not, the differences were too small to be considered as major improvements over the original *PruDent* algorithm. However, despite being small, in some datasets (such as `cal500` , `yeast`, and `imdb`), these differences would have made *PruDent* the winning algorithm when compared to existing competing methods; an outcome that is apparent when these results are compared to the previous ones in Tables 4.4 and 4.10[1]. In general, the *PruDent-A* version provided the best trade-off between `micro-` and `macro-f1`, especially considering that it does not require calibrating the m parameter to the individual classifier errors. Ultimately, the best configuration of the p and m param-

[1]Note that *PruDent* won in the other domains even without modifying the p and m parameters

eters is a subjective aspect that will depend on the end-user's needs; a higher bias (higher p) tends to increase `recall` and decrease `precision`, and vice versa. Additionally, having a high p value may increase `macro-f1` in some datasets at the cost of a possible decrease in `micro-f1`.

5.2 Applying PruDent to Other Base Classifiers

The *PruDent* algorithm was implemented by using the *C4.5* decision tree learner for the binary classification subproblems in *Stacking*. As introduced by the previous chapter, *PruDent* relies on the prediction confidences of the first and second layer classifiers (for each class) in order to choose the more confident class prediction. We know from Section 3.1.2 that decision trees can readily provide probabilistic confidence values by counting the training examples observed in the leaf nodes of the tree. In contrast to decision trees, many base learning algorithms induce classifiers that do not provide probabilistic confidence values associated with their class predictions. For base learning algorithms that rely on frequency counts, such as the *K-Nearest-Neighbors* from Section 3.1.5 , inferring probabilistic confidence values can be done in a fashion similar to that in decision trees. For example, one can consider the neighboring examples in *KNN* in the same way as examples observed in the leaf node. From a probabilistic standpoint, considering the nearest neighbors to a test example is similar to modeling the conditional probability that defines a leaf node in the tree, $P(y|\mathbf{x})$.

Unlike the above mentioned classifier types, inferring confidence estimates in continuous valued classifiers (i.e. regression-based) may require a more sophisticated approach. For the sake of illustration, the remainder of this section will suggest

some possible methods on how to adapt *PruDent* to one such classifier—namely, the non-probabilistic *Support Vector Machines (SVM)* described by Section 3.1.4.

Recall that *SVM* aims to separate a dichotomy of examples by finding the maximum-margin hyperplane which minimizes classification error. A depiction of such a hyperplane is in Figure 5.2 below. Once an optimal hyperplane is found, classifying unseen instances is achieved by multiplying the features of test examples by the set of weights that orient the hyperplane. As a result of this dot product, each test example is then mapped to a single dimensional output z that lies along the normal of the hyperplane. Based on which side of the hyperplane the test example lies, it is then classified to one of the two categories accordingly. More specifically, the test example is labeled as a positive class instance if the sign of the output is positive, $z > 0$. Otherwise, the example is deemed negative.

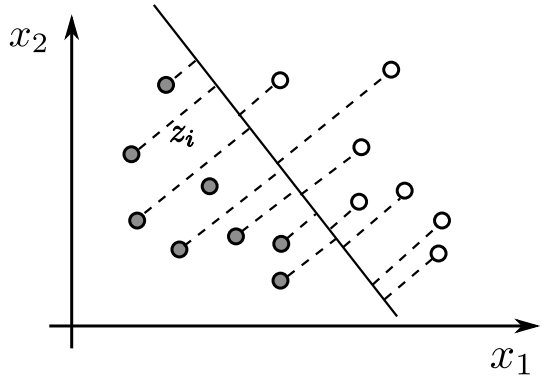


Figure 5.2: A linear *SVM* hyperplane separating a 2-dimensional dichotomy. Positive and negative examples are indicated using white and gray circles respectively. The distances of examples from the hyperplane are indicated with the dashed lines.

Per the classification process described in the previous paragraph, the *SVM* classifier does not provide a probabilistic confidence value associated with its class predictions; aside from the predicted class label, the only other output that is provided

is the signed distance to the separating hyperplane, z (see Figure 5.2). Since the prediction confidences play a crucial role in the *PruDent* algorithm, applying this technique to the *SVM* classifier will require a method to convert these hyperplane distances (z) into probabilistic values $P(y|z)$.

One of the earliest approaches to obtain probabilistic confidences from *SVM* suggested estimating the posterior probabilities $P(y|z)$ by treating the examples' hyperplane distances as a group of Gaussian distributions, one per class [95]. The parameters of the distributions are then estimated using the locations of training examples relative to the induced hyperplane. This approach was later criticized by [88] for assuming that the distances follow a Gaussian distribution. In his paper, the author showcased an example where this assumption does not hold. Instead, the author suggested fitting a sigmoid function which guarantees a monotonicaly increasing probability estimate with respect to the distance z; as typically desired from an estimation technique. Equation 5.4 below shows the employed sigmoid function, and an illustration of an example sigmoid is provided in Figure 5.3.

$$\begin{aligned} P(y_i = 1|z) &= \frac{1}{1 + \exp(A + Bz)} \\ P(y_i = 0|z) &= 1 - P(y_i = 1|z) \end{aligned} \tag{5.4}$$

In the above formulation, y_i is the label of class i, where $y_i = \{0, 1\}$ per the notation in Section 2.1. Note that the parameters A and B can be determined with the help of maximum likelihood estimation from the training set [88]. The underlying idea of the sigmoid approach is the following: as examples lie farther from the hyperplane, they are likely to be classified correctly (because of the error

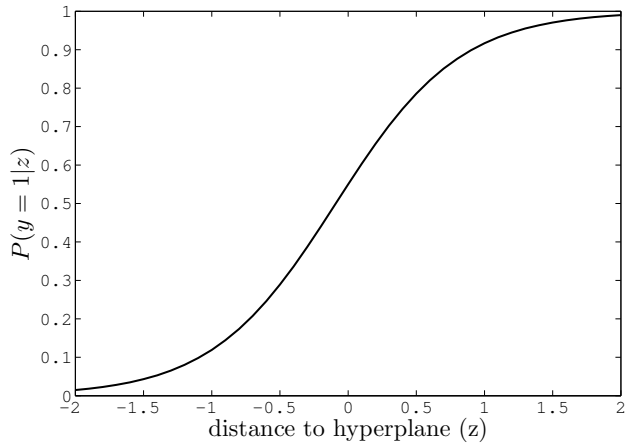


Figure 5.3: An example sigmoid function using the parameters $A = -0.2$ and $B = -2.2$.

minimization function of SVM). In contrast, when examples lie along the boundary, they are expected to be harder to classify since they become closer to the examples of the other respective class. In addition to showing more accurate probability estimates than previous methods, the above sigmoid function seems to be the method of choice currently adopted by recent versions of common machine learning software packages (e.g. WEKA [31] and LIBSVM [96]).

Although the sigmoid function detailed above is the most frequently used approach, one must be cautious about using this method when the inputs to the classifier are themselves prone to errors; a circumstance that is unavoidable in the stacking framework. One way to overcome this hurdle is to use an internal cross-validation process when optimizing the A and B sigmoid parameters. Specifically, the outputs of the first layer classifiers should be used during the optimization instead of the correct labels from the training set. This enables the optimization function to capture the possible errors introduced to the attributes by the additional class-features. Nev-

ertheless, this process may render *PruDent* with a high computational complexity similar to that of the *2BR* algorithm (see Section 4.3.1).

Another approach to estimate *SVM*'s prediction confidences is perhaps to obtain frequency counts of examples based on their relative locations to the induced hyperplane. This can be done by constructing a histogram following one of the different discretization methods detailed recently in [97]. After that, when a test example is submitted to the *SVM* classifier, a count of previously seen examples in the vicinity of the test example is provided along with the predicted class label. In turn, this information can be used in a similar way to the leaf-node example counts in decision trees. Specifically, one can apply the *Laplace* (Section 3.1.2) and *m-estimate* (Section 4.4.3) smoothing techniques whenever a test example falls within a region with few training examples. Perhaps class vector anomalies can then be avoided by preferring the predictions that have a similar output (distance-to-hyperplane) to those observed in the training set. Determining which estimation technique provides the best classification accuracy when *PruDent* is applied to *SVM* is a topic worth investigating in future research efforts.

CHAPTER 6

Conclusions and Future Directions

Targeting the domains where examples can be labeled with two or more classes at the same time, a new technique that belongs to the broader category of *Binary Relevance* methods, is presented. Relying on a stacking-based approach, the technique, called *PruDent*, addresses two drawbacks that negatively affect earlier methods: *unnecessary label dependencies* and *error-propagation*. Experimental results show that this indeed improves classification performance, and does so at a reasonable cost. The following conclusions can be drawn from the results of this work:

- Not all class correlations are necessary: while using all class correlations yields improved classification performance in some domains, others experienced improvements only when weak class relationships were removed.

- Using the *Information Gain* heuristic is a viable option to identify weak dependencies. Moreover, removing these dependencies leads to performance improvements when incorporating class correlations in *BR*.

- Adding the outputs of other classifiers to the vector of example-features can introduce noise in the examples because these outputs are not guaranteed to be

correct. Consequently, these errors lead to a significant degradation in classification performance.

- Classification confidence may provide a guideline to select which classifier is less likely to produce an error. Applying this concept in stacking is an essential step to reduce its associated error-propagation drawback.

- Since *BR* suffers from the imbalanced class representation phenomenon, biasing classifications towards the minority class may provide a more balanced distribution of predicted classes.

Let us discuss other observations made during the experiments. In particular, *PruDent* was able to reduce `hamming-loss` (Section 2.3), a criterion that measures partially correct classifications, even though `hamming-loss` can, in principle, be minimized without incorporating class dependencies [12]. Additionally, the experiments in [6] show that *BR* is hard to beat along this measure. Still, *PruDent* was able to improve `hamming-loss` because of the system's hybrid nature: based on classification confidence, dependent models are used in conjunction with independent ones. Reducing `hamming-loss` is crucial in applications that rely on partial label matches, such as keyword-based searches. It is also worth noting that although the pruning threshold was based on `micro-f1`, other measures may yield different outcomes—it is possible to optimize *PruDent* based on different performance criteria using the same validation-set technique used in the presented experiments.

Experimental evidence also indicates that the proposed system outperforms state-of-the-art algorithms, especially on majority classes (those most frequently repre-

sented in the training set). This is useful in applications where commonly occurring classes are more important than rare ones.

In terms of computational complexity, *PruDent* is quite efficient because it does not construct a second-layer classifier for a class that is independent of others. This is particularly beneficial in domains with large training sets.

One drawback in *PruDent* is the technique's reliance on confidence scores. The experiments in this dissertation employed the decision tree algorithm which provides probabilistic confidence values automatically. For classifiers with real-valued outputs however, a more complex method may be required. Here, one can use probabilistic fitting models, or the uncertainty methods from the field of active learning. Nevertheless, care must be taken when fitting the probabilistic models since the second layer incorporates class-features that are prone to errors. Alternatively, it is possible to discretize the continuous outputs of the classifier using arbitrary regions in the output space. Thus, every region becomes associated with a sample of training examples that can generate a confidence value similar to leaf nodes in decision trees. The latter approach can perhaps pave the way to using different classifier types in the two layers of stacking. The reader is referred to the work in [98] for a better insight into the problems encountered in this research direction.

Another aspect that may be worth future investigation is the modification of the confidence comparison method used in *PruDent*. Recall that *PruDent* chooses the final prediction by simply comparing the confidences of the pair of classifiers assigned to each class. Here, one might prefer to reuse the classification of the independent classifier only if it is *substantially* more confident than its dependent counterpart. Implementation of this idea might be accomplished by imposing a threshold on the

difference between the prediction confidence values—to detremine whether one classifier is indeed more confident than the other.

Bibliography

[1] A. Elisseeff and J. Weston, "Kernel methods for multi-labelled classification and categorical regression problems," *Advances in neural information processing systems*, vol. 14, pp. 681–687, 2002.

[2] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine learning*, vol. 30, no. 2-3, pp. 195–215, 1998.

[3] A. McCallum, "Multi-label text classification with a mixture model trained by em," in *AAAI99 Workshop on Text Learning*, 1999, pp. 1–7.

[4] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 10, pp. 1338–1351, Oct 2006.

[5] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *Principles of data mining and knowledge discovery*. Springer, 2001, pp. 42–53.

[6] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Dzeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, Sep 01 2012.

[7] P. Vateekul, S. Dendamrongvit, and M. Kubat, "Improving svm performance in multi-label domains: Threshold adjustment," *International Journal on Artificial Intelligence Tools*, vol. 22, no. 01, 2013.

[8] S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua, "A mfom learning approach to robust multiclass multi-label text categorization," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 42–.

[9] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2004, pp. 22–30.

[10] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[11] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine learning*, vol. 85, no. 3, pp. 333–359, 2011.

[12] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence and loss minimization in multi-label classification," *Machine Learning*, vol. 88, no. 1-2, pp. 5–45, 2012.

[13] S.-J. Huang and Z.-H. Zhou, "Multi-label learning by exploiting label correlations locally," vol. 2, 2012, pp. 949–955.

[14] A. Alali and M. Kubat, "Prudent: A pruned and confident stacking approach for multi-label classification," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 9, pp. 2480–2493, Sept 2015.

[15] R. E. Schapire and Y. Singer, "Boostexter: A boosting-based system for text categorization," *Machine learning*, vol. 39, no. 2-3, pp. 135–168, 2000.

[16] N. Ghamrawi and A. McCallum, "Collective multi-label classification," in *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 2005, pp. 195–200.

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

[18] E. L. Mencia and J. Furnkranz, "Pairwise learning of multilabel classifications with perceptrons," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 2899–2906.

[19] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004.

[20] C. G. Snoek, M. Worring, J. C. Van Gemert, J.-M. Geusebroek, and A. W. Smeulders, "The challenge problem for automated detection of 101 semantic concepts in multimedia," in *Proceedings of the 14th annual ACM international conference on Multimedia*. ACM, 2006, pp. 421–430.

[21] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world web image database from national university of singapore," in *Proceedings of the ACM international conference on image and video retrieval*. ACM, 2009, p. 48.

[22] E. Spyromitros-Xioufis, S. Papadopoulos, I. Kompatsiaris, G. Tsoumakas, and I. Vlahavas, "A comprehensive study over vlad and product quantization in large-scale image retrieval," *Multimedia, IEEE Transactions on*, vol. 16, no. 6, pp. 1713–1728, Oct 2014.

[23] C. J. Van Rijsbergen, *Information Retrieval*, 2nd ed. London, 1979.

[24] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information retrieval*, vol. 1, no. 1-2, pp. 69–90, 1999.

[25] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas, "Multi-label classification of music into emotions." in *ISMIR*, vol. 8, 2008, pp. 325–330.

[26] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas, "Multi-label classification of music by emotion," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2011, no. 1, pp. 1–9, 12 2011.

[27] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in neural information processing systems*, 2001, pp. 681–687.

[28] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet, "Semantic annotation and retrieval of music and sound effects," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 467–476, 2008.

[29] S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas, "Protein classification with multiple algorithms," in *Advances in Informatics*. Springer, 2005, pp. 448–456.

[30] J. P. Pestian, C. Brew, P. Matykiewicz, D. Hovermale, N. Johnson, K. B. Cohen, and W. Duch, "A shared task involving multi-label classification of clinical free text," in *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*. Association for Computational Linguistics, 2007, pp. 97–104.

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[32] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Machine learning: ECML 2004*. Springer, 2004, pp. 217–226.

[33] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multilabel text classification for automated tag suggestion," in *Proceedings of the ECML/PKDD*, vol. 18, 2008.

[34] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *The Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[35] A. N. Srivastava and B. Zane-Ulman, "Discovering recurring anomalies in text reports regarding complex space systems," in *Aerospace Conference, 2005 IEEE*. IEEE, 2005, pp. 3853–3862.

[36] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 1819–1837, Aug 2014.

[37] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[38] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine learning*, vol. 37, no. 3, pp. 297–336, 1999.

[39] K. Sarinnapakorn and M. Kubat, "Combining subclassifiers in text categorization: A dst-based solution and a case study," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 12, pp. 1638–1651, 2007.

[40] ——, "Induction from multi-label examples in information retrieval systems: A case study," *Applied Artificial Intelligence*, vol. 22, no. 5, pp. 407–432, 2008.

[41] S.-J. Huang, Y. Yu, and Z.-H. Zhou, "Multi-label hypothesis reuse," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 525–533.

[42] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[43] ——, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[44] J. Quinlan, "Improved use of continuous attributes in c4. 5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[45] B. Cestnik, "Estimating probabilities: a crucial task in machine learning." in *ECAI*, vol. 90, 1990, pp. 147–149.

[46] T. Niblett, "Constructing decision trees in noisy domains," in *Proceedings of the Second European Working Session on Learning*, Bled, Yugoslavia, 1987, pp. 67–78.

[47] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[48] F. De Comite, R. Gilleron, and M. Tommasi, "learning multi-label alternating decision trees from texts and data," *Lecture Notes in Computer Science*, vol. 2734, pp. 35–49, 2003.

[49] Y. Freund and L. Mason, "The alternating decision tree learning algorithm," in *ICML*, vol. 99, 1999, pp. 124–133.

[50] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[51] F. Rosenbaltt, "The perceptron–a perceiving and recognizing automation," Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep., 1957.

[52] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *IRE WESCON Convention Record*, 1960, pp. 96–104.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, p. 3, 1988.

[54] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[55] V. N. Vapnik, "The nature of statistical learning theory," 1995.

[56] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998.

[57] E. Fix and J. J. L. Hodges, "Discriminatory analysis, nonparametric discrimination: consistency properties," USAF School of Aviation Medicine, Randolf Field, Tex., Project 21-49-004, Contract AF 41(128)-31, Rep. 4, Tech. Rep., Feb 1951.

[58] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[59] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. John Wiley & Sons, 2001.

[60] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[61] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038 – 2048, 2007.

[62] N. J. Nilsson, *Learning Machines: Foundations of Trainable Pattern-classifying Sistems*. McGraw-Hill, 1965.

[63] E. Alvares-Cherman, J. Metz, and M. C. Monard, "Incorporating label dependency into the binary relevance framework for multi-label classification," *Expert Systems with Applications*, vol. 39, no. 2, pp. 1647–1655, 2012.

[64] M.-L. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 999–1008.

[65] E. Montañes, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hllermeier, "Dependent binary relevance models for multi-label classification," *Pattern Recognition*, vol. 47, no. 3, pp. 1494 – 1508, 2014.

[66] S. Dendamrongvit, P. Vateekul, and M. Kubat, "Irrelevant attributes and imbalanced classes in multi-label text-categorization domains," *Intelligent Data Analysis*, vol. 15, no. 6, pp. 843–859, 2011.

[67] J. Fürnkranz, E. Hüllermeier, E. L. Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking," *Machine Learning*, vol. 73, no. 2, pp. 133–153, 2008.

[68] D. Gjorgjevikj and G. Madjarov, "Two stage classifier chain architecture for efficient pair-wise multi-label learning," in *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on*. IEEE, 2011, pp. 1–6.

[69] G. Madjarov, D. Gjorgjevikj, and T. Delev, "Efficient two stage voting architecture for pairwise multi-label classification," in *AI 2010: Advances in Artificial Intelligence*. Springer, 2011, pp. 164–173.

[70] G. Tsoumakas, I. Katakis, and L. Vlahavas, "Random k-labelsets for multilabel classification," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 7, pp. 1079–1089, July 2011.

[71] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 995–1000.

[72] L. Tenenboim-Chekina, L. Rokach, and B. Shapira, "Identification of label dependencies for multi-label classification," in *Working Notes of the Second International Workshop on Learning from Multi-Label Data*, 2010, pp. 53–60.

[73] L. Chekina, D. Gutfreund, A. Kontorovich, L. Rokach, and B. Shapira, "Exploiting label dependencies for improved sample complexity," *Machine learning*, vol. 91, no. 1, pp. 1–42, 2013.

[74] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*. Springer-Verlag, 2009, pp. 254–269.

[75] K. Dembczyński, W. Waegeman, and E. Hüllermeier, "An analysis of chaining in multi-label classification," in *20th European conference on Artificial Intelligence (ECAI 2012)*, vol. 242. IOS Press, 2012, pp. 294–299.

[76] R. Senge, J. J. del Coz, and E. Hüllermeier, "On the problem of error propagation in classifier chains for multi-label classification," in *Data Analysis, Machine Learning and Knowledge Discovery*. Springer, 2014, pp. 163–170.

[77] ——, "Rectifying classifier chains for multi-label classification," *Space*, vol. 2, no. 8, p. 19, 2013.

[78] W. Cheng, E. Hüllermeier, and K. J. Dembczynski, "Bayes optimal multilabel classification via probabilistic classifier chains," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 279–286.

[79] N. Li and Z.-H. Zhou, "Selective ensemble of classifier chains," in *Multiple Classifier Systems*, ser. Lecture Notes in Computer Science, Z.-H. Zhou, F. Roli, and J. Kittler, Eds. Springer Berlin Heidelberg, 2013, vol. 7872, pp. 146–156.

[80] P. da Silva, E. Gonalves, A. Plastino, and A. Freitas, "Distinct chains for different instances: An effective strategy for multi-label classifier chains," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, vol. 8725, pp. 453–468.

[81] G. Tsoumakas, A. Dimou, E. Spyromitros, V. Mezaris, I. Kompatsiaris, and I. Vlahavas, "Correlation-based pruning of stacked binary relevance models for multi-label learning," in *Proceeding of ECML/PKDD 2009 Workshop on Learning from Multi-Label Data, Bled, Slovenia*, 2009, pp. 101–116.

[82] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[83] P. Vateekul and M. Kubat, "Fast induction of multiple decision trees in text categorization from large scale, imbalanced, and multi-label data," in *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*. IEEE, 2009, pp. 320–325.

[84] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo *et al.*, "Fast discovery of association rules." *Advances in knowledge discovery and data mining*, vol. 12, no. 1, pp. 307–328, 1996.

[85] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *Proceedings of the 21th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., 1995, pp. 432–444.

[86] M. Zaki, "Scalable algorithms for association mining," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, no. 3, pp. 372–390, May 2000.

[87] T. Meng, Y. Liu, M.-L. Shyu, Y. Yan, and C.-M. Shu, "Enhancing multimedia semantic concept mining and retrieval by incorporating negative correlations," in *Semantic Computing (ICSC), 2014 IEEE International Conference on*, June 2014, pp. 28–35.

[88] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances In Large Margin Classifiers*. MIT Press, 1999, pp. 61–74.

[89] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[90] X. Zhu and X. Wu, "Class noise vs. attribute noise: A quantitative study," *Artificial Intelligence Review*, vol. 22, no. 3, pp. 177–210, 2004.

[91] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *Journal of Artificial Intelligence Research*, pp. 131–167, 1999.

[92] X. Zhu, X. Wu, and Q. Chen, "Bridging local and global data cleansing: Identifying class noise in large, distributed data datasets," *Data mining and Knowledge discovery*, vol. 12, no. 2-3, pp. 275–308, 2006.

[93] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *ICML*, vol. 1, 2001, pp. 609–616.

[94] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[95] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," *Ann. Statist.*, vol. 26, no. 2, pp. 451–471, 04 1998.

[96] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[97] S. Garcia, J. Luengo, J. A. Sáez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 734–750, 2013.

[98] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *J. Artif. Intell. Res.(JAIR)*, vol. 10, pp. 271–289, 1999.