Learning Versatile Decision Trees

Towards Context-Awareness and Multi-Label Classification

By

REEM MOTEAB SULTAN ALOTAIBI

Intelligent Systems Laboratory
Department of Computer Science
Faculty of Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

SEPTEMBER 2016

ABSTRACT

Decision trees are predictive models that partition the training space into smaller subspaces in a divide-and-conquer manner. They are simple, self-explanatory, intuitive and powerful approaches to build classification and regression models. Due to their advantages, they have become one of the most common algorithms in the fields of machine learning and data mining. The main assumption in many classical decision tree algorithms is that both training and deployment contexts share the same characteristics, which may not be the case in many real-world applications. A change in data distribution, known as dataset shift, or a change in misclassification costs are examples of such context changes. Moreover, decision trees are tailored to automatically classify instances that belong to mutually exclusive labels; each instance is assigned to only one label out of a set of labels. However, in many realistic applications, instances often belong to more than one label at the same time, which is known as multi-label classification.

This thesis proposes versatile decision trees which are oriented towards two issues in existing standard decision tree algorithms: *context changes* and *multiple outputs*. Firstly, we propose the Versatile Model (**VM**) that is rich enough to handle different kinds of dataset shift without making strong assumptions about the training and deployment contexts. Furthermore, the VM does not require labelled data to identify the data shift at deployment. Our empirical evaluations and results on both synthetic shift and real datasets shift show strong performance gains by achieved the VM.

Secondly, we propose **LaCova** that uses the divide-and-conquer approach of decision trees and enables taking local decisions about label correlations. The resulting algorithm establishes a tree-based multi-label classifier that interleaves between two well-known baseline methods: Binary Relevance, which assumes all labels are independent, and Label Powerset, which learns the joint label distributions. **LaCova** introduces a novel splitting criterion designed specifically for multi-label data. The key idea is based on the label covariance matrix at each node, which allows the algorithm to choose between: splitting this matrix horizontally using attributes in the conventional way, or splitting it vertically by separating the labels.

Thirdly, the proposed algorithm is improved further by introducing **LaCovaC** that models cluster labels locally into several independent subsets at various points during training. The clusters are obtained locally by identifying the conditionally-dependent labels in localised regions of the attribute space using the label correlation matrix. Our findings demonstrate that our proposal achieves competitive performance over a wide range of evaluation metrics when compared with the state-of-the-art multi-label classifiers.

Finally, for each test instance at deployment time, the decision tree outputs scores or probabilities, which can be adapted to several deployment contexts by means of a decision threshold. We investigate and analyse various methods for tuning decision thresholds in a multi-label setting. We explore two possible scenarios: the selection of global and multiple thresholds. However, we use the logistic regression algorithm as a base classifier instead of the decision tree algorithm as this algorithm produces better calibrated probabilities than decision trees.

Experimental evaluation demonstrates that there is no significant difference by adjusting a global threshold in two situations: shared and variable costs. Moreover, it shows that there is no significant loss by adjusting a single global threshold rather than a threshold for each label (multiple thresholds) considering different misclassification costs. Although tuning multiple thresholds is the obvious solution, the global threshold can also be valid.

DEDICATION

I dedicate this thesis to...

My great parents,
My beloved husband,
My boys, Saif and Qussi,
My twins, Yazan and Battal, who came into the world during the early stages of this work.

ACKNOWLEDGEMENTS

Working with this thesis has been a very interesting and valuable experience to me and I have learned a lot. I want to express my sincerest thanks and appreciation to many people for being part of my PhD journey and making this thesis possible.

First and foremost, I would like to thank my passionate supervisor, Professor Peter Flach, whom I was very fortunate by his supervision, guidance, patience and constructive comments that have greatly improved this work. I appreciate the time he spent for discussing, reviewing my work and teaching me new stuff.

I would like to thank Dr.Meelis Kull who provided contributions, insight and expertise that greatly assisted this research. Thanks to Professor Tijl De Bie for his feedback during the annual review of the PhD. Thanks to all co-authors and reviewers of published work described in this thesis. I am also thankful to Flach's group and the members of the REFRAME project for useful discussions at meetings.

I would also like to thank my committee members, professor Alex Freitas, Dr.Raul Santos-Rodriguez and Dr.Steve Gregory for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you all.

My special gratitude goes to my loving parents whose love and affection is the source of motivation and encouragement for my studies. Thank you to my wonderful husband, Aesam Alsharif, for being patient, supportive and helpful at every stage of my study. I wish him good luck with submitting his own PhD thesis in April 2017. My boys, Saif, Qussi, Yazan and Battal, thanks for your understanding and patience for being busy especially at the last days of this work.

I would like to thank all my family, sisters and brothers for simply being there when needed. A special thanks to my beloved brother Badar and my sister Maha for unconditional love, encouragement and support. I am also deeply thankful to all friends in Bristol for continuous support, encouragement and making Bristol life a wonderful and unforgettable experience for my little family and me.

I acknowledge my gratitude to King Abdulaziz University for giving me the PhD scholarship and financial support for this research. Finally, I would like to thank again everyone who provided support and help throughout my PhD. If I miss someone's name here, it does not mean that I do not recognise your support and help.

Reem Alotaibi
September 2016

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

FIGURE **Page**

List of Abbreviations

AdaBoost	Adaptive Boosting
AdaCost	Adaptive Cost
ADTrees	Alternating Decision Trees
AUC	Area Under the ROC Curve
BC	Binary Classification
BCC	Bayesian Chain Classifiers
BR	Binary Relevance
BS	Brier Score
CD	Cost-Driven
CD	Critical Difference
CDA	Cost-Driven Average
CDF	Cumulative Distribution Functions
CDU	Cost-Driven Uniform
CLR	Calibrated Label Ranking
CS	Compressive Sensing
DAGs	Directed Acyclic Graphs
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DTs	Decision Trees
ECCs	Ensembles of Classifier Chains
fn	false negative

fp	false positive
FW	Four-class pairWise classification
GP	Genetic Programming
H-Loss	Hierarchical Loss
HC4.5	Hierarchical C4.5
HMC	Hierarchical Multi-label Classification
HMC-Loss	Hierarchical Multi-label Classification Loss
Homer	Hierarchy Of Multilabel classifiERs
ID3	Iterative Dichotomiser 3
IDD	Independently and Identically Distributed
IG	Information Gain
IGR	Information Gain Ratio
IOP	Integrated Optimisation Problem
IWCV	Importance Weighted Cross Validation
KMM	Kernel Mean Matching
KNN	K-Nearest Neighbour
KS	Kolmogorov-Smirnov test
LCD	Label-wise Cost-Driven
LOpt	Label-wise Optimal
LP	Label Powerset
LPBR	LP and BR
LRD	Label-wise Rate-Driven
MAE	Mean Absolute Error
MAP	Maximum A posteriori Principle
MCC	Multi-Class Classification

MCut	Maximum Cut
MetaCost	Meta-learning for Cost-sensitive classification
ML-C4.5	Multi-Label C4.5
ML-LOC	Multi-Label learning using LOcal Correlation
ML-SVMDT	SVM-based Decision Trees for Multi-Label learning
MLC	Multi-Label Classification
MLkNN	Multi-Label K-Nearest Neighbour
OOCC	One-To-One Classifier Chain
Opt	Optimal
PAM	Partitioning Around Medoids
PCA	Principal Component Analysis
PCC	Probabilistic Classifier Chain
PCut	Proportion Cut
PLST	Principle Label Space Transformation
PS	Pruned Sets
PW	PairWise classification
RAkEL	RAndom k-labELset
RCut	Rank Cut
RD	Rate-Driven
RKHS	Reproducing Kernel Hilbert Space
ROC	Receiver Operating Characteristic
SCut	Score-based Cut
SHC	Stochastic Hill Climbing
SMOTE	Synthetic Minority Over-sampling TEchnique
SVMs	Support Vector Machines

tn	true negative
tp	true positive
VM	Versatile Model

CHAPTER 1

INTRODUCTION

Decision trees (DTs) are one of the most popular algorithms for both classification and regression tasks in machine learning and data mining. They are intuitive, understandable, selective and easy to use. They employ a divide-and-conquer approach in building a model that can be used later for better understanding and future predictions. This thesis extends the standard decision tree algorithms to address two important and challenging issues in the machine learning field: 1) potential context changes between training and deployment data and 2) learning multiple labels simultaneously.

We propose three novel algorithms, which are based on decision trees. Firstly, we introduce the notion of versatile decision trees that are adaptable to context changes, dataset shift in particular. Secondly, we develop two novel algorithms in the context of multi-label classification that explore and utilise label correlations dynamically.

In this first chapter, we introduce research motivations, aims and objectives, contributions and publications and outline the reminder of the thesis.

1.1 Introduction

Machine learning and data mining are concerned with learning a model from training data and applying this model to new deployment or test data automatically without human assistance. Training data consists of a set of training instances, where each instance is represented by a number of input attributes, also known as features or covariates.

There are two paradigms of learning: supervised and unsupervised. Supervised learning needs labelled data, which is usually classified and labelled by human experts, whereas unsupervised learning does not require data to be labelled, as the task is to group and cluster data based on the input attributes. Regression and classification are examples of supervised learning tasks. In the former, the task is to estimate the numerical target, while in the latter; the task is to identify group membership.

Classification is a supervised learning task in which the goal is to categorise a new instance into one or more classes using the trained model. In traditional classification, each instance is assigned a single label. In binary classification, each instance can belong to one of two classes, for example, in medical diagnosis the task is to determine whether the patient is suffering from a particular disease or not (positive or negative classes). Whereas in multi-class classification the setting is more general, allowing each training instance to belong to one of more than two classes, for instance, to classify an image of fruits which may be oranges, apples, or pears [1].

On the other hand, over the last few years, multi-label classification has received interest of many researchers. It considers more than one label for each instance simultaneously [2, 3]. Thus, it allows for a wide range of applications, such as text categorisation, image and movie tagging, and gene functions. For example, a medical diagnosis might find a patient has multiple diseases at one time; an article that gives statistics about the number of students who have applied to medical schools in a country could be categorised as both educational and medical; and an image that captures a beach at sunset could belong to both beach and sunset groups. Thus, all these examples naturally yield multiple labels.

Decision trees are one of the most popular algorithms for classification. They are built recursively by partitioning the training data. They consist of nodes, starting from the root node, internal nodes that represent attributes, and leaf nodes that represent class labels or probability distributions over classes. Internal nodes split the training space into two or more based on the chosen splitting criterion. Among other advantages, decision trees are simple, intuitive and comprehensible such that the structure can be captured by inexpert users easily. Furthermore, they are selective, which means they select the most discriminative attributes from the data to construct the trees, which is usually much less than the actual number of attributes. This has an advantage especially in high-dimensional spaces with many attributes [4–6].

In machine learning, we distinguish between training and deployment contexts. The training context is when labelled data is used to learn a model, whereas the deployment context is when the model is actually used for predictions. Machine learning models are usually based on the assumption that both training and deployment contexts are the same. In violation of this assumption, for example, the model is built using data from one country but the deployment data are collected in another country, so the trained model cannot be applied to the deployment data and re-training a new model is necessary.

A context can be any meta information about the data, such as data distributions, class priors, misclassification costs and loss functions. Hernández-Orallo et al. [7] identified a set of context changes and suggested some possible solutions. However, this thesis addresses two types of context changes: data distributions and misclassification costs.

To begin with, this thesis proposes the Versatile Model (VM) that addresses a particular type of context change known as dataset shift. We first introduce the VM for single-label classification (binary and multi-class classification) as it is one of the most common machine learning paradigms. The VM is a decision tree model that is adaptable to different kinds of shift without making strong assumptions such as linear relationship across contexts.

While the first part of the thesis focuses mainly on single-label problems, we gradually shift the focus towards a more general setting, particularly multi-label classification. Firstly, we investigate decision tree algorithms for multi-label classification. We propose a novel decision tree algorithm to learn multi-label models by exploiting label correlations called **LaCova**. The key idea of the algorithm is to use a label covariance matrix at every node of the tree to decide how to split labels. **LaCova** is also extended to **LaCovaC** by studying ways in which the correlation matrix can suggest label clusters. Both proposed algorithms are evaluated and compared against baseline and state-of-the-art algorithms.

Finally, we consider a context-aware approach that adapts the learned model to several deployment contexts by means of threshold choice methods. We study different thresholding approaches in multi-label classification. We extend some binary classification thresholding methods to the multi-label context. We also analyse the empirical loss associated with each approach by varying misclassification costs across contexts. We adopt cost curves that have been used widely in binary classification to multi-label classification. In addition, we propose the use of scatter plots for some cases.

1.2 Research Motivations

Acknowledging the benefits of decision tree models, we focus our work on learning with decision trees. While the standard decision tree algorithms have several advantages, they still suffer from potential drawbacks.

Firstly, the key step in decision tree learning is identifying the best split of data based on an input attribute at each tree node. One difficulty, here, is the fact that the split is selected based on absolute scale relying on the assumption that both training and deployment data follow the same distribution. Accordingly, this split is inappropriate when there is a variation between training and deployment contexts. This situation is referred to as dataset shift in the literature.

Recently, several attempts have been made to address this variation in both domain adaptation and transfer learning areas. In this thesis, our ***first motivation*** is to focus on an unsupervised method that assumes unlabelled data at deployment. Moreover, this thesis proposes to enrich the model throughout the training process with more information about the current context in relation to possible future contexts. Such models are expected to perform very well in both the training context and the new contexts. Owing to this, we study context changes in single-label classification (binary and multi-class classification).

Secondly, standard decision tree algorithms allow only a single label for each instance, while there are many applications that require more than one label at the same time, text categorisation is an example of such applications. The main challenge to note is that real-world applications tend to have a large number of labels which often have a relationship or connection, whereby the presence of one label affects or depends on another label. Moreover, correlations may exist in a particular region of the data, which make them conditional on the input space for this region. Furthermore, multi-label classes are highly imbalanced since the class distributions for some labels are imbalanced. These labels are called minority

labels as they are assigned to only a few instances in the training data, which makes the prediction of those labels a hard task [8].

Several studies argue that exploiting label correlations is important in the area of multi-label classification [3, 9–13]. Thus, exploiting dependencies among labels without increasing complexity could improve a classifier's performance. Although a considerable amount of work has been done in this area, these have chiefly focused on a global approach, in which label correlations are identified as a pre-processing step prior to training the model.

The ***second motivation*** is to adapt decision tree algorithms to multi-label problems, which are more complicated than single-label problems, because of the extra label dimensions. This setting can be challenging in both learning and evaluation processes [14]. We investigate the potential and practical benefits of employing label correlations to capture dependencies among labels. To the best of our knowledge, so far, there is no work that considers label correlations dynamically as we propose in this thesis.

Finally, standard decision trees produce confidence scores for labels that need to be classified into relevant and irrelevant. They use 0.5 as the default threshold in order to produce labels. More importantly, the trained model can be learnt in a training context that most probably changes in the deployment context. More specifically, a model can be learnt with training label costs and deployed in other contexts, where these costs have been changed.

In multi-label scenarios, there exist several ways to adjust thresholds, which can be grouped into three different types, one per-dataset, one per-label and one per-instance. Threshold selection is an important research area as it can affect a classifier's performance. However, it is not clear which of these approaches to choose or how they relate to each other. Therefore, our ***third motivation*** is to explore different threshold choice methods in multi-label classification and analyse their performance under different deployment contexts.

1.3 Aim and Research Objectives

Considering the advantages and benefits of learning with decision trees, the aim of this thesis is to extend the standard decision tree algorithms to solve several challenging problems by achieving the following objectives.

- An implicit assumption in machine learning is that both training and deployment data follow the same distribution, which is known as Independently and Identically Distributed (IID). ***The first objective*** is to propose and implement a general framework based on decision trees that do not depend on the IID assumption.

- Despite the fact that there is a relatively large number of research targets multi-label classification, there still exist areas for further improvement. ***Our second objective*** is to develop decision tree algorithms that explore label correlations locally, which could improve the predictive performance of the learning algorithm.

- In multi-label classification, labels can have different misclassification costs. For example, a mistake in classifying a patient with serious and life-threatening illness is more costly than a mistake in noncritical illness. Thus, ***our third objective*** is to explore threshold selection approaches for multi-label classification over a range of label costs. Moreover, it involves analysing three feasible techniques for tuning thresholds in multi-label setting and adopting some binary classification thresholding strategies.

1.4 Summary of Contributions

This thesis makes several contributions towards single-label and multi-label classification. We summarise the main contributions as below.

- We propose a general framework called the Versatile Model (VM) that deploys different strategies in order to handle dataset shift. The proposed model does not require labelled data at deployment and does not assume linear relationship across contexts.

- We propose a novel approach to build decision trees by introducing percentiles in splitting decisions. For example, suppose the splitting attribute is patient's age, the values going into the left and right child nodes will be shown as values less than or greater than an absolute threshold in the standard algorithms. Suppose the threshold is 60, the left node consists of all patients who are older than 60, whereas the right node consists of all patients who are aged 60 or younger. By means of a percentile, we say, for example, 60 is the 70^{th} percentile, meaning that if 70% of people in the training data aged 60 or younger go to the right child node, then the 70% adult people in the deployment set also go to the right child node. Whilst the other 30% go to the left child node.

- We develop two novel multi-label classification algorithms based on standard decision trees that are able to deal with multiple labels at the same time. We design a novel type of node in the decision tree called "vertical split" that splits the label space vertically. All descendant branches of that node must be visited in order to collect predictions for all labels.

- We introduce a new feature selection method in the context of multi-label learning. The proposed method evaluates for each child node both its sum of label variances and its sum of absolute label covariances, and assigns as a splitting measure the minimum of these two (lower is better).

- Multi-label cost curves are developed to study the predictive performance over a range of misclassification costs under different thresholding techniques. We propose to use the cost curves when labels share the same cost, otherwise, we introduce scatter plots.

- We contribute two new threshold selection methods in multi-label classification that have been used in binary classification: score-driven and one optimal.

During the PhD study, we have several published contributions, which have been peer-reviewed in conferences and workshops. The contributions of the author of this thesis are stated in each case, which was all under the supervision of Prof. Peter Flach. Below, we list those papers and point out in which chapters of the thesis we discuss them. All source code is available in online repository [15].

1. Al-Otaibi, Reem, B.C. Prudêncio, Ricardo, Kull, Meelis, and Flach, Peter. Versatile decision trees for learning over multiple contexts. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML-PKDD 2015, Lecture Notes in Computer Science, pages 184-199, Porto, Portugal. Springer [16].

2. Al-Otaibi, Reem, B.C. Prudêncio, Ricardo, Kull, Meelis, and Flach, Peter. A Context-Aware Model based on Versatile Decision Trees. In Proceedings of the 9th Saudi Students Conference in the UK, SSC9, Birmingham, The UK. Springer (to appear) [17].

   Chapter 3 covers the work investigated in these papers. Ricardo B.C. Prudêncio and Meelis Kull provided valuable feedback. Ricardo helped with editing the introduction and dataset shift sections and provided some examples. My contribution was developing the models, designing and performing the experiments, analysing the results, writing the initial drafts, and revising the final papers.

3. Al-Otaibi, Reem, Kull, Meelis, and Flach, Peter. Declaratively capturing local label correlations with multi-label trees. In Proceedings of the 22nd Biennial European Conference on Artificial Intelligence, ECAI 2016, pages 1467 - 1475, The Hague, The Netherlands. IOS Press [18].

4. Al-Otaibi, Reem, Kull, Meelis, and Flach, Peter. LaCova: A tree-based multilabel classifier using label covariance as splitting criterion. In Proceedings of the 13th IEEE International Conference on Machine Learning and Application, ICMLA 2014, pages 74-79, Detroit, USA. IEEE [19].

5. Al-Otaibi, Reem, Kull, Meelis, and Flach, Peter. Multi-label classification by label clustering based on covariance. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Doctoral Consortium at ECML-PKDD 2015, pages 43-52, Porto, Portugal. Aalto University [20].

6. Al-Otaibi, Reem, Kull, Meelis, and Flach, Peter. Hybrid multi-label decision trees for classification. In Proceedings of the 8th Saudi Students Conference in the UK, SSC8, pages 323-334, London, The UK. Imperial College Press [21].

   Chapter 4 covers the work published in all these four papers. I contributed in developing the algorithms, designing and performing the experiments, analysing and reporting the results, writing the initial drafts, and revising the final papers. Meelis Kull proposed the analytical covariance threshold described in Section 4.2.3.

7. Al-Otaibi, Reem, Flach, Peter, and Kull, Meelis. Multi-label classification: A comparative study on threshold selection methods. In Proceedings of the First International Workshop on Learning over Multiple Contexts, LMCE at ECML-PKDD 2014, Nancy, France [22].

   This work is discussed in Chapter 5. My contribution was implementing the methods, analysing and interpreting the results, writing the initial draft, and revising the final manuscript. Meelis Kull provided useful comments through discussions.

Furthermore, some extra works, which are not directly linked to the main topic of the thesis, have been carried out and achieved in collaboration with others during the PhD research.

1. The first paper studies how to adapt the VM proposed in Chapter 3 in the data stream setting. It shows that the VM can cope with monotonic concept drift and reports good performance.
   Dos Reis, Denis, Kull, Meelis, Al-Otaibi, Reem, Flach, Peter, and Batista, Gustavo. Data stream classification under non-null verification latency (paper ready for submission).

2. The second paper proposes feature construction and calibration for clustering electricity daily load curves. The paper presents three new methods to construct features: conditional filters on time-resolution based features, calibration and normalisation, and using profile errors.
   Al-Otaibi, Reem, Jin, Nanlin, Wilcox, Tom, and Flach, Peter. Feature construction and calibration for clustering daily load curves from smart meter data. IEEE Transactions on Industrial Informatics, Volume 12, Issue 2, pp. 645-654, April 2016 [23].

1.5 Organisation of the Thesis

The rest of the thesis is organised as follows.

1. **Chapter 2: Background** begins with listing the notations used throughout the thesis. Next, we give background on decision tree algorithms and their key advantages that will be applied and built upon. Then, we give an overview of multi-label learning, algorithms and evaluation measures. A detailed description on multi-label benchmarks is also given. After that, we highlight the existing techniques in the context of cost-sensitive classification. Finally, we discuss existing approaches for unsupervised learning, and in particular clustering algorithms.

2. **Chapter 3: Context-aware Decision Trees: The Versatile Model** suggests a general framework to learn decision tree models. We address several important issues in the dataset shift problem. Firstly, how can we automatically detect that there is a significant difference between training and deployment data to take action or adjust the model appropriately? Secondly, different shifts can occur in real applications (e.g., linear and non-linear), which require the use of diverse solutions.

   This chapter offers two main contributions towards resolving these issues. We propose a Versatile Model (VM) that is rich enough to handle different kinds of shift without making strong assumptions such as linearity and furthermore does not require labelled data to identify the data shift at

deployment. The VM obtains good performance on both synthetic shift and real datasets in the case of single-label classification.

3. **Chapter 4: Multi-label Decision Trees that Capture Label Correlations Locally** introduces two novel tree-based algorithms to learn multi-label problems: **LaCova** and **LaCovaC**. Firstly, **LaCova** interleaves between two decisions: 1) learning labels separately (independence assumption); and 2) learning all labels together (dependence assumption). Additionally, we introduce a novel splitting criterion designed specifically for multi-label problems, which is based on the label covariance matrix. Furthermore, we propose novel nodes called vertical split in addition to the internal nodes (horizontal splits) in standard decision tree algorithms.

   Secondly, **LaCovaC** alternates between the previously mentioned decisions by introducing label clusters. It is important to notice that all these decisions are taken locally while growing the trees. Finally, we demonstrate the performance of the proposed algorithms experimentally.

4. **Chapter 5: Threshold Selection Methods for Multi-label Classification** explores the existing thresholding methods of multi-label classification by considering label misclassification costs as context change. It also contributes two new methods in multi-label classification that have been used in binary classification: score-driven and one optimal threshold choice methods. It suggests the use of cost curves and scatter plots to visualise classifier's performance over a wide range of misclassification costs.

5. **Chapter 6: Conclusions and Future Work** concludes the thesis by summarising the main findings and results of the previous chapters. It also highlights possible avenues of future work.

6. **Appendix A: Detailed Experimental Results** shows more detailed experimental results for Chapter 5.

CHAPTER 2

BACKGROUND

This chapter begins with an introduction and the notations used throughout the thesis. The remainder of the chapter is structured in the following order. Section 2.3 gives informative background topics relevant to traditional classification. A review on decision tree learning is set out in this section as well. Next, Section 2.4 presents an overview on multi-label classification, analyses different learning algorithms, discusses several evaluation measures for multi-label classification, and finally presents the multi-label benchmark datasets used in this thesis for experimental evaluation.

After that, Section 2.5 sheds some light on cost-sensitive classification. Finally, Section 2.6 provides the necessary background to understand different clustering algorithms.

2.1 Introduction

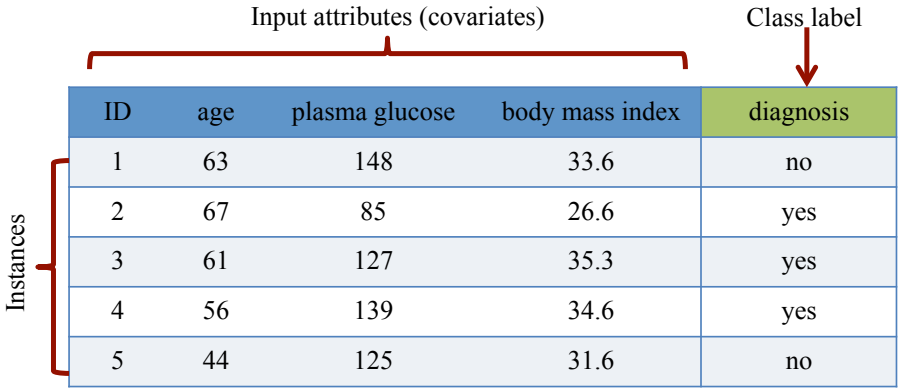
Machine learning plays an important role in today's research because of the ability to automate complex tasks. Such algorithms explore and learn from previous experience, knowledge, and data. They build a model from instance inputs called training data in order to use it later for future observations. Two of the most widely used types of machine learning methods are supervised learning and unsupervised learning.

In supervised learning, the training data needs to be labelled and classified by human experts. Classification and regression are considered types of supervised learning algorithms that are also called predictive learners [1]. On the contrary, unsupervised learning is used when we do not have the ground truth (labels) or the right answer. Clustering is the leading technique in unsupervised learning that segments training data into different groups [24].

In classification training data has discrete class labels, as the task is to predict which is the class label for unlabelled data unseen during training. In contrast in regression, the class is a real value rather than a category such as house prices or gas consumptions, and the task is to estimate those values. An example of classification task is the diagnosis of diabetes. The patient's record should have

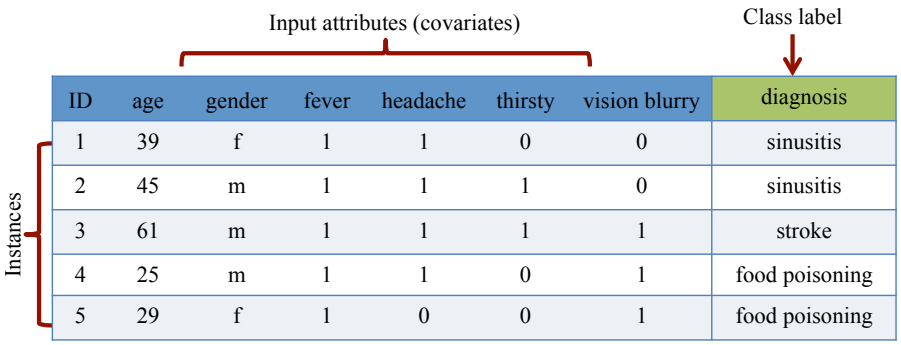
some measurements such as laboratory tests besides a class label: either positive "+" or negative "-". Accordingly, patient records are called labelled training data and they can be used to build the diagnosis model. At deployment, the trained model is available to provide predictions for unlabelled deployment (testing) data.

Traditional classification allows only a single label, and the task can be binary or multi-class classification. In the former, the class label can only be one of two categories or classes as in the diabetes example (Figure 2.1). Each row is a training instance, all columns represent input attributes except the last, which is the class label. In multi-class classification, the class label can be one out of many class values as shown in Figure 2.2.



ID	age	plasma glucose	body mass index	diagnosis
1	63	148	33.6	no
2	67	85	26.6	yes
3	61	127	35.3	yes
4	56	139	34.6	yes
5	44	125	31.6	no

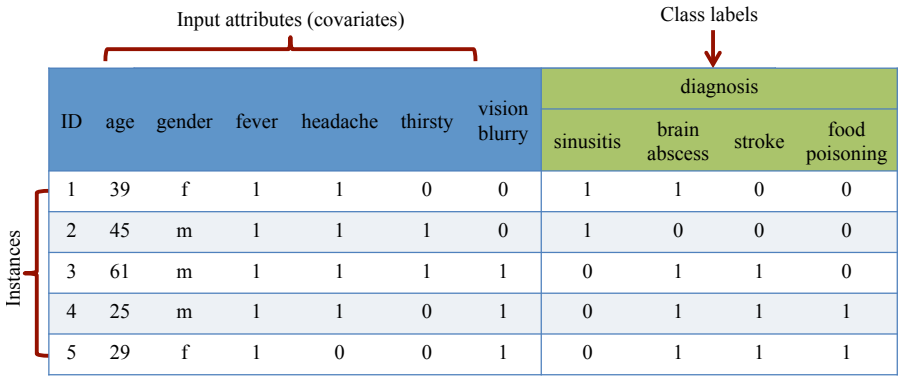
Figure 2.1: An example of binary classification task.



ID	age	gender	fever	headache	thirsty	vision blurry	diagnosis
1	39	f	1	1	0	0	sinusitis
2	45	m	1	1	1	0	sinusitis
3	61	m	1	1	1	1	stroke
4	25	m	1	1	0	1	food poisoning
5	29	f	1	0	0	1	food poisoning

Figure 2.2: An example of multi-class classification task.

On the other hand, multi-label classification considers more than one label per training instance. For example, in the area of automatic medical diagnosis, multiple diseases can be associated and predicted according to patient symptoms. An example of the representation of multi-label data can be seen in Figure 2.3. The patient record includes some of the medical symptoms labelled either as relevant "1" or irrelevant "0" to a particular disease. All of the above-mentioned topics will be discussed in more details in the following sections.



ID	age	gender	fever	headache	thirsty	vision blurry	diagnosis			
							sinusitis	brain abscess	stroke	food poisoning
1	39	f	1	1	0	0	1	1	0	0
2	45	m	1	1	1	0	1	0	0	0
3	61	m	1	1	1	1	0	1	1	0
4	25	m	1	1	0	1	0	1	1	1
5	29	f	1	0	0	1	0	1	1	1

Figure 2.3: An example of multi-label classification task.

2.2 Notations

Classification algorithms use training instances, which have been labelled previously in order to learn predictive models. Each instance is associated with attribute vector(s) and label vectors(s). In single-label classification, the label space is a single vector allowing only one label for each instance, while in the multi-label setting, it is extended to multiple vectors enabling multiple labels per instance.

We define next some key notations used throughout the thesis. Let $\mathscr{X}$ be the instance space and $\mathscr{Y} = \{l_1, l_2, l_3, \ldots, l_q\}$ be the label space with q possible labels. In single-label classification (binary or multi-class), q is equal to 1, while in multi-label classification, q is greater than 1. Each subset of $\mathscr{Y}$ is called label set. $D = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), (\mathbf{x}_3, \mathbf{y}_3), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ is any dataset with n instances, where $\mathbf{x}$ is an instance and $\mathbf{y}$ is a set of labels associated with that instance.

$$\mathbf{x} = (x_1, x_2, x_3, \ldots, x_{att}) \in \mathscr{X}$$
$$\mathbf{y} = \{y^1, y^2, y^3, \ldots, y^z\} \subseteq \mathscr{Y}, \quad z \leq q$$

where att, z, and q represent the number of input attribute(s), the number of relevant labels, for instance, $\mathbf{x}$, and the dimensionality of label space, respectively. Both att and q are fixed in the training dataset, whereas, z can be different from instance to instance. Labels can be represented as a q-dimensional binary vector $\{0,1\}^q$, where 1 means this label is relevant and 0 otherwise.

A model is a function $\mathscr{X} \rightarrow \Re^q$ that maps instances to vectors of scores. A classifier is a function that maps an instance $\mathbf{x}$ from $\mathscr{X}$ to labels from $\mathscr{Y}$. Let w be the number of possible classes per label. Single-label classification has only a single label, i.e., $q = 1$, whereas multi-label classification has more than one label, i.e., q>1. A multi-label classifier is a function $h_{mlc} : \mathscr{X} \rightarrow 2^{\mathscr{Y}}$. A multi-class classifier is defined as $h_{mcc} : \mathscr{X} \rightarrow \mathscr{Y}$, whereas a binary classifier is defined as $h_{bc} : \mathscr{X} \rightarrow \{0,1\}$. The possible values that can be taken by w and q to determine the type of classification are shown in Table 2.1. Note that multi-dimensional classification is out of the scope of this thesis. Table 2.2 summarises the notations used throughout the thesis.

Table 2.1: Classification taxonomy based on the number of labels q and the number of classes per label w.

	$q=1$	q>1
$w=2$	binary	multi-label [25]
w>2	multi-class	multi-dimensional [26]

Table 2.2: Notations used throughout the thesis.

Notation	Definition
D	Dataset
$\mathscr{X}$	Instance space
$\mathscr{Y}$	Output space
L	A finite set of labels
X	Attribute vector
Y	Label vector
$n=\|D\|$	# Instances
$q=\|L\|$	# Labels
att	# Attributes
w	# Classes per label
$\mathbf{x}=(x_1,x_2,...,x_{att})\in\mathscr{X}$	Instance
$\mathbf{y}=\{y^1,y^2,...,y^z\}\subseteq\mathscr{Y}$	A set of labels associated with an instance
z	# Relevant Labels for an instance $\mathbf{x}$
$\mathbf{y}_i$	Set of actual labels for i^{th} instance
$\hat{\mathbf{y}}_i$	Set of predicted labels for i^{th} instance
y_i^j	The binary relevance of the j^{th} label with respect to the i^{th} instance
$\hat{y}_i^j$	The prediction of the j^{th} label with respect to the i^{th} instance
pr_i^j	The probabilistic confidence value of the j^{th} label with respect to the i^{th} instance
l_j	A label
cl_d	A class label
p_j	Proportion of instances that have label l_j as 1
$C=\{tr,dep\}$	Context: tr=training context and dep=deployment context
D_{tr}	Training dataset
D_{dep}	Deployment dataset
$n_{tr}=\|D_{tr}\|$	# Training instances
$n_{dep}=\|D_{dep}\|$	# Deployment instances
ct	Cost
wt	Weight

2.3 Single-label Classification

Traditional classification, which we call here single-label classification, can be binary or multi-class classification. Binary classification is one of the classical tasks in machine learning, where an instance is associated with one of two possible classes. Another traditional setting is multi-class classification that allows more than two classes.

Many classification algorithms have been developed and used in machine learning such as logistic regression, naive Bayes, Support Vector Machines (SVMs) and decision trees. Decision tree algorithms are considered one of the most widely used algorithms for classification. They are used in various

domains including decision-making and medical applications due to their explanatory power. In this thesis, we mainly use the decision tree models and adapt them to more difficult problems as we will discuss in Chapters 3 and 4. Next, we discuss decision tree learning in much more detail.

2.3.1 Logistic Regression

Logistic regression is a simple classification algorithm for learning to make binary decisions. In linear regression, we learn to predict continuous target such as house prices as a linear function of input attributes such as the location and size of the house, while in logistic regression, the target is binary, for example, a patient is diabetic (positive) or not (negative) [27].

Logit function is used to model the relationship between one or more attribute variables and the target variable as can be seen in Equation 2.1.

$$P(Y|X) = \frac{1}{e^{-\alpha X+\beta}} \tag{2.1}$$

where X and Y are the attribute and target variables, respectively. α and β determine the logistic intercept and slope.

Logistic regression can handle non-linear relationships between attributes and target variables because it applies a non-linear log transformation of the linear regression.

2.3.2 Decision Tree Learning

Decision trees are predictive models that recursively partition the covariate's space (attribute's space) into subspaces. They can be used for both classification and regression tasks. Due to their advantages, decision trees have become one of the most powerful and popular types of models in machine learning. In this section, we give an overview of several classical decision tree algorithms, analyse different splitting criteria and point out their advantages and disadvantages. We mainly focus on decision trees for classification.

A decision tree algorithm constructs a model in a top-down approach, which is known as a "divide and conquer" algorithm. They are built recursively by partitioning the training data. Decision trees consist of nodes, starting from the root node, passing the internal nodes and reaching the leaf nodes. The root node has only outgoing edges and no incoming edge. Internal nodes have both outgoing and incoming edges, while leaves have no outgoing edges.

Each internal node represents an attribute and splits the instance space into two or more subspaces based on the chosen splitting criterion. If the attribute is nominal, the instance space will be divided based on the attribute's value. In the case of the numerical attribute, one of the simplest methods is to find a cut-point and split the instance space according to it. Each leaf node represents a class label or a probability distribution over classes. For classification, the tree should be navigated starting from the root node until stopping at a leaf. The majority class of the leaf is typically selected for the final classification.

Figure 2.4 illustrates an example of a decision tree that predicts whether a patient is diabetic (positive) or not (negative). The attributes used in the tree are plasma, body mass, and age. In this example, the leaves show the class labels. The main algorithm is given in Algorithm 1.

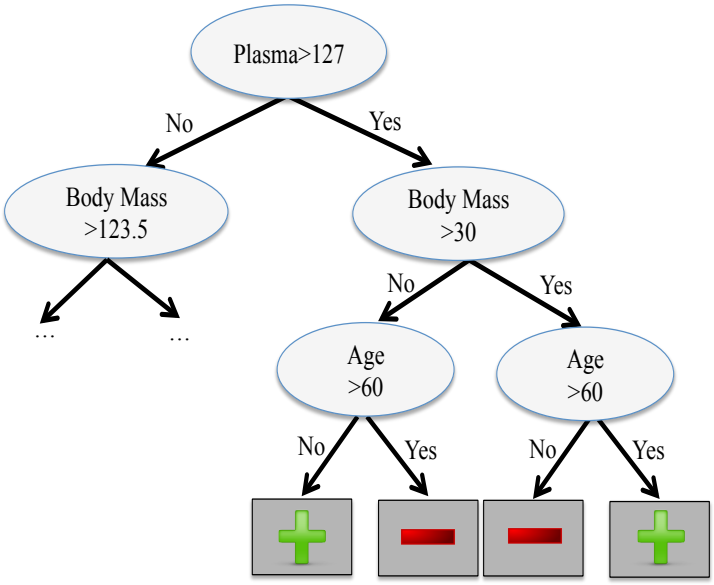


Figure 2.4: A sample decision tree showing decisions at the internal nodes and final classification at the leaves.

Algorithm 1 TreeLearning: General algorithm for learning a decision tree.

Input: Dataset D
Output: Tree

if One of the stopping criteria is fulfilled **then**
  Return Leaf with relative frequencies of labels or class label

else
  $f, \{D_i\} =$ **FindBestSplit**(D)
  for each child node D_i **do**
    Tree$_i =$ **TreeLearning**(D_i)
  end for
  Return Node splitting on f with subtrees Tree$_i$
end if

Numerous decision tree algorithms have been developed over the years. ID3, C4.5, and CART are the most common decision tree algorithms. In order to select the best attribute to split the training data according to it, the algorithm iterates over all attributes and then selects the attribute that satisfies the chosen criterion. When the best attribute is selected, the algorithm partitions the training space into smaller subsets based on the possible values of this attribute. However, in the case of numerical attributes, binary decision trees split the training space into two subsets based on a cut point. In this case, they search for the best consideration of an attribute and its cut point.

Additionally, decision tree algorithms require stopping rules such as reaching: purity, maximal depth, and minimal number of training instances at each leaf. One of the issues with such learning method is the overfitting problem, which means the generated tree may not generalise to other data not observed during training. This problem can be tackled by a bottom-up technique known as pruning [6].

2.3.3 Splitting Criteria

The selection of the best attribute for a node is made according to specific criteria, there are several splitting criteria proposed in the literature. Two well-known measures are Information Gain (IG) and Gini index. IG measures the impurity using Shannon's entropy measure [5]. Firstly, the algorithm computes the entropy before partitioning the data as follows,

$$\text{Entropy}(D) = -\sum_{d=1}^{w} P(cl_d)\log P(cl_d)$$

where f, w and $P(cl_d)$ are the examined attribute, the number of classes in a single-label problem and the probability of cl_d class, respectively.

Then the IG criterion compares the entropies before and after partitioning the data and selects the attribute that has the maximum information gain (i.e. entropy reduction) as follows,

$$\text{IG}(D|f) = \text{Entropy}(D) - \sum_{i=1}^{vals(f)} \frac{|D_i|}{|D|}\text{Entropy}(D_i|f)$$

where $vals(f)$ stands for the number of possible values of the examined attribute(s) and D is the data subset where instances have the $i-th$ values of attribute f.

Although the information gain criterion is widely used in decision tree algorithms, it is biased towards attributes with a large number of distinct values. Therefore, it is advised to use the Information Gain Ratio (IGR) instead as below [28].

$$\text{IGR}(D|f) = \frac{\text{IG}(D|f)}{\text{Entropy}(f)}$$

On the other hand, the Gini index is another way to measure the impurity as given below [5].

$$\text{Gini}(D) = \sum_{d=1}^{w} P(cl_d)(1-P(cl_d))$$

The average Gini index is computed as follows.

$$\text{Gini}(D|f) = \sum_{i=1}^{vals(f)} \frac{|D_i|}{|D|}\text{Gini}(D_i|f)$$

Both the entropy and the Gini index reach their minimum value when a set of training instances becomes pure, meaning that they share the same class value. Conversely, they reach their maximum value when a set of training instances has all classes with equal probabilities. Several studies in the

literature discussed whether the performance of decision tree algorithms could be affected by favouring one measure over the other. For example, Raileanu and Stoffel [29] concluded that both measures can perform equally with only 2% disagreement. Moreover, the choice between both measures does not have a significant impact on the performance of the algorithm.

The previous measures are called univariate attribute selection approaches as they examine only a single attribute at a time, which might not be the best approach for analysing the data [30]. In other words, there are some attribute relationships that these measures do not model. Beside the above methods, several studies proposed multivariate approaches to overcome the problems with univariate attribute selection [31].

2.3.4 Decision Tree Algorithms

There are several algorithms introduced in the literature and the most popular of them will be listed below.

ID3: It stands for Iterative Dichotomiser 3. It is a simple algorithm that is used for classification. It uses the IG to search for the best attribute to split on [28] and it is available in Weka as ID3 [32]. ID3 deals with only nominal attributes without missing values. Moreover, it does not support pruning. Although it is historically important, its strong limitations mean in practise it has been superseded by more advanced decision tree algorithms.

C4.5: It is an extension of ID3 that uses the IGR as the splitting criterion instead of IG [33]. It is known as J48 in Weka. C4.5 incorporates additional features such as the ability to handle numerical attributes, missing values, and pruning.

CART: It stands for Classification and Regression Trees. It selects the best attribute for a node using the Gini index [34]. It is known as rpart in R [35]. An important aspect of CART is its ability to deal with continuous target numbers as in regression. Moreover, it supports pruning and misclassification costs. Misclassification costs will be discussed later in Section 2.5.

These algorithms are basically top-down induction of decision trees. Barros et al. [36] proposed a system for automating the design of decision tree algorithms. They show how Genetic Programming (GP) is used for building decision trees, and the output algorithms by the GP can then be used for learning any given classification dataset.

2.3.5 Discussion

Among other advantages, decision trees are intuitive and comprehensible, since their structure can be captured by inexpert users easily. Furthermore, they are selective, which means they select the most discriminative attributes from the data to construct the trees, which usually are much less than the actual number of attributes. This is an advantage, especially in a high dimensional attribute space [5].

Nevertheless, decision trees suffer from several drawbacks. Firstly, decision trees have high variance, which means that small changes in the data may lead to a different tree and hence high variance in the predictive performance. Secondly, standard decision trees examine only a single attribution at a time, which might not be the best approach for analysing the data. In other words, there are some attribute relationships that decision trees do not model them.

2.4 Multi-label Classification

Multi-label learning has recently received attention from researchers who have proposed several learning algorithms. There exist two main tasks with respect to multi-label learning: multi-label classification (bipartition) and label ranking [8, 37, 38]. In the former, the task is to produce a bipartition of the labels into a relevant and an irrelevant set. While in the latter, the algorithm orders a set of predicted labels in such a way that the topmost labels are predicted to be the most relevant to the new instance.

Existing approaches can be categorised into two main types [25]. Decomposition approaches that transform multi-label data into several single-label problems. For example in the Binary Relevance (BR) approach, a multi-label problem with q labels results in q binary classifiers, in which all predictions are then merged to produce the final predictions. Alternatively, algorithm adaptation methods that handle multi-label data directly, such as C4.5 decision tree and neural network [25]. We extend this classification of multi-label methods with additional three groups, namely, input space transformation, output space transformation and incorporating label dependency using clustering.

In some domains, the relationships between labels may be structured hierarchically in the dataset and taking this structure into account during the classification process is called "hierarchical multi-label classification". Otherwise, ignoring this hierarchical structure simply leads to what is known as flat classification [39]. Several research attempts ignore the hierarchical relationship among labels, and simply transform the hierarchical problem into flat classification. Others rely on a tree hierarchy where each class label has a single parent (tree structure) or multiple parents as in Directed Acyclic Graphs (DAGs).

In this thesis we assume a flat label structure to keep the work well focused. The following sections explain in more details multi-label classification approaches.

2.4.1 Decomposition into Single-label Problems

A number of algorithms exist which actually decompose a multi-label problem into multiple single-label problems, performing either binary or multi-class classification. Popular decomposition approaches are one-vs-all and one-vs-one.

Binary relevance is the most common and the simplest algorithm used as one-vs-all decomposition method [25]. BR applies one binary classifier to each individual label. It transforms the original dataset D into q datasets, each of which comprises all instances of the original dataset. The instances are labelled positively if the label set for the original instance contains this label, and negatively if not.

To classify a new instance, BR outputs the union of labels that have been positively predicted by the q classifiers. Moreover, BR learns a number of classifiers equal to the number of labels, which can be hundreds or thousands in some domains. Figure 2.6 shows an example of the BR method, where it transforms the multi-label problem into four binary classification problems; one for each label and learns them independently.

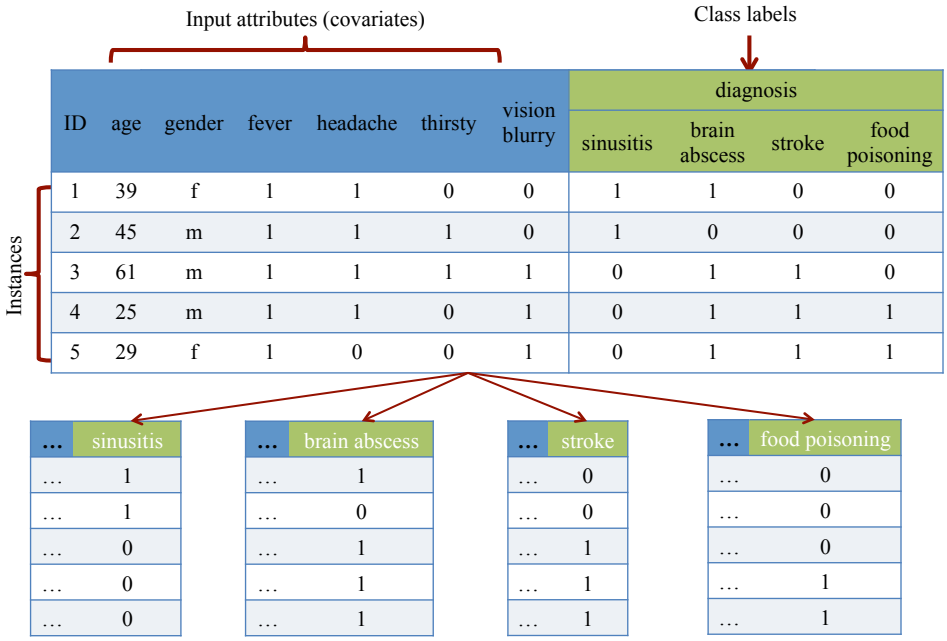


Figure 2.5: BR transforms the multi-label problem into four binary classification problems and predicts the union of them. For example, assume the base classifier is the majority class that ignores the input attributes and predicts the majority class for each label. The prediction for an unseen test instance will be two labels: *brain abscess* and *stroke*.

PairWise classification (PW) transforms a multi-label problem into several binary classification problems; one for each pair of labels l_j and l_k. Each classification problem includes instances belonging to either l_j or l_k, but not both labels. Figure 2.6 shows an example of PW, where it transforms the multi-label problem into six binary classification problems; one for each pair of labels. Similarly, Four-class pairWise classification (FW) transforms a multi-label problem into several multi-class classification problems; one for each pair of labels l_j and l_k. Each classification problem learns classes $\{00, 01, 10, 11\}$ for each pair of labels l_j and l_k. Figure 2.7 shows an example of FW, where it transforms the multi-label problem into six multi-class classification problems; one for each pair of labels.

In PW, instances labelled by l_j are considered as "positive", whereas instances labelled by l_k are considered as "negative" [40]. For testing, all classifiers are tested and the votes are counted for each label to provide the final predictions. For example in FW, 01 for the pair l_j and l_k gives one vote to label l_k and 10 gives one vote to label l_j and 11 gives one vote for the two labels l_j and l_k. Then, the accumulated votes can be converted into a binary class using a threshold. Moreover, the accumulated votes can be used to rank labels from the most relevant labels to the most irrelevant ones.

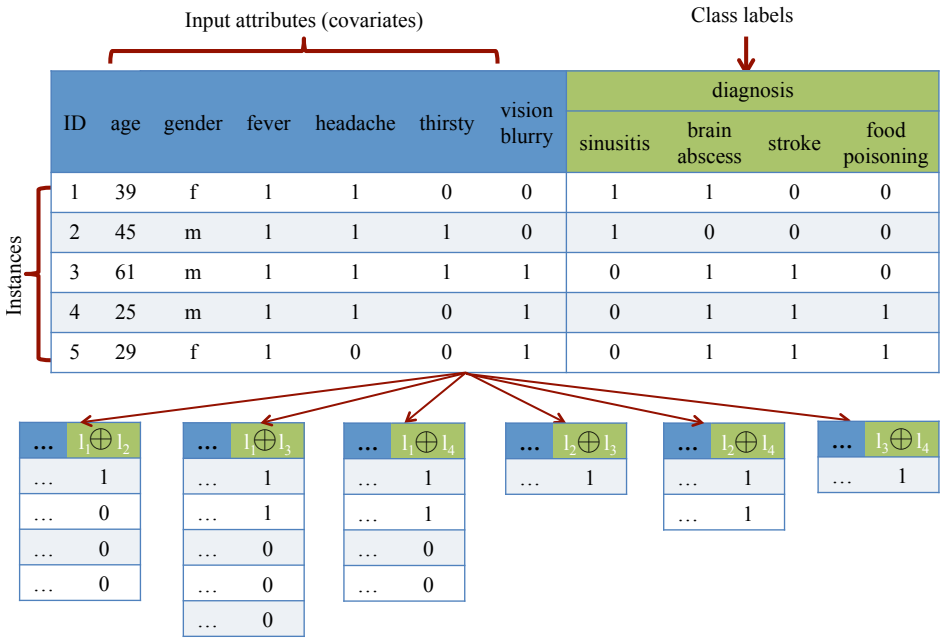


Figure 2.6: PW transforms the multi-label problem into six binary classification problems. Like in Figure , we assume the use of the majority class classifier. For a new test instance, all six models should be invoked to count the votes for each label. The number of votes for each label is $\{1,3,2,0\}$, which can be divided by $q-1=3$ (the number of classification problems where a label can be predicted) to produce the class probabilities $\{0.33,1,0.66,0\}$. The final predictions will be *brain abscess* and *stroke* using 0.5 as a threshold. l_1, l_2, l_3 and l_4 stand for labels *sinusitis*, *brain abscess*, *stroke* and *food poisoning*, respectively. In the event of a tie (having equal classes), classes are chosen at random.

PW and FW are one-vs-one decomposition methods that perform well in some contexts, but the complexity in terms of the number of models is $\frac{q(q-1)}{2}$ for q labels, meaning that this approach is usually intractable on problems with a large number of labels. FW uses multi-class classifiers which are always considered complex than binary classifiers that are used in PW. In binary classification, only the decision boundaries of one class are to be known and the rest is considered as second class whereas, in multi-class classification, several boundaries are essential. This may increase the probability of error because of constructions of many decision boundaries.

Another popular method that combines both one-vs-all (as in BR) and one-vs-one (as in PW and FW) is called Calibrated Label Ranking (CLR). CLR was introduced in [41], which extends the idea of PW by introducing an artificial label λv. It decomposes the multi-label data into q binary problems for each label. $q-1$ problems are like PW, where instances that belong to a particular label l_j are annotated "positive" and instances that do not belong to l_j are annotated negative for l_j. In addition, the $q-th$ problem for label l_j has classes l_j and λv, denoting any label different from l_j, like in one-vs-all methods. Thus, λv works as a natural cut-point that splits relevant and irrelevant labels, labels are considered relevant if they are ranked higher than λv, otherwise, they are considered irrelevant. The total number of models is $\frac{q(q-1)}{2}+q$ for q labels. Figure 2.8 gives an example of how the CLR method works.

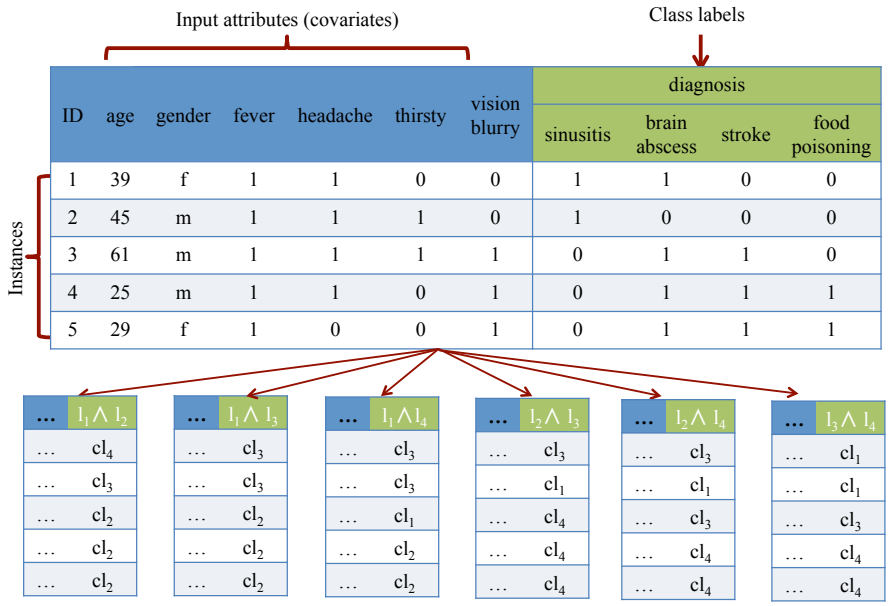


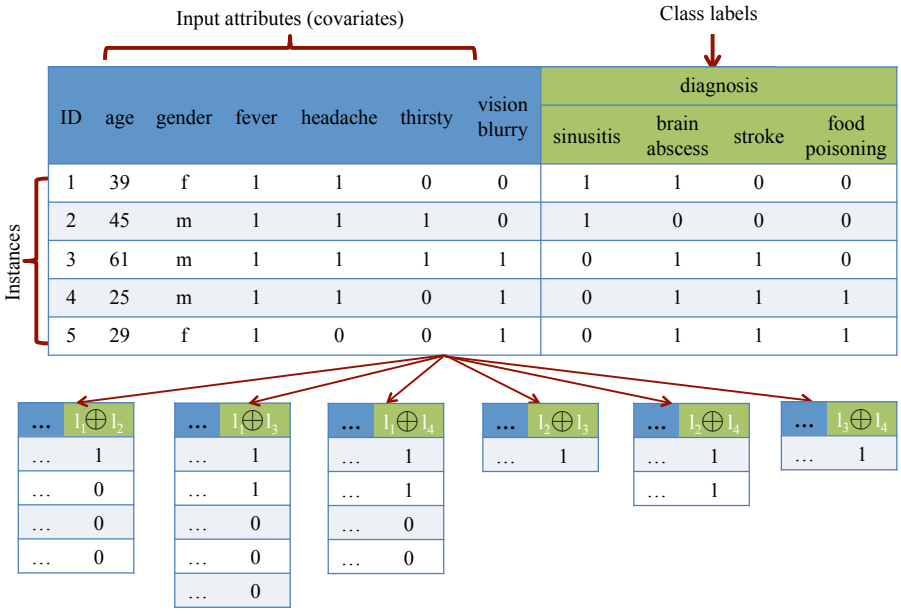
Figure 2.7: FW transforms the multi-label problem into six multi-class classification problems with four-classes ($cl_1 = 00, cl_2 = 01, cl_3 = 10, cl_4 = 11$). The accumulated number of votes computed by the majority class classifier for each label in this example are $\{1,3,3,1\}$ and the weighted votes (class probabilities) are $\{0.33,1,1,0.33\}$, respectively. The final prediction will be *brain abscess* and *stroke* assuming 0.5 is the threshold.

Additionally, Label Powerset (LP) or Label Combination (LC) is a simple and effective decomposition method, which considers each distinct set of labels that exists in a multi-label training set as a new class of a single-label classification task [42] as shown in Figure 2.9. For an unseen test instance, LP predicts the most probable class in the multi-class problem, which can be converted back to the original labels. The number of new classes in the transformed dataset is exponential with respect to q, 2^q, which can be a potential issue with this approach.

Some decomposition approaches such as BR suffer from the label independence assumption, while other methods such as LP overcome the label independence problem. However, these approaches can be inefficient for some datasets with a high number of label combinations. For example, label combinations can be exponential as in LP and some of the new labels are infrequent. In addition, new label combinations that occur only in the test set, not in the training set, cannot be handled, which cause overfitting the training data. Furthermore, the number of models can be expensive as in FW, PW, and CLR. Therefore, there are several attempts in the literature that aim to improve the aforementioned issues. We summarise these approaches below.

Pruned Sets (PS) prunes infrequent label combinations based on a pruning parameter. PS addresses the infrequent problem of LP by pruning training instances with label sets that occur less than a user-defined threshold. In fact, setting the pruning threshold to zero leads to the original LP technique. Additionally, pruned instances can be sampled and reintroduced in the training data with more frequent label sets [43]. Finally, a single-label classifier can be applied similarly to LP as seen in Figure 2.10.

1)



2)

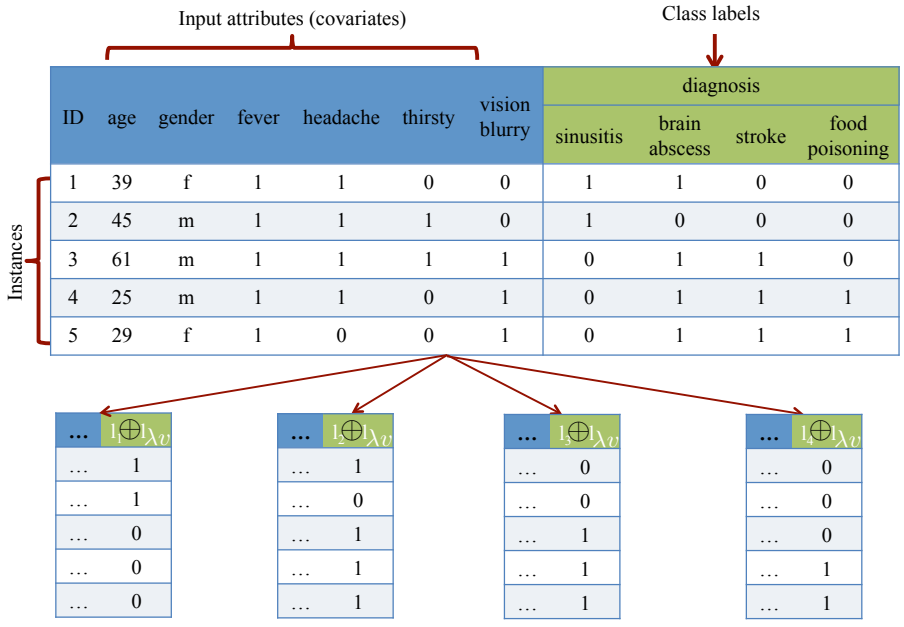


Figure 2.8: CLR involves two steps: PW and BR. The first step is to train six binary classifiers as in PW, while in the second step four binary classifiers are trained with respect to λv. Then the votes are computed by the majority class classifier for all labels including the λv by testing all classifiers in the two steps. The accumulated votes for labels in this example are as follow: $l_1 = 1, l_2 = 4, l_3 = 3, l_4 = 0$ and $\lambda v = 2$, which can be ranked as $\{l_2, l_3, \lambda v, l_1, l_4\}$. Given a new instance, the predictions will be *brain abscess* and *stroke*.

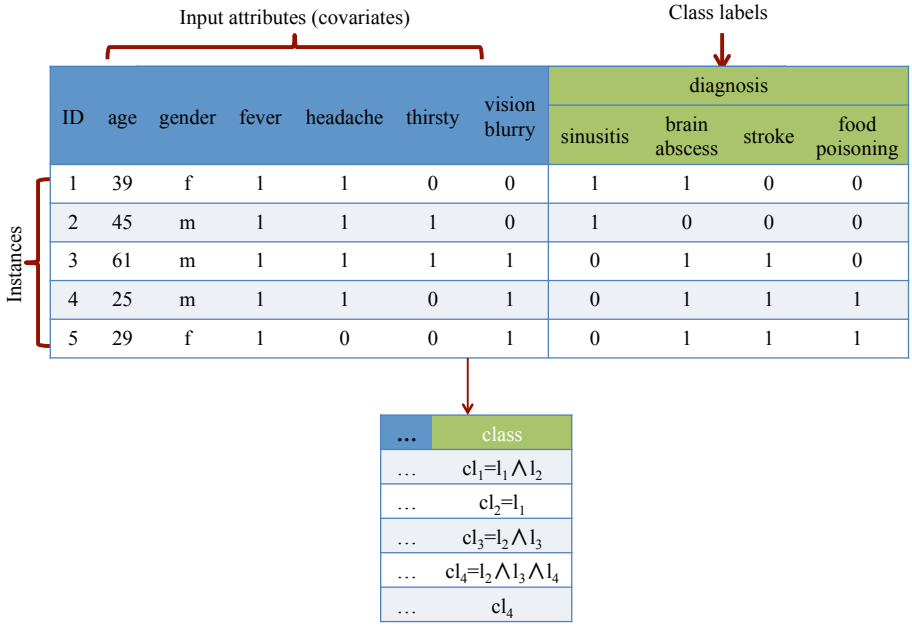


Figure 2.9: LP transforms the multi-label problem into a multi-class classification problem. In this example, we have four new classes; each of them represents a unique set of labels. Assuming the use of the majority class classifier, LP predicts the most probable class, which is $cl_4 = \{$*brain abscess*, *stroke*, *food poisoning*$\}$.

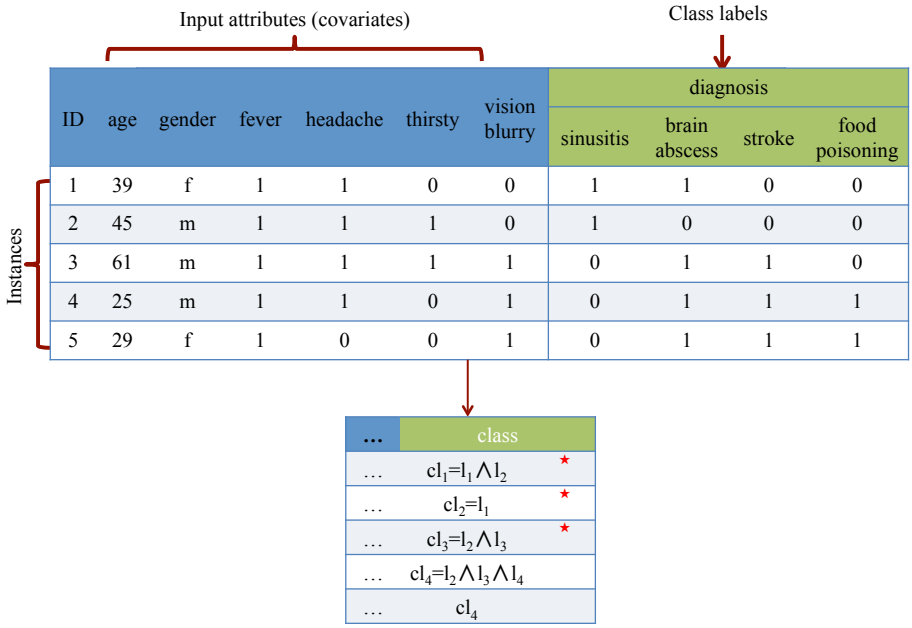


Figure 2.10: PS prunes infrequent label sets. Suppose we set a pruning parameter to 1. Training instances that are marked with * will be discarded.

Random K labelsets (RAkEL) is an ensemble version of LP that decomposes the set of labels into a random number of subsets of small size K and trains a multi-label classifier on each of them using the LP method. There are two strategies to construct the label sets: overlapping or disjoint. The overlapping case allows labels to be shared across different label sets. In other words, it allows multiple predictions for the same label by different classifiers. Conversely, the disjoint strategy builds classifiers in such a way that it guarantees distinct label sets for each classifier.

According to [42], both approaches show better performance compared to LP. Given a new instance, RAkEL queries all classifiers and determines the final classification set based on voting. The complexity of RAkEL is lower than LP [42]. An illustrative example of this approach is shown in Figure 2.11. Note that both PS and RAkEL need reasonable configuration parameters such as the pruning parameter and the size of label set.

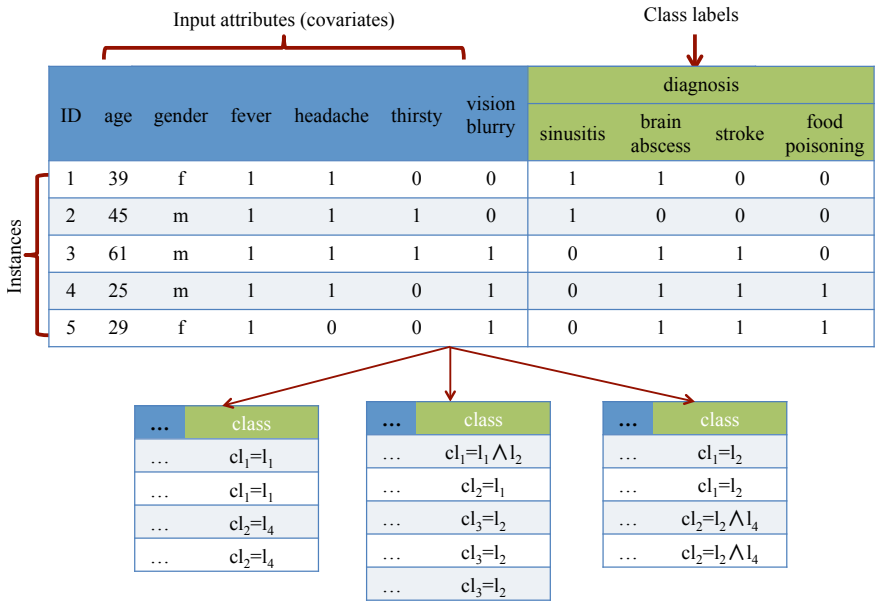


Figure 2.11: An example of the RAkEL method using the overlapping approach, where the number of models is $b = 3$ and the size of each label set is $K = 2$. RAkEL constructs three classifiers as follow: 1) the first classifier has $\{l_1, l_4\}$; 2) the second one has $\{l_1, l_2\}$; and 3) the third one has $\{l_2, l_4\}$. Using LP as multi-label classifier and the majority class as the base classifier, the output for each classifier will be as follow: $\{l_4\}$, $\{l_2\}$, $\{l_2, l_4\}$. The accumulated votes are $\{0, 2, 0, 2\}$. The prediction will be $\{$*brain abscess*, *food poisoning*$\}$. Note that we prune instances that do not belong to the chosen label set in each classifier.

2.4.2 Input Space Transformation

The aforementioned techniques decompose multi-label classification problems into several single-label problems without changing the input attribute space. In other words, the input attribute vectors are shared in all single-label problems. There exist recent attempts in the literature that try to enrich the input space by incorporating information about other labels.

A well-known algorithm called the Classifier Chain (CC) involves q binary classifiers as in BR, but links them along a chain, wherein each classifier predicts only one label as follows. The first classifier uses the original attribute space to predict the first label in the chain l_j. For the second classifier, the attribute space is extended with l_j as an input to predict the second label l_k, and so on until the last classifier has all previous labels as input attributes to predict the last label. The order of the chain can be selected at random or based on the predefined order in the dataset as shown in Figure 2.12. To predict a new test instance, the attribute space of each link in the chain is extended with the 0/1 label predictions for all previous links [9]. The same authors of CC have proposed the Ensembles of Classifier

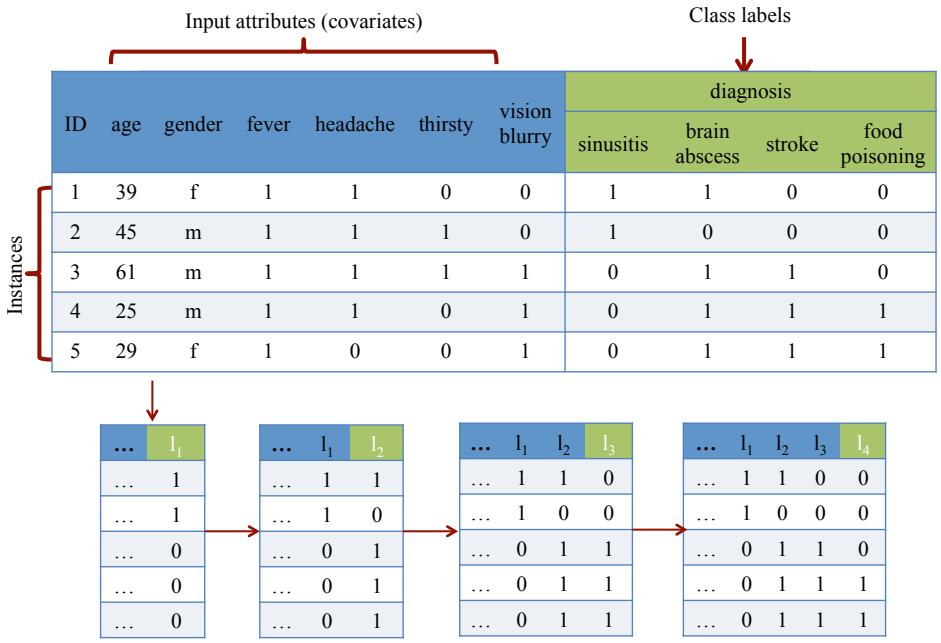


Figure 2.12: CC learns four classifiers, one for each label linked along the chain. We select the label order as they appear in the dataset. Previous labels in the chain are included as new input attributes for the current classifier.

Chains (ECCs), which are multi-label ensemble models that use CC as a base classifier by training a number of CC classifiers. Each of these classifiers is trained with a different order of labels in the chain and a random subset of the training data encouraging variability among the classifiers. Hence, each classifier is able to provide different predictions. For testing, the total number of predictions per label will be computed and used as label votes, which turn into final multi-label predictions by means of a threshold [9].

The Probabilistic Classifier Chain (PCC) was introduced in [10]. The training phase of PCC is identical to the one of CC, which selects a random label sequence. However, during the testing, PCC explores all possible chain orders to estimate the entire joint distribution and predicts the optimal label set (see Equation 2.2) given a loss function. Meaning that PCC does not depend on the prediction of the previous label, which is the case in CC.

$$P(\mathbf{y}|\mathbf{x}) = P(y^1|\mathbf{x}) \cdot \prod_{j=2}^{q} P(y^j|y^1,\ldots,y^{j-1},\mathbf{x})$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}\in\mathscr{Y}} P(\mathbf{y}|\mathbf{x}) \tag{2.2}$$

where $\mathbf{y}$ is the set of labels for an instance $\mathbf{x}$.

Although PCC achieves good performance compared to CC, it is computationally expensive due to the inference strategy to select the optimal predictions. PCC can be suitable for datasets with relatively small numbers of labels.

Clearly, CC, ECC, and PCC are sensitive to the label order in the training process. Accordingly, a number of extensions to the original CC method have been proposed in [44–47]. They aim at eliminating the key drawback in the CC, which is the lack of a principled way to decide on the label ordering.

Bayesian Chain Classifiers (BCCs) [44] model the label joint distributions conditioned on the attribute space using a Bayesian network. Based on the network, it builds a CC for correlated labels by selecting one of the nodes to be the starting point of the chain. Note that the length of the chain can be less than the size of the original label space (less than the chain length in the original CC). Finally, the prediction phase works similar to CC by including the predictions of the previous classifiers in the chain. Similarly, Gonçalves et al. [47] proposed a novel genetic algorithm that optimises label ordering. Both solutions can reduce the number of labels that can be concatenated as additional input attributes. Additionally, Monte Carlo optimisation for CC, which was proposed in [45], aims to find the best classifier chain in both the training and testing sets by sampling training data with different label orders.

da Silva et al. [46] introduced a One-To-One Classifier Chain (OOCC), which assigns a different label chain to each test instance using K-Nearest Neighbour. Both training and testing processes are different from the original CC. OOCC begins by training several classifier chains with random label orders and selects the best label orders for each training instance. In this phase, the number of classifier chains is defined by the user and the quality of each sequence is defined in terms of three evaluation measures: Hamming loss, exact-match, and multi-label accuracy, as we will discuss in Section 2.4.7. Given a test instance, the classification starts by finding the K nearest training instances to the test instance. Then, the best classifier chain is applied to the test instance.

BR plus (BR+) proposed in [48] extends the standard BR approach to overcome the independence assumption. The training phase of BR+ consists of two steps. In the first step, it learns q binary classifiers, one for each label as the standard BR. In the second step, each training binary dataset is replicated and its input attribute vectors are incremented with $q-1$ new attributes consisting of the labels of the other binary classifiers. Finally, BR+ learns a set of q new binary classifiers. Figure 2.13 illustrates the training phase for BR+.

There are two ways to classify a new instance: No Update (NU) and Update. In the case of THE No Update (NU) method, in order to compute the final prediction for each label, the attribute space of the test instances is incremented by the initial predictions produced by $q-1$ binary classifiers (see No Update in Figure 2.13).

While in the Update method, the first label uses the initial predictions as extra attributes and then the new prediction for this label is used to update the initial prediction for the second label and so on, considering a predefined label order (see Update in Figure 2.13), in a way analogy to the propagation of label predictions across a classifier chain.

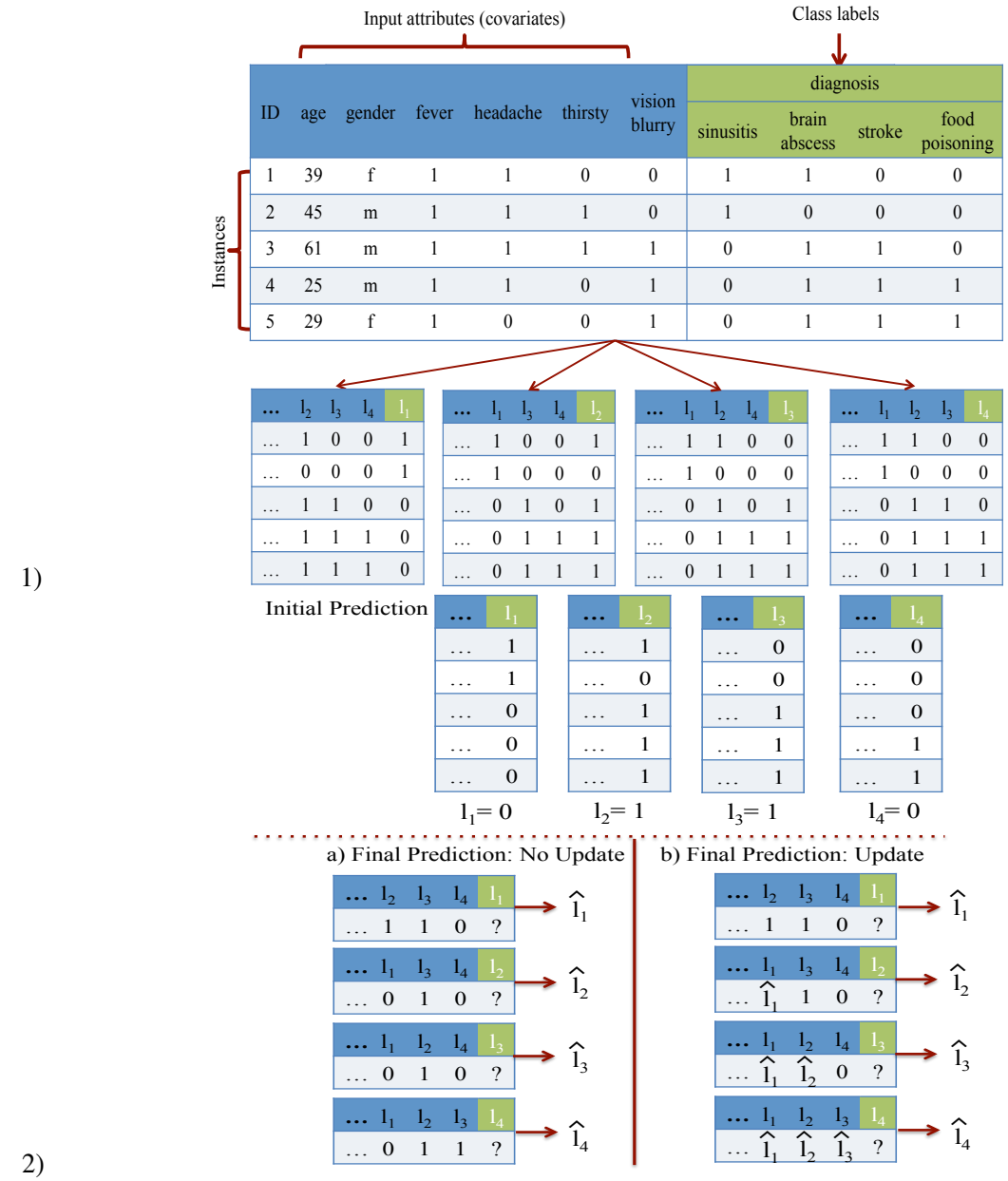


Figure 2.13: The training phase of BR+ is shown in 1). Assume the majority classifier is used, the initial predictions are as follow: $l_1 = 0, l_2 = 1, l_3 = 1, l_4 = 0$. Two ways to provide the final predictions are shown in 2). No Update (NU) uses the initial predictions as attributes and provides the final predictions. Update uses the initial predictions to provide the final prediction of the first label and then uses the new prediction as input attribute for the next label.

Although CC as well as BR+, consider label correlations by inserting labels as new attributes, there is an important point to consider. It is the fact that all labels are inserted as additional attributes. In the case of CC, labels are inserted one by one along the chain, until the last classifier in the chain uses all previous labels as extra input attributes. Whereas in BR+, each binary classifier uses all other labels as input attributes. This can be a limiting factor in high-dimensional label spaces. Furthermore, these methods force all labels to be additional attributes for the examined label, which might not be relevant or useful.

Huang and Zhou [49] proposed a method called Multi-Label learning using LOcal Correlation (ML-LOC), which first clusters the instances into K groups with respect to the similarity in the label space. Subsequently, the attribute space is extended with additional attributes encoding the cluster membership. Given a test instance, these additional attributes are predicted using a regression model. The final predictions are then obtained from any standard multi-label classifier, trained on the extended attribute space.

2.4.3 Output Space Transformation

Multi-label problems can be seen as a specific case of structured output prediction where the predictions are only multiple binary targets [50]. Although multi-label classification has become an increasingly important problem in machine learning, current approaches are restricted to learning in the original label space [51]. Guo and Schuurmans [51] proposed to use kernels on label space to significantly expand the forms of label dependence that can be captured. To express a problem in terms of a kernel over a label (output) space, it is assumed that there is a label mapping $\Psi : Y \mapsto F_Y$ that maps each label vector Y into a new representation $\Psi(Y)$. A kernel between label vectors can then be defined by an inner product between two label vectors in the new space.

Langford et al. [52] proposed a multi-label classifier called Compressive Sensing (CS), which assumes label sparsity. Firstly, they project the high-dimensional label space into a low-dimensional space using linear random projections. Then, they learn a regression model for each transformed label separately. Finally, to predict an unseen instance, they estimate the labels, which are then projected back to the original space. Tai and Lin [53] introduced Principle Label Space Transformation (PLST), which exploits label correlations globally, without requiring the sparsity assumption. It applies Principal Component Analysis (PCA) on the label space and shows performance improvement in terms of Hamming loss, especially in a high-dimensional label space.

2.4.4 Multi-label Specific Algorithms

Clare and King [54] adapted the C4.5 decision tree algorithm for binary classification to handle multi-label problems, which is called Multi-Label C4.5 (ML-C4.5). ML-C4.5 assumes labels are independent and computes the entropy per label separately as in Equation 2.3.

$$\text{Entropy}(D) = -\sum_{j=1}^{q} P(l_j)\log P(l_j) + (1-P(l_j))\log(1-P(l_j)) \tag{2.3}$$

where $P(l_j)$ is the probability that the j^{th} label is true.

To classify a new instance, ML-C4.5 predicts the majority value (positive or negative value) for each label, in the leaf node reached by the new instance.

Similarly, De Comité et al. [55] proposed Alternating Decision Trees (ADTrees) for multi-label classification based on Adaboost.MH [56]. There are two types of internal nodes: decision and prediction nodes. Nodes are generated by combining multiple decision stumps. Each node has a probability vector to support multiple labels. Another recent work was proposed in [57], which also builds a single tree for each multi-label problem. They proposed a hybrid decision tree architecture that utilises support vector machines (SVMs) at leaves. This approach is called SVM-based Decision Trees for Multi-Label learning (ML-SVMDT). It combines two types of models: ML-C4.5 and BR. It builds a single decision tree similar to ML-C4.5, where leaves contain BR classifiers that give multi-label predictions using SVM.

Multi-Label K-Nearest Neighbour (MLkNN) is an extension to k-Nearest Neighbour (kNN) algorithm for single-label classification. To make predictions for an unseen test instance, MLkNN identifies the K nearest instances to each test instance and counts the number of neighbours belonging to each label. Then, it uses the Maximum A posteriori Principle (MAP) to determine the label set for each test instance as shown below [58].

$$\hat{y}^j = \begin{cases} 1, & = \text{if } P(a_j|y^j=1)\cdot P(y^j=1) \geq P(a_j|y^j=0)\cdot P(y^j=0). \\ 0, & = \text{otherwise.} \end{cases}$$

where a_j is the number of instances having y^j as positive. $P(y^j=1)$ is the prior probability and $P(a_j|y^j=1)$ is the posterior probability for the j^{th} label within the K nearest neighbours.

The aforementioned algorithms are multi-label classifiers, but they do not consider the correlations between different labels, which is a key drawback. The next section focuses on learning algorithms that take into account label correlations during learning the model.

2.4.5 Incorporating Label Dependency using Clustering

In multi-label learning labels may be dependent or not, which means that the presence of a certain label affects the probability of other labels. Thus, exploiting dependencies among labels could improve the classifier's performance. Existing approaches for tackling label correlations are categorised into two key methods based on the degree of correlations: 1) the pairwise approach, which considers only pairwise correlation as in PW, FW, and CLR; and 2) the high-order approach, which considers correlations among all labels or subsets of them as in LP, PS, RAkEL, CC and BR+ [3, 11, 13, 49].

Dembczyński et al. [12] distinguished two types of dependencies among labels: unconditional dependencies (global) and conditional dependencies (local). In the former, dependencies are identified globally over the dataset, while in the latter, they are conditional to a subset of the dataset. In other words, the knowledge about attribute space should be incorporated in order to identify conditional (local) dependencies. For example, when two labels occur together regardless of the attribute values this is known as unconditional (global) dependencies. On the other hand, conditional dependencies between two or more labels are based on attribute values. This section reviews several existing methods proposed recently for multi-label classification considering high-order label dependencies.

Hierarchy Of Multilabel classifiERs (Homer) exploits correlations between labels by clustering them. Homer organises all labels into a tree-shaped hierarchy, with leaf nodes containing a single label. Each node has training instances that are annotated with at least one of its labels. In the training phase, a multi-label classifier is trained for each node to predict the subset of labels in that node. In particular, a leaf node constructs a binary classifier to predict its single label. Given an unseen instance, Homer starts from the root node and proceeds to any child node only if at least one of its labels was predicted by its parent node. In the end, this process reaches a subset of leaves and the final prediction is combined from predictions of these leaves.

One of the main Homer processes is the clustering of the label set into disjoint subsets using balanced K-means such that similar labels are placed together. A balanced K-means, which extends the standard K-means algorithm accepts a set of labels as input and produces K disjoint subsets of labels with roughly equal size. The main element of Homer's algorithm is that for each cluster, it maintains a list of labels in descending order based on the distance from the cluster's centroid [59].

A recent work suggested in [60] combines LP and BR methods and is called LPBR. Its first step is to explore dependencies between labels and then to cluster these labels into several independent subsets, according to the chi-squared statistic [61]. Subsequently, a multi-label classifier is learnt: if the label set contains only one label, BR is applied; for a group of dependent labels, LP is used. LPBR implements a greedy clustering algorithm that continues clustering as long as the loss function improves or one of the stop conditions is reached. The possible stop conditions are: no more label pairs to consider, all labels are clustered into one single group, exceeding a non-improving counter, which is a user-defined parameter, a pair correlation score is below some threshold value. While LPBR showed an improvement in terms of classification accuracy, it is computationally expensive.

2.4.6 Hierarchical Multi-label Classification

In Hierarchical Multi-label Classification (HMC) classes are organised in a hierarchy such that an instance that belongs to a particular class automatically belongs to all its superclasses. Silla and Freitas [62] have explained that any hierarchical classification problem could be considered a multi-label problem. For instance, if the output of a classifier is "bear" label, we can say that it also belongs to both "animal" and "mammal" labels, therefore the classifier can output these three classes similarly to multi-label classification.

On the contrary, others consider a hierarchical classifier to be a hierarchically multi-label classifier only if it can assign more than one class at any given level of the hierarchy to a given instance. This is an important aspect, as a hierarchically multi-label classification algorithm is more challenging to design than a hierarchically single-label one.

There are two methods for HMC: top-down (local) classifiers when the model employs a set of local classifiers and big-bang (global) classifiers when a single classifier deals with the entire hierarchy. Clare and King [63] have extended the C4.5 decision tree algorithm to handle a class hierarchy, which is called Hierarchical C4.5 (HC4.5). HC4.5 modifies the entropy calculation formula to consider some form of weighting to the upper levels of the hierarchy.

Blockeel et al. [64] presented clustering hierarchical multi-label classification where labels are organised in a hierarchy. All leaf nodes contain probabilities of all labels and a threshold is used to determine which labels are assigned to an instance. The threshold of the parent label will be greater than or equal to the threshold of the child to ensure the hierarchical relationship. They conducted a comparison between learning a separate decision tree for each label and learning a single tree that predicts all labels at once, where the latter was much faster and more efficient.

Another research that considers hierarchical relationships was proposed by Vens et al. [65], who compared three proposed approaches, which are learning a single tree for each label, learning a separate tree for each hierarchy edge and learning one tree that predicts all labels at the same time. They consider more complex class hierarchies, which is a DAG and show how decision tree approaches can be modified to support class hierarchies with a DAG structure.

Moreover, Bi and Kwok [66] introduced a hierarchical multi-label classification algorithm that can be applied to both tree and DAG hierarchies. Cesa-Bianchi et al. [67] introduced an on-line algorithm for hierarchical classification, which basically constructs a training data for each node in such a way that it only consists of instances belonging to its parent.

When a hierarchical classification method is used, loss functions should incorporate hierarchy information. The Hamming loss gives the same loss for two predictions if they have the same number of mistakes despite the fact that this error may occur at different levels of the hierarchy [68]. Hierarchical loss functions consider misclassifications at the higher levels of the hierarchy more important than the ones at lower levels [38]. Two well-known loss functions are studied in the literature, which are Hierarchical Loss (H-Loss) and Hierarchical Multi-label Classification Loss (HMC-Loss), they are described in [67, 69].

2.4.7 Evaluation Metrics

In this section, we explain evaluation metrics for multi-label classification, some of which have been extended from single-label classification. Moreover, we present a number of evaluation metrics that were designed specifically for multi-label classification. Multi-label classification measures are divided into two main categories: 1) instance-based measures that compute the average differences between

the actual and the predicted labels over all instances; and 2) label-based measures that decompose the evaluation for each label, which is then subsequently averaged over all labels [8, 14, 70, 71].

When multiple labels are to be retrieved, averaging the evaluation measures can be achieved for all labels using two averaging operations: macro-averaging and micro-averaging [72]. Macro-average averages the measure for each label independently, while micro-average integrates all label predictions and computes a single value over all of them. The most common evaluation metrics are summarised below.

Multi-label accuracy: It is an instance-based measure that was introduced in [73]. It calculates the ratio of the union over the intersection of the predicted and actual label sets and averages over the number of instances (Equation 2.4). It is also known as the Jaccard index [71]. Figure 2.14 gives an example of how to calculate this metric.

$$\text{mla} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\mathbf{y}_i \cap \hat{\mathbf{y}}_i|}{|\mathbf{y}_i \cup \hat{\mathbf{y}}_i|} \tag{2.4}$$

where $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ are the set of actual and predicted labels for an instance, respectively.

	actual labels				predicted labels			
	l_1	l_2	l_3	l_4	l_1	l_2	l_3	l_4
Instances	**1**	1	0	0	**1**	0	1	0
	1	0	0	0	**1**	0	0	0
	0	**1**	**1**	0	1	**1**	**1**	0
	0	**1**	**1**	1	0	**1**	**1**	0
	1	0	**1**	0	**1**	0	**1**	0

$$mla = \frac{1}{5}(\frac{1}{3} + \frac{1}{1} + \frac{2}{3} + \frac{2}{3} + \frac{2}{2}) = 0.73$$

Figure 2.14: An example of how to compute multi-label accuracy (the Jaccard index) for five instances, which is the ratio of the intersecting set to the union set. Red colour shows the intersection between the actual and the predicted label sets.

Exact-match: It is an instance-based metric, which is also called subset or classification accuracy. It examines whether the predicted and actual label sets are equal or not [74]. Subset accuracy is known as a very strict measure since it requires the predicted set of labels to be matched exactly to the actual set of labels. An example of how to compute the exact-match metric is shown in Figure 2.15.

$$em = \frac{1}{n}\sum_{i=1}^{n} I(\mathbf{y}_i = \hat{\mathbf{y}}_i) \tag{2.5}$$

where $I(true) = 1$ and $I(false) = 0$.

	actual labels				predicted labels			
	l_1	l_2	l_3	l_4	l_1	l_2	l_3	l_4
Instances	1	1	0	0	**1**	**0**	**1**	**0**
	1	0	0	0	1	0	0	0
	0	1	1	0	**1**	**1**	**1**	**0**
	0	1	1	1	**0**	**1**	**0**	**0**
	1	0	1	0	1	0	1	0

$$em = \frac{(0+1+0+0+1)}{5} = 0.4$$

Figure 2.15: An example of how to compute exact-match, where five is the number of instances. Red colour shows the three misclassified label sets, the remaining two instances (in black colour) are correctly predicted label sets.

Hamming loss: It is one of the popular instance-based metrics that has been used widely in the literature. It takes the proportion of misclassified labels (labels predicted incorrectly and actual labels that are not predicted) averaged over all instances [75]. The lower the value of Hamming loss is, the better. Figure 2.16 gives an example of how to compute Hamming loss in multi-label problems.

$$hl = \frac{1}{q*n}\sum_{i=1}^{n}|\mathbf{y}_i \triangle \hat{\mathbf{y}}_i| \tag{2.6}$$

where $\triangle$ is the symmetric difference.

	actual labels				predicted labels			
	l_1	l_2	l_3	l_4	l_1	l_2	l_3	l_4
Instances	1	1	0	0	1	**0**	**1**	0
	1	0	0	0	1	0	0	0
	0	1	1	0	**1**	1	1	0
	0	1	1	1	0	1	**0**	**0**
	1	0	1	0	1	0	1	0

$$hl = \frac{5}{5*4} = 0.25$$

Figure 2.16: An example of how to compute Hamming loss, where four is the number of labels, five is the number of instances and five is the total number of mistakes.

Log-loss: It evaluates a classifier's confidence by punishing errors with higher probability more severely [9]. It is considered as a label-based evaluation measure. The lower the value of log-loss is, the better.

$$ls = \frac{1}{q*n}\sum_{i=1}^{n}\sum_{j=1}^{q}\log(pr_i^j)\cdot y_i^j + \log(1-pr_i^j)\cdot(1-y_i^j) \tag{2.7}$$

where y_i^j indicates whether the j^{th} label is relevant for the i^{th} instance (value 1) or irrelevant (value 0). pr_i^j is the probability estimate for the i^{th} instance and the j^{th} label.

Brier score: It evaluates the estimated a posteriori probabilities for each label. It is considered as a label-based evaluation measure where the lower the value of Brier score is, the better [76].

$$BS = \frac{1}{q*n}\sum_{i=1}^{n}\sum_{j=1}^{q}(pr_i^j - y_i^j)^2 \tag{2.8}$$

where y_i^j indicates whether the j^{th} label is relevant for the i^{th} instance (value 1) or irrelevant (value 0). pr_i^j is the probability estimate for the i^{th} instance and the j^{th} label.

Mean Absolute Error: It computes the average absolute error between the estimated probability and the actual label. The lower the value of mean absolute error is, the better [77]. It is a label-based evaluation measure.

$$MAE = \frac{1}{q*n}\sum_{i=1}^{n}\sum_{j=1}^{q}|pr_i^j - y_i^j| \tag{2.9}$$

where y_i^j indicates whether the j^{th} label is relevant for the i^{th} instance (value 1) or irrelevant (value 0). pr_i^j is the probability estimate for the i^{th} instance and the j^{th} label.

micro F_1: It corresponds to the harmonic mean of precision and recall [71]. It is an instance-based metric.

$$\text{micro } F_1 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{q} 2\cdot y_i^j \cdot \hat{y}_i^j}{\sum_{i=1}^{n}\sum_{j=1}^{q}(y_i^j + \hat{y}_i^j)} \tag{2.10}$$

macro f_l: It is the averaged F_1 score over labels [71]. It is a label-based evaluation measure.

$$\text{macro } f_l = \frac{1}{q}\sum_{j=1}^{q} F_1^{(j)} \tag{2.11}$$

where $F_1^{(j)}$ is the F_1 score for the j^{th} label vector.

macro f_e: It is the averaged F_1 score over instances [71]. It is an instance-based metric.

$$\text{macro } f_e = \frac{1}{n}\sum_{i=1}^{n} F_1^i \tag{2.12}$$

where F_1^i is the F_1 score for the i^{th} instance vector.

2.4.8 Discussion

We summarise the aforementioned algorithms based on the number of trained models and their computational time complexity as shown in Table 2.3.

For some methods, for example, LP and PS, the time complexity includes running time as well as the pre-processing time required for transforming the data. We also consider in our comparison whether label correlations are exploited or not. In addition, Table 2.4 shows the total number of label votes for algorithms that used voting schema such as RAkEL.

Table 2.3: Comparison of multi-label classification methods. q and b denote the number of labels and the number of classifiers, respectively. K could be the number of clusters for Homer and ML-LOC, the size of label subsets for RAkEL and LPBR, and the number of neighbours in MLkNN.

Category	Algorithm	# classifiers	Computational time complexity in terms of q	Correlation level
Decomposition	Binary Relevance (BR)	q	linear in q	no
	PairWise classification (PW)	$\frac{q(q-1)}{2}$	quadratic in q (q^2)	pairwise
	Four-class pairWise (FW)	$\frac{q(q-1)}{2}$	quadratic in q (q^2)	pairwise
	Calibrated Label Ranking (CLR)	$\frac{q(q-1)}{2}+q$	quadratic in q (q^2)	pairwise
	Label Powerset (LP)	1	exponential in q (2^q)	high-order
	Pruned Sets (PS)	1	exponential in q (2^q)	high-order
	Random K Labelset (RAkEL)	b	exponential in K (2^K)	high-order
Input space transformation	Classifier Chain (CC)	q	linear in q	high-order
	BR+	$2 \cdot q$	linear in q	high-order
	ML-LOC	K+q	linear in q	high-order
Multi-label algorithms	ML-C4.5	1	linear in q	no
	ADTrees	1	linear in q	no
	ML-SVMDT	1	linear in q	no
	MLkNN	1	linear in q	no
Label clustering	Homer	K	linear in q	high-order
	LPBR	q	exponential in K (2^K)	high-order

Table 2.4: Comparison of multi-label classification methods that use voting schema. q, b and K denote the number of labels, the number of classifiers and the size of clusters, respectively.

Algorithm	Maximum number of votes per label
PairWise classification (PW)	$q-1$
Four-class pairWise (FW)	$q-1$
Calibrated Label Ranking (CLR)	q
Random K Labelset (RAkEL)	q

It is widely recognised in the literature that there is a wide range of evaluation metrics in multi-label classification that measure different aspects of predictive accuracy. There is no algorithm that performs best in all of them [3, 8]. Therefore, the selection of the learning algorithm depends on the problem, data and the requirements such as the evaluation measures and what is going to be optimised. For example, Hamming loss measures the incorrect predictions for each label independently, whereas exact-match is a very strict metric that requires the entire label set to be predicted correctly.

Hamming loss is clearly oriented to learners that model each label separately such as BR, whereas exact-match can be considered an optimal measure for classifiers that consider label correlations such as LP. One issue with LP is the overfitting problem, as it models the label combinations that appear only in training data.

On the other hand, multi-label accuracy is a more balanced measure and can be considered in between Hamming loss and exact-match metrics. It gives importance to the true labels among others. Whereas Hamming loss gives equal importance to all labels and exact-match focuses on the whole set of labels.

However, multi-label accuracy is relatively sensitive to label cardinality, which means it can be lower in datasets with higher label cardinality [78]. Rokach et al. [78] in their work concluded that multi-label accuracy and micro-averaged F-measure are better general evaluation measures for most regular multi-label classification problems, whereas Hamming loss and exact-match measures may be appropriate for some specific multi-label classification problems.

We believe that by modelling label correlation we could improve the accuracy as well as the Hamming loss for minority labels. However, algorithms that exploit label correlations can be useful in some datasets that include high label correlations.

2.4.9 Multi-label Benchmark Datasets

We select 13 commonly used benchmarks from the Meka[1] and Mulan[2] repositories [79]. The key statistics of these datasets are given in Table 2.5, where q is the number of labels, n is the number of instances and att is the number of attributes. Label cardinality and density are denoted by $card(D)$ and $dens(D)$, respectively. $Lsets$ is the number of unique label sets in the dataset.

Label cardinality is the average number of labels per instance in D, while $dens$ is the density with respect to labels in D, as shown below [25].

$$card(D) = \frac{1}{n}\sum_{i=1}^{n}|\mathbf{y}_i|$$

$$dens(D) = \frac{card(D)}{q}$$

where $\mathbf{y}_i$ is the set of actual labels for the i^{th} instance and q is the number of labels in the dataset D.

A model's performance can be affected by the cardinality as discussed in [14, 80, 81]. They concluded that a lower cardinality value can achieve better results in terms of Hamming loss, multi-label accuracy and F_1 measure. Next, we describe each dataset in more details. Also, we use two kinds of plots in order to get more insights on multi-label datasets. Both plots are generated in R using the mldr package for multi-label datasets [82]. Firstly, we use the histogram, which shows the frequency for each label in the dataset. This plot provides information on the majority and minority labels.

[1] http://meka.sourceforge.net/

[2] http://mulan.sourceforge.net/

Table 2.5: The statistics of multi-label benchmark datasets. q is the number of labels, n is the number of instances, att is the number of attributes, $card(D)$ and $dens(D)$ are the label cardinality and density in the dataset, respectively. $Lsets$ is the number of unique label sets in the dataset. Datasets are sorted in descending order with respect to q.

	q	n	att	$card(D)$	$dens(D)$	$Lsets$
Corel5k	374	4500/500	499	3.522	0.94%	3175
Cal500	174	502	68	26.044	14.96%	502
Bibtex	159	4880/2515	1836	2.402	1.51%	2856
Language log	75	1460	1004	1.179	1.57%	304
Enron	53	1702	1001	3.378	6.37%	753
Medical	45	978	1449	1.245	2.76%	94
Genbase	27	662	1186	1.252	4.63 %	32
Slashdot	22	3782	1079	1.180	5.36%	157
Birds	19	322	242	1.014	5.30%	89
Yeast	14	2417	103	4.237	30.26%	198
Flags	7	194	19	3.392	48.45%	54
Emotions	6	593	66	1.869	31.14%	27
Scene	6	2407	294	1.074	17.89%	15

Secondly, we use a circular plot that indicates label correlations. The circular plot is partitioned into several disjoint arcs with different lengths, where each arc represents a label in the dataset. The length of the arc is adjusted to the total number of pairwise correlations between a particular label and other labels. The pairwise correlation between two labels is represented in the band that joins them, where the width of the band is proportional to the number of instances belonging to both labels. Thus, it gives an indication of the strength of the correlation. It is not expected to represent a label that does not interact with other labels [83].

Corel5k: It contains 499 attributes extracted from a collection to describe images. Attributes were extracted using 5000 images, 4500 used for training and 500 for testing. The task is to annotate an image with a set of tags. The number of possible tags is 374, which represents the number of labels. Examples of image tags are Waves, Dogs, Ships, Island, Jet and Plane [84]. Figure 2.17 shows the relative frequencies for the first 40 labels (on the left) and their interactions (on the right). We can notice that Sky label has more interactions with other labels, it has 1529 pairwise correlations.

Cal500: It has information about 502 songs collected from Computer Audition Lab [3]. Each song has 68 extracted attributes. Each song has been manually annotated by at least three different human annotators and the task is to classify the song into 174 tags. Tags are classified into six categories: Emotions, Genres, Instrumentation, Usage, Vocals and Features [85]. Figure 2.18 shows the relative frequencies for the first 40 labels (on the left) and their interactions (on the right).

[3] http://eceweb.ucsd.edu/ gert/calab/

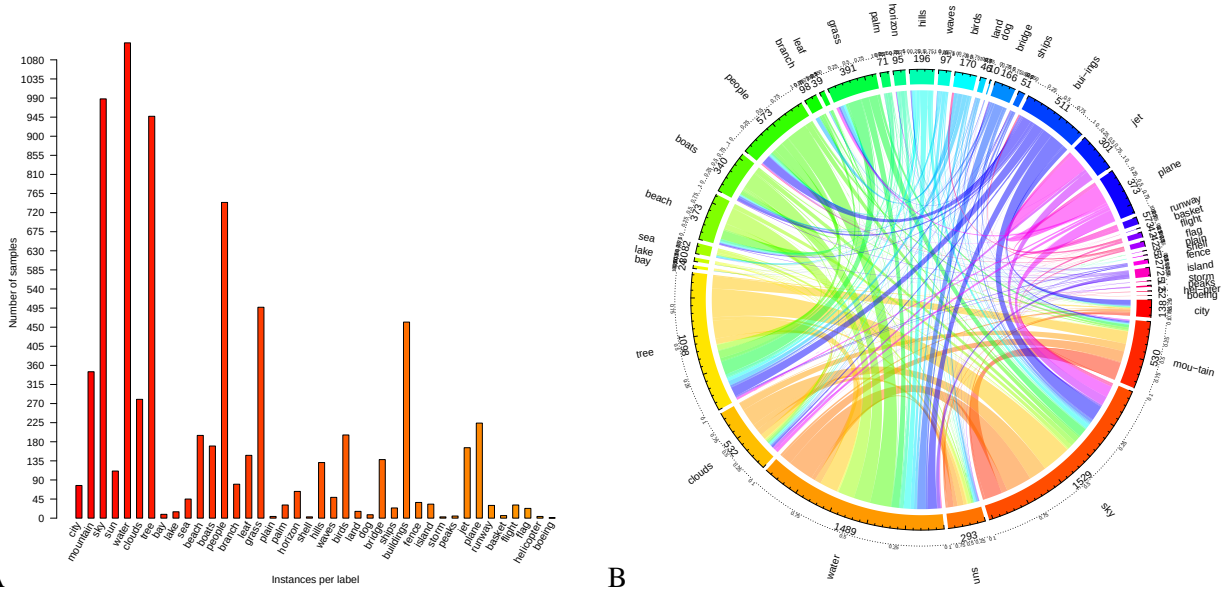


Figure 2.17: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Corel5k dataset. On the left plot, we can see that Water is the majority label, while there are some minority labels such as Shell and Storm. On the right side of the figure, Sky has more interactions with other labels, it has 1529 pairwise correlations.

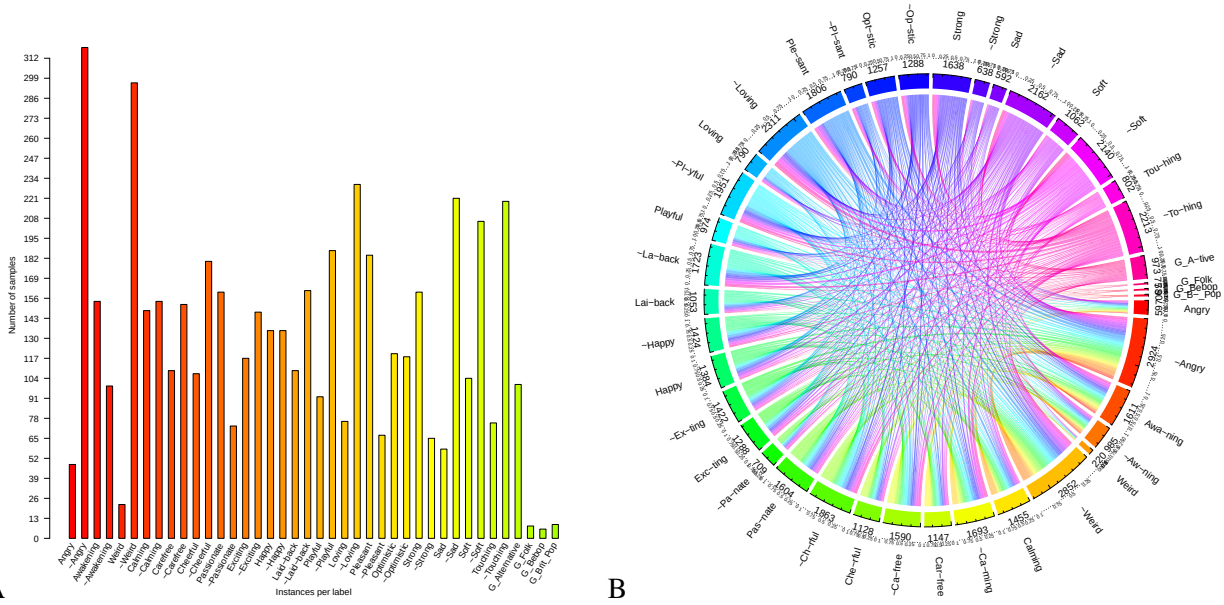


Figure 2.18: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Cal500 dataset.

Bibtex: It has 7395 bookmark entries in the blue social bookmark and publication sharing system [4]. The dataset is partitioned into 4880 instances for training and 2515 for testing. The task is to tag the bookmark entries with a set of relevant tags. The total number of candidate tags is 159. Examples of the tags include Algorithms, Architecture, Classification, Clustering, Cognition, and Computer [86]. Figure 2.19 (A) shows the distribution of the first 40 labels in the dataset, while (B) gives an indication of the concurrence among labels.

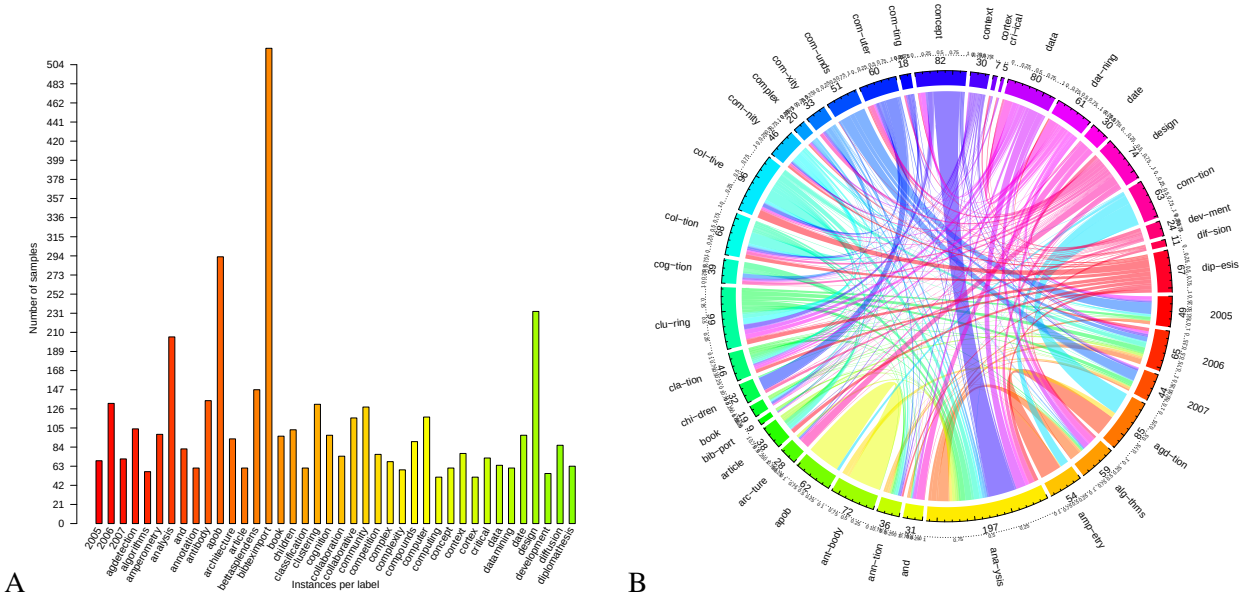


Figure 2.19: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Bibtex dataset.

Language log: It has 1460 posts collected from the language log website [5]. It has 1004 bag-of-words attributes describing each post. It contains 75 tags such as Errors, Humor, Culture, Sports, Comics and History [39]. Figure 2.20 (A) shows the distribution of the first 40 labels in the dataset, while (B) gives an indication of the concurrence among labels.

Enron: This dataset was collected by a Cognitive Assistant that Learns and Organizes (CALO) project [6]. It contains 1702 email messages that have been classified into four main categories: A) Coarse genre, B) Included/forwarded information, C) Primary topics and D) Emotional tone. Each of these categories is further divided into several tags. For example, A1 and A2 represent Company Business and Personal emails, respectively. There is a total of 53 email tags [7] [87]. Figure 2.21 (A) shows the distribution of the first 40 labels in the dataset, while (B) gives an indication of the concurrence among labels.

[4] http://www.bibsonomy.org
[5] http://languagelog.ldc.upenn.edu/nll/
[6] http://www.cs.cmu.edu/∼enron/
[7] http://bailando.sims.berkeley.edu/enron/enron_categories.txt

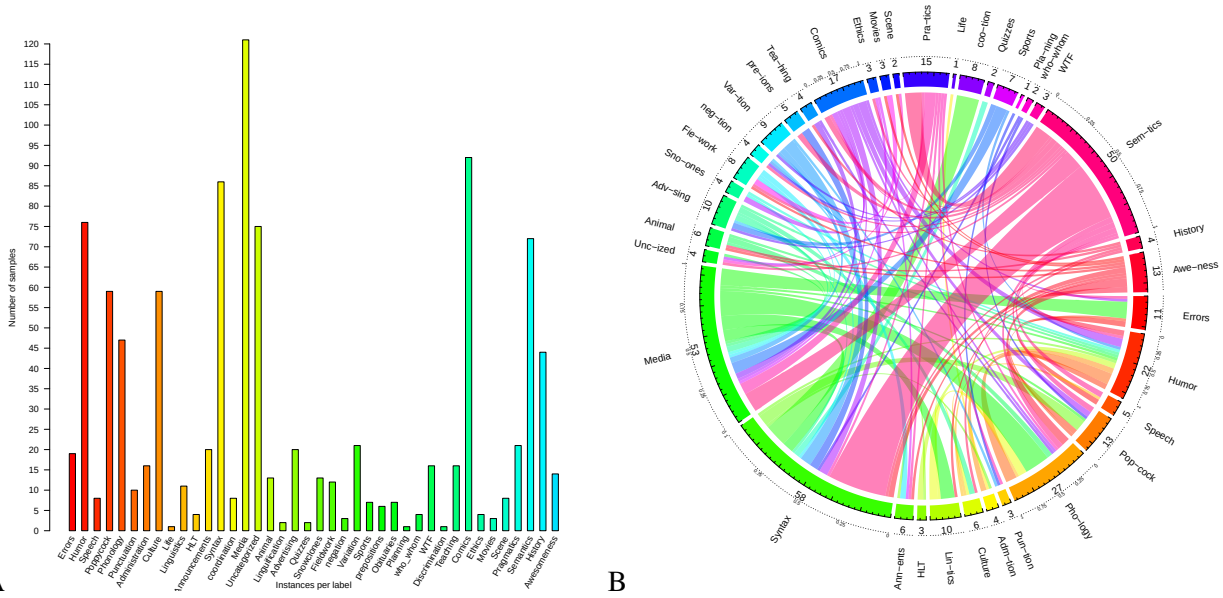


Figure 2.20: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Language log dataset. As can be seen in A, there are several infrequent labels such as Life, Planning, and Discrimination that appear in only one post.

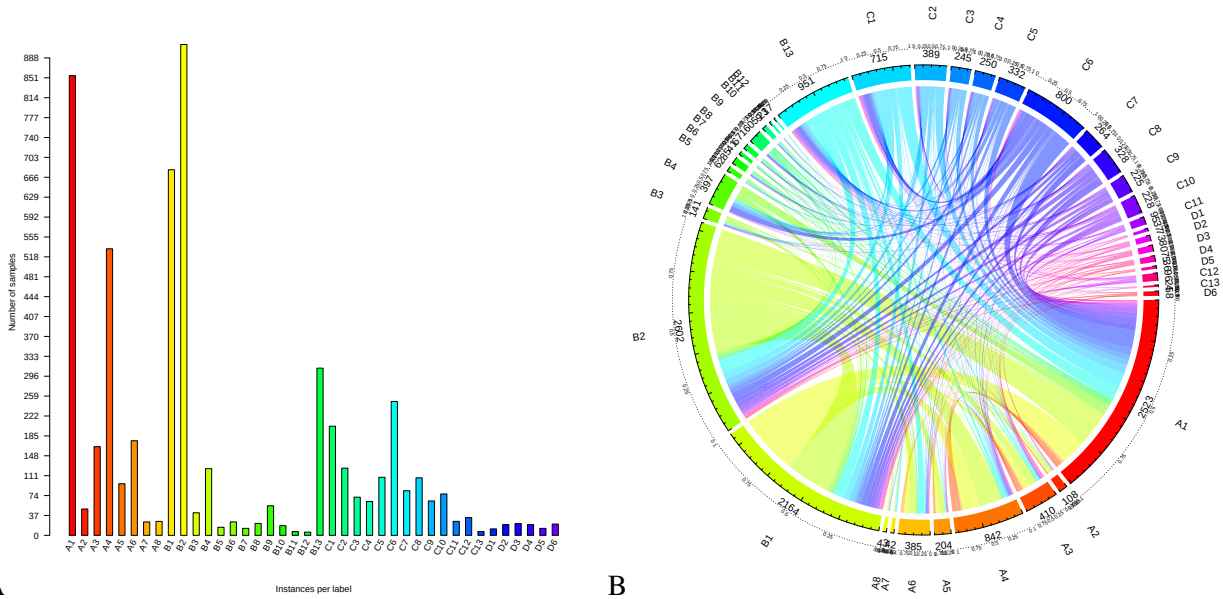


Figure 2.21: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Enron dataset.

Medical: It is concerned with classifying Clinical free text including information about patients and their clinical history. It has 978 documents collected from Cincinnati Children's Hospital Medical Center's, Department of Radiology. The task is to classify these documents into a subset of 45 standard

codes called the International Classification of Diseases, Ninth Revision, Clinical Modification (ICD-9-CM) [88]. In other words, each code represents a possible prognosis. Figure 2.22 (A) shows the distribution of the first 40 labels in the dataset, while (B) gives an indication of the concurrence among labels.

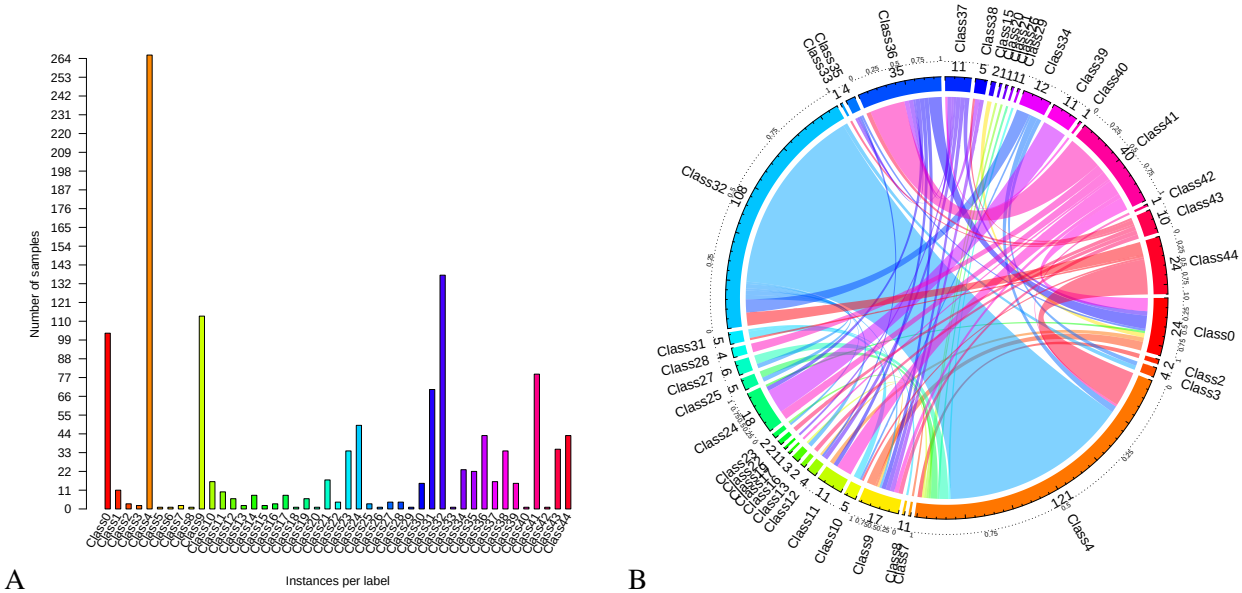


Figure 2.22: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Medical dataset. For example, Class4 and Class32 represent 753_0 (Renal agenesis) and 486 (Pneumonia) respectively. Interestingly, as can be noticed in B, there is a strong relationship between these two prognosis labels.

Genbase: It is a biological dataset that contains 662 proteins and the task is to predict a set of their functions among 27 class labels. Examples of class labels are PDOC00154 (isomerases), PDOC00064 (oxydoreductases) and PDOC00791 (protein secretion and chaperones) [89]. Figure 2.23 (A) shows label distributions, while (B) gives an indication of the concurrence among them. We can notice from A that PDOC00791 is the majority label.

Slashdot: It has 3782 articles labelled with 22 tags and 1079 attributes extracted from the Slashdot website [8]. Examples of these tags are Entertainment, Interviews, Book Reviews, Politics, News, and Technology [9]. From the left side of Figure 2.24 we observe the imbalance of labels. For example, Apache and Search appear in only three and eight articles, respectively. Moreover, these two labels have relationships with other frequent labels such as Technology. An implication of this is the possibility that modelling the correlation could improve the prediction of these hard labels. There are two labels with zero frequencies: Main and Meta. Meaning that there is no article tagged with these labels in the data.

[8] http://slashdot.org

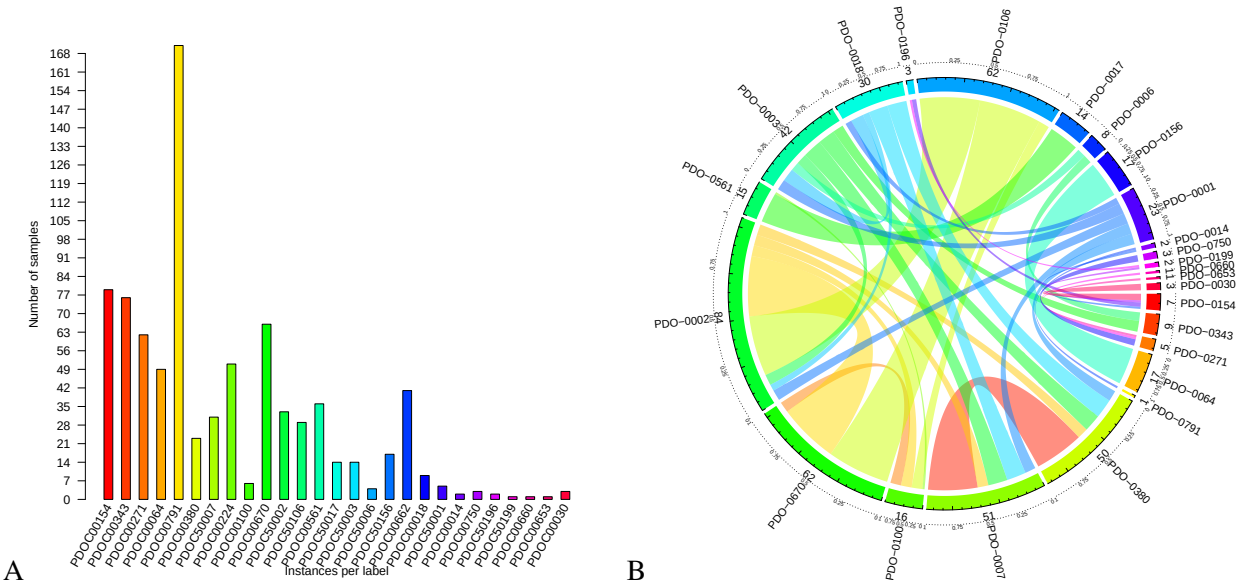


Figure 2.23: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Genbase dataset. We can notice from A that PDOC00791 is the majority label.

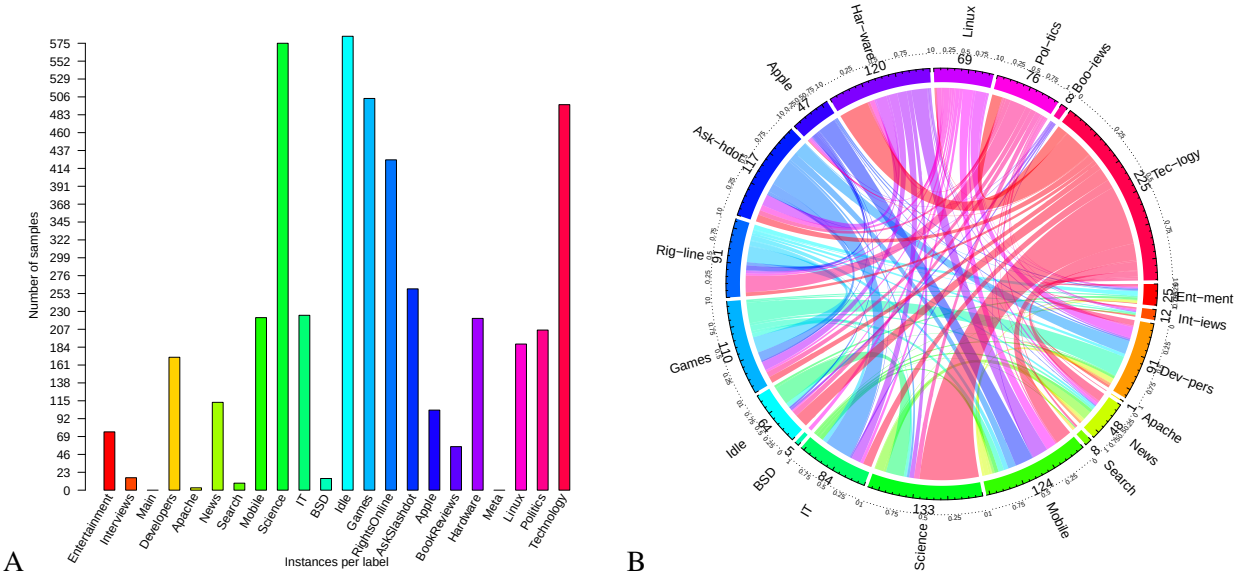


Figure 2.24: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Slashdot dataset. We can observe label imbalance as seen in A. For example, Apache and Search appear in only three and eight articles, respectively. Moreover, these two labels have a relationship with other frequent labels such as Technology. There are two labels with zero frequencies: Main and Meta.

Birds: It has information about birds collected from 645 audio clips. The task is to predict the set of bird species among 19 species [90]. Figure 2.25 depicts label's distributions and relationships.

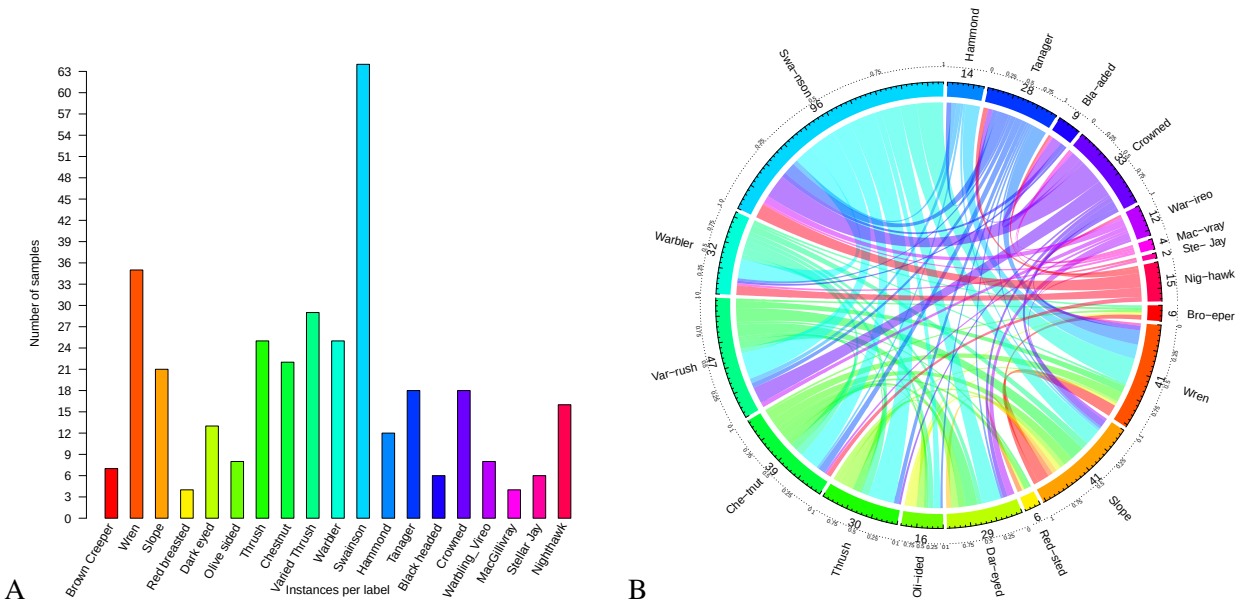


Figure 2.25: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Birds dataset. As can be seen in A, Swainson is the majority species.

Yeast: It represents gene expression for yeast. It has 103 extracted attributes from 2417 yeast genes. The task is to predict the set of gene categories among 14 available in the data [91]. Figure 2.26 (A) shows that the last category is the minority label. Moreover, the circular plot (B) shows that there are many interactions among labels.

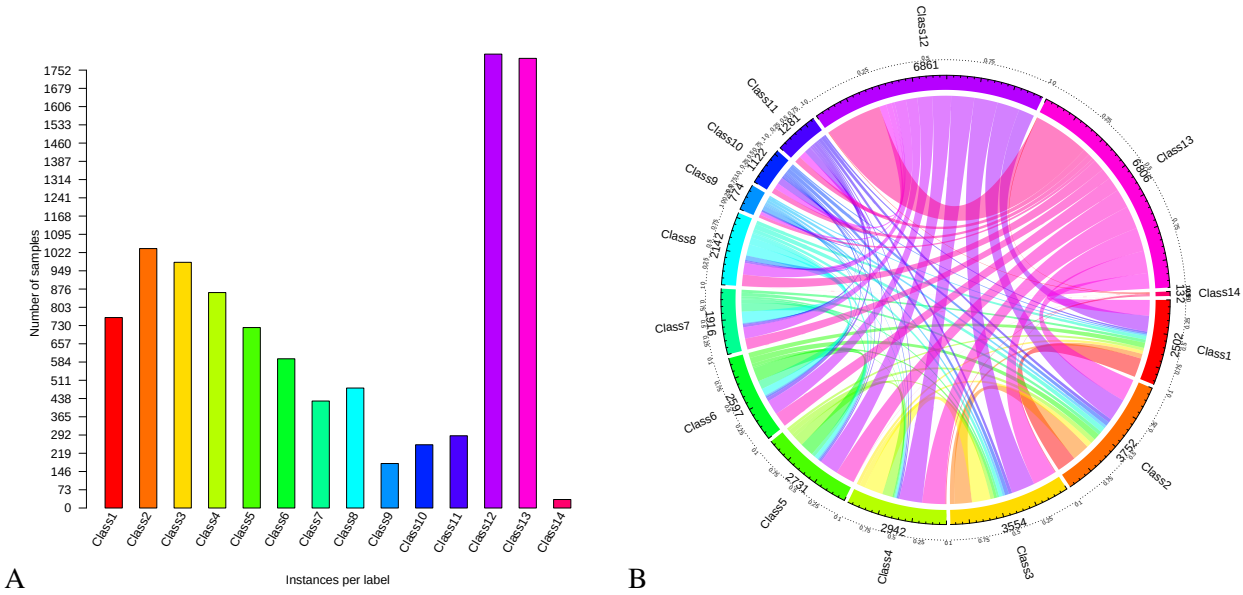
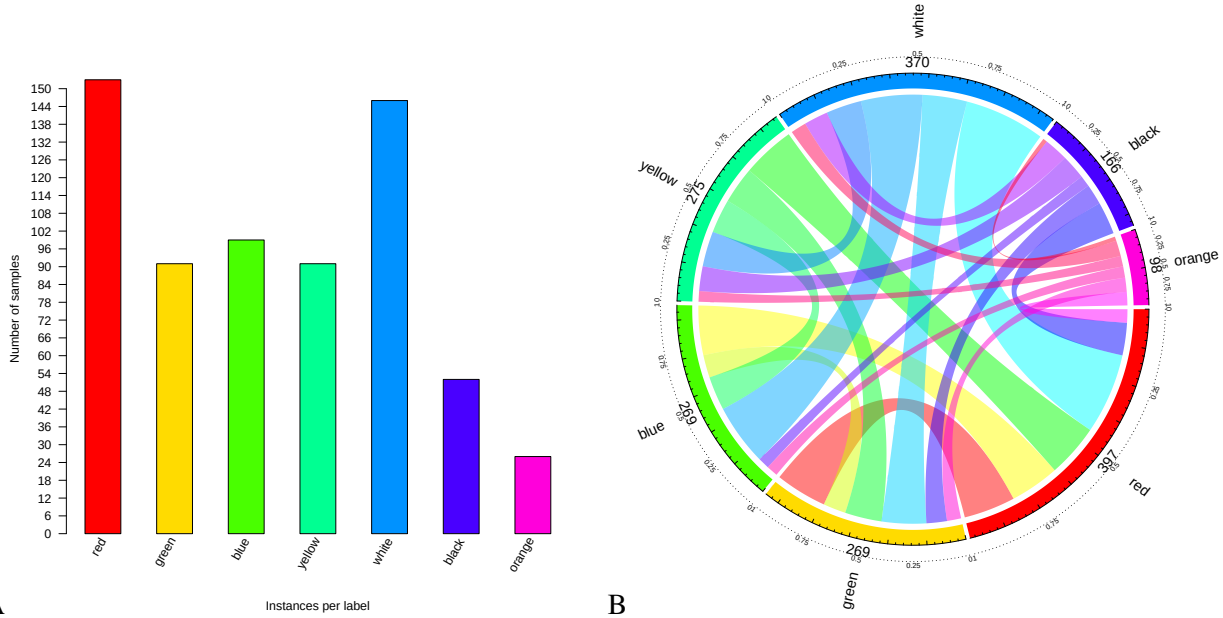


Figure 2.26: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Yeast dataset. A shows that the last category Class14 is the minority label and B shows that there are many interactions among labels.

Flags: It contains information about nation flags. It has only 194 countries and 19 attributes. Examples of attributes are Area, Population, Language, and Religion. The task is to predict a set of flag colours based on these attributes. There are seven labels, each label represents a colour [92]. Figure 2.27 (A) shows the distribution of the labels, while (B) gives an indication of the concurrence among them.



A B

Figure 2.27: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Flags dataset.

Emotions: It classifies music clips into emotions. It has 72 attributes extracted from 593 songs. The task is to classify the song into a set of six emotions (labels). Examples of emotion labels are Amazed-surprised, Happy-pleased and Relaxing-calm [93]. Figure 2.28 (A) shows the distribution of labels, while (B) gives an indication of the concurrence among them.

Scene: It consists of 2407 images and 294 attributes describing these images. It has six labels to describe the scene in each image, which can be Beach, Sunset, Foliage, Field, Mountain and Urban [94]. Figure 2.29 shows label distributions which indicate that Scene is a well-balanced dataset. However, the circular plot (B) shows the interactions among labels.

2.5 Cost-sensitive Classification

Ling and Sheng [95] defined cost-sensitive learning as "*a type of learning in data mining that takes the misclassification costs and possibly other types of cost into consideration*". In cost-sensitive learning, the key idea is to treat different misclassification costs differently to achieve higher classification accuracy. Costs are not necessarily monetary but can be for example a waste of time or an illness.

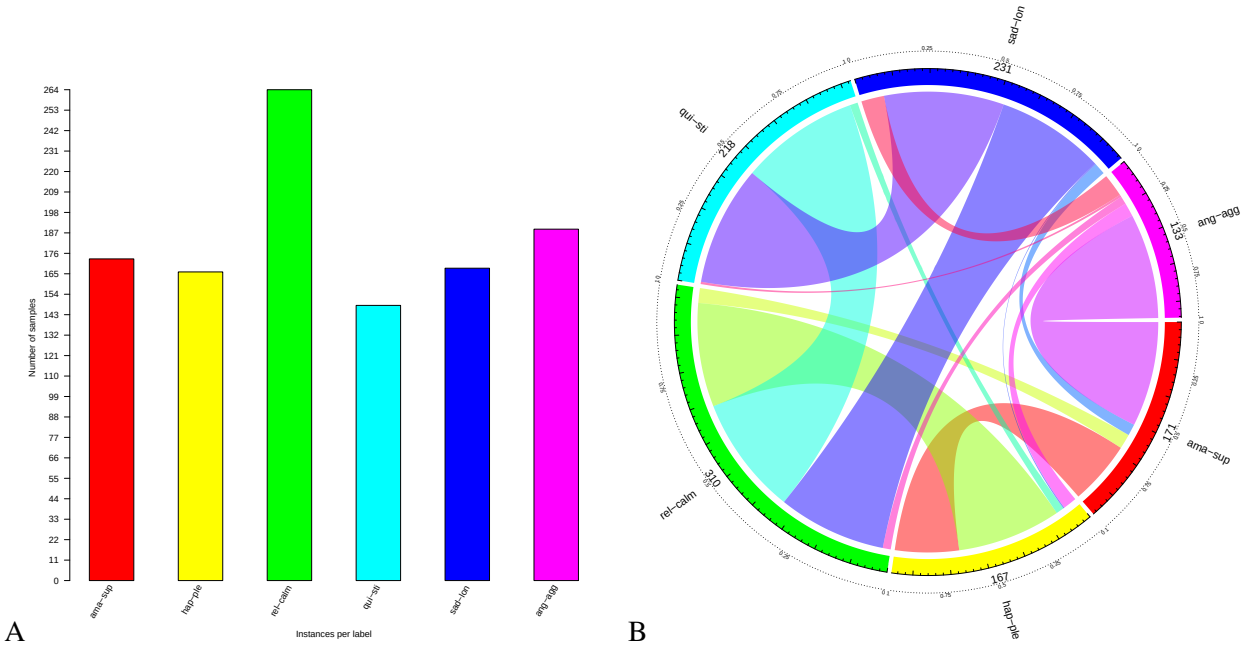


Figure 2.28: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Emotion dataset.

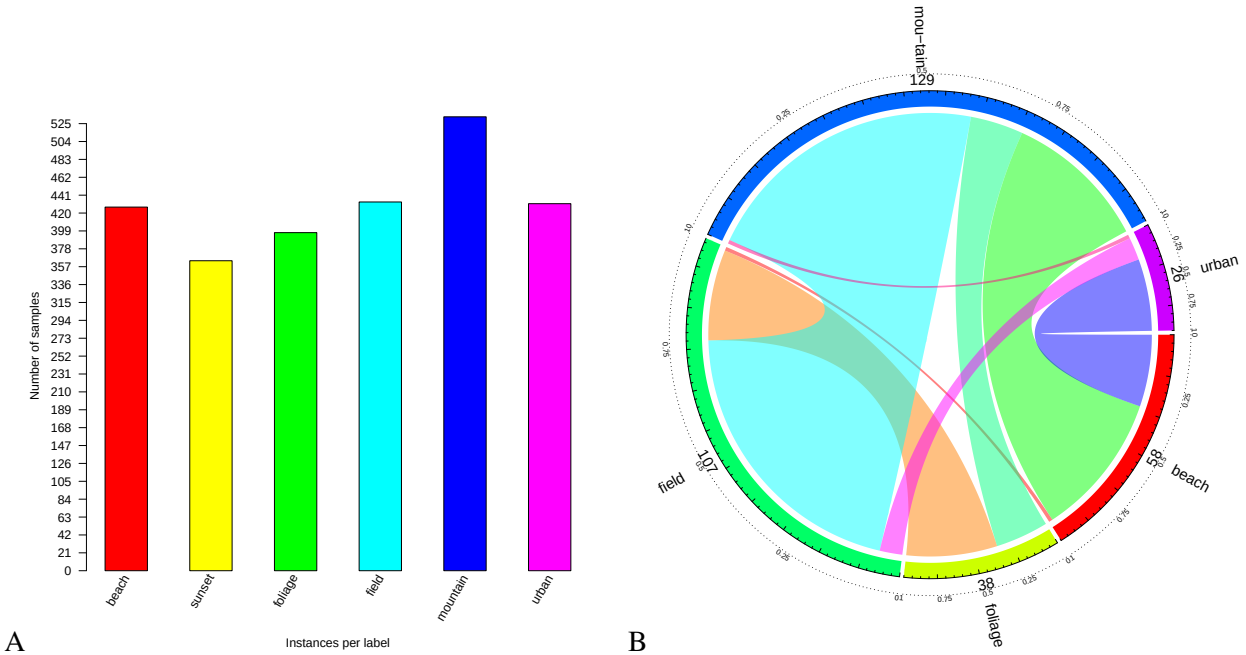


Figure 2.29: A is the histogram showing relations between instances and labels and B is the relationship among labels in the Scene dataset. Note that Sunset label is missing in this plot, as it does not have any correlations with other labels. In general, Scene dataset also shows low correlations between pairs of labels.

Many real-world applications such as medical decision making, target marketing, and object recognition are cost-sensitive classification problems [96, 97]. Cost-sensitive classification can be categorised into two different types: instance-dependent cost and label-dependent cost. In the former, the cost is associated with instances, whereas in the latter, each label has its own cost [98]. Cost-sensitive classification can be performed using the following types of methods [95, 99].

Direct method: It includes cost-sensitive learning algorithms that utilise misclassification costs directly during the learning process.

Meta-learning method: It transforms any cost-insensitive classifiers, such as C4.5, into cost-sensitive ones. It is based on pre-processing the training instances or post-processing the output of cost-insensitive learning algorithms. This type of method can be further categorised into thresholding and rescaling.

The thresholding technique is based on using different thresholds to classify instances into different classes. It can be used when classifiers are able to produce probability estimates. However, this method can be also applied to classifiers that do not produce reliable probability estimates by using techniques such as bagging to improve those estimates, as we will explain later in MetaCost or calibration.

The second method is rescaling (or rebalancing), which is one of the most popular techniques in cost-sensitive learning. It tries to change the proportion of negative and positive classes in such a way that their influences on the classification algorithm match their costs. Rescaling can be achieved by re-sampling or re-weighting training instances [96, 98].

Re-sampling modifies the class proportion of training instances in order to apply any cost-insensitive classifiers on the sampled data directly. This method can be applied to any classifier, meaning that it is not necessary to produce probability estimates. Sampling can be achieved either by oversampling instances from the high costly class or by undersampling instances from the less costly class.

Re-weighting assigns a normalised weight to each training instance proportional to misclassification costs; high weight means high cost and vice versa. That is, instances of the majority class are assigned low weights and the opposite is true for the minority class. This method works with any classifier which is able to handle different instances' weights, such as C4.5.

Boosting can be classified as a mixed approach between re-sampling and re-weighting. It generates a set of different classifiers in sequential iterations and on each round each of the training data should be re-weighted and then constructs a composite classifier by voting these classifiers. The re-weighting increases the weights of misclassified instances and decreases the weights of correctly classified instances, which causes the learner to focus on those misclassified instances in the next round.

2.5.1 Binary Cost-sensitive Classification

Binary cost-sensitive classification considers the cost of mispredicting each class as the other. The cost of predicting an actual negative as positive is denoted by ct_{+-} and the cost of predicting an actual positive as negative is represented by ct_{-+}.

False positive (fp) and false negative (fn) are wrong decisions, while true positive (tp) and true negative (tn) represent the correct predictions; therefore, there is no misclassification cost [95, 96]. Table 2.6 shows the cost matrix where the columns indicate actual classes and the rows predicted classes.

Table 2.6: Cost matrix of binary classification.

	Actual negative	Actual positive
Predict negative	ct_{--}	ct_{-+}
Predict positive	ct_{+-}	ct_{++}

Given a cost matrix, an instance should be classified into the category that has the minimum expected cost as defined below [95].

$$\text{Loss}(\mathbf{x}, cl_g) = \sum_{d=1}^{w} P(cl_d|\mathbf{x}) ct_{gd} \tag{2.13}$$

where $\text{Loss}(\mathbf{x}, cl_g)$ is the expected loss of classifying an instance $\mathbf{x}$ into class cl_g, $P(cl_d|\mathbf{x})$ is the probability estimation of classifying $\mathbf{x}$ into class cl_d and w is the number of classes. g and d are indices in the cost matrix.

Binary cost-sensitive classification is well studied in the literature. Ling and Sheng [95] in their study concluded that if a classifier can produce a posterior probability estimation for an instance $\mathbf{x}$, cost-sensitive classification can be simply applied using thresholding strategies by calculating the classification threshold as follows.

$$thr = \frac{ct_{+-}}{ct_{+-} + ct_{-+}} \tag{2.14}$$

Adaptive Cost (AdaCost) is a binary cost-sensitive version of Adaptive Boosting (AdaBoost), which builds a sequence of classifiers. AdaCost introduced a cost adjustment function into the AdaBoost weighting function. This function will be able to increase the weights of misclassified training instances and decrease those of correctly classified instances based on misclassification costs. Hence, each weak classifier classifies more costly instances correctly, which yields an overall cost reduction and better ensemble [100].

Zadrozny et al. [101] proposed transparent and black box approaches that use weighting and sampling methods, respectively. In the former technique, they examined the importance of weights but as we mentioned before it is only applied in the classifier that is able to handle weights. The latter is called Costing, which uses rejection sampling to avoid duplication in training data after sampling. It changes the distribution of training data according to misclassification costs shown in a cost matrix ct_{gd}, such that each instance in the training data is used once with accepting probability $\frac{ct_g}{cont}$, where ct_g is the misclassification cost of class cl_g, and $cont$ is an arbitrary constant chosen with the condition $max(ct_g) \leq cont$. However, the sample produced by this method can be much smaller than the original training data; even in the case $cont = max(ct_g)$.

Synthetic Minority Over-sampling Technique (SMOTE) is an over sampling method used to deal with the issue of class imbalance. It synthesises new minority class instances between several minority instances that lie together in the data space, rather than simply duplicating them as in random over-sampling. Samples are produced as follows: firstly, it finds the K nearest neighbours of each minority instance; next, it selects one of the nearest neighbours randomly; then, it creates a new minority class instance joining the K minority class nearest neighbours [102].

2.5.2 Multi-class Cost-sensitive Classification

Multi-class cost-sensitive classification can be more difficult than the binary version, because of the number of classes (more than two). The cost matrix can be extended from binary to any number of classes [97]. Let w be the number of classes, then the cost of misclassifying an instance of the g^{th} class to the d^{th} class, where the element at the g^{th} row and the d^{th} column can be defined as ct_{gd}. One of the direct methods to handle multi-class cost sensitive classification is the reduction, which reduces multi-class to binary classification and then applies binary cost-sensitive classification.

Domingos [103] proposed the Meta-learning for Cost-sensitive classification (MetaCost) algorithm, which is essentially a thresholding method based on relabelling training instances with their optimal classes according to the cost matrix and then any classifier can be applied. Firstly, it uses bagging to obtain probability estimations of training instances, then, it relabels the classes of training instances according to Equation 2.13. Finally, it uses these labelled training instances with any cost-insensitive classifier.

Zhou and Liu [98] examined multi-class cost-sensitive classification in a different way. They proposed another weighting approach, which assigns different weights for training instances with different classes, such that the weights are proportional to misclassification costs. The weighted instances are then passed to any cost-insensitive learner such as C4.5 decision tree. Suppose each class cl_g can be assigned a weight wt_g, the pairwise weights for two classes cl_g and cl_d are described as follows:

$$\frac{wt_g}{wt_d} = \frac{ct_{gd}}{ct_{dg}} \quad g,d \in 1,...,w$$

where w is the number of classes in single-label classification.

Santos-Rodríguez et al. [104] explored a different approach, based on designing loss functions specifically tailored to cost matrices. They proposed a novel parametric family of Bregman divergences that are used to train cost-sensitive classifiers in multi-class problems.

2.5.3 Multi-label Cost-sensitive Classification

Currently, cost-sensitive learning research focuses on binary or multi-class classification but not on multi-label classification [105]. Multi-label cost-sensitive classification extends multi-label classification by adding a cost vector to each training instance, where a misclassification cost is assigned to each label.

Lo et al. [105] have won the MIREX 2009 audio tagging competition by considering the cost for each tag. Audio tags are keywords that are used to describe different kinds of music clips. Usually, these tags are assigned by users with different musical knowledge, which may contain a lot of noisy information. In this work, the total tag counts have been treated as a cost to predict the most frequent used tags. They extended two multi-label classification methods, namely, stacking and RAkEL, to solve multi-label cost-sensitive classification.

In conclusion, cost-sensitive classification is an important research area with many real world applications. Therefore, making better decisions not only depends on minimising the expected errors but also on minimising the total cost associated with those decisions. Current research has been focused on binary or multi-class cost-sensitive classification, but not on multi-label classification.

2.6 Clustering

Clustering is an unsupervised method that groups instances based on their input attributes, unlike the supervised algorithms such as classification, that assign predefined class labels to instances. The task is to partition training data into groups called "clusters" based on their similarity or distance measures.

Clustering algorithms operate either on the original input attributes such as sex, height, and weight or on relationships that describe instances. The relationship can be described by similarity that measures how close two instances are to each other or dissimilarity that measures the opposite, which is how far the two instances are from each other [106, 107].

There are several dissimilarity measures such as Euclidean distance and similarity measures such as Pearson's correlation, Jaccard and cosine similarity [108]. Similarity (correlation) and dissimilarity (distances) matrices are square n by n matrices, where n is the number of instances. The diagonal of the similarity matrix is usually 1, which represents similarities between instances and themselves, whereas it is zeros in the dissimilarity matrix. Since most of the popular clustering algorithms take a dissimilarity matrix as their input, we must first construct pairwise dissimilarities between the instances.

There is a large number of clustering algorithms available to use in the literature. The choice of the algorithm depends on the problem and the dataset. Xu and Wunsch [24], Jain [109], Berkhin [110] provided a brief summary of clustering methods and addressed the key issues in designing clustering algorithms. Next, we discuss several clustering algorithms that can be categorised as in the following subsections [110–113].

2.6.1 Partitioning Clustering

Algorithms that belong to this group tend to partition and break training data up into groups by minimising the distance between instances and cluster centers. K-means is the most widely used clustering algorithm that searches for K centroids (one for each cluster), where each centroid is the mean of all instances belonging to the cluster. It assigns each instance to the nearest centroid cluster [109, 114].

The algorithm can be summarised as follows. The first step is to choose K arbitrary instances as centroids. Then it assigns instances to the cluster that has the nearest centroid. After that, it recomputes the centroids, which become the mean of all instances in the cluster. The algorithm repeats until convergence. The Euclidean distance is chosen as the dissimilarity measure. K-means is a simple and effective algorithm but it has several weaknesses such as sensitivity to outliers and assignment of K (the number of clusters). K-means has been extended in different ways, including K-means++, Partitioning Around Medoids (PAM) or K-medoids and Fuzzy C-Means.

K-means++ is different from the standard K-means algorithm as it initialises the centroids carefully, unlike K-means that selects the initial K centroids randomly [115, 116]. In K-means++, the first centroid is chosen randomly as in K-means, while other centroids are chosen in such a way to maximise the distance between them. In general, the algorithm is similar to K-means except for the first step, which is assigning initial centroids.

In contrast to the K-means algorithm, k-medoids chooses actual instances in the dataset as centers (medoids). Consequently, it is known to be more robust to outliers than K-means. The main difference between K-means and K-medoids is that the latter assigns each instance to the nearest cluster medoids, which are roughly comparable to the median, rather than the mean [117].

The aforementioned clustering algorithms are crisp algorithms in which each instance should have a membership to only one cluster. On the other hand, Fuzzy clustering is more flexible such that an instance can have a membership to more than one cluster. Fuzzy C-Means is one of the most widely used Fuzzy clustering methods [118, 119].

2.6.2 Hierarchical Clustering

This technique decomposes training data into clusters by imposing a hierarchical structure. Unlike partitioning clustering that constructs a single-level partition, hierarchical clustering divides instances into groups at a high level such that each level can be further divided into sub-levels. A hierarchy of clusters can be achieved by two modes: agglomerative and divisive. The agglomerative approach starts by assigning each instance to a separate cluster. Then it iteratively merges the most similar pair of clusters based on a dissimilarity metric, until a stopping criterion is satisfied. In contrast, the divisive mode begins with one cluster including all instances and then divides that cluster into smaller and smaller clusters recursively [112, 120].

Single-linkage and complete-linkage are the most well-known agglomerative clustering methods [120]. Single-linkage merges two clusters whose two members (one instance from each of the two clusters) have the minimum distance. Whereas in complete-linkage the merging depends on the maximum distance among all pairs of instances, where the two instances being considered are in different clusters. Figures 2.30 and 2.31 explain the difference between both techniques.

The first step in both algorithms is to assume that each instance is in its own cluster. Then they merge two instances B and C as they have the lowest distance (0.95). The calculation of the distance between the new cluster {BC} and the remaining cluster {A} is based on the type of linkage. In single-linkage

the smallest distance between AB and AC (0.96) is used as shown in Figure 2.30, while the largest distance (0.98) is assigned in complete-linkage as shown in Figure 2.31. In other words, the distance in single-linkage and complete-linkage is the smallest and the largest of all pairwise comparisons (considering one instance from each cluster), respectively. The process is repeated until all instances are assigned to one cluster, there are no more instances to consider or other user-defined stopping criteria (like a distance threshold in Figures 2.30 and 2.31) are satisfied.

Single-linkage suffers from a problem known as a chaining (or merging). In more detail, single-linkage needs only one pair of instances to be close to each other in order to merge them in one cluster. It therefore can link a chain of instances for long distances. On the other hand, clusters resulting from complete-linkage are more compact than these found by single-linkage. Yet, it suffers from another problem called crowding. Meaning that an instance can be closer to instances in other clusters than its own cluster. Additionally, it is sensitive to outliers, which can cause undesirable clusters [120, 121].

2.6.3 Distribution-based Clustering

Distribution-based clustering partitions training data based on distribution models. Instances are grouped into clusters that are most likely sharing the same probability distribution. Additionally, there is no need to specify the number of clusters as it is determined automatically [122, 123]. DBCLASD is one of the typical algorithms for distribution clustering [124], which can be described as follows. At first, the algorithm begins with no cluster assigned to data instances. Then, it creates a new cluster for the first examined instance. Incrementally, it joins the cluster with its nearest instances if the distance between the cluster and its nearest instances has the expected distribution. The cluster distribution is generated from all instances belonging to it.

2.6.4 Density-based Clustering

Most clustering methods listed previously assume a mixture of K densities, one for each cluster. In contrast, density-based clustering partitions a set of instances by identifying dense regions, which result in clusters with any shape. In addition, it does not require a predefined K as the number of clusters and does not assume a centre for each cluster. However, it requires parameters related to the definition of the minimum density required to create a cluster. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is the most common type of density-based algorithms [24, 123, 125].

2.6.5 Spectral Clustering

Spectral clustering relies on the spectrum of similarity matrices derived from training data. The algorithm begins by reducing dimensions, which makes it competitive in high-dimensional attribute spaces. Then it connects instances in such a way that instances that belong to the same cluster should have high similarity values, while instances from different clusters should have low similarity values [126–128].

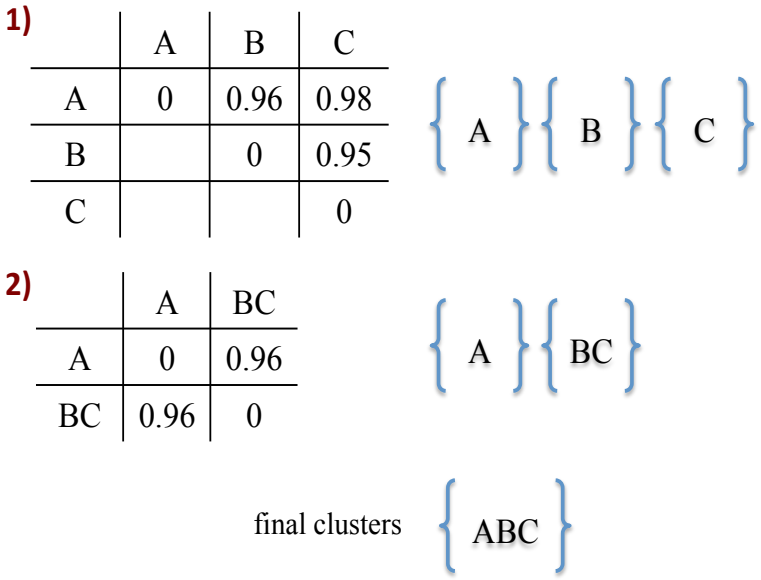


Figure 2.30: The result of single-linkage clustering assuming 0.97 as a distance threshold for stopping the agglomerative process. In the second iteration, the distance between two clusters {A} and {BC} is the smallest between AB and AC, which is 0.96.

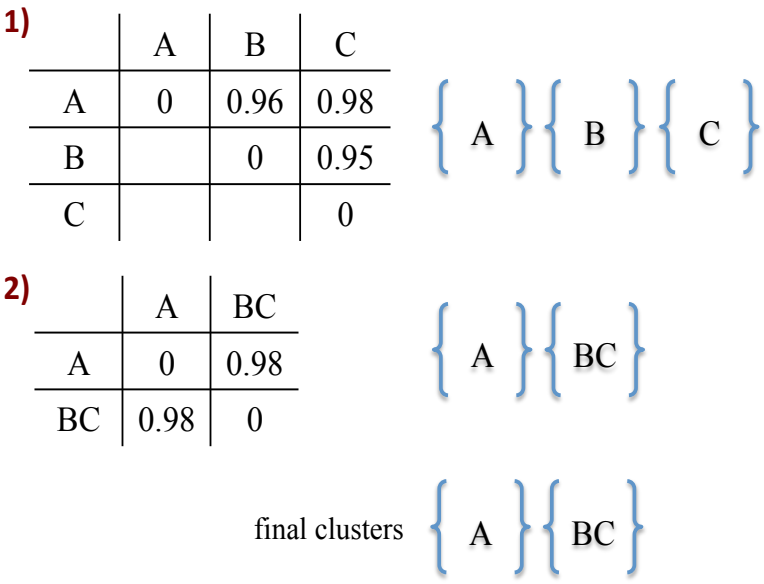


Figure 2.31: The result of complete-linkage clustering assuming 0.97 as a distance threshold. The distance between two clusters {A} and {BC} is the largest between AB and AC, which is 0.98. The algorithm stops without merging those clusters, as the distance between them is higher than the threshold.

2.7 Summary

In this chapter, we have covered most of the essential and relevant background to read and understand this thesis. We have begun by introducing our notations that are used throughout the thesis. We have reviewed related work in the area of single-label and multi-label classification and focused on modelling label correlations. We have briefly discussed multi-label evaluation measures. We have described the multi-label datasets that used in our experiments. Finally, we have given an overview of cost-sensitive classification and clustering algorithms.

The work done in order to achieve the objectives of this thesis will be discussed in the following chapters: 3, 4 and 5.

CHAPTER 3

CONTEXT-AWARE DECISION TREES: THE VERSATILE MODEL

This chapter discusses the dataset shift problem and introduces the Versatile Model (VM) for single-label classification. Section 3.2 presents dataset shift and existing approaches addressing it are discussed in Section 3.3. In Section 3.4 we introduce the versatile model, which adopts multiple strategies in order to handle context changes. Section 3.5 presents the experiments performed to evaluate the versatile model on both synthetic and non-synthetic dataset shifts. Finally, Section 3.6 concludes the chapter. This chapter is based on published work but it includes additional experiments and results [16]. The code is available in an online repository [15].

3.1 Introduction

Supervised machine learning is typically concerned with learning a model using training data and applying this model to new data. An implicit assumption made for successfully deploying a model is that both training and deployment data follow the same distribution. However, the distribution of the covariates (attributes) can change, especially when the training data is gathered in one context, but the model is deployed in a different context. For example, the training data is collected in one country but the predictions are required for another country. The presence of such *dataset shifts* can negatively affect the performance of a learned model.

Different kinds of dataset shift have been investigated in the literature [129, 130]. In this chapter, we focus on shifts in continuous attributes caused by hidden transformations from one context to another. For instance, a diagnostic test may have different resolutions when produced by different laboratories, or the average temperature may change from city to city or season to season. In such cases, the distribution of one or more of the covariates in X changes. This problem is referred as a covariate observation shift [130].

We address this problem by proposing the Versatile Model (VM) based on Decision Trees (DTs). We build Decision Trees (DTs) using percentiles for each attribute to deal with covariate observation shifts. In this proposal, if a certain percentage of training data reaches a child node after applying a decision test, the decision thresholds in deployment are redefined in order to preserve the same percentage (for example 60%) of deployment instances in that node. In the original learned DT, the learned threshold in a decision node corresponds to the 60^{th} percentile of the training data. The updated threshold in deployment will be the 60^{th} percentile of the deployment data.

The percentile approach assumes that the shift is caused by a monotonic function preserving the ordering of attribute values but ignoring the scale. For some shifts, it may be more appropriate to assume a transformation from one linear scale to another. We, therefore, develop a more general and versatile DT that can choose between different strategies (percentiles, linear shifts or no shift) to update the DT thresholds for each deployment context, according to the shifts observed in the data. A non-parametric statistical test called Kolmogorov-Smirnov test was adopted to identify shifts in the attributes by verifying differences in the empirical cumulative distribution function of the attributes among training and deployment data [131].

The VM targets single-label classification, where leaves can predict only a single label (binary or multi-class problems). Nevertheless, the VM can be extended to other learning settings, for example, multi-label problems.

3.2 Dataset Shift

In this section, we initially make a distinction between *training* and *deployment* contexts. The training context is when and where the model is built from a set of labelled instances, whereas the deployment context is when and where the learned model is actually used for predictions. These contexts are often different in some non-trivial way. For instance, a model may be built using training data collected in a certain period of time and/or in a particular country and deployed to deployment data in a future time and/or in a different country. A model built in a training context may fail in a deployment context due to different reasons: in this thesis, we focus on performance degradation caused by *dataset shifts* across contexts.

The obvious solution to deal with shifts would be to train a new model for each new deployment context. However, if there are not enough available labelled instances in the new context, training a new model specific to that context is then unfeasible as the model would not sufficiently generalise. More accurately, it can be feasible, but the quality of the model will not be good. Alternative solutions have to be applied to reuse or adapt existing models to several deployment contexts, which will depend on the kind of shift observed.

Dataset shift occurs when the distribution of inputs and/or output changes. Moreno-Torres et al. [129] defined dataset shift as follows.

Definition 3.1. Dataset shift happens when training and deployment joint distributions are different.

$$P(Y,X|C=tr) \neq P(Y,X|C=dep)$$

where X and Y are random variables corresponding to covariate (attributes) and target variables, respectively.

A shift can occur in the output when $P(Y,C=tr) \neq P(Y,C=dep)$, while the conditional probability distribution remains the same $P(X|Y,C=tr) = P(X|Y,C=dep)$. This is referred in the literature as the prior probability shift and can be addressed in different ways as for example in [132].

In our work, we are mainly concerned with situations where the marginal distribution of a covariate changes across contexts, more specifically, covariate shift. Covariate shift is a specific case of dataset shift and has received more attention in today's research. Covariate shift happens when the distribution of the input attributes (covariates) changes between training and deployment stages. Given a change in the marginal distribution, we can further distinguish two different kinds of shifts depending on whether the conditional distribution of the target also changes between training and deployment.

We consider two settings to define the changes that can happen to the data under covariate shift. Firstly, the situation when the covariate distributions change while the conditional distributions remain the same. In fact, there are two definitions for this type of shift as follows.

Definition 3.2. The covariate shift general definition that is implicitly assumed in many papers is the following.

$$P(X|C=tr) \neq P(X|C=dep)$$

Definition 3.3. The more specific definition of covariate shift was proposed by [129], which assumes that the conditional distribution of the target variable given the covariates does not change.

$$P(X|C=tr) \neq P(X|C=dep)$$
$$P(Y|X,C=tr) = P(Y|X,C=dep)$$

For instance, the smoking habits of a population may change over time due to public initiatives but the probability of lung cancer given smoking is expected to remain the same [133] as shown in Figure 3.1. In the same problem, a labelled training set may be collected biased to patients with bad smoking habits. Again, the marginal distribution in deployment may be different from training while the conditional probability is the same. The covariate shift is referred in the literature by different terms such as simple covariate shift [133] or sample selection bias [134].

Secondly, covariate observation shift occurs when both the marginal and the conditional distributions change, which was introduced in [130]. This is a more difficult situation that can be hard to solve and requires additional assumptions. A suitable assumption in many situations is that there is a hidden transformation of the covariates $\Phi(X)$, for which the conditional probability is unchanged across contexts as follows.

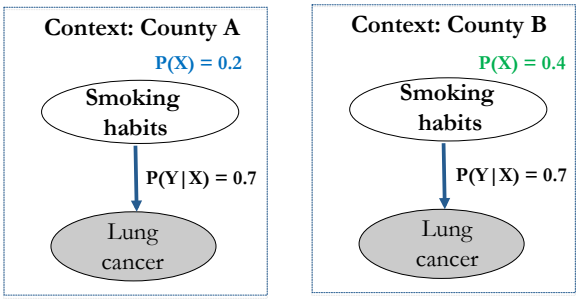


Figure 3.1: Simple covariate shift occurs when the marginal distribution X of the training and deployment data changes, while the conditional probability $P(Y|X)$ remains the same.

Definition 3.4. Let $\Phi(X)$ be a hidden variant of the covariate such that there is no dataset shift. Covariate observation shift happens when both the marginal distribution of the covariates and the conditional distribution of the target variable given the covariates change while there is no shift in the hidden true distribution across contexts.

$$P(X|C=tr) \neq P(X|C=dep)$$
$$P(Y|X,C=tr) \neq P(Y|X,C=dep)$$
$$P(Y|X,C=tr) = P(Y|\Phi(X),C=dep)$$

For instance, in prostate cancer detection, shifts can be observed in data from different laboratories due to differences in their equipments and resolution of diagnostic tests [135]. A mapping between attributes can be performed to correct the existing differences in data [135]. Another example, suppose that in an image recognition problem, pictures are taken by a camera with two different colour adjustment settings, thus representing two different contexts. This can result in a shift in the covariates. The conditional probability, however, may be the same given an invariant hidden raw camera representation [130]. Furthermore, a sensor used to detect an event may degrade over time or two different sensors use a different scale to observe skin temperatures. Such degradation/differences can be seen as a transformation function in the sensor outputs that causes a covariate observation shift.

Figure 3.2 gives an example of covariate observation shift. Suppose we train a model from sensors in building A and the task is to predict body activities based on skin temperature. Building A has sensors that are only able to provide skin temperature in degree Celsius $^{\circ}C$. At deployment, the available sensors can observe the skin temperature on a different scale, which is Fahrenheit $^{\circ}F$. However, we know that there is a hidden transformation function that can transform the skin temperature back to $^{\circ}C$, in order to use the trained model.

In summary, we emphasise that it can be difficult to recognise or distinguish between different kinds of shifts that may occur in a dataset. It can be simple in some cases to identify a shift in the covariates, depending on a sufficient number of unlabelled instances in the deployment context. On the other hand, verifying a shift in the conditional probabilities of the target variable given the covariates $P(Y|X)$ is

not possible if there are only unlabelled instances in deployment or may be unreliable if the number of labelled instances in deployment is low.

Additionally, suppose that we have evidence that a change is caused by a transformation in a covariate. Trying to detect a linear transformation to apply to the deployment data may be counter-productive if the true transformation is non-linear instead. Finally, applying a shift-aware method in a deployment context that did not actually change compared to the training context may be detrimental as well. These considerations motivated our proposal of a more sophisticated approach that can adapt to different kinds of dataset shifts under different assumptions. In this thesis, we focus on the second type of covariate shift in which both the marginal and the conditional distributions can change across contexts.

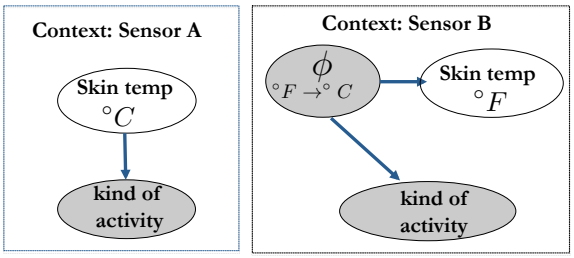


Figure 3.2: Covariate observation shift occurs when both the marginal distribution X and the conditional probability $P(Y|X)$ change while the hidden variant of the covariates do not change.

3.3 Related Work

Many solutions can be applied in case a shift is detected or known in the data. The obvious one is to train a new classifier for the deployment data. However, there are some attempts in the literature to avoid the re-training process and they can be grouped as follows:

A common solution to deal with simple covariate shifts is to modify the training data distribution by considering the deployment data distribution. The basic idea is to minimise the difference between the two distributions by reweighting the training data. A new model can then be learned using the shift-corrected training data distribution. This strategy is adopted by different authors using importance sampling which corrects the training distribution using instance weights proportional to $\frac{P(X,C=dep)}{P(X,C=tr)}$. Examples of such solutions include Integrated Optimisation Problem (IOP) [136], Kernel Mean Matching (KMM) [137] and Importance Weighted Cross Validation (IWCV) [138]. These approaches overcome the computational cost associated with estimating the distribution densities by estimating the importance weights.

IOP learns the weights that maximise the conditional probability $P(Y|X)$ using the labelled training data. Each training instance needs to be weighted with the importance ratio. The weights are learned in a discriminative way that shows how much likely an instance is occurred in the deployment data than in the training data. The optimisation problem leads to a kernel logistic regression and an exponential model classifier for covariate shift [136].

KMM approach reweights each training instance such that the difference between the mean of the training and deployment input attributes is minimised in Reproducing Kernel Hilbert Space (RKHS). KMM can be rewritten as a quadratic programming optimisation problem as follows [137].

$$\beta^* = \arg\min_{\beta} \| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \beta_i X_{tr} - \frac{1}{n_{dep}} \sum_{i=1}^{n_{dep}} X_{dep} \|^2$$

$$\text{subject to:} \beta_i \in [0, B] \text{ and } | \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \beta_i - 1 | \leqslant \varepsilon$$

where X_{tr}, X_{dep}, n_{tr} and n_{dep} represent the training attribute, the deployment attribute, the number of training instances and the number of deployment instances, respectively. β_i is the instance weight, B is a parameter and ε is a small quantity.

On the availability of some information about the deployment context (few labelled data), modifying the deployment data can be a solution. Previous authors have dealt with covariate observation shifts by finding a transformation function Φ to correct the deployment data [139]. Once transformed or 'unshifted' using Φ, the deployment data is given as input to the model learned in the training context. Finding a linear transformation is a natural choice in this approach. In [139], for instance, the authors adopted Stochastic Hill Climbing (SHC) [140] to find the best linear transformation to apply to the given deployment data. In that work, labelled deployment instances are required in order to evaluate the suitability of a candidate linear map. The parameters of the linear transformation are then optimised to maximise the accuracy obtained by the learned model on the labelled deployment instances (once transformed). A similar idea was proposed in [135], using genetic programming [141] techniques to find more complex transformation functions (both linear and non-linear). As stated in [139], it is a requirement that labelled instances are available in the deployment context to evaluate the adequacy of the transformation functions.

In this thesis, we propose learning a rich model named here the versatile model that handles both input and output data shifts. Such model concerns the deployment context and the changes that may happen in advance. Moreover, the proposed model learned on the training data is useful in both contexts: training and deployment. Furthermore, the proposed model does not require labelled data at deployment.

3.4 Versatile Decision Trees

In this work, we propose different strategies to build Decision Trees (DTs) in the presence of covariate observation shifts. We make two main contributions. Firstly, we propose a novel approach to build DTs based on percentiles (see Sections 3.4.1 and 3.4.2). The basic idea is to learn a conventional DT and then to replace the internal decision thresholds by percentiles, which can deal with monotonic shifts. Secondly, we propose a more general Versatile Model (VM) that deploys different strategies (unshifted data, linearly and non-linearly shifted data) to update the DT thresholds for each deployment context, according to the shifts observed in the data (see Section 3.4.3). The shifts are identified by applying a non-parametric statistical test called Kolmogorov-Smirnov Test (see Section 3.4.4).

3.4.1 Adapting for Input Shifts

We consider an example using the diabetes dataset from the UCI repository [142], which has 8 input attributes and 768 instances. Suppose we train a decision stump and the discriminative attribute is the Plasma glucose concentration attribute, which is a numerical attribute. Also, suppose the decision threshold is 127, meaning that any patient with plasma concentration above 127 will be classified as diabetic, otherwise classified as non-diabetic as seen in Figure 3.3 (left). If there is no shift in the attribute from training to deployment, the decision node can be directly applied, for example, the threshold 127 is maintained to split data in deployment. However, if the attribute is shifted in deployment, the original threshold may not work well.

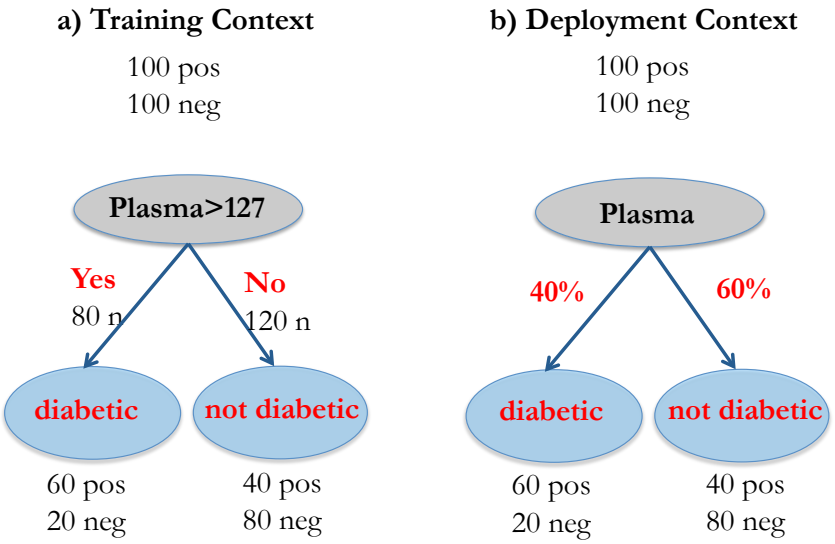


Figure 3.3: Two types of models; on the left is the model using a fixed threshold while on the right is the model using percentiles. For each deployment context, the decision tree is deployed in such a way that the deployment instances are split to the leaves in the same percentile amounts of 60% and 40%.

In the current work, we propose to adopt the percentiles[1] of continuous attributes to update the decision thresholds for each deployment context. Back to the example, instead of interpreting the data split in an absolute sense, we will interpret it in terms of ranks: 40% of the training instances with the *highest* values of Plasma reach the left child, while 60% of the training instances with the *lowest* values of Plasma, in turn, reach the right child (see the right side of Figure 3.3). We can say that the data was split at the 60^{th} percentile in training.

Given a batch set of instances in deployment to classify, the DT can apply the same split rule: the 40% of the instances in deployment with the highest values of Plasma are associated with the left child, while 60% of the instances with the lowest values of Plasma in deployment are associated to the right child. The decision threshold in deployment is updated in such a way that the percentage of instances assigned to each child is maintained.

[1]Percentile is the value below which a given percentage of observations in a group is observed.

In our approach, it is assumed that the shift is caused by a *monotonic* transformation Φ. Such functions, when applied to an input attribute, preserve the order of its original values. Different from the previous work [139] the transformation function in the versatile DT is not explicitly estimated, but it is implicitly treated by deploying the percentiles.

Formally, let $\mathscr{Y} = \{cl_1, \ldots, cl_w\}$ be the set of class labels in a single-label problem (for example $w = 2$ in binary classification). Let thr_{tr} be the threshold value applied on a numerical attribute X in a decision node. In the previous example $thr_{tr} = 127$ for the Plasma attribute. Let n^{cl}_{right} be the number of training instances belonging to class cl that are associated to the right child node after applying the decision test, for example, for which $X \leq thr_{tr}$. The total number of instances n_{right} associated to this node is:

$$n_{right} = \sum_{cl \in \mathscr{Y}} n^{cl}_{right} \tag{3.1}$$

Let $R_{tr}(thr_{tr}) = 100 \cdot \frac{n_{right}}{n_{tr}}$ be the percentage of training instances in the right node, where n_{tr} is the total number of training instances. Then, thr_{tr} is the percentile associated to $R_{tr}(thr_{tr})$ for the attribute X. In the above example: $R_{thr}(127) = 60\%$ and thr_{tr} is the 60^{th} percentile of Plasma in the training data. Then the threshold adopted in deployment is defined as:

$$thr_{dep} = R^{-1}_{dep}(R_{tr}(thr_{tr})) \tag{3.2}$$

In the above equation, the threshold thr_{dep} is the attribute value in deployment that, once adopted to split the deployment data, maintains the percentage of instances in each child node: $R_{dep}(thr_{dep}) = R_{tr}(thr_{tr})$.

3.4.2 Adapting for Output Shifts

The percentile rule can be adapted to additionally deal with shifts in the class distribution across contexts. Figure 3.4 illustrates a situation where the prior probability of the positive class was 0.5 in training and then shifted to 0.6 in deployment. In Figure 3.4(a) we observe a certain number of positives and negatives internally in each child node, which is used to derive the percentiles. If a shift is expected in the target, the percentage of instances expected in deployment for each child node may change as well. For instance, a higher percentage of instances may be observed in the right node in deployment because the probability of positives has increased and the proportion of positives related to negatives in this node is high. In our work, we estimate the percentage of instances in each child node according to the class ratios between training and deployment.

Let P^{cl}_{tr} and P^{cl}_{dep} be the probability of class cl, respectively in training and deployment. P^{cl}_{dep} can be estimated using available labelled data in deployment or just provided as input. There is a prior shift related to this class label when $P^{cl}_{tr} \neq P^{cl}_{dep}$. For each instance belonging to cl observed in training we expect to observed $\frac{P^{cl}_{dep}}{P^{cl}_{tr}}$ instances of cl in deployment. The number of instances associated to the right child node in deployment is then estimated by the following equation:

(3.3) $$\hat{n}_{right} = \sum_{cl \in \mathscr{Y}} n^{cl}_{right} \frac{P^{cl}_{dep}}{P^{cl}_{tr}}$$

The percentile is then computed using the corresponding percentage: $R_{tr}(thr_{tr}) = 100 * \frac{\hat{n}_{right}}{n_{tr}}$. In Figure 3.4(b), the class ratios of $\frac{0.6}{0.5}$ and $\frac{0.4}{0.5}$ are respectively adopted to correct the number of positive and negative instances in each node. In the right node, for instance, the expected number of positive and negative instances is, respectively, 48 and 64, resulting in 112 instances. The percentage to be adopted in deployment is now 56%, instead of 60% if no correction is performed. The 56^{th} percentile in the deployment data is then adopted as the decision threshold.

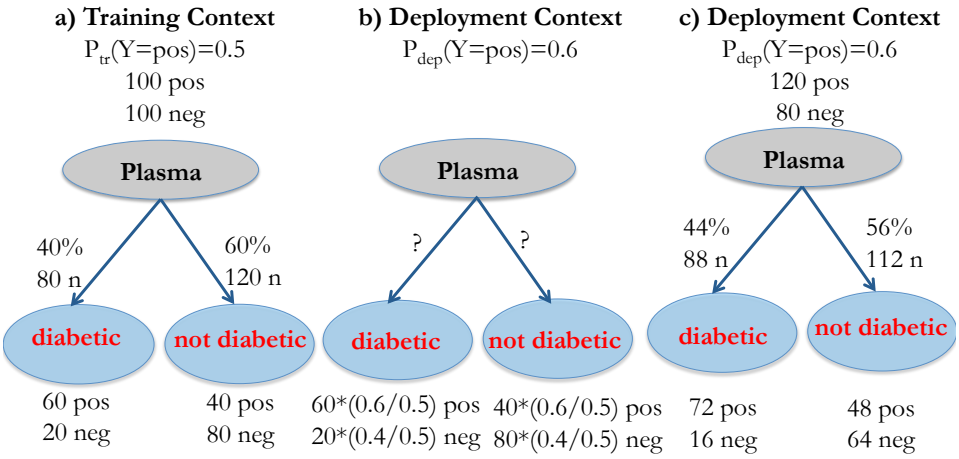


Figure 3.4: Example of DT with percentiles when a shift is identified in the class distribution. Part (a) illustrates the percentiles of each leaf for the training context, with the prior probability equal of each class to 0.5. Part (b) illustrates the correction of the percentiles for a new deployment context where the prior probability of the positive class is 0.6. The correction of performed using the ratios of $\frac{0.6}{0.5}$ and $\frac{0.4}{0.5}$ respectively for the positive and negative instances. The corrected number of instances expected at each leaf resulted in new estimated percentiles (c).

3.4.3 The Versatile Model

In this section, we propose the versatile model that employs different strategies to choose the decision threshold according to the shift observed in the deployment context. Figure 3.5 presents the proposed VM, which receives as input the original threshold applied on an attribute, the training, and the deployment data of that attribute and returns the appropriate threshold to adopt in deployment. This VM can be described in three steps as below (see Algorithm 2).

1. Initially a statistical test is applied to verify whether the distribution of the attribute differs between training and deployment. In this step, we aim to avoid dealing with shifts when they do not really exist, which could lead to overfitting. In our implementation, the non-parametric Kolmogorov-Smirnov (KS) test was adopted (see Section 3.4.4). We employed KS test on the values of the

attribute: training X_{tr} and deployment X_{dep}. The KS test tests the null hypothesis that the empirical Cumulative Distribution Functions (CDF) of X_{tr} and X_{dep} are identical against the alternative hypothesis that the two distributions are different. If there is no shift in the attribute, the versatile DT is applied using the original thresholds learned in the training context, such as $thr_{dep} = thr_{tr}$.

2. If a shift is detected by the previous test, a linear transformation is fitted and applied to the attribute in deployment, aiming to correct a potential linear shift. In our implementation, α and β parameters were estimated based on the change in the mean and standard deviation of the attribute in training and deployment (see Algorithm 3). We then apply KS test again to compare the distribution of the transformed attribute in deployment and the attribute in the training data. If no shift is observed now, we assume that the linear transformation applied was adequate. The versatile DT is then deployed with a threshold $thr_{dep} = \alpha \cdot thr_{tr} + \beta$.

3. Finally, if the second test indicates that there is still a shift in the attribute (for example, the shift was not corrected using the linear transformation), then the percentiles are deployed, assuming a non-linear monotonic shift. In this case the adopted threshold is: $thr_{dep} = R_{dep}^{-1}(R_{tr}(thr_{tr}))$.

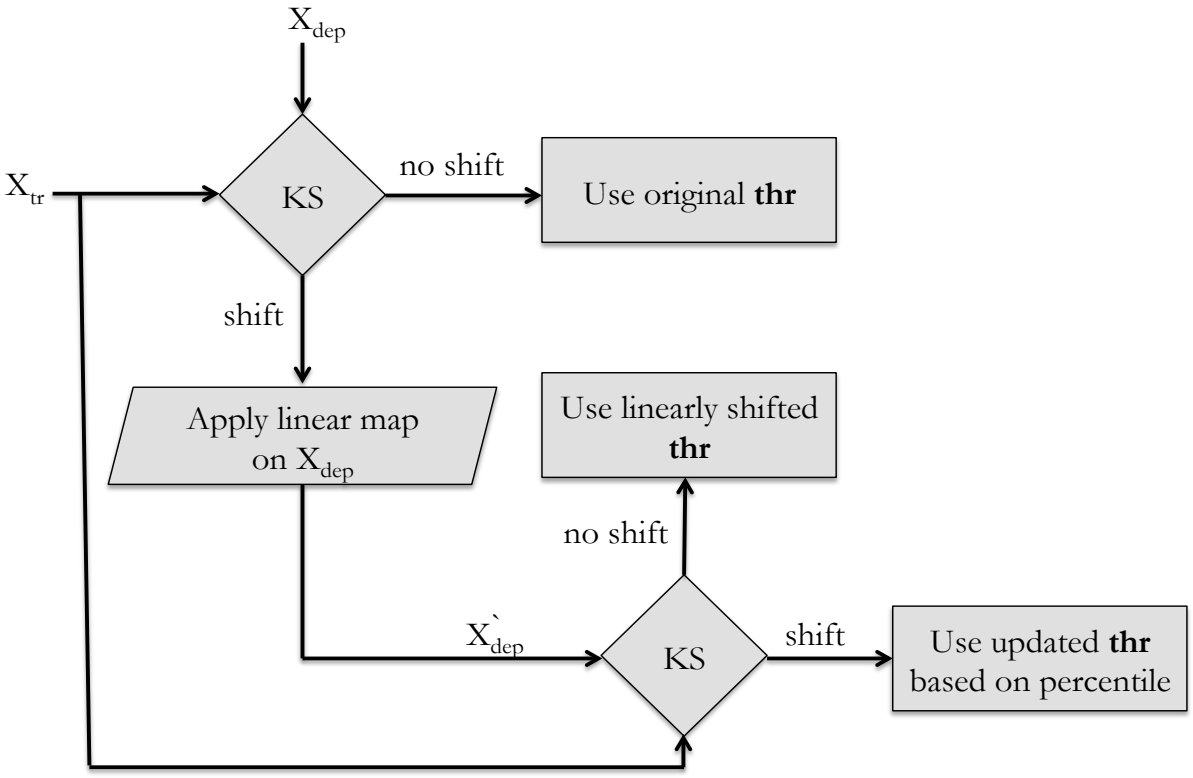


Figure 3.5: VM architecture. Three methods can be alternatively adopted for defining thresholds of numerical attributes in the decision nodes. The selection of the appropriate method is based on deploying KS test to verify differences in the attribute distribution between the training and deployment contexts.

Algorithm 2 Versatile model threshold selection algorithm

Input: training attribute vector $X_{tr} = (x_1, \ldots, x_{n_{tr}})$ with n_{tr} the number of training instances;
deployment attribute vector $X_{dep} = (x'_1, \ldots, x'_{n_{dep}})$ with n_{dep} the number of deployment instances;
decision threshold in training thr_{tr} for attribute X_{tr}.
Output: deployment decision threshold thr_{dep}.

/* Test for no shift. Null hypothesis $H_0 : CDF(X_{tr}) = CDF(X_{dep})$ */
p_{value}=**Kolmogorov-Smirnov**(X_{tr},X_{dep})

if $p_{value} < 0.05$ **then**

  /* Reject H_0, X_{dep} is shifted. Try a linear transformation */
  $(X^u_{dep}, \alpha, \beta)$= **Linear Transformation**(X_{tr},X_{dep})

  /* Test corrected shift. Null hypothesis $H_0 : CDF(X_{tr}) = CDF(X^u_{dep})$ * /
  p_{value}=**Kolmogorov-Smirnov**(X_{tr},X^u_{dep})

  if $p_{value} < 0.05$ **then**
    /* Reject H_0, X^u_{dep} is still shifted. Use the percentile. */
    $thr_{dep} = R^{-1}_{dep}(R_{tr}(thr_{tr}))$
  else
    /* Accept H_0, X^u_{dep} is not shifted. Use the linearly corrected threshold */
    $thr_{dep} = \alpha \cdot thr_{tr} + \beta$
  end if

else
  /* Accept H_0, X_{dep} is not shifted. Use training threshold. */
  $thr_{dep} = thr_{tr}$
end if

Return thr_{dep}

By adopting percentiles in the DT, we are assuming a monotonic transformation Φ across contexts. In this sense, our work is more general compared to the previous work [139] that assumes a linear transformation. Monotonic shifts can not only cover the linear case but also a broad range of non-linear monotonic transformations (for example, piecewise linear transformations). Even the case where there is no shift can be seen as a monotonic transformation when Φ is the identity function.

Despite this generality, the use of percentiles has limitations too. Firstly, percentile estimates (either in training or deployment) can be inaccurate when there is few or sparse data for estimation. Also, it may be worth trying alternative methods if the assumptions made by these methods about the context shifts are actually met. For instance, if we expect the shift to be linear we might be better off fitting an explicit linear transformation between training and deployment.

Algorithm 3 Linear Transformation

Input: training attribute vector $X_{tr} = (x_1, \ldots, x_{n_{tr}})$;
deployment attribute vector $X_{dep} = (x'_1, \ldots, x'_{n_{dep}})$.
Output: 'Unshifted' deployment attribute vector X^u_{dep} and corresponding parameters α, β.

/* Estimate the mean and standard deviation of X in training and deployment */
$\mu_{tr} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} x_i$, $\sigma_{tr} = \sqrt{\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (x_i - \mu_{tr})^2}$
$\mu_{dep} = \frac{1}{n_{dep}} \sum_{i=1}^{n_{dep}} x'_i$, $\sigma_{dep} = \sqrt{\frac{1}{n_{dep}} \sum_{i=1}^{n_{dep}} (x'_i - \mu_{dep})^2}$

/* Estimate α and β considering that: $\sigma_{dep} = \alpha\sigma_{tr}$ and $\mu_{dep} = \alpha\mu_{tr} + \beta$ */
$\alpha = \frac{\sigma dep}{\sigma_{tr}}$
$\beta = \mu_{dep} - \alpha \cdot \mu_{tr}$

/* Produce unshifted deployment data X^u_{dep} according to α and β */
$X^u_{dep} = \emptyset$
for $i = 1$ **to** n_{dep} **do**
  $x^u_i = \frac{(x'_i - \beta)}{\alpha}$
  $X^u_{dep} = X^u_{dep} \cup x^u_i$
end for

Return X^u_{dep}, α, β

3.4.4 The Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test is a non-parametric test that can be used to compare two distributions [131]. Suppose we have two distributions of covariates: training X_{tr} and deployment X_{dep}. The null hypothesis is that these two distributions are independent and identically distributed (IID). We would like to know whether to reject or accept the null hypothesis with a desired significant level. The KS test works as follows. The first step is to sort both sets of values separately. Then, it computes the maximum absolute difference between the two distributions as the following:

$$MD = \max_x |CDF_{tr}(x) - CDF_{dep}(x)|$$

where MD is the Kolmogorov-Smirnov test statistic and CDF is the empirical cumulative distribution function. x is an index of the values in the training and deployment instances.

Now, the p-value is computed from all permutations as the following:

$$per = \sqrt{\frac{n_{tr} \cdot n_{dep}}{n_{tr} + n_{dep}}}$$
$$\text{p-value} = e^{-2(MD \cdot per)^2}$$

where n_{tr} and n_{dep} are the size of training and deployment data, respectively.

The p-value is compared against a desired significant level (for example 0.05). We can accept the null hypothesis and reject the alternative hypothesis if the following is satisfied: p-value ≥ 0.05

3.5 Experimental Evaluation

The VM combines three strategies for defining the DT thresholds in deployment: original thresholds, linear transformations and monotonic transformations using percentiles. In the experiments, each strategy was individually compared to the VM, respectively named as Original Model (OM), (α,β) and Perc. Additionally, (α,β) and the Percentile methods were combined with the KS test, referred to in the experiments as KS+(α,β) and KS+Perc, respectively. In the former, linear transformation is applied to all shifted attributes, whereas, in the latter, Percentiles are utilised for all attributes. In both approaches, for each attribute, the original model was applied if there is no shift detected by KS test.

The first set of experiments applies synthetic shifts to benchmark datasets to analyse the performance of the shift detection approach adopted by the VM. We inject two types of shifts into the deployment data to test the VM: a non-linear monotonic transformation and linear shifts with different degrees of shifting (see Sections 3.5.1 and 3.5.2). In Section 3.5.3 we report on experiments with actual context changes occurring in real-world datasets.

3.5.1 Generating Synthetic Shifts

We simulate four contexts, namely, unshifted, linear shift, non-linear shift, and mixture shift. Linear shift was applied using $X_{dep} = \alpha \cdot X_{tr} + \beta$. In this experiment, parameters α and β are chosen to vary the mean and standard deviation in deployment in a controlled way. Let μ_{tr} and σ_{tr} be the mean and standard deviation of attribute X_{tr} in training. When X_{tr} is shifted using the parameters α and β, the mean and standard deviation of the transformed attribute become:

$$\mu_{dep} = \alpha \cdot \mu_{tr} + \beta$$
$$\sigma_{dep} = \alpha \cdot \sigma_{tr}$$

Assume the mean is shifted by γ times the original standard deviation: $\mu_{dep} = \mu_{tr} + \gamma \cdot \sigma_{tr}$. μ_{dep} can be written:

$$\begin{aligned} \mu_{dep} &= \mu_{tr} + \gamma \cdot \sigma_{tr} \\ \alpha \cdot \mu_{tr} + \beta &= \mu_{tr} + \gamma \cdot \sigma_{tr} \\ \beta &= (1-\alpha) \cdot \mu_{tr} + \gamma \cdot \sigma_{tr} \end{aligned}$$

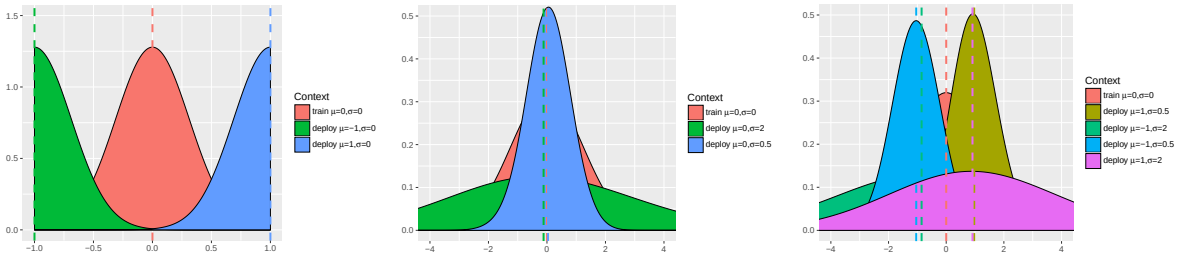
α and β were parametrised as follows:

$$\alpha = 2^{\varphi}$$
$$\beta = (1-2^{\varphi}) \cdot \mu_{tr} + \gamma \cdot \sigma_{tr}$$

If φ is negative the data is compressed across contexts, and if φ is positive the data is *stretched*. Table 3.1 shows possible values used in the experiments for φ and γ. Figure 3.6 visualises effects of the linear map on the distribution by changing the μ or σ or both.

Table 3.1: Values used in the experiments for φ and γ in order to generate the synthetic linear shifts.

φ	γ	Effect
0	0	unshifted data (original)
0	1	μ_{dep} shifted to right
0	-1	μ_{dep} shifted to left
1	0	stretch data
-1	0	compress data
1	1	μ_{dep} shifted to right and stretch the data
1	-1	μ_{dep} shifted to left and stretch the data
-1	1	μ_{dep} shifted to right and compress the data
-1	-1	μ_{dep} shifted to left and compress the data



(a) mean shift (b) variance shift (c) mean and variance shift

Figure 3.6: Different shift degrees used to inject linear shift.

Considering the non-linear shift, we use a cubic function which is a monotonic (order-preserving) transformation rather than a square transformation. Applying the cubic transformation only can cause outliers in one direction. Meaning that the original models might put all data in one bin (one side of the tree). In order to preserve the mean and standard deviation of the data we first convert the data into z-score, then, we apply the cubic function. Equation 3.4 shows the non-linear monotonic map.

$$X_{dep} = \sigma_{tr} \cdot \left(\frac{X_{tr} - \mu_{tr}}{\sigma_{tr}} \right)^3 + \mu_{tr} \tag{3.4}$$

The unshifted data means we do not change the 5^{th} fold, meaning that the training and deployment data follow the same distribution. However, the mixture shift combines different shifts: one-third of the attributes was shifted linearly, one-third non-linearly and one-third remained unchanged. Each of the attributes has been randomly assigned to a shifting group and the results are averaged over five different runs of 5-fold cross-validation [143].

3.5.2 Results of the Synthetic Shifts

In total, we have selected 15 datasets from UCI [142] and KEEL [144] with all numerical (real-value as well as integer-value) attributes. Ten datasets for binary classification, namely, Phoneme, Liver Disorders (Bupa), Appendicitis, Pima Indian Diabetes, Breast Cancer Wisconsin Original (breast-w), MAGIC Gamma Telescope, Threenorm, Ringnrom, Ionosphere, and Connectionist Bench (sonar).

Five datasets for multi-class classification: Balance scale, Landsat satellite, Image segmentation, Yeast, and winequality-red. Characteristics of these datasets, namely the number of instances $|D|$, number of input attributes *att* and the number of classes *w* are shown in Table 3.2.

Each dataset was randomly partitioned into five folds. Using four folds for training and the 5^{th} fold for deployment, after shifting according to each set of parameters in Table 3.1. The same shift is applied to all attributes in each dataset.

Table 3.2: Datasets characteristics used for the synthetic shift. *att* and *w* denote the number of attributes and the number of classes, respectively.

Dataset	$\|D\|$	*att*	*w*
Phoneme	5404	5	2
Bupa	345	6	2
Appendicitis	106	7	2
Pima	106	8	2
Breast-w	683	9	2
Magic	19020	10	2
Threenorm	300	20	2
Ringnrom	300	20	2
Ionosphere	351	33	2
Sonar	208	60	2
Balance scale	625	4	3
Landsat satellite	4435	36	7
Image segmentation	2310	19	7
Yeast	1484	8	10
Winequality-red	1599	11	11

Results are averaged over five cross-validation runs for each dataset. Tables 3.3 and 3.4 report the average accuracy of five different runs for all used methods in four cases: unshifted, linear shift, non-linear and mixture shift data. The performance of these methods applied to linear shifts is the average of eight degrees of linear shift as reported in Table 3.1. In Tables 3.3 and 3.4, the best result in each row is shown in boldface.

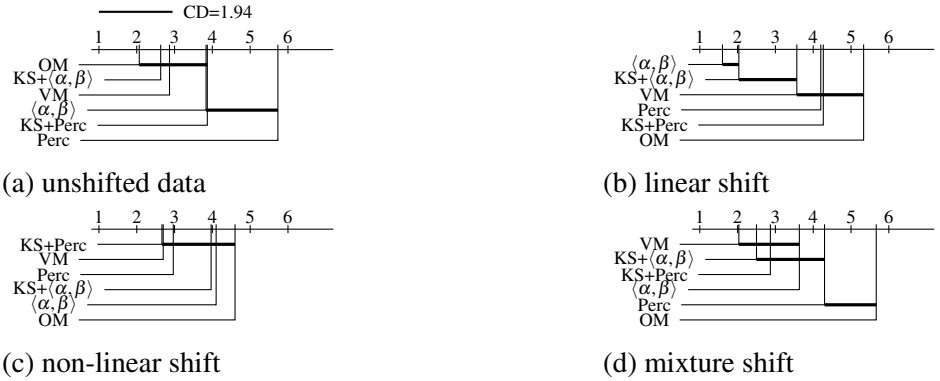
We conducted the Friedman test based on the average ranks for all datasets in order to verify whether the differences between all algorithms are statistically significant [145]. At significance level 0.05 the Friedman test gives significance for all experiments, so we show critical difference diagrams based on the Nemenyi post-hoc test for the former in Figure 3.7. We proceed to discuss the results of each experiment.

Unshifted data: Unsurprisingly, the original model performs best when there is no shift from the training context to the deployment context, but the Critical Difference (CD) diagram demonstrates that the VM is not significantly worse. Percentiles don't work well, in this case, confirming the need for a multi-strategy approach.

Linear shifts: Estimating a linear shift is the right thing to do here, so it is not surprising that $\langle \alpha, \beta \rangle$ performs strongest, with KS+$\langle \alpha, \beta \rangle$ trailing slightly behind as KS test may sometimes fail to detect the shift. The original model is significantly outperformed by all context-sensitive models except the percentiles.The VM is a good representative of the context-sensitive models and is only outperformed significantly by the model which always expects a linear shift.

Non-linear shift: Here the KS+Perc outperforms all other methods in terms of the average ranks followed by the VM. The Friedman test shows a significant difference, but the Nemenyi post-hoc test fails to report the difference in the pairwise comparisons. Regarding the original model, as shown in Table 3.4, we notice that, while it performs worst, there are three datasets, where the original model performs best: in these datasets many attribute values are in the range $[-1, 1]$ where the cubic transformation has less effect. In more details, the non-linear shift has no effect, which results in overlapping between two distributions: the training and the deployment. This explains the strong performance of the original model in this case.

Mixture shift: The aim of this experiment was to test how well the VM deals with a mixture of different shifts: one-third of the attributes was shifted linearly, one-third non-linearly, and one-third remained unchanged. The results demonstrate that the VM derives a clear advantage from the ability of being able to distinguish between these different kinds of shift and adapt its strategy. The Versatile Model has the best average rank among others and significantly wins against the percentiles, $\langle \alpha, \beta \rangle$, and the original models. Furthermore, it wins in 9 out of 15 datasets in the case of binary and multi-class classification, respectively.



(a) unshifted data

(b) linear shift

(c) non-linear shift

(d) mixture shift

Figure 3.7: Critical difference diagrams for both binary and multi-class classification using pairwise comparisons for those experiments where the Friedman test gives significance at 0.05. The Nemenyi post-hoc test shows no significant pairwise comparisons in the case of non-linear shift.

Table 3.3: Cross-validated binary and multi-class classification accuracy for both unshifted and linear shift. The numbers between brackets are ranks. VM is the Versatile Model, OM is the original model, $\langle \alpha, \beta \rangle$ corresponds to a linear shift, Perc corresponds to a percentile shift, and KS+$\cdots$ indicates that the Kolmogorov-Smirnov test is used for detecting the shift.

	VM	OM	$\langle \alpha, \beta \rangle$	KS+$\langle \alpha, \beta \rangle$	Perc	KS+Perc
unshifted data						
Phoneme	0.851(3)	**0.856(1)**	0.846(5)	0.854(2)	0.819(6)	0.850(4)
Bupa	0.631(3)	**0.632(1.5)**	0.619(5)	**0.632(1.5)**	0.578(6)	0.625(4)
Appendicitis	0.846(5.5)	0.849(4)	**0.853(1)**	0.846(5.5)	0.851(2.5)	0.851(2.5)
Pima	**0.728(1.5)**	0.725(4)	**0.728(1.5)**	0.727(3)	0.711(6)	0.721(5)
Breast-w	**0.947(3)**	**0.947(3)**	**0.947(3)**	**0.947(3)**	0.820(6)	**0.947(3)**
Magic	0.821(4)	**0.834(1)**	0.830(3)	0.833(2)	0.773(6)	0.814(5)
Threenorm	0.674(2.5)	**0.682(1)**	0.673(4)	0.674(2.5)	0.635(6)	0.671(5)
Ringnorm	0.735(2.5)	0.731(4.5)	**0.744(1)**	0.735(2.5)	0.678(6)	0.731(4.5)
Ionosphere	0.893(2.5)	**0.894(1)**	0.851(4)	0.893(2.5)	0.825(5.5)	0.825(5.5)
Sonar	0.752(2.5)	**0.754(1)**	0.739(5)	0.752(2.5)	0.716(6)	0.746(4)
Winequality-red	0.574(3)	**0.580(1)**	0.466(5)	0.576(2)	0.462(6)	0.573(4)
Yeast	0.542(2.5)	**0.548(1)**	0.353(5)	0.542(2.5)	0.330(6)	0.540(4)
Landsat satellite	0.852(3.5)	**0.854(1)**	0.669(5)	0.853(2)	0.662(6)	0.852(3.5)
Image segmentation	**0.961(2.5)**	**0.961(2.5)**	0.871(5)	**0.961(2.5)**	0.864(6)	**0.961(2.5)**
Balance scale	**0.773(1.5)**	0.772(3.5)	0.719(5)	0.772(3.5)	0.708(6)	**0.773(1.5)**
Average	0.772(2.86)	**0.774(2.06)**	0.727(3.83)	0.773(2.63)	0.695(5.73)	0.765(3.86)
linear shift						
Phoneme	0.825(3)	0.660(6)	**0.846(1.5)**	**0.846(1.5)**	0.819(4.5)	0.819(4.5)
Bupa	0.601(3)	0.558(6)	**0.619(1.5)**	**0.619(1.5)**	0.578(4.5)	0.578(4.5)
Appendicitis	0.844(4.5)	0.776(6)	**0.853(1)**	0.844(4.5)	0.851(2)	0.846(3)
Pima	0.726(3)	0.624(6)	**0.728(1.5)**	**0.728(1.5)**	0.711(4.5)	0.711(4.5)
Breast-w	0.820(4)	0.782(6)	**0.947(1.5)**	**0.947(1.5)**	0.820(4)	0.820(4)
Magic	0.761(5)	0.579(6)	**0.830(1.5)**	**0.830(1.5)**	0.773(3.5)	0.773(3.5)
Threenorm	0.672(2.5)	0.606(6)	**0.673(1)**	0.672(2.5)	0.635(4.5)	0.635(4.5)
Ringnorm	**0.744(1.5)**	0.608(6)	**0.744(1.5)**	0.743(3)	0.678(4.5)	0.678(4.5)
Ionosphere	0.810(5)	0.694(6)	**0.851(1.5)**	**0.851(1.5)**	0.825(3.5)	0.825(3.5)
Sonar	**0.739(2)**	0.624(6)	**0.739(2)**	**0.739(2)**	0.716(4.5)	0.716(4.5)
Winequality-red	0.465(3)	0.426(6)	**0.466(1.5)**	**0.466(1.5)**	0.462(4.5)	0.462(4.5)
Yeast	0.328(6)	**0.354(1)**	0.353(2.5)	0.353(2.5)	0.330(4.5)	0.330(4.5)
Landsat satellite	0.663(3)	0.538(6)	**0.669(1.5)**	**0.669(1.5)**	0.662(4.5)	0.662(4.5)
Image segmentation	0.869(3)	0.379(6)	**0.871(1.5)**	**0.871(1.5)**	0.864(4.5)	0.864(4.5)
Balance scale	0.708(5)	**0.731(1)**	0.719(2.5)	0.719(2.5)	0.708(5)	0.708(5)
Average	0.705(3.56)	0.595(5.33)	**0.727(1.6)**	0.726(2.03)	0.695(4.2)	0.695(4.26)

Table 3.4: Cross-validated binary and multi-class classification accuracy for both non-linear shift and mixture shift. The numbers between brackets are ranks. VM is the Versatile Model, OM is the original model, $\langle \alpha, \beta \rangle$ corresponds to a linear shift, Perc corresponds to a percentile shift, and KS+$\cdots$ indicates that the Kolmogorov-Smirnov test is used for detecting the shift.

	VM	OM	$\langle \alpha, \beta \rangle$	KS+$\langle \alpha, \beta \rangle$	Perc	KS+Perc
			non-linear shift			
Phoneme	**0.819(2)**	0.746(4)	0.720(5.5)	0.720(5.5)	**0.819(2)**	**0.819(2)**
Bupa	**0.594(1)**	0.506(6)	0.571(4.5)	0.571(4.5)	0.578(2.5)	0.578(2.5)
Appendicitis	0.847(4.5)	0.240(6)	0.849(3)	0.847(4.5)	**0.851(1.5)**	**0.851(1.5)**
Pima	0.715(3)	0.478(6)	**0.728(1.5)**	**0.728(1.5)**	0.711(4.5)	0.711(4.5)
Breast-w	0.820(4)	0.464(6)	**0.916(1.5)**	**0.916(1.5)**	0.820(4)	0.820(4)
Magic	0.761(3)	0.398(6)	0.744(4.5)	0.744(4.5)	**0.773(1.5)**	**0.773(1.5)**
Threenorm	0.651(2.5)	**0.671(1)**	0.607(6)	0.635(4.5)	0.635(4.5)	0.651(2.5)
Ringnorm	0.698(2.5)	**0.731(1)**	0.667(6)	0.680(4)	0.678(5)	0.698(2.5)
Ionosphere	**0.825(2)**	0.820(4)	0.781(5.5)	0.781(5.5)	**0.825(2)**	**0.825(2)**
Sonar	**0.744(2)**	0.478(6)	**0.744(2)**	**0.744(2)**	0.716(4.5)	0.716(4.5)
Winequality-red	**0.462(2)**	0.446(4)	0.444(5.5)	0.444(5.5)	**0.462(2)**	**0.462(2)**
Yeast	0.328(5)	0.117(6)	**0.354(1.5)**	**0.354(1.5)**	0.330(3.5)	0.330(3.5)
Landsat satellite	**0.662(2)**	0.217(6)	0.592(4.5)	0.592(4.5)	**0.662(2)**	**0.662(2)**
Image segmentation	**0.864(2)**	0.207(6)	0.542(4.5)	0.542(4.5)	**0.864(2)**	**0.864(2)**
Balance scale	0.708(3)	**0.790(1)**	0.631(5.5)	0.631(5.5)	0.708(3)	0.708(3)
Average	0.699(2.7)	0.487(4.6)	0.659(4.1)	0.661(3.96)	0.695(2.96)	**0.697(2.66)**
			mixture shift (unshifted, linear shift, non-linear)			
Phoneme	**0.828(1)**	0.749(6)	0.787(5)	0.789(4)	0.819(3)	0.823(2)
Bupa	**0.605(1)**	0.551(6)	0.595(2)	0.594(3)	0.578(5)	0.592(4)
Appendicitis	0.843(4.5)	0.718(6)	0.847(2.5)	0.843(4.5)	**0.851(1)**	0.847(2.5)
Pima	0.710(5)	0.512(6)	**0.727(1)**	0.724(2)	0.711(4)	0.712(3)
Breast-w	0.935(3.5)	0.797(6)	**0.947(1.5)**	**0.947(1.5)**	0.819(5)	0.935(3.5)
Magic	**0.805(1.5)**	0.510(6)	0.802(3)	**0.805(1.5)**	0.773(5)	0.785(4)
Threenorm	**0.672(1)**	0.647(4)	0.635(5.5)	0.653(2)	0.635(5.5)	0.649(3)
Ringnorm	**0.739(1)**	0.674(6)	0.728(2.5)	0.720(4)	0.678(5)	0.728(2.5)
Ionosphere	0.843(3.5)	0.792(6)	0.843(3.5)	**0.865(1)**	0.825(5)	0.848(2)
Sonar	**0.740(1)**	0.631(6)	0.737(3)	0.738(2)	0.716(4)	0.712(5)
Winequality-red	**0.504(1)**	0.459(5)	0.443(6)	0.472(3)	0.462(4)	0.498(2)
Yeast	0.426(2)	0.319(6)	0.354(4)	**0.460(1)**	0.330(5)	0.413(3)
Landsat satellite	**0.769(1)**	0.467(6)	0.646(5)	0.762(3)	0.662(4)	0.768(2)
Image segmentation	**0.906(1)**	0.461(6)	0.736(5)	0.767(4)	0.864(3)	0.905(2)
Balance scale	0.748(2.5)	0.727(4)	0.712(5)	**0.756(1)**	0.708(6)	0.748(2.5)
Average	**0.738(2.03)**	0.600(5.66)	0.702(3.63)	0.726(2.5)	0.695(4.3)	0.730(2.86)

3.5.3 Results on Non-Synthetic Shifts

The aim of this experiment is to evaluate the VM on real dataset shifts and compare it with the state-of-art covariate shift solvers. For comparisons, we included two sophisticated methods to cope with covariate dataset shift, which are IOP and KMM [146, 147]. IOP and KMM algorithms were retrieved from [148] and run using the default parameters. The available implementation supports only binary classification problems, therefore, for the last dataset, which is a multi-class problem, we have not reported the results for both IOP and KMM.

Diabetes: Our first benchmark is a dataset of 4 different ethnic groups of diabetes patients [2] [149]. The original dataset consists of 47 attributes and 101 766 instances. Each instance corresponds to a unique patient diagnosed with diabetes. The attributes describe the diabetic encounters such as diagnoses, medications, and number of visits in the year preceding the encounter. We rank the attributes using information gain ratio then we select the best 8 numerical attributes, 8 is an arbitrary number. The classification task is whether the patient was re-admitted to the hospital. The values of the readmission class attribute are: "yes" or "no". In the original dataset, these classes are: readmitted within 30 days " 30", readmitted after 30 days " 30" (these two classes are considered "yes" in this thesis) or no.

In our experiment, we split the dataset into 4 subsets according to the "ethnic group" the patient belongs to. There are 4 different groups: Caucasian, African American, Asian, and Hispanic. We evaluate the predictive performance of models trained on ethnic group X and deployed on ethnic group Y, denoted by X-Y. Table 3.5 shows the performance of the VM against the original model (OM), IOP and KMM. We also report the number of shifted attributes according to the KS test. The VM wins most often (6 wins), followed by the original model (5 wins).

Table 3.5: Classification accuracy for Diabetes dataset. Symbols denote ethnic groups as follows: African-American (AA), Asian (A), Caucasian (C), Hispanic (H). X-Y denotes trained on X, deployed on Y.

	A-AA	A-C	A-H	AA-A	AA-C	AA-H	C-A	C-AA	C-H	H-A	H-AA	H-C
# shifted	6	5	4	6	5	4	5	5	4	4	4	4
VM	0.569	0.529	**0.576**	**0.653**	**0.530**	**0.590**	0.645	0.546	0.588	0.624	**0.565**	**0.564**
OM	**0.574**	**0.538**	0.554	0.642	0.526	0.587	0.641	**0.566**	**0.595**	**0.642**	0.562	0.563
IOP	0.526	0.499	0.547	0.500	0.494	0.463	0.520	0.488	0.469	0.519	0.509	0.452
KMM	0.467	0.499	0.419	0.352	**0.530**	0.474	**0.647**	0.557	0.580	0.400	0.442	0.507

Heart: Our next benchmark is the heart disease dataset [142]. We split it into two subsets according to gender: male and female. In this dataset there are 5 continuous attributes, 3 of them are indicated as shifted between genders according to the KS test, which are age, heart rate, and serum cholesterol. Table 3.6 shows the performance of the versatile method against the original model, IOP, and KMM. In both contexts, the VM has the best accuracy among all four methods including the original model.

[2] https://www.hindawi.com/journals/bmri/2014/781670/sup/

Table 3.6: Classification accuracy for Heart dataset, with contexts by gender (F: Female, M: Male).

	M-F	F-M
# shifted	3	3
VM	**0.735**	**0.568**
OM	0.712	0.557
IOP	0.703	0.500
KMM	0.724	0.540

Bike Sharing: This dataset contains the hourly and daily count of rental bikes between years 2011 and 2012 in addition to weather information [142]. It contains 4 continuous attributes: actual and apparent temperature in Celsius, humidity and wind speed. The classification task is whether there is a demand in this period of time or not. In order to evaluate the shift effects, we split the dataset as proposed in [139] to obtain the 4 seasons datasets. According to KS, all these 4 attributes are detected as shifted except in 3 cases. Firstly, between Summer-Spring, wind speed is not shifted. Secondly, in both Summer-Autumn and Autumn-Winter, humidity is not shifted. The performance of VM and others are shown in Table 3.7. Again we note the solid performance of the VM, which was the best method in 6 out of the 12 contexts.

Table 3.7: Classification accuracy for Bike Sharing dataset, with contexts by season (Sp: Spring, S: Summer, A: Autumn, W: Winter).

	Sp-S	Sp-A	Sp-W	S-Sp	S-A	S-W	A-Sp	A-S	A-W	W-Sp	W-S	W-A
# shifted	3	4	4	3	3	4	4	3	3	4	4	3
VM	**0.641**	**0.558**	**0.601**	0.519	**0.579**	0.601	**0.602**	**0.543**	0.556	0.646	0.565	0.526
OM	0.538	0.468	0.544	0.607	0.547	0.612	0.574	0.521	0.528	0.718	**0.657**	**0.558**
IOP	0.489	0.468	0.533	**0.635**	0.510	**0.657**	0.585	0.534	0.522	0.658	0.630	0.510
KMM	0.559	0.468	0.522	**0.635**	0.521	0.651	0.585	0.521	**0.589**	**0.690**	0.521	0.521

AutoMPG: This dataset concerns the consumption in miles per gallon of vehicles from 3 different regions: USA, Europe and Japan [142]. It contains 4 numerical attributes: displacement, horsepower, weight, and acceleration. All these input attributes have been detected as shifted between regions using the KS test. This dataset has been binarised according to the mean value of the target. We split the dataset as proposed in [139] to obtain the 3 region datasets. The performance of VM and others are shown in Table 3.8. The VM outperforms all other methods and has only one loss against the original model.

Table 3.8: Classification accuracy for AutoMPG dataset, with contexts by origin (U: USA, E: Europe, J:Japan).

	U-E	U-J	E-U	E-J	J-U	J-E
# shifted	4	4	4	3	4	3
VM	**0.676**	**0.759**	**0.873**	**0.772**	**0.780**	0.647
OM	0.544	0.607	0.670	0.746	0.747	**0.691**
IOP	0.558	0.493	0.400	0.417	0.600	0.441
KMM	0.558	0.582	0.600	0.582	0.400	0.485

Steel Plates Faults: This dataset has seven types of steel plate faults and the task is to classify the type of fault. It is a multi-class dataset with 25 numerical attributes. We split the dataset into two subsets based on the type of steel, which is a categorical attribute with values A300 and A400. In our experiment, we concentrate on only three types of faults: Z-Scratch, Bumps, and Other-Faults due to highly imbalanced other classes. Table 3.9 shows the performance of the versatile method against the original model.

Table 3.9: Classification accuracy for Steel Plates Faults dataset, with contexts by type of steel (T1: A300, T2: A400).

	T1-T2	T2-T1
# shifted	25	25
VM	**0.678**	0.369
OM	0.403	**0.384**

Finally, we report the result of a Friedman test and post-hoc analysis using the Nemenyi test (both test at significant confidence level 0.05) on all non-synthetic shifts. Figure 3.8 demonstrates that the VM outperforms all others, significantly so except for the original model.

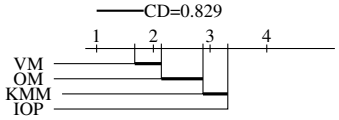


Figure 3.8: Critical difference diagram using pairwise comparisons using the Nemenyi post-hoc text for datasets with non synthetic shift. Average ranks as follows: VM=1.671, OM=2.140, KMM=2.875 and IOP= 3.312. The Friedman test gives significance at 0.05.

3.6 Concluding Remarks

We proposed a model for adapting to covariate observation shift using unlabelled deployment data. The proposed model is called the Versatile Model and is a Decision Tree model with enhanced splits. The main idea of the VM is that it captures more information about the context during the training process in order to be able to adapt this model for deployment contexts. The VM trains a classifier over the available data and then adapts some of its decisions according to the (usually unlabelled) deployment data. We used a non-parametric statistical significance test to choose among different strategies to update the decision thresholds in a DT. The VM does not make any strong assumptions such as linear transformation between contexts. Furthermore, it does not need any tuning parameters to adjust the model. Finally, empirical results on both synthetic shift and real dataset shift show noticeable performance gains achieved by the proposed method.

This work opens up many avenues for future work. One direction is to adapt the VM to other predictive problems, such as regression. One possibility is to assume that the deployment data is partially labelled and utilise this knowledge in the VM. We could learn the distribution parameters of the target using some labelled data such as the mean and the standard deviation. Another direction is to adapt the

VM to multi-label classification. It would be interesting to study the percentile in multi-label data where multiple labels are present. Percentile can be based on the marginal distribution (independent label), the joint distribution (dependent labels), or both.

CHAPTER 4

MULTI-LABEL DECISION TREES THAT CAPTURE LABEL CORRELATIONS LOCALLY

This chapter introduces **LaCova** for locally modelling label dependencies based on the covariance matrix. The proposed algorithm interleaves between two main decisions: learning separate classifiers for independent labels and modelling labels together for dependent labels. Finally, it shows the feasibility of extending **LaCova** by incorporating label clusters, which is called **LaCovaC**.

This chapter is organised as follows. Sections 4.2 and 4.3 introduce the proposed algorithms **LaCova** and **LaCovaC**, respectively. Experimental results are presented in Section 4.4 and Section 4.5 concludes the chapter. The work described in this chapter has been published in [18–20]. The complete source code is available in an online repository [15].

4.1 Introduction

Decision tree algorithms are considered one of the most widely used algorithms for classification [6, 150]. In fact, with regards to multi-label learning, where there is more than one target variable, there are many possible strategies to apply decision trees. On one hand, a single decision tree can be learnt for each label ignoring the correlation between different labels. On the other hand, a single tree can be learnt, which makes predictions for all labels together. Obviously, the former method ignores any correlations among labels, while they can be exploited by the latter approach.

Considering the advantages of decision tree models, this chapter proposes **LaCova**, which is a multi-label decision tree classifier. **LaCova** is different from the previous methods as it tests dependencies dynamically during construction of the decision tree. Where the other models attempt to capture label dependencies, they do so over the entire dataset whereas in practice dependencies may be more local and depend on specific attribute values. In addition to internal nodes that split the dataset horizontally based

on selected attributes, **LaCova** introduces a second kind of node for splitting the label space vertically. At deployment, a horizontal split routes to exactly one child node according to an attribute value as normal, while a vertical split tests all outgoing edges to collect predictions about the entire label set.

We then improve **LaCova** further by incorporating label clusters, which we call **LaCovaC**. It utilises the label correlation matrix at every node of the tree to find possible clusters among the labels. Vertical nodes allow multiple labels for outgoing edges based on label clusters, while in **LaCova** each vertical split has the path for only one label. **LaCovaC** interleaves between two main decisions on the label matrix with training instances in rows and labels in columns: splitting this matrix vertically by partitioning the labels into subsets, or splitting it horizontally using attributes in the conventional way.

4.2 The LaCova Algorithm

The key theoretical underpinning of decision trees is that they identify regions in the instance space with low label variance, which is built into the splitting criterion. Naive extensions to the multi-label setting would either lead to a separate tree for each label as in Binary Relevance (BR) or keep all labels together as in Label Powerset (LP) (as described previously in Section 2.4.1 of Chapter 2).

Learning a separate tree for each label would result in as many trees as there are labels, which can be hundreds or thousands in some domains (in our experiments it can be up to 374). Furthermore, the explanatory power of the decision tree models would be reduced, which would be important in various domains including decision making and medical applications, among many others. The key idea here is to model a single tree for correlated labels without reducing the predictive performance in such a way that the structure can be captured by inexpert users easily.

4.2.1 An Illustrative Example

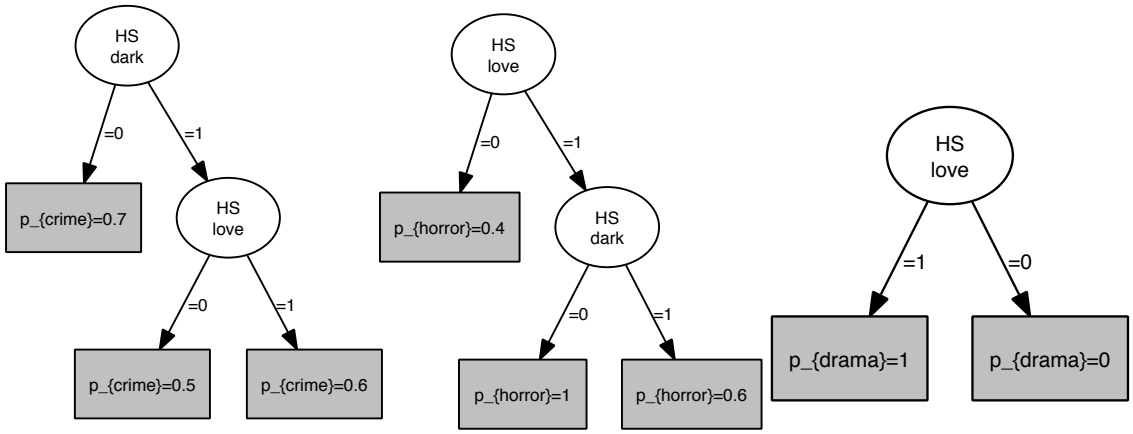
Before providing a formal definition, a simple example is introduced here to illustrate the proposed algorithm. IMDB[1] is a multi-label dataset that contains keywords describing movies, and the classification task is to predict the movie's genre. Each movie can be assigned multiple genres from among 27 labels. For simplicity, we have selected two input attributes: `dark` and `love`; and three movie genres: `crime`, `horror` and `drama`. Inspired by the IMDB dataset we have created a toy dataset, shown in Table 4.1, which we use to demonstrate the advantages of **LaCova** over binary relevance.

Figure 4.1 shows the simplest multi-label approach that learns independent trees (BR), one for each label considering the toy example in Table 4.1. The leftmost tree is for the `crime` label; the root node splits based on the attribute `dark`, when `dark=0` it leads to a leaf due to a minimum cardinality constraint (two instances). On the right-hand side of the tree, it splits further based on the attribute `love`. Leaves show the estimated label probabilities. The middle and right-most tree were constructed using the same method to learn models predicting the labels `horror` and `drama`, respectively.

[1] http://meka.sourceforge.net/

Table 4.1: A Simple example of a multi-label dataset with 12 instances that used in movie genres classification.

	labels			attributes	
	crime	horror	drama	love	dark
1	1	1	1	1	1
2	1	0	0	0	1
3	1	0	1	1	1
4	1	0	0	0	1
5	1	0	1	1	1
6	0	1	0	0	1
7	0	1	1	1	1
8	0	1	0	0	1
9	0	1	1	1	1
10	1	1	1	1	0
11	1	0	0	0	0
12	0	1	1	1	0



(a) crime (b) horror (c) drama

Figure 4.1: A separate decision tree for each label as learned by the binary relevance baseline method. HS stands for horizontal split based on the selected attribute. Two instances is the minimum cardinality constraint at leaves. For example, in order to predict genres for a new movie with the following attributes {dark=1 and love=1}, all three trees should be tested. The prediction for this movie will be three genres {crime, horror, drama}, using 0.5 as a threshold.

Suppose that we want to classify a movie with the following attributes: {dark=1 and love=1}, then according to binary relevance all three trees should be invoked. In this example the prediction for this movie will be three genres {crime, horror, drama}, using 0.5 as a threshold for label's probability.

Alternatively, ML-C4.5 learns a single decision tree for all labels as shown in Figure 4.2 and predicts all labels of a new test instance once. This algorithm is an extension of the C4.5 decision tree algorithm for binary classification. The splitting criterion is the sum of per label entropy as explained in Chapter 2. For example, to predict the genres for a movie with the following attributes: {dark=1 and love=1}, the predictions will be: {crime, horror, drama}.

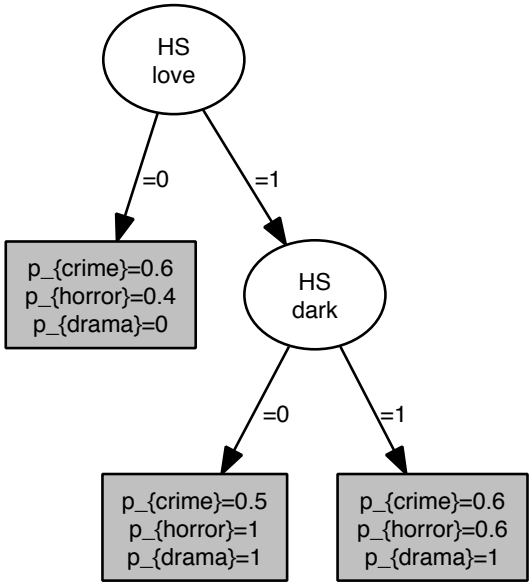


Figure 4.2: An example of a tree learned by ML-C4.5. HS denotes a horizontal split. To predict genres for a new movie with the following attributes {`dark=1` and `love=1`}, only one tree should be tested. The prediction will be {`crime`, `horror`, `drama`}, using 0.5 as a threshold.

LaCova produces a hybrid tree as in Figure 4.3. Generally speaking, the algorithm tests dependencies among labels at each iteration of constructing the tree (i.e. at each internal tree node): if labels are independent, a single tree for each label will be created; otherwise dependency is assumed, so it is better to grow a single tree.

Going back to the previous toy example, the proposed algorithm calculates the covariance at each iteration. As can be seen in Figures 4.3 and 4.4, the first covariance is 0.25. Assume 0.13 is used as a covariance threshold, which means labels are independent when the sum of absolute pairwise covariances is lower than this threshold, otherwise, they are dependent. Then, the algorithm tries to select the best attribute, which is `love` in the root node. When `love=0` we reach a leaf as splitting further would violate the cardinality constraint. On the right-hand side of the tree, we have a vertical split that leads to three binary classifiers as the covariance 0.12 is less than the threshold.

More importantly, the covariance threshold was set to an absolute value in this example, yet, we show in a later section of this chapter (Section 4.2.3) how to compute it locally within the tree. In other words, each region of the data will have its own threshold. To classify the following movie {`dark=1` and `love=1`}, one outgoing branch of a horizontal split should be followed based on the attribute value as in a standard decision tree. In contrast, in a vertical split, all outgoing branches should be followed to collect predictions for all labels that are reached from the current node. The prediction in this instance will be {`crime`, `horror`, `drama`}.

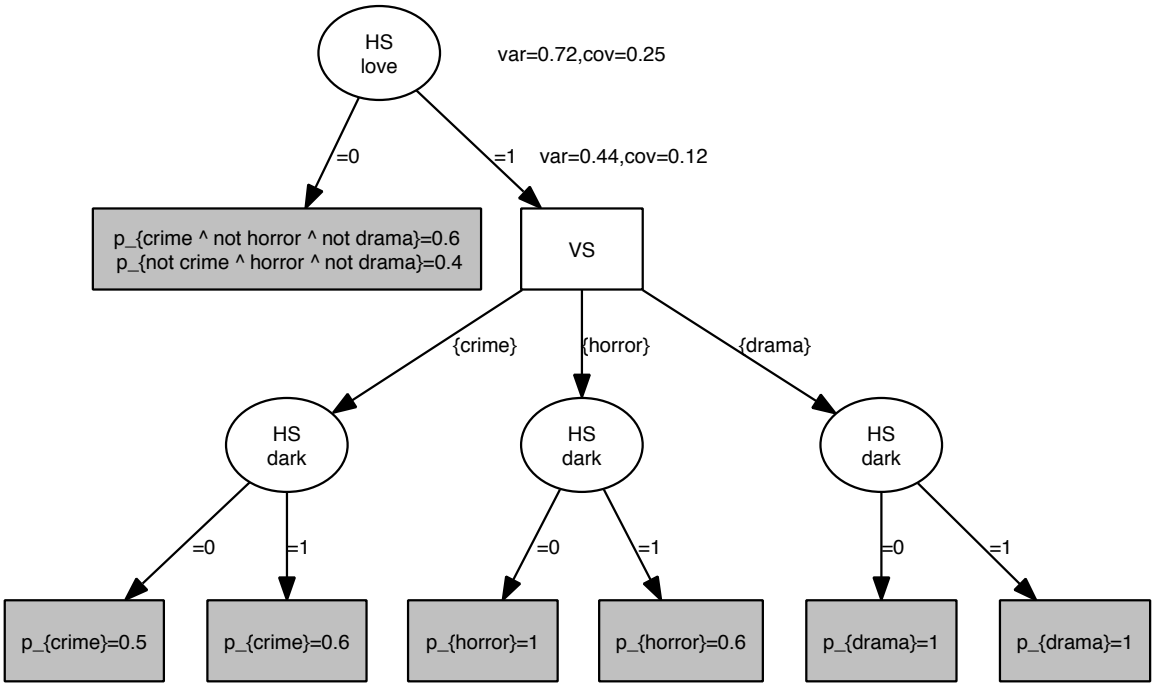


Figure 4.3: An example of a tree learned by **LaCova**. VS, HS, *cov* and *var* stand for vertical split, horizontal split, the sum of the absolute pairwise covariances and the sum of the variances, respectively. The leaves show the labels' probabilities, which can be turned into labels using the thresholding method. The predictions for a new movie with the following attributes {`dark=1` and `love=1`} will be {`crime`, `horror`, `drama`}.

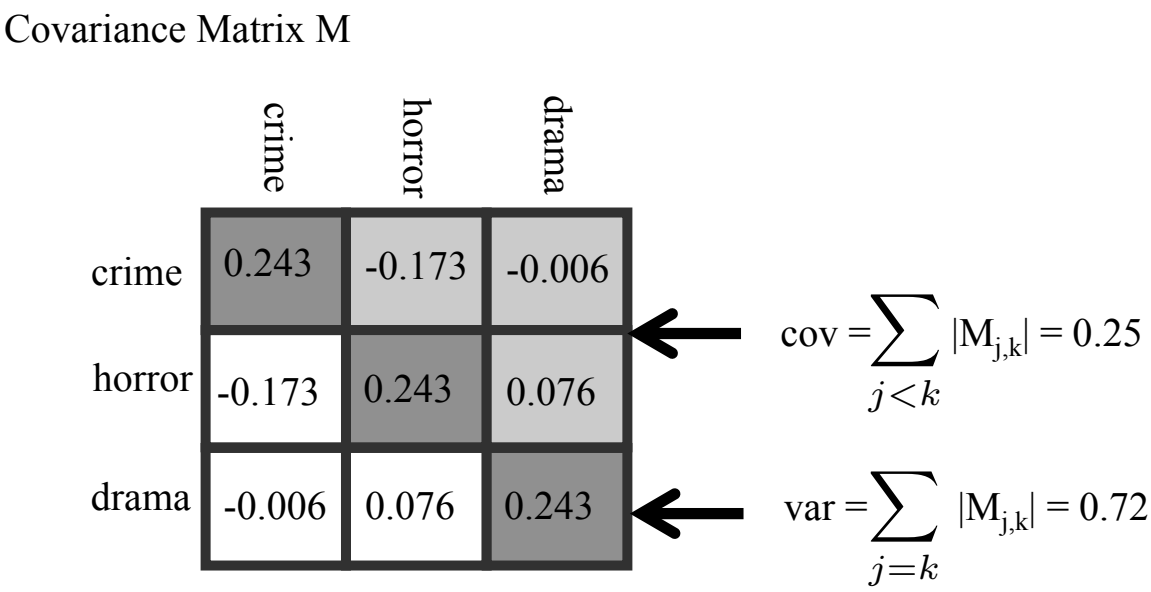


Figure 4.4: An example of how to calculate the sum of absolute pairwise covariances and the sum of the variances in **LaCova**.

4.2.2 Splitting Criterion based on Label Covariance

Standard decision tree algorithms use greedy search to select splits that maximally decrease the impurity from a node to its children. There are many ways to measure the impurity of a set of instances including entropy and Gini index. We note that the Gini index $p_j(1-p_j)$ for a binary class label l_j with proportion of positives p_j is the variance of a Bernoulli distribution with success probability p_j. With multiple binary labels, we can also consider label covariance. The covariance of two random variables is:

$$\begin{aligned} cov(V_1,V_2) &= E\left[\left(V_1 - E(V_1)\right)\left(V_2 - E(V_2)\right)\right] \\ &= E(V_1V_2) - E(V_1)E(V_2) \end{aligned}$$

where V_1 and V_2 are two random variables. $E[V_1]$ and $E[V_2]$ are the expected values of V_1 and V_2, also known as the mean.

Consider two binary labels as Bernoulli variables l_j and l_k, their expected values are the success probabilities $E[l_j] = p_j$ and $E[l_k] = p_k$:

$$cov_{jk} = p_{jk} - p_j p_k$$

where p_{jk} is the joint probability of l_j and l_k.

For a set of q labels, we can form a q by q covariance matrix with label variances on the diagonal and pairwise covariances off-diagonal. Clearly, if the trace of this matrix (the sum of the diagonal entries) is small then the set of instances is nearly pure in a multi-label sense (all labels are almost constant). The question now is what information can we glean from the off-diagonal covariances. Suppose the labels are actually independent – for example, each pairwise covariance is low in magnitude – then it would be reasonable to apply BR at this point. We might assess this by calculating the total absolute covariance, where the absolute value is taken to avoid positive and negative covariances cancelling out.

Our first innovation, therefore, is to implement a **three-way** splitting criterion:

- if the sum of variances (SumOfVar) is low, stop growing the tree;
- else, if the sum of absolute covariances (SumOfAbsCov) is low, apply BR at this point and recurse (*vertical split*);
- else find a good attribute to split on and recurse (*horizontal split*).

The first two choices require a threshold on the sum of variances and the sum of absolute covariances, respectively. In our experiments, we used the combination of a minimum number of instances in a leaf and a variance of 0 (all labels are pure) for the variance threshold as in standard decision tree algorithms. The covariance threshold requires a second innovation, which is presented in the next section. This leads to the main algorithm given in Algorithm 4, which implements the above three-way split.

Algorithm 4 LaCova (D,L): Learn a tree-based multi-label classifier from training data.

Input: Dataset D; Label set L
Output: Tree-based multi-label classifier
$f, \{D_i\} =$ **FindBestSplit**(D)
/*$f = -1$ is when there is no more attributes to consider or all remaining attributes violate the minimum number of instances in a leaf.*/

if $\mathsf{SumOfVar}(D) = 0$ or $f = -1$ **then**
  Return Leaf with relative frequencies of labels

else
  if $\mathsf{SumOfAbsCov}(D) < \lambda(|D|)_{cov}$ **then**
    for each label j in D **do**
      Tree$_j$ = Learn a decision tree for single label j
    end for
    Return Node with single-label decision tree Tree$_j$

  else
    for each child node D_i **do**
      Tree$_i$ = **LaCova** (D_i,L)
    end for
    Return Node splitting on f with subtrees Tree$_i$
  end if

end if

This leaves one issue open, which is how to find a good attribute f to split on. Clearly, the split quality measure should evaluate whether an attribute leads us closer to the goal. In traditional decision trees the goal is to have pure leaves, and so we are interested in the weighted impurity of the children of a split. Here, it is very well possible that one child in a split is near-pure over all the labels, while another child renders the labels nearly independent. Our third innovation, then, is to define a quality criterion which evaluates for each child **both** its sum of label variances and its sum of absolute label covariances, and to assign as a quality measure the **minimum** of these two (low is better). The FindBestSplit algorithm is given in Algorithm 5; Q_i measures the node quality of one of the children in the split, and Q averages this over all children.

In order to deal with numerical attributes that may have a lot of numerical values, a binary split is performed. The values of the attribute are sorted in ascending order. Then, the quality is computed for every possible cut point of an attribute as can be seen in Algorithm 6 and the best cut-point is selected. To avoid computational overhead the covariance matrix is recalculated in an incremental way. We maintain two matrices M_{right} and M_{left}. M_{right} is initialised by zeros whereas M_{left} is initialised by counts Co_{jk} of the number of instances having both the j^{th} and the k^{th} label. Each time a new cut point is tested, Co_{jk} is decreased in M_{left} and added to M_{right}. Hence, the covariance matrix can be calculated for each split using these matrices.

Algorithm 5 FindBestSplit(D): Find the best attribute to split on.

Input: Dataset D; Minimum number of instances *min*
Output: Attribute f; Data split into child nodes $\{D_i\}$
Initialise $Q^{best} = \infty$; f^{best}=-1

for each attribute f in D **do**
  if f is a numerical attribute **then**
    Cut_f=**FindBestCut**(D, f)
    Split D into child nodes $\{D_i\}$ according to value of Cut_f
  else
    Split D into child nodes $\{D_i\}$ according to values of f
  end if

  /* Check that each child node has the number of instances $|D_i| \geq min$, otherwise, the attribute f will be ignored. If there is no attribute to consider, f^{best} will be returned as -1.*/

  for each child node D_i **do**
    $Q_i = \mathsf{minimum}(\mathsf{SumOfVar}(D_i), \mathsf{SumOfAbsCov}(D_i))$
  end for
  $Q = \sum_i \frac{|D_i|}{|D|} Q_i$

  if Q<Q^{best} **then**
    $Q^{best}, f^{best}, \{D_i^{best}\} = Q, f, \{D_i\}$
  end if

end for
Return $f^{best}, \{D_i^{best}\}$

4.2.3 Estimation of the Covariance Threshold

Algorithm 4 requires a threshold $\lambda(|D|)_{cov}$ to decide whether there is a significant dependence between labels or not. The algorithm needs to take this decision locally at each node while growing the tree. We consider labels to be independent if the sum of pairwise absolute covariances is less than a threshold. One way to do that statistically is by imposing the null hypothesis in bootstrapping, which produces the sampling distribution of the covariance. For each of these bootstrap samples, the covariance matrix is computed and then the sum of the absolute values of the upper triangle (covariance) is computed. However, since bootstrapping is slow, we will derive an estimating formula (Equation 4.3) and show experimentally that it matches well with the thresholds obtained by bootstrapping.

Let co_j and co_k be two independent binomially distributed random variables that represent the number of instances for label l_j and l_k, respectively, which can be written as:

$$co_j \sim B(n, p_j)$$
$$co_k \sim B(n, p_k)$$

where p_j and p_k are the estimated probabilities that both labels l_j and l_k are $1's$ (rather than $0's$) and n is the number of training instances.

Algorithm 6 FindBestCut(D,f): Find the best cut-point to split the numerical attribute.

Input: Dataset D; Attribute f
Output: Best cut-point Cut_f^{best}
Initialize $Q_f^{best} = \infty$
Sort D according to f in ascending manner
Find all possible cut-points Cut

for each possible cut Cut_i in Cut **do**
  Split D into two child nodes $\{D_1\}$ and $\{D_2\}$ according to value of Cut_i

  for each child node D_i **do**
    $Q_i = \mathsf{minimum}(\mathsf{SumOfVar}(D_i), \mathsf{SumOfAbsCov}(D_i))$
  end for

  $Q = \sum_i \frac{|D_i|}{|D|} Q_i$

  if $Q < Q_f^{best}$ **then**
    $Q_f^{best} = Q$
    $Cut_f^{best} = Cut_i$
  end if

end for
Return Cut_f^{best}

Assume the samples are generated using random shuffle, which keeps the proportions of $1's$ constant: co_j and co_k, whereas the proportion of both labels simultaneously having value 1 may change from a sample to another: Co_{jk}. Therefore, to estimate the mean and variance of the covariance between two labels it is important to estimate the parameter Co_{jk} that represents this change. Co_{jk} of instances with both labels is distributed hypergeometrically. The hypergeometric distribution for a random variable has the following properties [151]:

$$P \sim \text{Hypergeometric}(r, m, n)$$
$$\mu = \frac{rm}{n}$$
$$\sigma^2 = \frac{n-r}{n-1} r \frac{m}{n} (1 - \frac{m}{n})$$

where r is number of successes in m draws and n is the sample size.

Therefore, the parameters of the Hypergeometric distribution for Co_{jk} are:

$$Co_{jk} \sim \text{Hypergeometric}(co_j, co_k, n)$$
$$\mu = np_j p_k$$
$$\sigma^2 = \frac{n^2}{n-1} p_j p_k (1 - p_j)(1 - p_k)$$

where $p_j = co_j / n$ and $p_k = co_k / n$.

Then due to $\widehat{cov}_{jk} = \frac{Co_{jk}}{n} - p_j p_k$ the covariance estimate has the following mean and variance:

$$\begin{aligned} E[\widehat{cov}_{jk}] &= E\left[\frac{Co_{jk}}{n} - p_j p_k\right] \\ &= \frac{E[Co_{jk}]}{n} - p_j p_k \\ &= \frac{1}{n} n p_j p_k - p_j p_k \\ &= 0 \end{aligned}$$

$$\begin{aligned} Var[\widehat{cov}_{jk}] &= Var\left[\frac{Co_{jk}}{n} - p_j p_k\right] \\ &= \frac{Var[Co_{jk}]}{n^2} \\ &= \frac{1}{n^2}\frac{n^2}{n-1} p_j p_k (1-p_j)(1-p_k) \\ &= \frac{1}{n-1} p_j p_k (1-p_j)(1-p_k) \end{aligned}$$

To estimate the mean and variance of absolute covariance we approximate the distribution of $\widehat{cov}_{jk}$ with a normal distribution and then can use the mean and variance of half-normal distribution as follows:

$$\begin{aligned} \text{mean} &= \sigma\sqrt{\frac{2}{\pi}} \\ \text{variance} &= \sigma^2\left[1 - \frac{2}{\pi}\right] \end{aligned}$$

$$\begin{aligned} E\left[|\widehat{cov}_{jk}|\right] &\approx \sqrt{\frac{2 \cdot Var[\widehat{cov}_{jk}]}{\pi}} \\ &= \sqrt{\frac{p_j p_k (1-p_j)(1-p_k)}{n-1}}\sqrt{\frac{2}{\pi}} \\ &= \sqrt{\frac{2 p_j p_k (1-p_j)(1-p_k)}{\pi(n-1)}} \\ &= \sqrt{\frac{2}{(n-1)\pi}}\sqrt{p_j p_k (1-p_j)(1-p_k)} \end{aligned} \tag{4.1}$$

$$\begin{aligned} Var\left[|\widehat{cov}_{jk}|\right] &\approx \left(1 - \frac{2}{\pi}\right) Var[\widehat{cov}_{jk}] \\ &= \frac{p_j p_k (1-p_j)(1-p_k)}{n-1}(1 - \frac{2}{\pi}) \\ &= \frac{1 - \frac{2}{\pi}}{n-1} p_j p_k (1-p_j)(1-p_k) \end{aligned} \tag{4.2}$$

As estimates of mean and variance of $\sum_{jk} |\widehat{cov}_{jk}|$ we will use the sum of means (4.1) and sum of variances (4.2):

$$\hat{\mu} = \sqrt{\frac{2}{(n-1)\pi}} \sum_{jk} \sqrt{p_j p_k (1-p_j)(1-p_k)}$$

$$\hat{\sigma}^2 = \frac{1-\frac{2}{\pi}}{n-1} \sum_{ij} p_j p_k (1-p_j)(1-p_k)$$

Finally, by assuming normal distribution, we define $\lambda(|D|)_{cov}$ to be within two standard deviation intervals to achieve 95% confidence interval.

$$\lambda(|D|)_{cov} = \hat{\mu} + 1.96 \cdot \hat{\sigma} \tag{4.3}$$

Below in Section 4.4.1 we show experimentally that this analytic approximation matches the random reallocation bootstraps very well.

4.3 The LaCovaC Algorithm

The principle of **LaCova** is to use the label covariance matrix at each node of the tree to treat labels independently or keep them together. It is based on making two extreme decisions: splitting all labels into binary classifiers or keeping them all together. In this section, we explore the value of mediating between these extreme decisions at each node in the tree.

We propose **LaCovaC**, which – different from previous methods – clusters labels *dynamically* during the construction of the decision tree, and hence models conditional label correlations at every node of the tree. Although other models attempt to cluster labels, they do so over the entire dataset, e.g. Homer and LPBR (as described previously in Section 2.4.5 of Chapter 2). In practice, conditional correlations that are used for clustering may be local and depend on specific attribute values. Hence **LaCovaC** will not separate labels automatically as happens in BR, but only in cases when labels are uncorrelated. Additionally, **LaCovaC** would not model the joint distribution at all times, as this can cause overfitting, as in LP. These decisions are taken locally at every node in the tree.

Figure 4.5 shows the tree generated using **LaCovaC** for the simple dataset introduced in Section 4.2.1. In this example, **LaCovaC** initially finds two label clusters: {`crime`, `horror`} and {`drama`}. Therefore, it splits the dataset vertically, and recursively creates a tree for each cluster of labels. On the left-hand side of the tree, a horizontal split can be observed at the next level using the attribute `dark`, leading to another vertical split on the left and a node that requires further horizontal splitting. Note that the second vertical split did not split further due to a minimum cardinality constraint (two instances in this toy example). As can be seen on the left-hand side of the tree, {`crime`, `horror`} are independent when `dark` is false (0), and dependent otherwise.

When the attribute `love` is true in addition to the attribute `dark` being true, both labels share the same estimated marginal probability of 0.6, which can lead to predicting both of them as relevant labels,

assuming 0.5 as a threshold for label's probability. However, the proposed model suggests that the two label subsets {`crime=0`, `horror=1`} or {`crime=1`, `horror=0`} with the highest probability among all possible subsets can lead to a better prediction. In other words, leaves predict the label subset that has the highest probability among all possible label subsets, regardless of the labels' marginal probabilities. With regards to the third label {`drama`}, the proposed algorithm learns it separately as can be seen on the right-hand side of the tree. This example demonstrates the power of modelling label correlations locally in the tree instead of globally in pre-processing, and also that the tree model represents such local correlations in a declarative way.

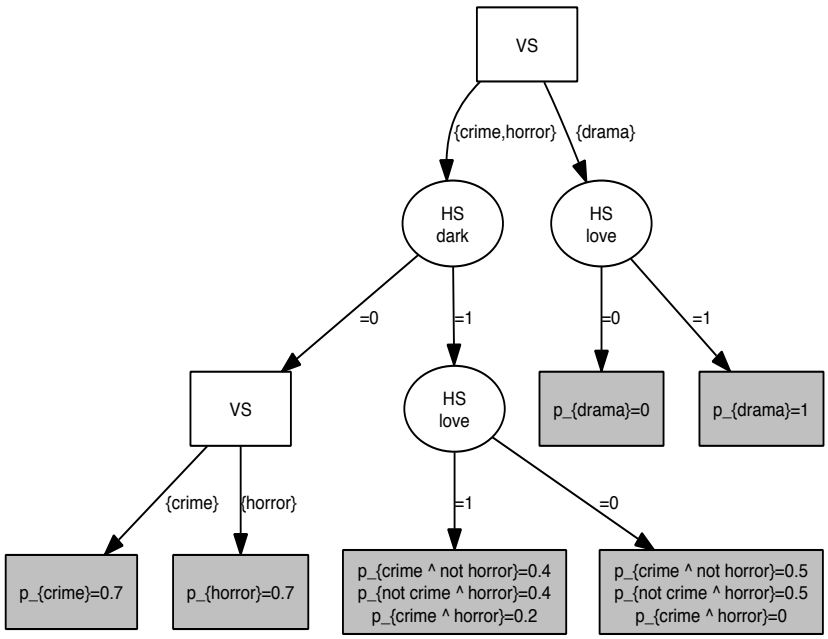


Figure 4.5: An example of a tree learned by **LaCovaC**. VS and HS stand for vertical and horizontal splits, respectively. The leaves show the probabilities of each label subset, except for the empty subset, which can be calculated as 1 minus the sum of the given probabilities. For testing, **LaCovaC** predicts the label set that has the highest probability among other subsets.

When applying the learned model on a test instance, one outgoing branch of a horizontal split should be followed based on the attribute value as in a standard decision tree. In contrast, in a vertical split, all outgoing branches should be followed to collect predictions for all labels that are reached from the current node. For example, in order to label a movie which has attributes {`dark=1` and `love=1`}, the two branches of the first vertical split should be visited. On the right-hand side of the tree, we find that the `drama` label does apply (has a probability of value 1). On the left-hand side of the tree, we will have a horizontal split based on the `dark` attribute leading to the second horizontal node based on the `love` attribute. The right leaf of this second horizontal split suggests the following label subsets: {`crime`, `not horror`} or {`not crime`, `horror`}. Suppose that the first set is chosen, then the overall predictions for this movie will be the following labels {`crime`, `not horror`, `drama`}, which

is different from what the binary relevance and **LaCova** would suggest as can be seen previously in Figures 4.1 and 4.3.

BR and **LaCovaC** suggest different predictions. BR predicts all the three labels as relevant {`crime`, `horror`, `drama`}. while **LaCovaC** proposes a different predicted set, which is {`crime`, `not horror`, `drama`}. Each of these algorithms optimises a specific evaluation metric, for example, **LaCovaC** is a good candidate for optimising exact-match, which considers only complete correct label predictions.

4.3.1 The Main Algorithm

LaCovaC implements three key decisions at every node of the decision tree, and the process can be summarised as follows. Firstly, if labels are pure or the set of instances reaches the minimum number of instances, the algorithm stops and returns a leaf. Secondly, if the labels' correlation matrix suggests the presence of a cluster structure, the algorithm splits the dataset reaching that node according to the label clusters (vertical split). Finally, a set of labels located together at a node is considered, and the best attribute to split is determined (horizontal split).

Algorithm 7 details the main algorithm implementing the approach described here. The first decision requires a threshold on the label variance. In our experiments, we used a combination with a minimum number of instances at a leaf and a variance threshold of 0 (all labels are pure) as variance threshold as in **LaCova**. The detection of a cluster structure is presented in the next section. The third decision that splits the instance space horizontally also demands a splitting criterion.

The most popular splitting criteria in standard decision trees learning are the Gini index and information gain. However, **LaCovaC** implements the splitting criterion designed specifically for multi-label data as discussed previously in Algorithm 5, which can be summarised as follows. It evaluates for each child both its sum of label variances and its sum of absolute label covariances and assigns as a splitting quality measure the minimum of these two (low is better). The attribute's splitting quality measure is then averaged over the children's splitting qualities. Then, it selects the attribute that has the lowest splitting quality measure. This criterion can identify regions with either a low multi-label variance or low label covariance. We improve on this by clustering the labels in a principled way into independent groups of correlated labels.

4.3.2 Label Clustering

The labels are clustered based on the correlation matrix Cor, which is a square q by q matrix for q labels that contains the Pearson's correlation coefficients cor_{jk} between all pairs of labels l_j and l_k, $j,k = 1,\cdots,q$. As labels are binary in the multi-label setting, the Pearson's coefficient is reduced to the Phi coefficient, a well-known measure of association between two binary variables, also called dichotomous variables [152, 153]. Whenever the absolute value of correlation between two labels is greater than a threshold $\lambda_{|D|}$ calculated from the number of instances (derivation given below), we decide that these labels are correlated and should be in the same cluster.

Algorithm 7 LaCovaC (D,L): Learn a tree-based multi-label classifier from training data.

Input: Dataset D; Label set L
Output: Tree-based multi-label classifier

$f, \{D_i\} =$ **FindBestSplit**(D)
/*$f = -1$ is when there is no more attributes to consider or all remaining attributes violate the minimum number of instances in a leaf.*/

if Labels in D are pure or $f = -1$ **then**
  Return Leaf with relative frequencies of label sets and the predicted label set that has the highest probability.
else
  clust=**CLUST**(D, L)

  if $|clust|$>1 **then**
    for each label cluster (set) s in *clust* **do**
      $D_s =$ Split D vertically based on the label cluster s.
      /* Learn a decision tree for the label cluster s. */
      Tree$_s =$ **LaCovaC** (D_s,s).
    end for
    Return Node splitting on *clust* with subtrees Tree$_s$

  else
    for each child node D_i **do**
      Tree$_i =$ **LaCovaC** (D_i,L)
    end for
    Return Node splitting on f with subtrees Tree$_i$
  end if

end if

Algorithmically, this can be achieved by single linkage agglomerative hierarchical clustering as shown in Algorithm 8. The algorithm starts by creating a separate cluster for each label. It then proceeds iteratively by taking pairs of labels in the order of increasing distance between them, where distance is defined as one minus absolute correlation, $dist_{jk} = 1 - |cor_{jk}|$. For each pair of labels, the clusters containing these labels are merged (unless they belong to the same cluster already). Such merges are performed until the distance between labels becomes greater than $1 - \lambda(|D|)_{cor}$, that is absolute correlation less than $\lambda(|D|)_{cor}$.

We now derive the threshold $\lambda(|D|)_{cor}$ to decide whether a pair of labels are correlated or not, before merging them into one cluster. The idea is to set this threshold equal to two standard deviations in the distribution of correlation under the assumption of independent labels, hence enclosing a 95% confidence interval. The threshold depends on n, which is the number of instances reaching the current node in the tree. To derive the threshold we consider two labels with empirical frequencies p_j and p_k, respectively. The empirical Pearson's correlation between these labels as Bernoulli variables can be calculated as follows:

$$\widehat{cor_{jk}} \approx \frac{\frac{Co_{jk}}{n} - p_j * p_k}{\sqrt{p_j p_k (1-p_j)(1-p_k)}} \tag{4.4}$$

where Co_{jk} is the number of instances with both of those labels, and hence $\frac{Co_{jk}}{n}$ is the proportion of such instances. The terms $p_j(1-p_j)$ and $p_k(1-p_k)$ in the denominator are the variances of these Bernoulli variables.

Algorithm 8 CLUST(D,L): Cluster labels based on the correlation matrix.

Input: Dataset D; a set of labels $L = l_1, \cdots, l_q$;
Output: *currClust* - the clusters of labels

/* Compute the Pearson's correlation coefficients between each pair of labels.*/
Cor = Correlation Matrix

/* Compute the distance between each pair of labels using the absolute correlations.*/
$Dist = 1 - |Cor|$

/* Build initial clusters with each label in a separate cluster.*/
$currClust = \{l_1\}, \{l_2\}, \cdots, \{l_q\}$

/* Create a list of label pairs sorted in ascending order of distance value.*/
$pairList =$ all pairs of labels (l_j, l_k) with $dist_{jk}$<$1 - \lambda(|D|)_{cor}$, sorted by $dist_{jk}$ ascendingly

for each label pair (l_j, l_k) in pairList **do**
  if labels l_j and l_k are in different clusters **then**
    merge clusters containing l_j and l_k within *c*urrClust
  end if
end for

Return *currClust*

Next, we study the distribution of $\widehat{cor_{jk}}$ under the assumption that labels l_j and l_k are independent. This distribution can be generated by randomly reassigning one of those labels between the instances, keeping the total frequency constant. In such case Co_{jk} has a hypergeometric distribution, and we can then directly calculate its mean μ and variance σ^2:

$$\begin{aligned} Co_{jk} &\sim \text{Hypergeometric}(np_j, np_k, n) \\ \mu &= np_j p_k \\ \sigma^2 &= \tfrac{n^2}{n-1} p_j p_k (1-p_j)(1-p_k) \end{aligned}$$

From this we can calculate the mean and variance of the correlation $\widehat{cor_{jk}}$ between labels l_j and l_k, as follows:

$$E[\widehat{cor_{jk}}] = \frac{\frac{1}{n} * np_j p_k - p_j p_k}{\sqrt{p_j p_k (1-p_j)(1-p_k)}} = 0$$

$$Var[\widehat{cor_{jk}}] = \frac{\frac{1}{n^2} * \frac{n^2}{n-1} p_j p_k (1-p_j)(1-p_k)}{p_j p_k (1-p_j)(1-p_k)} = \frac{1}{n-1}$$

We define $\lambda(|D|)_{cor}$ as the value enclosing a 95% confidence interval assuming a normal distribution for $\widehat{cor_{jk}}$ which can be calculated in the usual way:

$$\begin{aligned} \lambda(|D|)_{cor} &= 1.96 \cdot \sqrt{var(\widehat{cor_{jk}})} \\ &= 1.96 \cdot \frac{1}{\sqrt{n-1}} \end{aligned} \tag{4.5}$$

where n is the number of instances reaching a particular decision node in the tree. Note that this threshold is always less than 1 as n>5 in our experiments.

4.4 Experimental Evaluation

Experimental evaluation is divided into two parts. Firstly, we performed experiments to evaluate the analytical threshold for the covariance that has been implemented in **LaCova**. Secondly, we performed experiments and compared **LaCova** and **LaCovaC** against other baseline approaches such as BR and LP and other state-of-the-art multi-label algorithms.

4.4.1 Analytical Threshold Evaluation

In Section 4.2.3 we have derived the analytical covariance threshold. In this section we show that the analytical derivation of the threshold approximates the random reallocation of labels (bootstrapping) in 13 benchmark datasets that were presented previously in Section 2.4.9 of Chapter 2.

The threshold was estimated by performing many repetitions of simulated reallocation of a label across instances (1000 reallocations in our experiments). Then, we computed the covariance matrix for each sample and the sum of the absolute values of the upper triangle (covariance). We estimated the distribution parameters: mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$. Finally, we computed the threshold by assuming a normal distribution and 95% confidence interval. The formula for calculating $\lambda(|D|)_{cov}$ using bootstrapping is the following:

$$\lambda(|D|)_{cov} = \hat{\mu} + 1.96 \cdot \hat{\sigma} \tag{4.6}$$

where n is the number of instances in the dataset.

We demonstrate experimentally on the 13 datasets that $\lambda(|D|)_{cov}$ obtained by bootstrapping is comparable to the analytical $\lambda(|D|)_{cov}$ as shown in Table 4.2. Therefore, was used the analytical approach for computing the covariance threshold in the next experiments as the bootstrapping is computationally expensive.

Table 4.2: Thresholds obtained from both bootstrapping and analytical methods at the root level of a decision tree.

	$\lambda(\lvert D\rvert)_{cov}$ based on bootstrapping	$\lambda(\lvert D\rvert)_{cov}$ based on the analytical solution
Corel5k	3.50	4.11
Cal500	45.99	45.84
Bibtex	1.57	1.58
Language log	2.83	2.85
Enron	0.85	0.86
Medical	0.35	0.39
Genbase	0.34	0.35
Slashdot	41.65	49.82
Birds	0.34	0.34
Yeast	0.26	0.26
Flags	0.27	0.26
Emotions	0.14	0.14
Scene	0.03	0.03

4.4.2 Experimental Setup

LaCova, **LaCovaC**, and ML-C4.5 were implemented in Java on the Meka platform [2]. The minimal number of instances in the leaves in these methods was set to five. BR, LP, and CC algorithms were run in Meka, whereas Mulan [3] was used for Homer and LPBR algorithms. The Homer algorithm was run using the recommended setting for its parameters, as reported by [59].

LPBR requires parameter settings, such as the non-improving counter to prevent clustering. The default parameters settings in Mulan were five for the non-improving counter and a five-fold cross-validation for testing the clustering performance. The target loss function to evaluate the clustering was set to exact-match according to Mulan. Exact-match is a strict measure that examines if the predicted and actual label sets are equal or not.

10-fold cross-validation was performed throughout the experiments, except for the two largest datasets in terms of both the number of labels and number of instances: Corel5k and Bibtex. The exception is because cross-validation was too computationally intensive for LPBR in these two datasets as we were not able to run this algorithm using the available resources [4]. Therefore, a single train/test split was used instead, which was provided by Mulan. Moreover, the cluster machine was used for LPBR with the 8 largest datasets in terms of number of labels [5] using a single core and allocating 50Gb of RAM.

When employing all these methods, the trees were produced with Weka's J48 algorithm [6]. For all methods, we used the Pcut method to convert the label probabilities into relevant labels [9]. This method uses a single threshold for all labels, chosen such that the average label density in the test data is the

[2] http://meka.sourceforge.net/

[3] http://mulan.sourceforge.net/

[4] 2.7 GHz Intel Core i5 processor and 8GB of memory

[5] https://www.acrc.bris.ac.uk/acrc/phase3.htm

[6] http://sourceforge.net/projects/weka

same as in the training data. Laplace correction was used to smooth the estimated probabilities [154].

All algorithms were evaluated under the Meka framework. We used the most common multi-label metrics, namely multi-label accuracy, exact-match, Hamming loss, micro F_1 and macro-averaged F_1 score per instance and per label [73–75]. We also used a multi-label version of log-loss [9] to evaluate the estimated probabilities. We used a maximum of $\log(\frac{1}{|D|})$ depending on the dataset's size to restrict the amount of penalty as suggested in [9]. The description and formal definition of these evaluation measures for multi-label data have been presented previously in Chapter 2.

4.4.3 Results and Discussion

We conducted the Friedman test based on the average ranks for all datasets [145]. It ranks the algorithms across datasets separately, thus the best algorithm gets the rank of 1, the second best the rank of 2, etc. Then, it calculates the test statistic on the ranks averaged over all datasets in order to verify whether the differences between all algorithms are statistically significant. The Friedman test indicated a significant difference at the 5% significant level for all metrics except multi-label accuracy; therefore, we carried out post-hoc Nemenyi tests for the other 6 measures [145] as shown in Figure 4.6.

Table 4.3 shows the average ranks for each algorithm over 13 datasets and seven evaluation measures. We also show the mean of these average ranks aggregated over all evaluation measures, which shows that overall **LaCovaC** scores highest (lowest mean rank), followed by CC and BR, as can be seen in the right-most column of Table 4.3, and is in fact, the only algorithm with this property. Furthermore, on each of the metrics individually **LaCovaC** is not significantly worse than the top contender. It is widely recognised in the literature that different multi-label evaluation metrics measure different things; hence there is value in demonstrating that an algorithm performs well across the board, which is why we included the right-most column in Table 4.3.

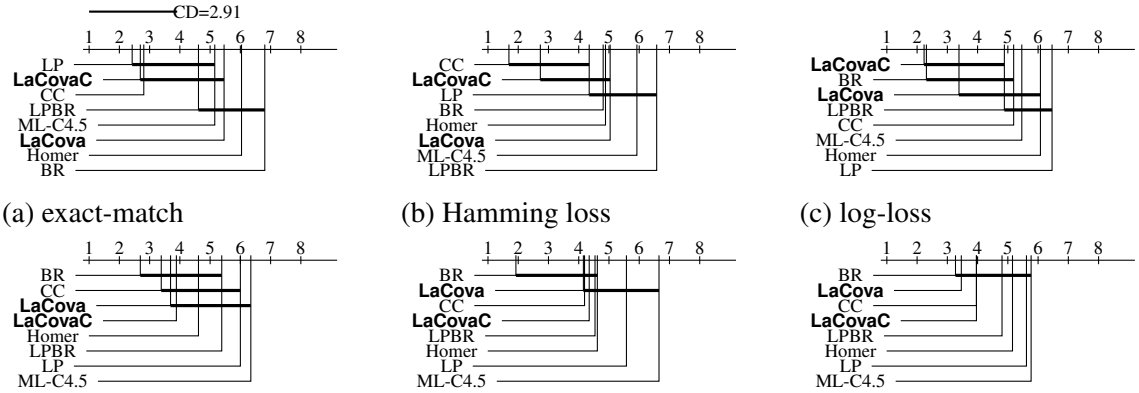
Table 4.3: Average ranks obtained by the Friedman test over 13 datasets. The last column shows the mean of the average ranks obtained by each algorithm over all the evaluation measures, and the algorithms are sorted on increasing mean. *mla*, *em*, *hl* and *ls* denote multi-label accuracy, exact-match, Hamming loss and log-loss, respectively. micro F_1 is the micro averaged $F1$. macro f_l and macro f_e are F_1 measures averaged in terms of labels and instances, respectively.

	mla	*em*	*hl*	*ls*	micro F_1	macro f_l	macro f_e	mean
LaCovaC	3.57	2.69	2.73	**2.23**	3.88	4.34	3.96	3.34
CC	**3.46**	2.8	**1.69**	5.19	3.38	4.19	3.69	3.49
BR	4.57	6.8	4.8	2.3	**2.69**	**1.92**	**3.26**	3.76
LaCova	3.65	5.46	5.03	3.38	3.69	4.15	3.46	4.12
LP	4.92	**2.42**	4.34	6.46	6	5.57	5.61	5.05
LPBR	4.84	4.61	6.57	4.88	5.38	4.53	4.8	5.09
Homer	5.3	6.03	4.88	6.07	4.61	4.61	5.15	5.24
ML-C4.5	5.65	5.15	5.92	5.46	6.34	6.65	5.76	5.85

Some further conclusions that can be drawn from the experiments are as follows. **LaCovaC** outperforms Homer, ML-C4.5, and LPBR in the average ranks with respect to all evaluation measures

used in this work, and in several cases statistically significantly. **LaCovaC** is not significantly worse than the best performing algorithm (LP) in terms of the exact-match measure. In fact, according to Table 4.5 **LaCovaC** loses to LP in the 4 datasets with the smallest numbers of labels (up to 14) but wins in 6 out of the 9 datasets with more labels. This fact confirms the assumption that LP can overfit the training data in case of a large number of labels and label combinations because it can only model labelsets observed in the training. Notably, for exact-match **LaCovaC** is the overall best method or tied with the best in the three datasets with the biggest numbers of labels (Corel5k, Cal500, and Bibtex with 374, 174 and 159 labels, respectively). **LaCovaC** has a better Hamming loss than LP in 9 out of 13 datasets.

Additionally, **LaCovaC** outperforms BR in terms of exact-match, log-loss and Hamming loss. BR has the best average rank considering F_1 score per instance average f_e, F_1 per label average f_l and micro F_1 as it decomposes the multi-label problem into several binary classifiers, which can be better for F_1 score. We can further observe that **LaCovaC** has the best average rank in terms of log-loss, which evaluates the classifiers' scores regardless of the thresholding technique. It is significantly better than CC, ML-C4.5, Homer and LP. Finally, CC gets top average rank for Hamming loss and **LaCovaC** is the second best without a significant loss against CC.

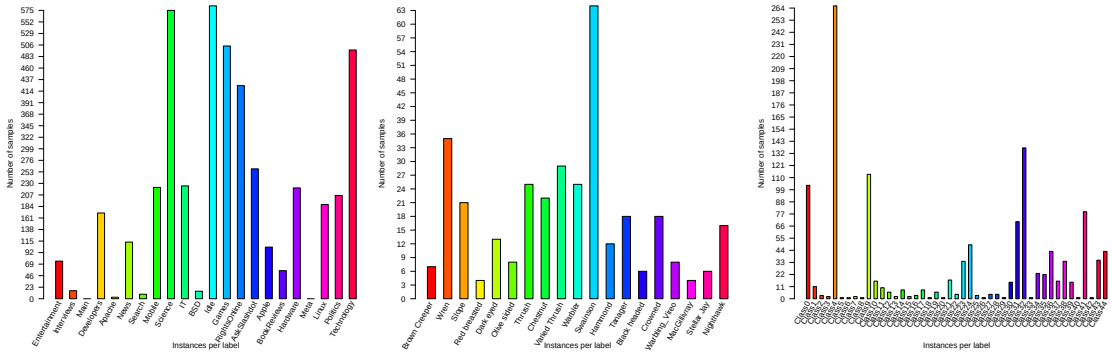


(a) exact-match (b) Hamming loss (c) log-loss

(d) micro F_1 (e) f_l (f) f_e

Figure 4.6: Critical Difference diagrams using pairwise comparisons with the post-hoc Nemenyi tests for experiments where the Friedman test yields significance at the 0.05 significance level.

It is known that Hamming loss can be minimised without incorporating label correlations suggesting that BR is hard to beat with respect to this metric [12, 71]. Interestingly, **LaCovaC** was able to improve Hamming loss. Figure 4.7 shows the label frequencies for three datasets where **LaCovaC** outperforms BR in Hamming loss, which suggests that by modelling label correlations we can improve the predictions of hard labels. Considering **LaCova**, it is better than **LaCovaC** in terms of F_1. It ranks the third and the second for micro F_1 and both f_l and f_e, respectively. Detailed results of the experiments are given in Tables 4.5, 4.6 and 4.7.



(a) Slashdot (b) Birds (c) Medical

Figure 4.7: Label frequencies for three datasets. The common characteristic among these datasets is the class imbalance.

4.5 Concluding Remarks

In this chapter, we presented two novel algorithms for multi-label classification, both of which take label correlation into account. The key idea of the **LaCova** algorithm is to use a label covariance matrix at every node of the tree in order to decide to stop growing, split vertically (treat labels independently) or split horizontally (find an attribute to split on). Its main novelty is to introduce vertical splits that separate locally independent labels, in addition to the traditional attribute-based horizontal splits that divide the instance space as in the standard decision tree algorithms. Moreover, a new decision tree splitting criterion has been proposed, which rewards splits that reduce either variance or covariance among labels.

We then presented **LaCovaC**, which computes the label correlation matrix at each node of the tree in order to identify label correlations and then cluster them locally. Vertical splits are improved further in **LaCovaC** as they handle dependent label subsets, not only single label or the full label set as in **LaCova**. The clusters are obtained locally by identifying the conditionally dependent labels in localised regions of the attribute space using the label correlation matrix. **LaCovaC** incorporates two main decisions on the label matrix with training instances in rows and labels in columns: splitting this matrix vertically by partitioning the labels into subsets, or splitting it horizontally using attributes in a conventional way.

We carried out an empirical evaluation on 13 benchmark datasets to compare the proposed methods with baseline and state-of-art approaches. We used seven common evaluation metrics for multi-label classification. **LaCovaC** has the best average rank for log-loss and the second best for Hamming loss (without significant loss). For exact-match, **LaCovaC** shows a comparable performance to the best algorithm LP, which is a strong baseline for exact-match and hard to beat. In addition, it outperforms LP on 6 of the 9 largest datasets in terms of the number of labels. This suggests that combining the LP and BR as in **LaCovaC** leads to improvement over those metrics and reduces the overfitting risk.

Moreover, experimental results indicate that the binary relevance approach (BR) produced arguably good predictive performance, given the simplicity of this method (ignoring label dependencies), by comparison with more advanced multi-label classification methods, which consider label dependencies and were used in the experiments. Therefore, BR remains a good candidate, especially in cases where the learnt model is well fitted to the data [155, 156]. There is an interesting interpretation in [157], which showed that BR can achieve optimal performance in any dataset using a suitable base classifier. Furthermore, they showed that modelling label correlations only helps when a base classifier behind one or more labels is inadequate.

Overall, the main strength of **LaCovaC** is its strong performance across all these metrics, as can be seen in the right-most column of Table 4.3. Furthermore, on each of the metrics individually **LaCovaC** is not significantly worse than the top contender. It is widely recognised in the literature that different multi-label evaluation metrics measure different things; hence there is value in demonstrating that an algorithm performs well across the board.

The general conclusion of these experiments, which compare **LaCova** and **LaCovaC** with other algorithms, are summarised in the following points:

- There is no algorithm which performs best across a range of evaluation measures. Multi-label classification algorithms can be selected depending on the dataset and the desired evaluation metrics.

- Classifiers that transform multi-label problems into several binary classification problems are good in terms of both Hamming loss and F-measure. Binary relevance and classifier chain are good examples for such classifiers. However, exploiting label correlations may improve the prediction of hard labels, which in turn improves their Hamming loss.

- Exact-match is a strict measure that requires the entire label set to be predicted correctly. By considering correlation among labels as in label powerset (LP), we can achieve higher exact-match than almost all other methods (except LP). As **LaCovaC** adopts label powerset for dependent labels, they achieve better exact-match. CC also considers label correlations, however, it depends on the label orders. For that reason, the authors of CC proposed an ensemble of classifier chains that aggregates the results of different label orders.

- **LaCova** is better than **LaCovaC** in terms of F_1. Whereas the latter outperforms the former in terms of exact-match as it models the joint label distributions. Moreover, **LaCovaC** is better than **LaCova** with respect to Hamming loss, log-loss, and multi-label accuracy.

- We finally discuss the efficiency of the proposed methods in terms of training time compared with baseline and state-of-art approaches. The results are given in Table 4.4. The Friedman test indicated a significant difference at the 5% significant level; therefore, we carried out post-hoc Nemenyi tests as shown in Figure 4.8.

Table 4.4: Comparison of training time in hundreds of seconds for all algorithms across 13 datasets. * shows the results when the cluster machine was used.

	BR	LP	CC	Homer	LPBR	ML-C4.5	**LaCova**	**LaCovaC**
Corel5k	13.101(4)	**0.398(1)**	4.139(3)	1.310(2)	1603.200(8)	93.895(5)	467.183(6)	617.478(7)
Cal500	0.091(5)	**0.003(1)**	0.045(3)	0.076(4)	9.581(8)	0.022(2)	0.339(6)	0.362(7)
Bibtex	13.501(4)	**1.457(1)**	12.752(3)	7.426(2)	134.304(5)	523.799(6)	938.052(8)	698.793(7)
Language log	0.536(2)	**0.092(1)**	0.854(4)	0.599(3)	19.160(6)	29.781(7)	15.708(5)	40.238(8)
Enron	0.756(2)	**0.145(1)**	0.920(4)	0.835(3)	20.893(7)	6.938(5)	35.224(8)	20.291(6)
Medical	0.137(2)	**0.025(1)**	0.143(3)	0.171(4)	8.684(7)	16.052(8)	1.844(6)	0.773(5)
Genbase	0.020(3)	**0.005(1)**	0.014(2)	0.026(4)	1.952(8)	0.330(6)	0.343(7)	0.209(5)
Slashdot	1.882(3)	**0.548(1)**	2.860(4)	1.846(2)	72.396(8)	35.067(6)	43.995(7)	18.229(5)
Birds	0.013(2.5)	**0.006(1)**	0.017(4)	0.013(2.5)	0.365(6)	0.025(5)	3.118(8)	0.429(7)
Yeast	0.014(2.5)	**0.007(1)**	0.014(2.5)	0.018(4)	0.557(8)	0.029(5)	0.084(7)	0.037(6)
Flags	**0.000(2)**	**0.000(2)**	**0.000(2)**	0.002(5.5)	0.007(8)	0.001(4)	0.003(7)	0.002(5.5)
Emotions	0.001(2.5)	**0.000(1)**	0.001(2.5)	0.003(6)	0.030(8)	0.002(4)	0.003(6)	0.003(6)
Scene	0.014(2)	**0.005(1)**	0.017(3)	0.019(4)	0.321(8)	0.056(6)	0.059(7)	0.026(5)
Average	2.80	**1.07**	3.07	3.53	7.30	5.30	6.76	6.11

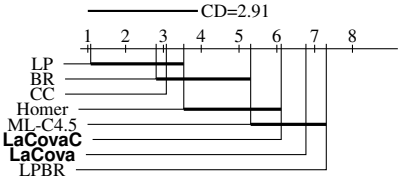


Figure 4.8: Critical Difference diagrams using pairwise comparisons with the post-hoc Nemenyi tests for experiments where the Friedman test yields significance at the 0.05 significance level.

Several directions can be taken for further work. We plan to investigate the parameter configuration for **LaCovaC**, for example, a stopping criterion for the clustering algorithm. Moreover, it would be interesting to investigate alternate clustering approaches, such as spectral clustering [128]. Both algorithms can also be used as a meta-classifier, where any single-label base classifier can be applied after a vertical split when labels become independent. Furthermore, to counteract the large variance problem associated with the standard decision tree learning – which carries over to the proposed algorithms –, we could use bootstrap aggregates as in random forests [150].

Finally, using the significance threshold on the correlation balances the sample size and the strength of the correlation, such that both low correlations on a large sample size and high correlations on a small sample are detected as significant. However, finding an even better balance between the effect size and sample size is an interesting future research direction.

Table 4.5: Results on 13 datasets with regards to multi-label accuracy and exact-match, the higher the value the better.

	BR	LP	CC	Homer	LPBR	ML-C4.5	**LaCova**	**LaCovaC**
	multi-label accuracy							
Corel5k	0.089(2)	0.080(7)	0.083(6)	**0.092(1)**	0.023(8)	0.087(4)	0.084(5)	0.088(3)
Cal500	0.201(6)	0.204(5)	0.208(4)	0.197(7)	0.067(8)	**0.278(1)**	0.254(2)	0.209(3)
Bibtex	0.259(5)	0.244(6)	0.276(3)	0.208(7)	**0.296(1)**	0.176(8)	0.267(4)	0.280(2)
Language log	0.245(2)	0.234(4)	**0.247(1)**	0.233(5)	0.184(8)	0.224(7)	0.235(3)	0.232(6)
Enron	0.296(3)	0.277(5)	0.315(2)	0.281(4)	**0.320(1)**	0.207(8)	0.256(6)	0.246(7)
Medical	0.729(4)	0.724(6)	0.728(5)	0.700(7)	0.745(2)	0.363(8)	0.732(3)	**0.762(1)**
Genbase	0.901(8)	0.980(3)	**0.986(1)**	0.952(5)	0.955(4)	0.946(7)	0.950(6)	0.981(2)
Slashdot	0.425(4)	0.420(5)	0.429(3)	0.379(6)	0.376(7)	0.372(8)	0.439(2)	**0.449(1)**
Birds	0.491(3)	0.453(7)	0.486(4)	0.493(2)	0.411(8)	0.466(6)	0.481(5)	**0.499(1)**
Yeast	0.391(6.5)	0.396(4.5)	0.397(3)	0.381(8)	0.411(2)	0.396(4.5)	**0.416(1)**	0.391(6.5)
Flags	0.534(6)	0.531(7)	0.540(5)	0.544(4)	0.509(8)	**0.572(1)**	0.556(3)	0.561(2)
Emotions	0.402(4)	0.416(2)	0.388(7)	0.390(6)	**0.417(1)**	0.406(3)	0.400(5)	0.384(8)
Scene	0.403(6)	0.432(2.5)	**0.464(1)**	0.380(7)	0.421(5)	0.366(8)	0.432(2.5)	0.429(4)
Average ranks	4.57	4.92	**3.46**	5.30	4.84	5.65	3.65	3.57
	exact-match							
Corel5k	0.000(8)	0.008(2)	0.006(3)	0.004(4.5)	0.002(6.5)	0.002(6.5)	0.004(4.5)	**0.018(1)**
Cal500	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**	**0.000(4.5)**
Bibtex	0.079(8)	0.137(2)	0.122(4)	0.084(7)	0.123(3)	0.092(6)	0.108(5)	**0.149(1)**
Language log	0.184(5)	**0.205(1)**	0.203(2)	0.183(6)	0.153(8)	0.195(4)	0.177(7)	0.197(3)
Enron	0.022(7)	**0.079(1)**	0.076(2)	0.031(5.5)	0.045(3)	0.020(8)	0.031(5.5)	0.032(4)
Medical	0.607(6)	0.642(3)	0.641(4)	0.598(7)	0.655(2)	0.293(8)	0.625(5)	**0.673(1)**
Genbase	0.809(8)	0.962(3)	**0.971(1)**	0.921(6)	0.938(4)	0.923(5)	0.914(7)	0.967(2)
Slashdot	0.296(7)	0.368(2)	0.356(3)	0.262(8)	0.306(6)	0.308(5)	0.353(4)	**0.392(1)**
Birds	0.398(4)	0.376(7)	**0.410(1)**	0.404(3)	0.339(8)	0.379(6)	0.395(5)	0.407(2)
Yeast	0.035(8)	**0.134(1)**	0.118(4)	0.050(7)	0.122(3)	0.109(5)	0.107(6)	0.124(2)
Flags	0.119(8)	0.202(2)	0.182(3)	0.155(5)	0.145(6)	**0.207(1)**	0.135(7)	0.171(4)
Emotions	0.125(8)	**0.202(1)**	0.157(4)	0.142(7)	0.165(3)	0.179(2)	0.143(5.5)	0.143(5.5)
Scene	0.284(7)	0.393(2)	**0.416(1)**	0.277(8)	0.375(3)	0.307(6)	0.354(5)	0.366(4)
Average ranks	6.80	**2.42**	2.80	6.03	4.61	5.15	5.46	2.69

Table 4.6: Results on 13 datasets with regards to Hamming loss and log-loss, the lower the value the better.

	BR	LP	CC	Homer	LPBR	ML-C4.5	**LaCova**	**LaCovaC**
	Hamming loss							
Corel5k	0.017(5.5)	0.016(3)	**0.011(1)**	0.016(3)	0.023(7)	0.203(8)	0.016(3)	0.017(5.5)
Cal500	0.223(7)	0.201(5)	0.189(2.5)	0.204(6)	0.540(8)	**0.188(1)**	0.189(2.5)	0.198(4)
Bibtex	0.022(5.5)	0.020(4)	**0.017(1)**	0.022(5.5)	0.018(2.5)	0.026(8)	0.024(7)	0.018(2.5)
Language log	0.031(6)	0.026(3)	**0.023(1)**	0.027(4)	0.069(8)	0.029(5)	0.033(7)	0.025(2)
Enron	0.082(6)	0.078(4.5)	**0.062(1)**	0.078(4.5)	0.077(2.5)	0.202(8)	0.085(7)	0.077(2.5)
Medical	0.013(3.5)	0.014(5.5)	0.011(2)	0.016(7)	0.013(3.5)	0.163(8)	0.014(5.5)	**0.010(1)**
Genbase	0.008(4.5)	0.002(2.5)	**0.001(1)**	0.008(4.5)	0.019(7.5)	0.019(7.5)	0.012(6)	0.002(2.5)
Slashdot	0.055(3)	0.058(4.5)	0.045(2)	0.075(7)	0.086(8)	0.066(6)	0.058(4.5)	**0.028(1)**
Birds	0.072(3)	0.076(5.5)	**0.063(1.5)**	0.075(4)	0.113(8)	0.077(7)	0.076(5.5)	**0.063(1.5)**
Yeast	0.296(7)	0.288(4)	0.281(2)	0.290(6)	0.363(8)	0.289(5)	**0.276(1)**	0.285(3)
Flags	0.307(6)	0.309(7)	0.302(5)	0.301(4)	0.332(8)	**0.273(1)**	0.293(3)	0.280(2)
Emotions	0.296(2)	0.304(3)	**0.286(1)**	0.305(4.5)	0.309(7.5)	0.305(4.5)	0.309(7.5)	0.306(6)
Scene	0.193(3.5)	0.200(5)	**0.185(1)**	0.193(3.5)	0.238(7)	0.294(8)	0.201(6)	0.188(2)
Average ranks	4.80	4.34	**1.69**	4.88	6.57	5.92	5.03	2.73
	log-loss							
Corel5k	**0.048(1)**	0.101(5)	0.070(4)	0.206(6)	0.275(8)	0.216(7)	0.064(3)	0.061(2)
Cal500	0.481(4)	0.786(7)	0.741(6)	0.586(5)	0.853(8)	**0.381(1)**	0.407(3)	0.394(2)
Bibtex	**0.068(1)**	0.158(6)	0.132(5)	0.194(7)	0.075(3)	0.220(8)	0.078(4)	0.070(2)
Language log	**0.080(1)**	0.130(6)	0.113(4)	0.165(7)	0.122(5)	0.349(8)	0.085(2)	0.093(3)
Enron	**0.197(1)**	0.402(7)	0.353(5)	0.367(6)	0.230(3)	0.437(8)	0.261(4)	0.218(2)
Medical	**0.035(1)**	0.064(5)	0.052(3.5)	0.084(7)	0.052(3.5)	0.364(8)	0.066(6)	0.037(2)
Genbase	0.008(3)	0.007(2)	**0.005(1)**	0.022(6)	0.012(4)	0.064(8)	0.032(7)	0.017(5)
Slashdot	**0.163(1)**	0.346(7)	0.269(5)	0.333(6)	0.241(4)	0.368(8)	0.204(3)	0.171(2)
Birds	0.181(2)	0.262(8)	0.217(5)	0.238(6)	0.215(4)	0.239(7)	0.190(3)	**0.180(1)**
Yeast	0.671(4)	1.583(8)	1.540(7)	0.900(6)	0.799(5)	0.585(2)	**0.566(1)**	0.603(3)
Flags	0.619(4)	0.918(8)	0.897(7)	0.716(6)	0.713(5)	**0.546(1)**	0.616(3)	0.559(2)
Emotions	0.640(4)	1.243(7)	1.245(8)	0.832(6)	0.815(5)	**0.603(1)**	0.630(3)	0.619(2)
Scene	0.492(3)	1.095(8)	1.014(7)	0.726(5)	0.733(6)	0.530(4)	0.482(2)	**0.475(1)**
Average ranks	2.30	6.46	5.19	6.07	4.88	5.46	3.38	**2.23**

Table 4.7: Results on 13 datasets with regards to micro F_1, macro f_l and macro f_e, the higher the value the better.

	BR	LP	CC	Homer	LPBR	ML-C4.5	**LaCova**	**LaCovaC**
	micro F_1							
Corel5k	0.149(2)	0.114(6)	0.148(3)	**0.154(1)**	0.038(7)	0.031(8)	0.141(4)	0.126(5)
Cal500	0.334(5)	0.332(6)	0.338(4)	0.328(7)	0.182(8)	**0.420(1)**	0.397(2)	0.341(3)
Bibtex	0.345(4)	0.285(6)	0.361(2)	0.272(7)	**0.391(1)**	0.195(8)	0.311(5)	0.351(3)
Language log	0.175(2)	0.122(7)	**0.177(1)**	0.158(3)	0.123(6)	0.106(8)	0.152(4)	0.142(5)
Enron	0.428(2)	0.364(7)	0.426(3)	0.399(4)	**0.442(1)**	0.313(8)	0.373(5)	0.366(6)
Medical	0.778(3)	0.744(6)	0.786(2)	0.726(7)	0.774(4)	0.355(8)	0.761(5)	**0.806(1)**
Genbase	0.921(4)	0.982(2)	**0.988(1)**	0.920(5)	0.853(7)	0.841(8)	0.893(6)	0.981(3)
Slashdot	0.506(2)	0.429(6)	**0.512(1)**	0.456(5)	0.381(8)	0.383(7)	0.468(4)	0.496(3)
Birds	**0.366(1)**	0.295(7)	0.312(6)	0.334(4)	0.281(8)	0.330(5)	0.336(3)	0.346(2)
Yeast	0.541(2)	0.522(8)	0.528(5)	0.531(3)	0.530(4)	0.525(6)	**0.549(1)**	0.523(7)
Flags	0.690(5)	0.677(7)	0.680(6)	0.696(4)	0.665(8)	**0.718(1)**	0.704(3)	0.706(2)
Emotions	0.539(2)	0.521(4)	0.499(8)	0.52(5)	**0.54(1)**	0.511(6.5)	0.526(3)	0.511(6.5)
Scene	**0.493(1)**	0.445(6)	0.487(2)	0.466(5)	0.429(7)	0.380(8)	0.473(3)	0.472(4)
Average ranks	**2.69**	6	3.38	4.61	5.38	6.34	3.69	3.88
	macro f_l							
Corel5k	**0.040(1)**	0.030(2)	0.027(4)	0.024(6)	0.012(8)	0.021(7)	0.029(3)	0.026(5)
Cal500	0.161(2)	0.147(6.5)	0.146(8)	0.156(3.5)	**0.168(1)**	0.147(6.5)	0.155(5)	0.156(3.5)
Bibtex	0.275(2)	0.193(6)	0.258(3)	0.147(7)	**0.280(1)**	0.118(8)	0.240(5)	0.248(4)
Language log	**0.068(1)**	0.039(6)	0.057(2)	0.050(4.5)	0.050(4.5)	0.038(7.5)	0.054(3)	0.038(7.5)
Enron	**0.132(1)**	0.090(7.5)	0.122(3)	0.123(2)	0.111(4)	0.098(5)	0.097(6)	0.090(7.5)
Medical	**0.342(1)**	0.311(6)	0.329(3)	0.304(7)	0.323(4)	0.172(8)	0.314(5)	0.334(2)
Genbase	0.518(4)	**0.556(1.5)**	**0.556(1.5)**	0.510(5)	0.504(6)	0.474(8)	0.492(7)	0.544(3)
Slashdot	**0.332(1)**	0.267(6)	0.315(2)	0.280(5)	0.235(8)	0.239(7)	0.294(4)	0.310(3)
Birds	**0.208(1)**	0.140(8)	0.143(7)	0.182(4)	0.164(5.5)	0.164(5.5)	0.19(2)	0.185(3)
Yeast	0.394(2)	0.372(7)	0.381(5)	0.389(3)	**0.406(1)**	0.359(8)	0.385(4)	0.379(6)
Flags	0.600(6)	0.601(5)	0.581(7)	0.616(3)	0.567(8)	**0.622(1)**	0.610(4)	0.620(2)
Emotions	**0.530(1)**	0.503(4)	0.488(8)	0.502(5)	0.527(2)	0.494(7)	0.511(3)	0.495(6)
Scene	0.244(2)	0.221(7)	**0.247(1)**	0.233(5)	0.223(6)	0.213(8)	0.236(3)	0.235(4)
Average ranks	**1.92**	5.57	4.19	4.61	4.53	6.65	4.15	4.34
	macro f_e							
Corel5k	0.140(2)	0.115(7)	0.121(6)	**0.143(1)**	0.032(8)	0.131(3)	0.127(4)	0.125(5)
Cal500	0.330(5)	0.327(6)	0.334(4)	0.323(7)	0.119(8)	**0.424(1)**	0.393(2)	0.335(3)
Bibtex	0.339(4)	0.292(6)	0.341(2)	0.266(7)	**0.369(1)**	0.216(8)	0.338(5)	0.340(3)
Language log	**0.133(1)**	0.104(6)	0.124(2)	0.115(4)	0.096(7)	0.094(8)	0.121(3)	0.106(5)
Enron	0.416(2.5)	0.368(5)	0.416(2.5)	0.392(4)	**0.434(1)**	0.297(8)	0.360(6)	0.342(7)
Medical	0.770(3)	0.752(6)	0.757(5)	0.735(7)	0.776(2)	0.389(8)	0.769(4)	**0.793(1)**
Genbase	0.931(8)	0.985(3)	**0.990(1)**	0.962(4)	0.961(5)	0.952(7)	0.960(6)	0.986(2)
Slashdot	**0.473(1)**	0.438(5)	0.454(4)	0.422(6)	0.408(7)	0.394(8)	0.471(2)	0.469(3)
Birds	**0.182(1)**	0.142(6)	0.138(7.5)	0.150(4)	0.138(7.5)	0.146(5)	0.172(2)	0.160(3)
Yeast	0.520(3)	0.495(8)	0.502(4)	0.500(6)	**0.526(1)**	0.501(5)	0.524(2)	0.498(7)
Flags	0.647(6)	0.636(7)	0.650(5)	0.655(4)	0.618(8)	**0.678(1)**	0.673(3)	0.674(2)
Emotions	0.500(2)	0.494(3)	0.469(7.5)	0.483(6)	**0.505(1)**	0.487(5)	0.488(4)	0.469(7.5)
Scene	0.446(4)	0.445(5)	**0.481(1)**	0.416(7)	0.441(6)	0.394(8)	0.460(2)	0.451(3)
Average ranks	**3.26**	5.61	3.96	5.15	4.80	5.76	3.46	3.96

CHAPTER 5

THRESHOLD SELECTION METHODS FOR MULTI-LABEL CLASSIFICATION

This chapter investigates existing threshold selection methods for both binary and multi-label classification. It explores the selection of single and multiple thresholds and extends some of the current techniques to support multi-label problems. Moreover, it proposes cost curves and scatter diagrams for performance evaluation in the multi-label setting.

The remainder of this chapter is organised as follows. Section 5.1 introduces the problem and draws attention to the motivation for this work. Section 5.2 lists additional notations used in this chapter. Sections 5.3 and 5.4 present both binary and multi-label classification thresholding methods, respectively. Section 5.6 analyses the performance of different thresholding approaches. Finally, Section 5.7 concludes the chapter. Some of the work and results described in this chapter are published in [22]. The code is available in an online repository [15].

5.1 Introduction

In multi-label learning, each instance is associated with a set of labels simultaneously. It involves two main tasks: partitioning labels into relevant and irrelevant sets and/or ranking labels from the most relevant to the least. A number of learning algorithms, such as naive Bayes, logistic regression and decision trees, produce a confidence score for each label, which needs to be classified as relevant or irrelevant by choice of thresholding [158, 159].

There exist various solutions to tune the thresholds in multi-label learning, which can be grouped into three categories: a threshold per dataset (globally), one threshold per label (label-wise), or one threshold per instance (instance-wise). However, there are only a few empirical studies that compare different thresholding approaches for multi-label classification.

Fan and Lin [160] published a technical report that studies different multi-label threshold selection methods. Experimental results on real-world datasets demonstrate that tuning a separate threshold per label with respect to a loss function is simply better than a fixed threshold, 0.5 is an example. Tang et al. [161] conducted an extensive empirical study on different thresholding approaches and proposed a MetaLabeler method. This method determines the number of relevant labels for each test instance based on the training data.

Cost-sensitive learning is an important research area in many real-world applications. Making better decisions depend not only on minimising the expected errors, or the empirical errors on the training data but also on minimising the total cost associated with those decisions. In many real-world applications, misclassification costs can be obtained from domain experts. However, the misclassification costs are generally unknown at the training context [162, 163]. In this chapter, we want the models to perform well in a wide range of misclassification costs, from high to low costs [164].

While Chapter 3 addressed the problem of context change between training and deployment in single-label classification, it focused on a specific type of context change, which is dataset shift. In this chapter, we introduce another type of context change, which is changes in misclassification cost across contexts. We explore different thresholding methods in the multi-label setting under various deployment contexts. We consider two possible scenarios: the selection of global and multiple thresholds. We examine a classifier's performance considering two situations: shared cost among labels and per-label cost.

Throughout this chapter, we identify some open questions: which thresholding method to choose for multi-label classification, how these approaches relate to each other and how would the classifier perform when a global threshold is used instead of a per-label threshold. We summarise the motivations behind this work as follows.

Firstly, thresholding is an important and relevant topic in multi-label learning, as most applications require bipartitions. Several thresholding methods have been introduced in the literature. Some of these methods are extensions of single-label methods, while others are specifically designed for multi-label classifiers. Therefore, it would be interesting to explore possible thresholding methods, namely, per-dataset, per-label and per-instance and how they relate to each other.

Secondly, most of the multi-label classifiers are meta-classifiers that decompose such problems into several binary problems. Hence, any single-label classifier can be used as a base classifier. In some thresholding methods, it is important to produce well-calibrated probability estimates. Predicted probabilities are called calibrated if they can be interpreted as a confidence level. For example, given a set of test instances with 0.6 scores, we can expect that 60% of these instances have a negative class ($-$), assuming higher scores indicate higher confidence towards negative class [165, 166].

Finally, in the context of cost-sensitive learning, current research has focused on binary or multi-class cost-sensitive classification, but not on multi-label classification [105]. A multi-label model can be learnt in a context with specific label costs but deployed in another context, where these costs have been changed. For example, in medical diagnosis, some illness (labels) have misclassification costs higher

than others due to the popularity and spreading of this condition in some countries. On the other hand, in the deployment context, the same conditions might be rare and have very low costs.

5.2 Notations

In addition to the previous notations listed in Chapter 2, we define the following notations. In binary classification, p and $(1-p)$ denote class ratios for positive $(+)$ and negative $(-)$ classes, respectively. The misclassification costs for positive and negative classes are ct_+ and ct_-. Whereas the total cost is $o = ct_+ + ct_-$. Thus, the relative cost is $ct = \frac{ct_+}{o}$. For simplicity, hereafter we refer to the relative misclassification cost ct simply as the misclassification cost, for short. Both ct_+ and ct_- can be written in terms of ct as below.

$$ct_+ = o \cdot ct$$
$$ct_- = o \cdot (1-ct)$$

By assuming $o = 2$ for all labels as proposed in [164, 167], a cost-based loss function for binary classification is defined as follows.

$$\begin{aligned} G(thr;ct) &\triangleq ct_+ \cdot p \cdot fnr(thr) + ct_- \cdot (1-p) \cdot fpr(thr) \\ &= o\{ct \cdot p \cdot fnr(thr) + (1-ct) \cdot (1-p) \cdot fpr(thr)\} \\ &= 2\{ct \cdot p \cdot fnr(thr) + (1-ct) \cdot (1-p) \cdot fpr(thr)\} \end{aligned} \tag{5.1}$$

where fnr and fpr represent false negative and false positive rates at threshold thr, respectively.

Given a training data D_{tr} of size n_{tr} and a deployment data D_{dep} of size n_{dep}, we denote by $card(D_{tr})$ the average number of labels per instance in D_{tr}. A multi-label model is a function $\mathscr{X} \to \mathfrak{R}^q$ that maps instances to vectors of scores. A multi-label classifier is a function that maps instance $\mathbf{x}$ from $\mathscr{X}$ to labels from $\mathscr{Y}$ using a threshold thr on the scores.

$$h_{mlc} : \mathscr{X} \to 2^{\mathscr{Y}}$$

Given a probabilistic confidence value $pr \in [0,1]$, the instance $\mathbf{x}$ is classified as negative $(-)$ class if $pr > thr$, and positive $(+)$ class otherwise. To be clear, we assume that higher scores (close to 1) indicate a higher confidence that instance $\mathbf{x}$ is of negative class. In the multi-label setting, the threshold can be assigned per label, resulting in a vector of q thresholds $thr = (thr_1, thr_2, \cdots, thr_q)$, or the number of thresholds can be equal to the number of test instances $thr = (thr_1, thr_2, \cdots, thr_{n_{dep}})$.

Multi-label cost-sensitive classification extends multi-label classification by considering the cost for each label. Hence, the loss in multi-label classification is defined as the average over all labels as follows.

$$\text{Multi-label loss function} \triangleq \frac{1}{q}\sum_{j=1}^{q} G_j(thr_j; ct_j)$$

$$= \frac{1}{q}\sum_{j=1}^{q} 2\{ct_j \cdot p_j \cdot fnr_j(thr_j) + (1-ct_j)\cdot(1-p_j)\cdot fpr_j(thr_j)\} \tag{5.2}$$

where G_j is the loss for the j^{th} label at threshold thr_j. q is the number of labels.

5.3 Thresholding Approaches for Binary Classification

This section presents the most common techniques used to adjust the thresholds in binary classification. Hernández-Orallo et al. [164] introduced different approaches, namely, fixed, score-driven, rate-driven and the optimal. In this thesis, we call the score-driven threshold choice method as the cost-driven method as it sets the threshold equal to the cost. A thresholding method is defined as a function T that takes misclassification cost ct and returns a threshold thr [164, 167]. We use different superscripts to identify particular thresholding methods as below.

Fixed: The simplest way is to set the threshold to a fixed value t, which is independent of the model and the deployment context [164]. The traditional technique is to simply use 0.5 as a default cut-off [3, 168].

Definition 5.1. The fixed threshold choice method is defined as follows:

$$T^{fixed}(ct) \triangleq thr \tag{5.3}$$

where ct is the misclassification cost and thr is any fixed value threshold.

Cost-driven: This approach takes into account the deployment context by adjusting the threshold to the cost [164]. For example, if the cost at deployment context is $ct = 0.4$, the cost-driven method adjusts the threshold to 0.4.

Definition 5.2. The cost-driven (CD) threshold choice method is defined as below:

$$T^{CD}(ct) \triangleq ct \tag{5.4}$$

where ct is the misclassification cost for the "+" class in the deployment context.

Rate-driven: It sets the threshold such that it achieves a desired predictive positive rate equal to the cost ct [164]. For instance, if the cost is $ct = 0.7$ at deployment time, this method tunes the threshold in such a way that 0.7 positive predictive rate is achieved. The predictive positive rate is defined as the proportions of instances predicted as positives and is computed on the training set.

Definition 5.3. The rate-driven (RD) threshold choice method is defined as follows:

$$T^{RD}(ct) \stackrel{\Delta}{=} PR^{-1}(ct) \tag{5.5}$$

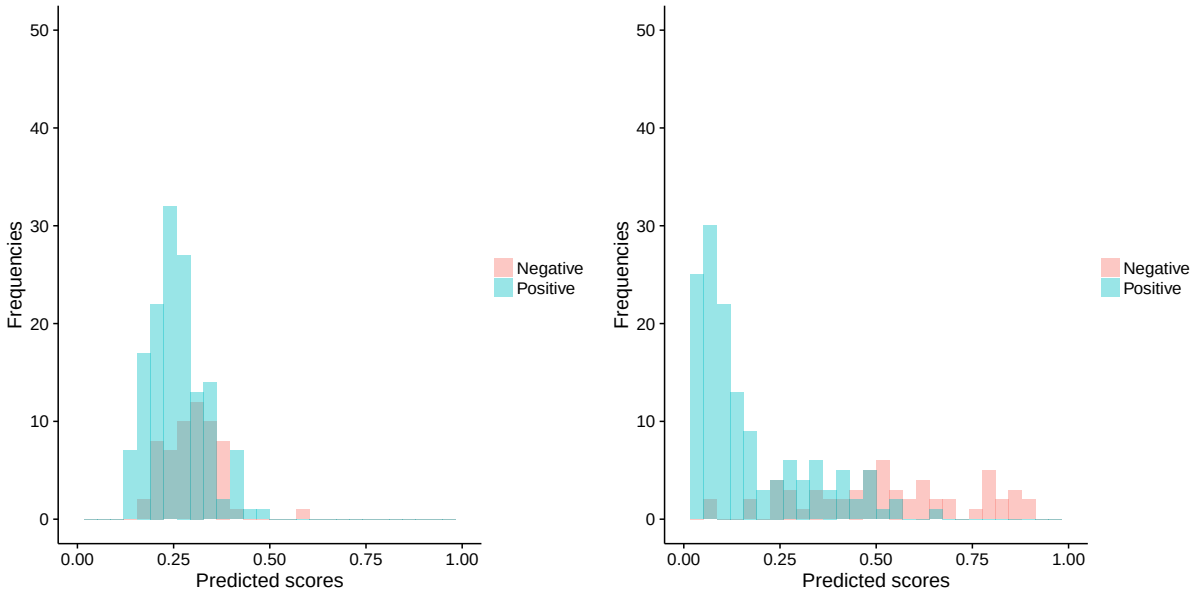
where PR is the predicted positive rate computed on the training set at threshold thr and $PR^{-1}(ct)$ is the threshold that achieves predictive positive rate ct. PR is assumed to be invertible.

Optimal: This method finds the threshold that minimises a specific loss function G. The optimal threshold can be selected empirically using a training or validation set [164].

Definition 5.4. The optimal (opt) threshold choice method for loss function G is defined as follows:

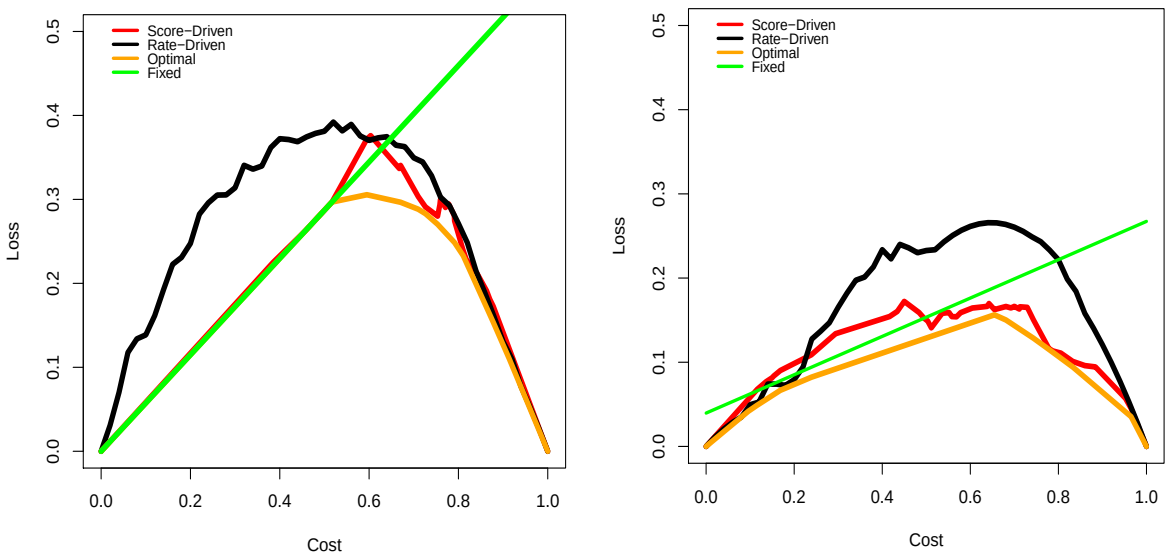
$$T^{opt}(ct) \stackrel{\Delta}{=} \underset{thr}{\arg\min}\ G(thr;ct) \tag{5.6}$$

We use cost curve for performance visualisation to understand the differences among those types of thresholding. It is a powerful tool that is used widely in binary classification to evaluate the learning performance. It draws the loss on the y-axis against misclassification costs on the x-axis [169]. As an illustrative example, we retrieved the Emotions dataset and selected two labels: `Happy-pleased` and `Quiet-still` [79]. Then, a logistic regression was trained on a training set and evaluated against a test set for each label separately. Figure 5.1 shows the score histograms for both labels, which are in the [0,1] interval. Figure 5.2 depicts the cost curves for four different thresholding approaches assuming uniform misclassification costs at deployment.



(a) `Happy-pleased` (b) `Quiet-still`

Figure 5.1: Histograms for the scores obtained from a logistic regression algorithm.



(a) `Happy-pleased` (b) `Quiet-still`

Figure 5.2: Cost curves measure the performance of four different thresholding methods used in binary classification assuming uniform misclassification costs. Scores are obtained from logistic regression.

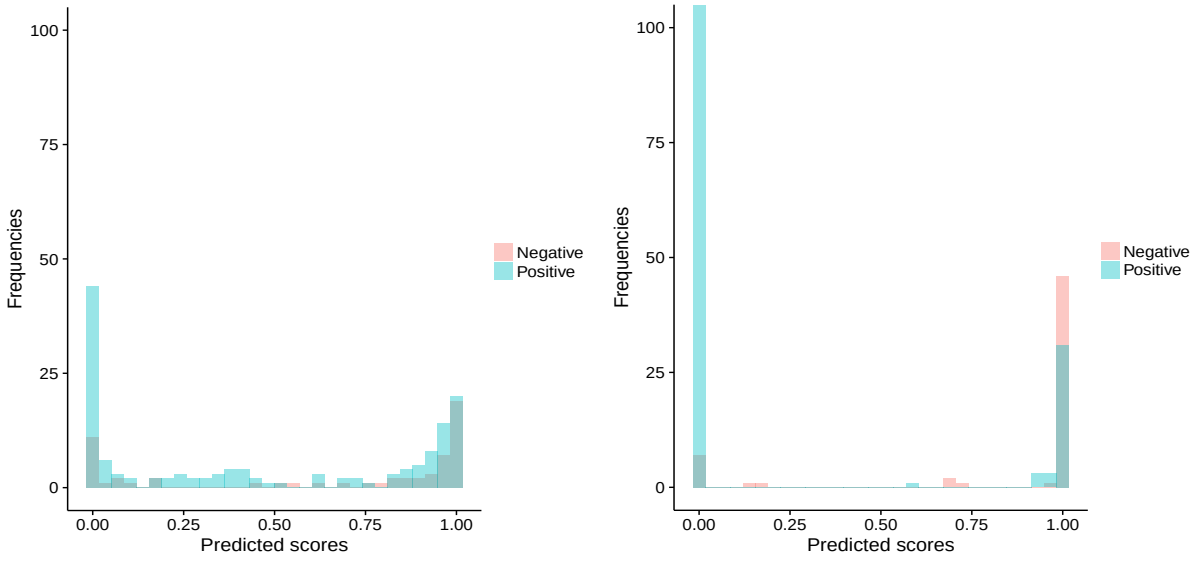
We observe that using a fixed threshold (the green lines), which produces a constant rate of false positives and negatives over a range of misclassification costs, there is a linear relationship between cost and loss. Note that the line goes up on leading to the highest loss when the cost is 1. This is because of the imbalance in the number of false positives and false negatives. In this case, having lower cost leads to lower loss as the false positives are weighted more heavily as seen in Equation 5.1. When the false positives and false negatives are uniformly distributed, the line of the fixed threshold method would be horizontal.

Additionally, the curves of the rate-driven method, which tries to achieve a predictive positive rate equal to the cost, are in black colour. Whereas the red curves that represent the cost-driven method are below these black curves. It shows that the cost-driven method achieves lower loss when we compare the area under these curves. Finally, the optimal threshold choice method is represented by the lower envelop of the model's cost lines [167], as seen in the orange curves. Note that the optimal threshold in these two plots is acquired from deployment data. In reality, the optimal threshold should be selected based on the training or validation set, as the deployment context is unknown at training time. Cost-driven and the optimal threshold choice approaches are equal when the model is well-calibrated [164].

We repeat the experiment but using another base classifier, which is the Naive Bayes algorithm. Figures 5.3 and 5.4 show the score histograms for `Happy-pleased` and `Quiet-still`, and the cost curves for four different thresholding approaches, respectively. The first observation is that the green lines that represent the fixed threshold go down leading to the lowest loss when the cost is 1. The black curves of the rate-driven method perform generally better than the red cost-driven curves. Next the red curve on Figure 5.4(a), which represents the cost-driven method, is different from 5.4(b). This is due

to the score variations resulting from the learning algorithm. To be clear, scores obtained on the right side are extremely 1 or 0 as seen in Figure 5.3(b).

We also examine the predictions made by the two models as shown in Figures 5.1 and 5.3. Note that for Figure 5.1(a) most of the scores are around the centre with few of them reaching 0 or 1. While the Naive Bayes scores are extremely 1 or 0 as seen in Figure 5.3(a). It is well known in the literature that the logistic regression algorithm returns well-calibrated scores [170], whereas the Naive Bayes classifier is a poorly calibrated classifier because of the independence assumption [166].



(a) `Happy-pleased` (b) `Quiet-still`

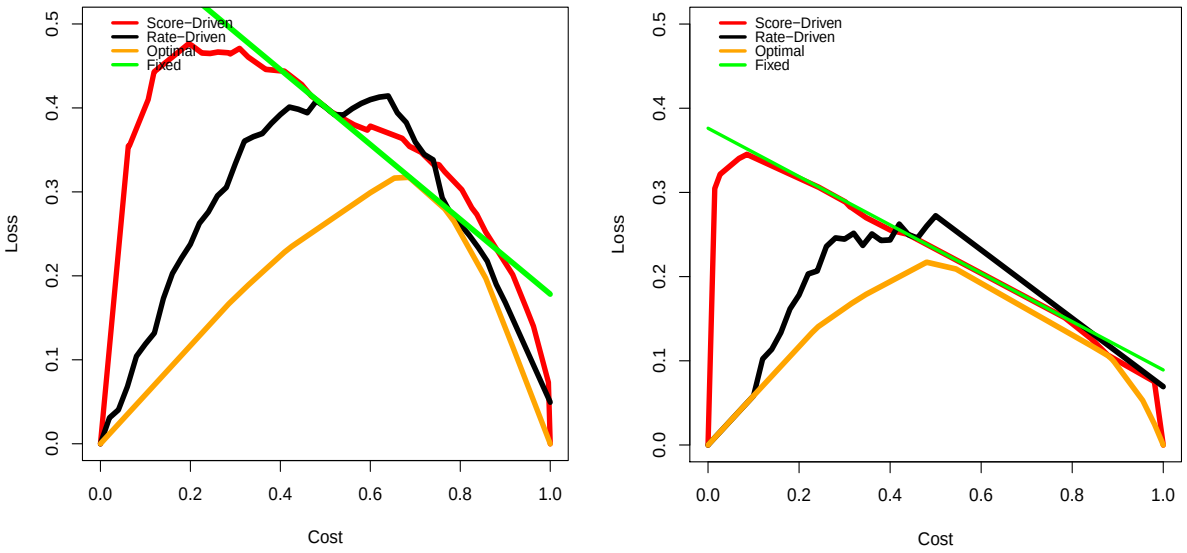
Figure 5.3: Histograms for the scores obtained from Naive Bayes algorithm.

5.4 Thresholding Approaches for Multi-label Classification

All the aforementioned methods can be applied directly to multi-label problems by treating each label separately, which leads to several binary problems, one for each label. Multi-label thresholding methods can be grouped into global and multiple thresholds. A global threshold means that only a single threshold is shared by all instances and labels (a threshold per dataset). Whereas the number of multiple thresholds can be equal to the number of labels (label-wise) or the number of instances (instance-wise). We further distinguish between score-based and rank-based approaches. In the former, they rely on label scores, whereas in the latter, they rank instances or labels in descending order of their scores.

5.4.1 Label-wise Thresholding

It computes a different threshold for each label. Given a test instance, it predicts a negative class for each label that has a score higher than its threshold and positive otherwise. They can be grouped into score-based and rank-based methods.



(a) `Happy-pleased` (b) `Quiet-still`

Figure 5.4: Cost curves measure the performance of four different thresholding methods used in binary classification assuming uniform misclassification costs. Scores are obtained from Naive Bayes.

Score-based Methods: They adjust a different threshold for each label independently based on the scores produced by the learning algorithm. There are several label-wise score-based thresholding methods as mentioned below.

Fixed: It tunes a fixed threshold for each label, which can be equal or different across labels. For example, using 0.5 as a default threshold for all labels. On the other hand, we can assign q different fixed thresholds, one for each label. In both cases, this method ignores the labels' misclassification costs.

Definition 5.5. The label-wise fixed threshold choice method is defined as:

$$T_j^{Fixed}(ct_j) \triangleq thr_j \tag{5.7}$$

where ct_j and thr_j denote the misclassification cost and the fixed threshold for the j^{th} label, respectively.

Cost-driven: This approach is used in binary classification as we explained in Section 5.3. However, no multi-label version of this method was found in the literature. Therefore, we define the label-wise cost-driven as follows.

Definition 5.6. The label-wise cost-driven threshold choice method is defined as:

$$T_j^{LCD}(ct_j) \triangleq ct_j \tag{5.8}$$

where ct_j is the misclassification cost for the j^{th} label, $0 \leq ct_j \leq 1$.

Optimal: The optimal threshold strategy is also known as the Score-based Cut (SCut) [171]. It adjusts the threshold for each label in such a way that it minimises a specific loss function using a validation set or cross-validation. It does not allow the user to specify the parameter to control the number of instances assigned to each label.

Definition 5.7. Given a label l_j and a loss function G, the label-wise optimal threshold choice method is defined as:

$$T_j^{LOpt}(ct_j) \triangleq \underset{thr_j}{\arg\min}\ G_j(thr_j; ct_j) \tag{5.9}$$

where ct_j and thr_j denote the misclassification cost and the selected threshold for the j^{th} label, respectively.

Rank-based Methods: For each label, label-wise rank-based methods rank test instances in decreasing order of their label scores. Then, they adjust a different threshold for each label as below.

Rate-driven: Label-wise rate-driven (LRD) sets a threshold for each label that guarantees that the predicted positive rate for each label equals to the label's positive rate in the training set as seen in Figure 5.5. It is also known as the Proportion Cut (PCut).

Definition 5.8. Given a label l_j, the label-wise rate-driven threshold choice method is defined as:

$$T_j^{LRD}(ct_j) \triangleq PR_j^{-1}(ct_j) \tag{5.10}$$

where PR_j is the predicted positive rate for the j^{th} label at threshold thr_j and $PR_j^{-1}(ct_j)$ is the threshold that achieves predictive positive rate ct_j. PR_j is assumed to be invertible.

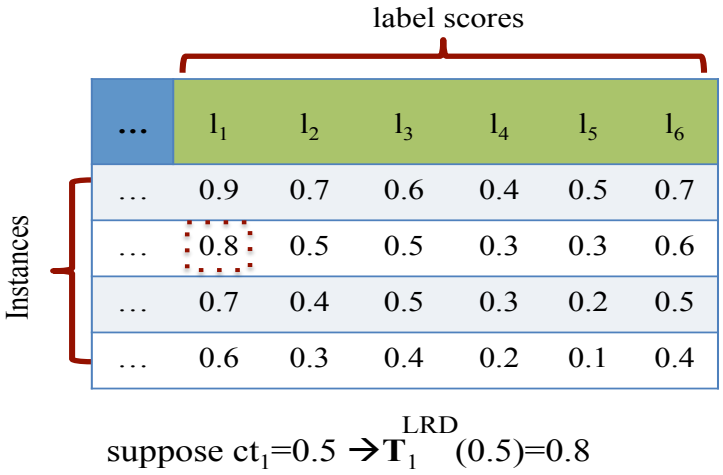


Figure 5.5: Suppose $ct_1 = 0.5$ and $n_{dep} = 4$, LRD assigns 0.8 threshold for this label as it achieves 0.5 predictive positive rate, which means that instances with scores higher than or equal to 0.8 will have this label predicted as negative, and predicted as positive otherwise.

5.4.2 Instance-wise Thresholding

It estimates a different threshold for each test instance. Then, it predicts a number of labels based on the assigned threshold; this is variable from instance to instance. Next, we discuss several instance-wise thresholding techniques that are basically considered as rank-based methods.

Rank-based Methods: Instance-wise rank-based techniques sort labels based on the scores for each test instance. Based on the ranking, each instance is assigned a set of labels using any of the following procedures.

RCut: The Rank Cut (RCut) is an instance-wise thresholding strategy, which outputs the K labels with the lowest scores for each test instance. The parameter K can be either a user-defined parameter or automatically determined using a validation set or a cross-validation. In the former, K can be any value between 1 and q, however, a sensible setting is *card*, which represents the average label set cardinality (number of labels) per instance in the training data. When $K = 1$, only the most relevant label is assigned to the test instance, while $K = q$ means that all labels are predicted as relevant for the test instance.

Definition 5.9. Given an instance $\mathbf{x}_i$ and $\mathbf{pr}_i = \{pr_i^1, pr_i^2, \ldots, pr_i^r\}$ is the sorted probabilistic labels' confidence score list for this instance. pr_i^1 and pr_i^r represent the lowest and highest values in $\mathbf{pr}_i$, respectively. The instance-wise RCut threshold choice method is defined as:

$$T_i^{RCut} \triangleq pr_i^K \tag{5.11}$$

where K is the number of labels to be predicted as positive.

It is important to emphasise that even though K is a fixed parameter, the thresholds obtained using RCut are not fixed as seen in Figure 5.6. Each instance will have its own threshold that guarantees K labels to be retrieved as relevant.

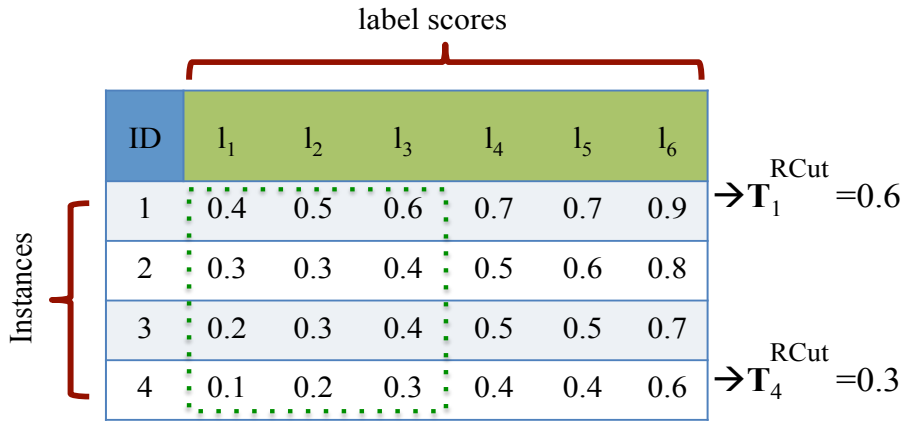


Figure 5.6: Assume $K = 3$, RCut assigns only the first three labels for all instances. Note that each instance has its own threshold.

MCut: The Maximum Cut (MCut) automatically determines a threshold for each instance. Given the sorted probabilistic labels' confidence score list for each instance, it selects two consecutive labels with the highest difference in terms of their scores among all pairs of consecutive labels. This leads to the selection of the middle of the interval defined by these two label scores as the threshold [172].

Definition 5.10. Given an instance $\mathbf{x}_i$ and $\mathbf{pr}_i = \{pr_i^1, pr_i^2, \ldots, pr_i^r\}$ is the sorted probabilistic labels' confidence score list for this instance. pr_i^1 and pr_i^r represent the lowest and highest values in $\mathbf{pr}_i$, respectively. The instance-wise MCut threshold choice method is defined as:

$$T_i^{MCut} \triangleq \frac{pr_i^{ind} + pr_i^{(ind+1)}}{2}, \quad ind = \arg\max_j \{(pr_i^j - pr_i^{(j+1)}), j = 1, \ldots, q-1\} \tag{5.12}$$

where pr_i^j and $pr_i^{(j+1)}$ represent the probabilistic confidence values for two consecutive labels considering the i^{th} instance.

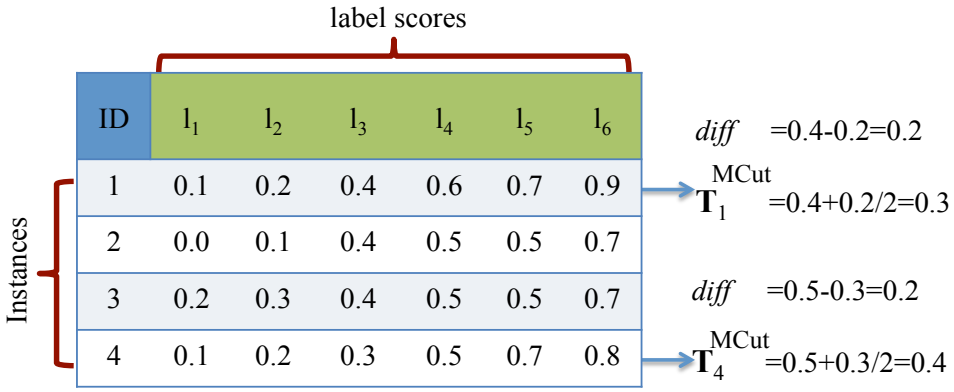


Figure 5.7: MCut tunes the threshold per instance. For example, the first instance has a maximum difference $diff$ between scores equals to 0.2, the threshold for this instance will be $\frac{0.4+0.2}{2} = 0.3$.

In case of tie, which means two pairs of consecutive scores have the same maximum difference, MCut selects the scores that occur first in the rank.

5.4.3 Global Thresholding

It considers the selection of a single threshold per dataset. A global threshold that is shared by all labels and instances is used to convert the scores into predictions. In a multi-label setting, we consider two situations with respect to the misclassification cost: shared cost and per label cost. When labels share the misclassification cost, the global threshold would perform equally well compared with per label thresholds such as the cost-driven and the optimal techniques. However, when there is a different cost for each label and we aim to set up a global threshold, it is possible to use the average misclassification cost to represent the deployment context. In other words, the average misclassification cost will be used as the cost for all labels at the deployment, as follows. Another possibility is to select one label and use its cost for other labels. Next, we propose two global cost-driven approaches.

Score-based Methods: They tune a single threshold for each dataset, which is then shared by all labels and instances in that dataset. The previous label-wise score-based strategies can also be used globally as follows.

Cost-driven average: Given a different cost for each label, we use the average label cost to tune a single cost-driven threshold. The global cost-driven average is defined as follows.

Definition 5.11. The global cost-driven average threshold choice method is defined as:

$$T^{CDA}(ct_j) \triangleq \overline{ct} \tag{5.13}$$

where ct_j is the misclassification cost for the j^{th} label and $\overline{ct}$ is the average misclassification cost over all labels.

Cost-driven uniform: Given a different cost for each label, a representative label can be randomly selected (uniform distribution) and its cost will be used for all other labels at the deployment as follows.

Definition 5.12. The global cost-driven uniform threshold choice method is defined as:

$$T^{CDU}(ct_j) \triangleq ct_k \tag{5.14}$$

where ct_j is the misclassification cost for the j^{th} label and ct_k is the misclassification cost for the k^{th} label that has been selected randomly.

The global versions of the fixed and the optimal threshold are similar to binary classification. They assign one global threshold by combining the output of all binary classifiers in one bin as in a binary classification. In the context of hierarchical cost-sensitive ensemble methods, Cesa-Bianchi et al. [173] introduced the use of a single (global) cost-sensitive parameter, which replaces the threshold 0.5 using F_1 score.

Rank-based Methods: In label-wise rank-based methods, the label scores in each test instance are sorted by the scores for each label. Here, the global rank-based methods combine all label scores for all instances in the dataset in one list and then do the ranking.

Global Rate-driven: The global rate-driven, known as PCut1, uses a single global threshold for all labels, chosen such that the average label cardinality *card* in deployment data is as close as possible to the training data [9]. Figure 5.8 gives an example of how this method works, assuming $card = 3$ in the training set.

Definition 5.13. A classifier h will make predictions for the deployment data D_{dep} using a threshold thr based on global rate-driven as follows:

$$T^{RD}(ct_j) \triangleq \arg\min_{thr} |card(D_{tr}) - card(h(D_{dep}))| \tag{5.15}$$

where $h(D_{dep})$ is the predicted scores at deployment. $card(D_{tr})$ and $card(h(D_{dep})$ represent the average number of labels for each instance in the training and deployment data, respectively.

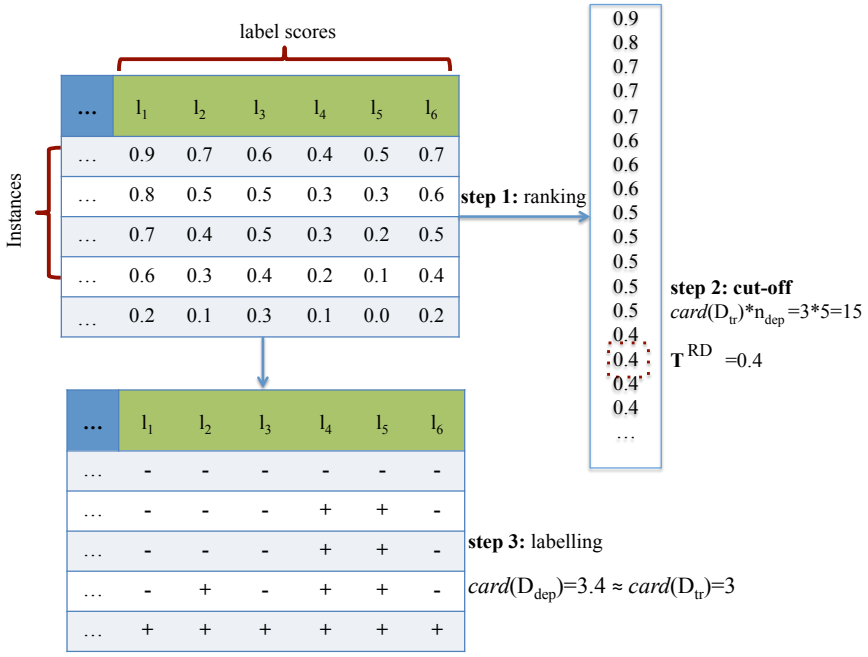


Figure 5.8: The global rate-driven is a context-independent method. Assuming $card = 3$ in the training set D_{tr}, it selects only a single threshold that approximates the label cardinality across contexts. The label cardinality at deployment is 3.4, which is the average number of labels per instance in D_{dep}.

5.4.4 Discussion

In summary, some of the aforementioned thresholding methods use the information about the deployment context to adjust the threshold, whereas other methods do not. For example, the global rate-driven method, which minimises the difference in label cardinality between training and deployment data, tunes the threshold independently from the deployment context. Likewise, RCut and MCut do not consider the deployment context when assigning the thresholds. In this respect, all these methods can be seen as similar to the fixed threshold that uses constant threshold across contexts.

Label-wise rate-driven is the only method that uses the prior relative frequencies of the labels observed in the trained model. It can provide good results if the labels' relative frequencies remain the same in the training and deployment. Label-wise versions of rate-driven and the optimal thresholding techniques compare instances scores by fixing the label, while RCut and MCut compare the label scores by fixing the example. MCut can assign different numbers of labels to different instances, whereas RCut assigns the same number of labels for all instances. RCut requires a user-specified parameter, but the others do not need any parameterisation. The rate-driven and optimal methods of binary classification are similar to the rate-driven and the optimal methods of multi-label classification, respectively.

We establish the relationship between multi-label classification and binary classification threshold choice methods as shown in Table 5.1.

Table 5.1: The link between multi-label and binary classification thresholding methods. q is the number of labels and n_{dep} is the number of deployment instances.

The approach	Applied level	#Thresholds	Cost-based	Link to binary classification
Fixed	Globally	1	No	Fixed
	Label-wise	q		
CDA	Globally	1	Yes	Cost-driven
LCD	Label-wise	q	Yes	
CDU	Globally	1	No	-
Opt	Globally	1	Yes	Optimal
LOpt	Label-wise	q	Yes	
RD	Globally	1	No	-
LRD	Label-wise	q	Yes	Rate-driven
RCut	Instance-wise	n_{dep}	No	-
MCut	Instance-wise	n_{dep}	No	-

5.5 Threshold Choice Methods and Expected Loss

The goal in classification is to find a threshold and classify the data in such a way that minimises the expected or the empirical classification error. Hernández-Orallo et al. [164] studied the relationship between binary classification threshold choice methods and their expected loss over a wide range of misclassification costs. They established a connection between threshold choice methods and their expected losses in terms of evaluation measures.

In a multi-label context, treating each label separately leads to multiple binary classification problems, therefore, the results obtained in Hernández-Orallo et al. [164] can also be generalised to multi-label classification as follows. The first approach is to set a fixed threshold for each label independently from the deployment context, which leads to linear cost curves [164]. The expected loss is the average of the per label error rate, which is known in multi-label classification literature as the Hamming loss. Additionally, uniform random thresholding ignores the deployment context and tunes different random thresholds for each label [164]. The expected multi-label loss is equal to the Mean Absolute Error (MAE) averaged over all labels.

Alternatively, some threshold choice methods take the deployment context into account. Firstly, if we set a threshold for each label that equals to its misclassification cost as proposed in the score-driven threshold choice method, then the per-label cost curve is a Brier curve [164, 174]. The expected loss is equal to the Brier score averaged over all labels. Secondly, the rate-driven threshold choice method considers misclassification cost for each label as the desired predictive positive rate and sets per-label thresholds in such a way that it achieves this rate. The expected loss for each label is linearly related to the Area Under the ROC Curve (AUC) [164].

The score-driven uniform technique that we proposed for the multi-label context is a combination of score-driven and uniform random threshold choice methods. Therefore, the expected loss is expressed in terms of both Brier score and MAE. The expected loss for the selected label will be equal to Brier score, whereas for other labels it will be equal to the MAE.

5.6 Experimental Evaluation

13 multi-label datasets, which were described in Section 2.4.9 of Chapter 2, have been used in our experiments to clarify the differences among different thresholding techniques. We used the train/test splits that are provided in Meka[1] and Mulan[2] repositories [79]. None of these repositories provide train/test splits for three datasets, namely, Cal500, Language log and Slashdot. We instead used the random splits (75% train, 25% test) that were made by the KDIS research group [3]. Table 5.2 provides some information about these datasets. It includes the number of labels and the size of the train/test splits. Additionally, the last two columns show the average number of labels and label density in the training context, respectively.

Table 5.2: The statistics of the datasets used in the experiments. q is the number of labels. n_{tr} and n_{dep} are the number of training and deployment instances, respectively. $card(D_{tr})$ and $dens(D_{tr})$ are label cardinality and density at the training context, respectively.

	q	n_{tr}	n_{dep}	$card(D_{tr})$	$dens(D_{tr})$
Corel5k	374	4500	500	3.521	0.009
Cal500	174	376	125	26.045	0.150
Bibtex	159	4880	2515	2.380	0.015
Language log	75	1095	365	1.169	0.015
Enron	53	1123	579	3.386	0.064
Medical	45	645	333	1.240	0.027
Genbase	27	463	199	1.261	0.046
Slashdot	22	2837	945	1.184	0.054
Birds	19	322	323	1.059	0.056
Yeast	14	1500	917	4.228	0.302
Flags	7	129	65	3.418	0.488
Emotions	6	391	202	1.813	0.302
Scene	6	1211	1196	1.061	0.176

Consider two situations, training and deployment contexts, a logistic regression was trained using the binary relevance approach, on the training set and evaluated against a test set to produce the scores. We used Weka logistic regression implemented as Logistic, which uses ridge regularisation, and we set the ridge parameter $r = 100$. We have varied label costs as we will describe later in Section 5.6.1 and computed the empirical loss that associated with each approach.

[1] http://meka.sourceforge.net/

[2] http://mulan.sourceforge.net/

[3] http://www.uco.es/grupos/kdis/kdiswiki/index.php/Resources

We have used cost curve analysis as in binary classification when labels share the misclassification cost. In addition, we introduce the use of scatter diagrams to determine the loss associated using variable misclassification costs, in addition to cost curves for the shared cost.

5.6.1 Experimental Setup

In cost-sensitive learning, when the targets are multiple as in multi-label data, we consider two settings with regards to misclassification costs, as follows: 1) assigning the same cost for all labels (shared cost) or 2) assigning a different cost for each label (per-label cost). Next, we discuss these two situations in more details.

Shared misclassification cost: When labels have shared cost ct, each threshold choice method has a different setting as below.

- **Global threshold:**
  - **Fixed** we set a single threshold 0.5 for all labels, which means that the fixed threshold does not consider the cost changes between training and deployment.
  - **Cost-driven average** the threshold will be equal to $\overline{ct}$, which also leads to a single threshold for all labels because the cost is identical.
  - **Optimal** finds a single threshold by putting all labels in one bin and treating them as in binary classification.
  - **Rate-driven** assigns a single threshold that achieves the best approximation of $card_{tr}$ in deployment such that $card(D_{dep}) = card(D_{tr})$.
- **Label-wise thresholds:**
  - **Rate-driven** adjusts a threshold for each label leading to q thresholds. Although the costs for all labels are equal, the thresholds will be different.
  - **Optimal** selects a threshold for each label that optimises the loss for this label.
- **Instance-wise thresholds:**
  - **RCut** assigns a different threshold for each instance at deployment. We set the cut K equal to $card(D_{tr})$, hence, this also leads to different thresholds for each instance.
  - **MCut** also assigns a different threshold for each instance at deployment, however, it does not need any parameterisation.

Variable misclassification cost: Labels' costs can be different $(ct_1, \cdots, ct_q)$, which means some labels have higher costs than others. We repeat the aforementioned settings of the shared cost scenario, but the difference is in some of the global methods. More specifically, in the cost-driven average approach, the threshold is equal to the average label cost $\overline{ct}$. In addition, the global optimal technique uses $\overline{ct}$ to finds the threshold that optimises the loss function. With regards to the cost-driven uniform (CDU) method, we select one label randomly and use its misclassification cost as a threshold for all labels.

5.6.2 Results and Discussion

Figure 5.9 compares the cost curves of the global methods in two cases: shared and per-label costs. In all cost curves shown in this Section, the loss function is the one defined in Equation 5.2. The global method assigns only a single threshold for all labels at deployment. The linear relationship between the losses and costs are represented by the black and green lines. The black line provides the loss associated with the global rate-driven, which assigns one threshold that achieves $card(D_{dep}) = card(D_{tr})$ independently from the context.

The global rate-driven is also called fixed, but it is fixed to a calibrated number. For example, the green line in B is a horizontal line, which means that the false positive and false negative rates are very close. Whereas in the other sub-figures, it goes up leading to the highest loss when the cost is 1. The cost-driven average curves shown in red are based on the scores. In C for example, the cost-driven average and the optimal curves are equivalent because the scores are uniformly distributed.

In most cases, by setting the threshold to the average cost, the scatter plots show clouds somehow average over the curve, and therefore, end up in the middle of the curve. We notice that some points have the same average but different loss due to the change in the average of false positive and false negative rates.

Figure 5.10 shows cost curves for the cost-driven uniform (CDU), which selects one label randomly and uses its cost as a threshold for other labels. The x-axis represents the selected label cost and the y-axis is the loss averaged over all labels. The pink curve is the cost-driven average curve as all labels have the same cost. While the pink scatter shows the case when each label has its own cost but its threshold is randomly decided. Meaning that only the selected label will have a threshold equal to its cost. The pattern in the scatter plots is that if the selected cost is very high or close to 1 and the per-label cost is very low, then the loss is also very high. For some labels, the per-label loss is higher than 1. Note that in Figure 5.10(A) for example, the scatter curve is not symmetric. This is actually justified by the imbalance of positive/negative values for the majority of class labels.

Figure 5.11 shows the cost curves obtained by applying multiple threshold methods, label-wise and instance-wise. Regarding both the purple and the blue lines, specified as MCut and RCut, respectively, there is also a linear relationship between loss and cost. They are also called fixed threshold methods because they choose thresholds independently of the deployment context. Black curves, which represent the label-wise rate-driven method, are clearly shown to have the highest loss in general. They are clearly affected by the imbalance of positive/negative values for the class labels.

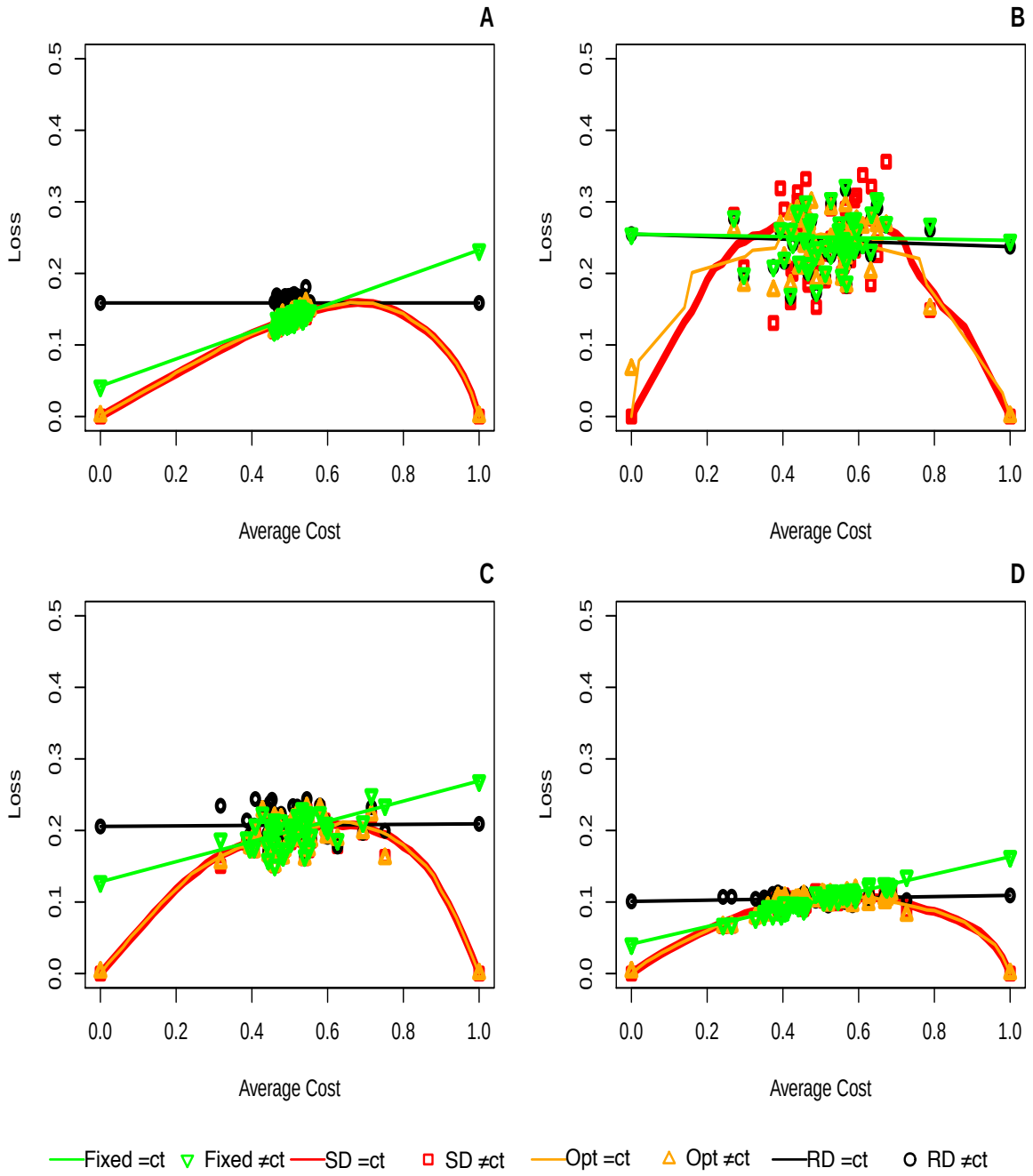


Figure 5.9: Cost curves for global threshold methods using shared costs ($= ct$) and per-label cost ($\neq ct$). Logistic regression is used as a base classifier, and the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

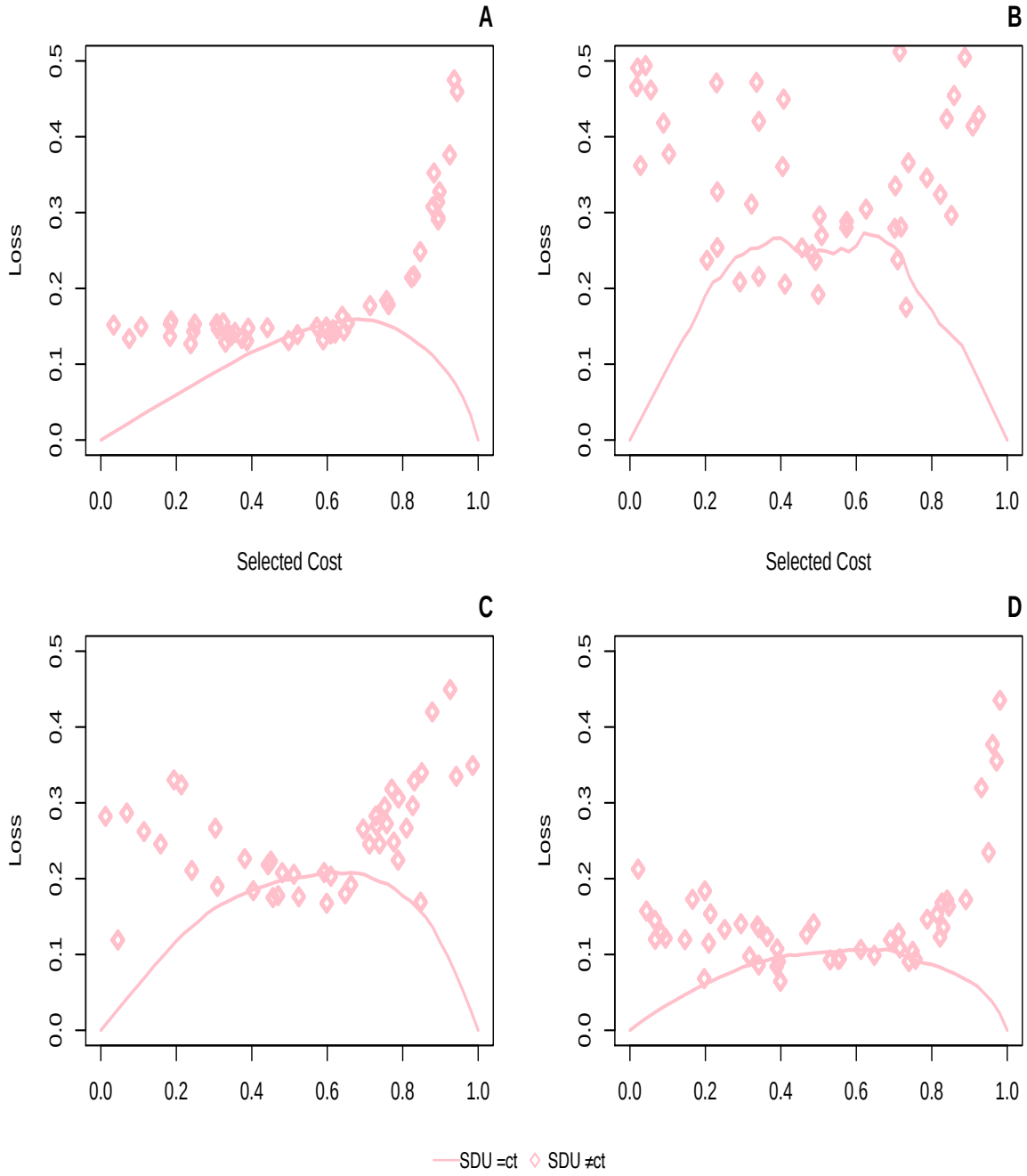


Figure 5.10: Cost curves for the cost-driven uniform (CDU) method where the x-axis represents the selected label cost which has been used as a threshold for all labels. Logistic regression is used as a base classifier, the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

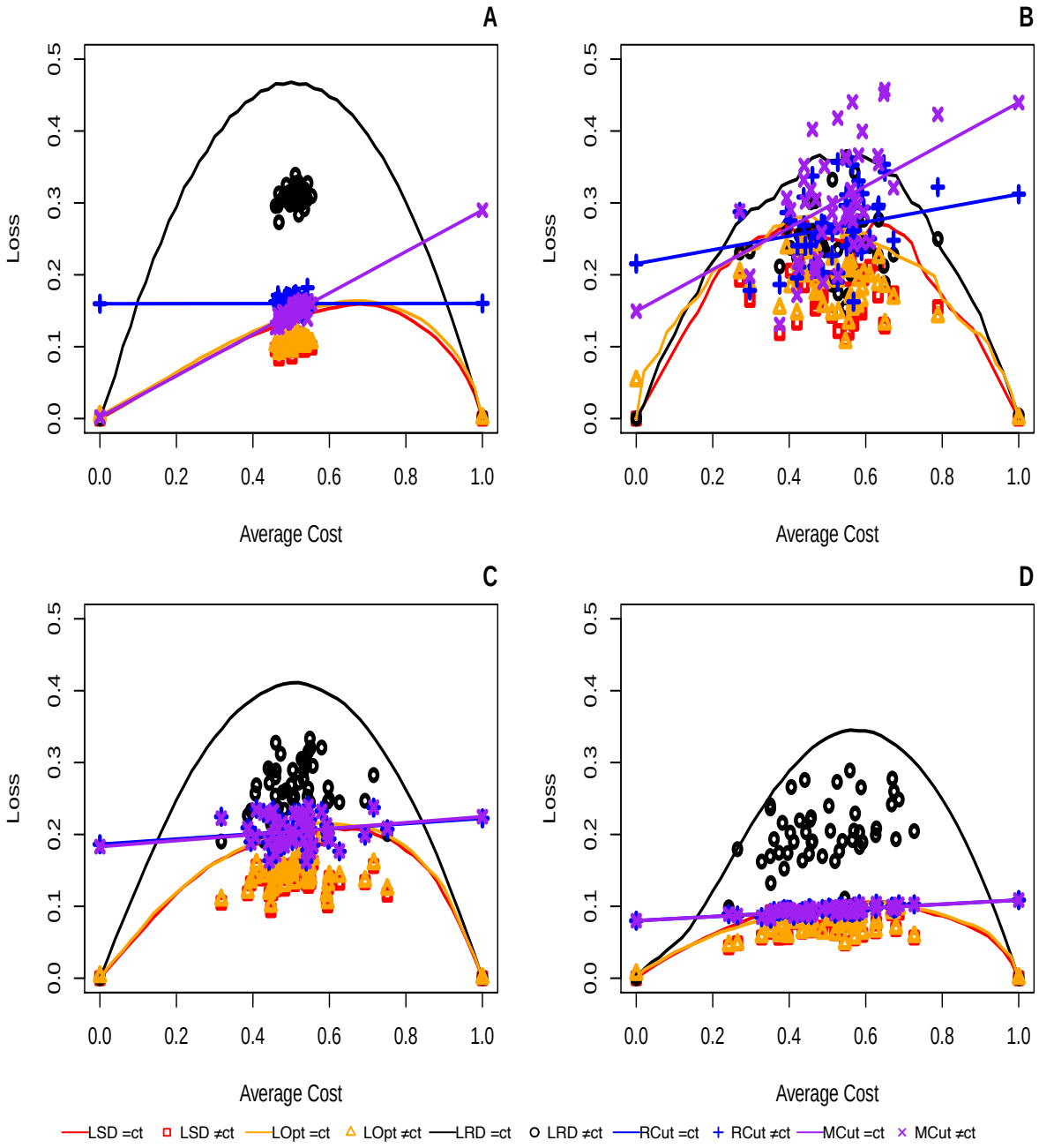


Figure 5.11: Cost curves for multiple threshold methods using shared cost ($= ct$) and per-label cost ($\neq ct$). Logistic regression is the base classifier, and the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

Figures 5.12, 5.13 and 5.14 repeat the same approaches on the same datasets but using the J48 algorithm as a base classifier. As expected, the cost-driven and the optimal threshold choice methods perform worse than using the logistic regression algorithm. However, the label-wise rate-driven method is shown to perform better using the J48 algorithm.

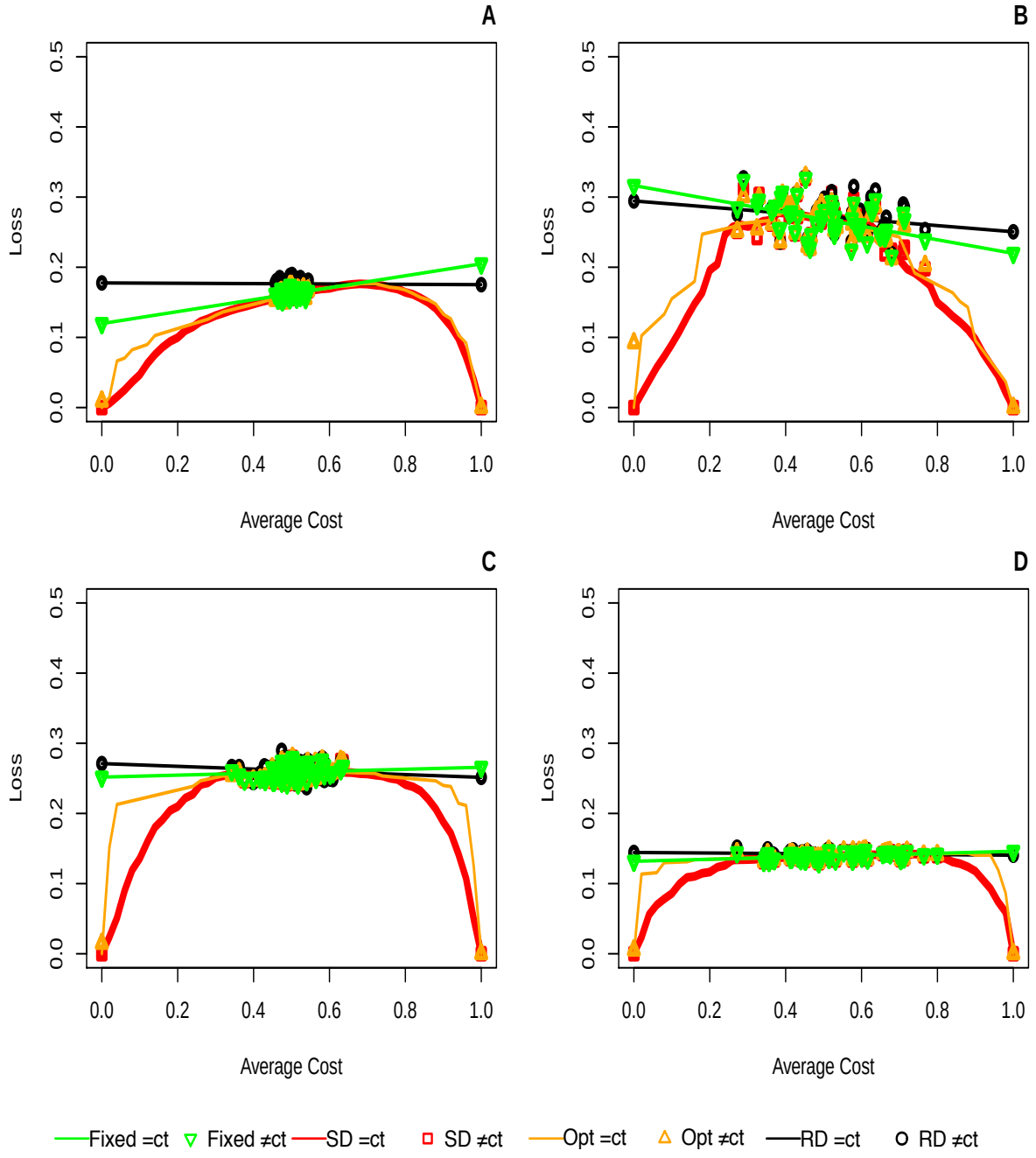


Figure 5.12: Cost curves for global threshold methods using shared costs ($= ct$) and per-label cost ($\neq ct$). J48 is used as a base classifier, and the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

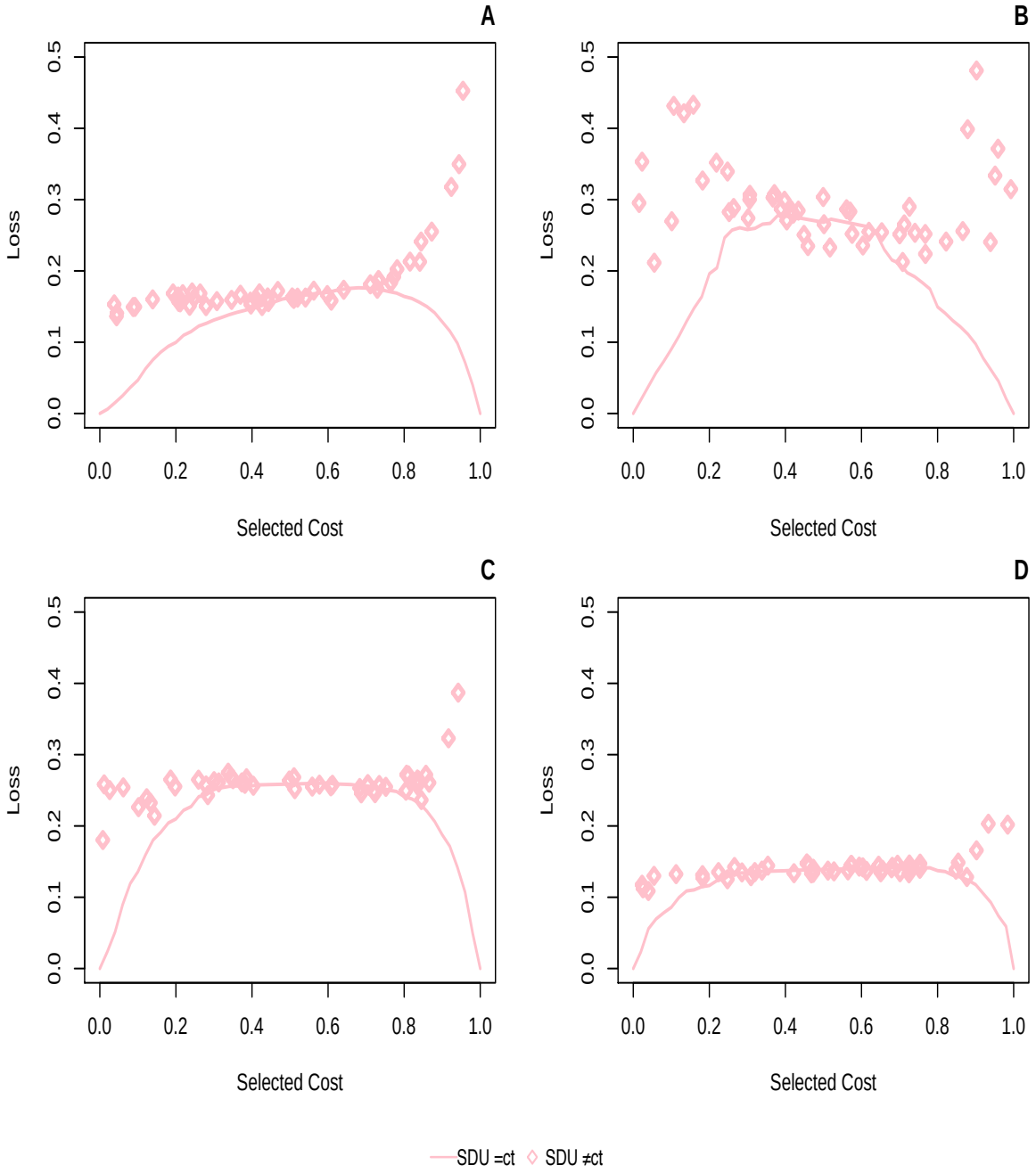


Figure 5.13: Cost curves for the cost-driven uniform (CDU) method where the x-axis represents the selected label cost which has been used as a threshold for all labels. J48 is used as a base classifier, the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

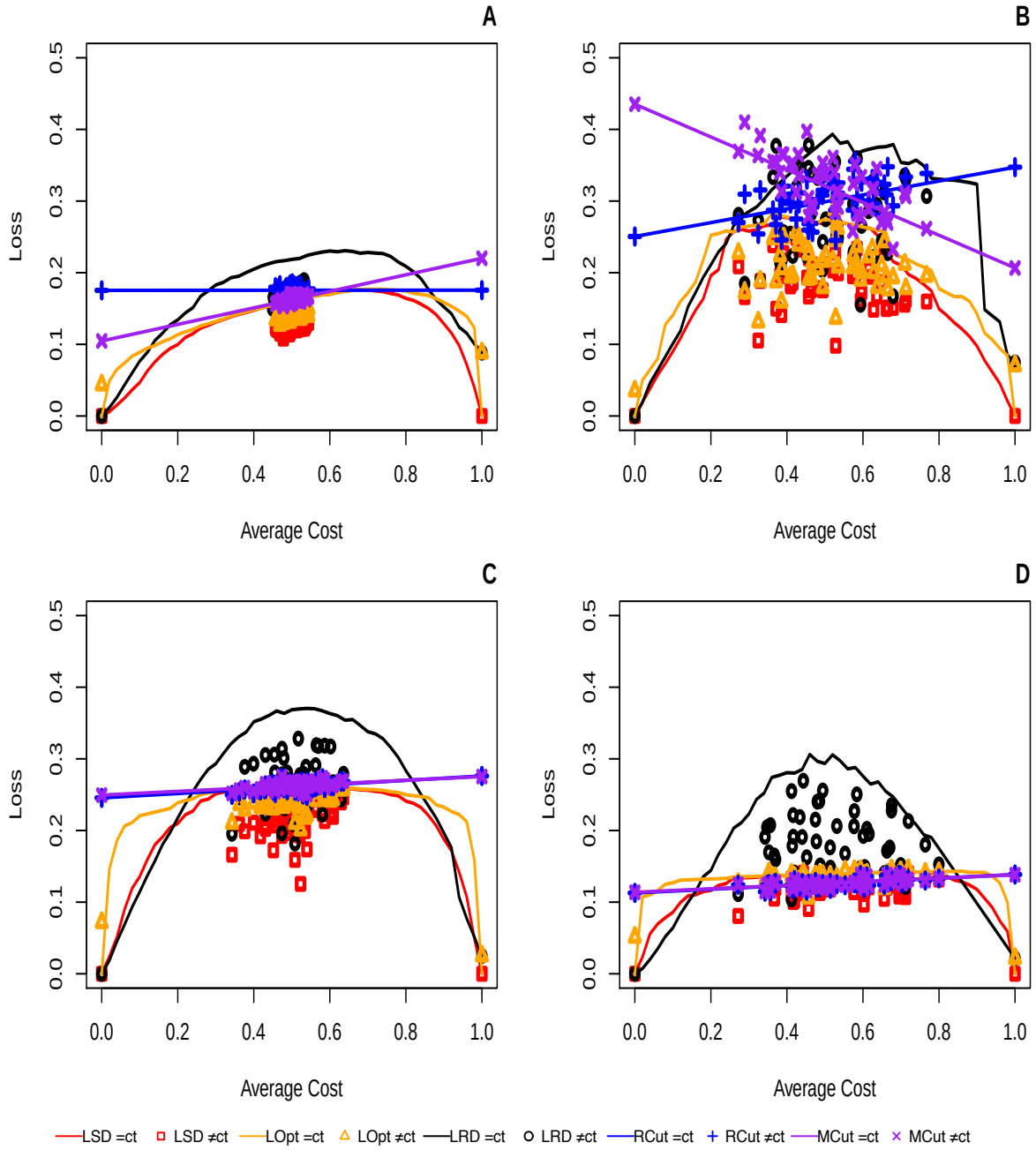


Figure 5.14: Cost curves for multiple threshold methods using shared cost ($= ct$) and per-label cost ($\neq ct$). J48 is the base classifier, and the datasets are: A) Cal500 B) Flags C) Yeast D) Scene.

Tables 5.3 and 5.4 show the empirical loss of using global and multiple thresholds, respectively. In the former, we tested the global threshold for each dataset in two settings, shared and per label cost, whereas in the latter, we tuned multiple thresholds, per label and per instance. In both cases, we used the binary relevance (BR) as a multi-label classifier and the logistic regression as a base classifier. However, the empirical loss can be measured by the Area under the Cost Curve over uniform cost(s) parameter(s) when the curve is obvious as seen previously using the shared cost [175]. In Tables 5.3 and 5.4, the best (smallest) loss for each dataset is shown in boldface.

Table 5.3: Empirical loss of adjusting ***a global threshold*** over multiple datasets and a range of misclassification costs.

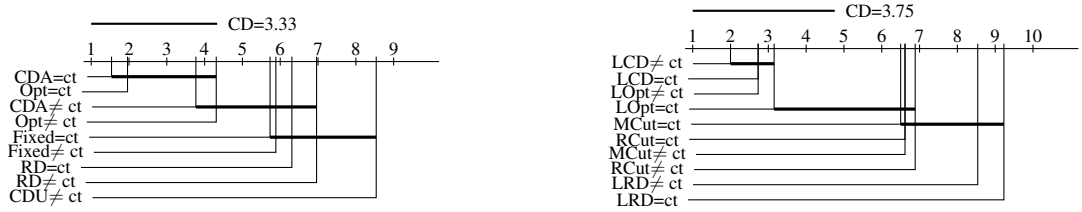
	shared cost					per-label cost				
	Fixed	CDA	CDU	Opt	RD	Fixed	CDA	CDU	Opt	RD
Corel5k	0.009	**0.008**	**0.008**	0.009	0.009	0.009	0.009	0.013	0.010	0.014
Cal500	0.138	**0.098**	**0.098**	0.099	0.159	0.138	0.132	0.201	0.133	0.160
Bibtex	0.013	**0.010**	**0.010**	0.011	0.016	0.013	0.012	0.013	0.012	0.016
Language log	0.015	**0.013**	**0.013**	0.015	0.022	0.016	0.015	0.022	0.016	0.022
Enron	0.047	**0.035**	**0.035**	0.036	0.050	0.046	0.044	0.062	0.044	0.051
Medical	0.022	0.015	0.015	**0.013**	0.018	0.022	0.021	0.022	0.017	0.018
Genbase	0.025	0.016	0.016	**0.002**	0.005	0.026	0.024	0.049	**0.002**	0.005
Slashdot	0.038	**0.030**	**0.030**	0.034	0.046	0.037	0.036	0.051	0.040	0.046
Birds	0.050	**0.040**	**0.040**	0.047	0.062	0.050	0.048	0.073	0.064	0.063
Yeast	0.199	**0.142**	**0.142**	**0.142**	0.207	0.200	0.188	0.283	0.188	0.208
Flags	0.251	**0.180**	**0.180**	0.181	0.246	0.246	0.232	0.357	0.232	0.242
Emotions	0.235	0.158	0.158	**0.151**	0.218	0.241	0.213	0.294	0.199	0.220
Scene	0.102	**0.073**	**0.073**	**0.073**	0.105	0.102	0.095	0.142	0.096	0.105
Average ranks	5.73	**1.53**	**1.53**	1.96	6.30	5.88	3.76	8.53	4.30	6.96

Table 5.4: Empirical loss of adjusting ***multiple thresholds*** over multiple datasets and a range of misclassification costs.

	shared cost					per-label cost				
	LCD	LOpt	LRD	RCut	MCut	LCD	LOpt	LRD	RCut	MCut
Corel5k	**0.008**	0.009	0.318	0.014	0.010	**0.008**	0.009	0.311	0.014	0.010
Cal500	0.098	0.104	0.304	0.160	0.146	**0.097**	0.103	0.297	0.161	0.148
Bibtex	**0.010**	0.012	0.314	0.016	0.014	**0.010**	0.011	0.306	0.016	0.014
Language log	**0.013**	0.014	0.297	0.022	0.022	**0.013**	0.014	0.293	0.022	0.022
Enron	**0.035**	0.039	0.291	0.053	0.054	**0.035**	0.039	0.286	0.053	0.054
Medical	0.015	**0.012**	0.259	0.020	0.020	0.015	**0.012**	0.247	0.020	0.020
Genbase	0.016	**0.001**	0.228	0.010	0.010	0.015	**0.001**	0.235	0.010	0.010
Slashdot	0.030	0.034	0.260	0.042	0.042	**0.029**	0.034	0.255	0.042	0.042
Birds	0.040	0.051	0.293	0.077	0.126	**0.039**	0.051	0.294	0.077	0.127
Yeast	0.142	0.145	0.263	0.204	0.204	**0.138**	0.142	0.255	0.205	0.204
Flags	0.180	0.188	0.235	0.264	0.295	**0.175**	0.183	0.234	0.262	0.298
Emotions	0.158	**0.150**	0.195	0.257	0.257	0.155	0.152	0.193	0.258	0.255
Scene	0.073	0.073	0.203	0.094	0.094	**0.071**	**0.071**	0.193	0.094	0.094
Average ranks	2.73	3.15	9.23	6.61	6.5	**2**	2.73	8.53	6.88	6.61

As can be seen in both tables, when labels have shared misclassification cost, the cost-driven average threshold achieves the lowest loss among others. On the other hand, the label-wise cost-driven thresholds lead to better performance when costs are different.

We conducted the Friedman test based on the average ranks for all datasets [145]. It ranks the methods across datasets separately, thus the best approach gets the rank of 1, the second best the rank of 2, etc. Then, it calculates the test statistic on the ranks averaged over all datasets in order to verify whether the differences between algorithms are statistically significant. The Friedman test gave a significant difference at the 5% significance level; therefore, we carried out post-hoc Nemenyi tests [145] as shown in Figure 5.15. Note that the cost-driven uniform (CDU=ct) method is not included in Figure 5.15(a) as it is similar to the cost-driven average (CDA=ct) method.



(a) global threshold (b) multiple thresholds

Figure 5.15: Critical difference diagrams for multi-label thresholding approaches using the Nemenyi test with pairwise comparisons for those experiments where the Friedman test gives significance at the 0.05 level. $= ct$ and $\neq ct$ denote shared and per-label cost, respectively.

The cost-driven average (CDA) technique has the best average rank followed by the global optimal method considering shared misclassification cost. In Figure 5.15(a), we observe that the global cost-driven average (CDA=ct) and the global optimal (Opt=ct) techniques are significantly better than most of the other methods. However, we notice that there is no significant loss on assigning a single threshold when the costs also are variable. To be clearer, the global threshold is still valid considering per label cost.

On the other hand, as shown in Figure 5.15(b), the label-wise cost-driven (LCD) method has the best average rank considering multiple thresholds and per label cost. There is no significant loss between label-wise versions of cost-driven (LCD) and optimal (LOpt) approaches. However, they are significantly better than MCut, RCut, and label-wise rate-drive (LRD) methods. Overall, the cost-driven threshold choice method obtains the best performance among others and can be a good candidate for multi-label cost-sensitive learning.

5.7 Concluding Remarks

There is a great deal of literature on multi-label learning. To the best of our knowledge, none of these studies has introduced changes in misclassification costs across contexts. In this chapter, we explored multi-label threshold choice methods: fixed, rate-driven, optimal, RCut and MCut over a wide range of

misclassification costs. In addition, we introduced two novel thresholding methods for multi-label data: cost-driven and the global optimal methods. The cost-driven can be adjusted globally (per dataset) or locally (per label).

We investigated two settings in multi-label cost-sensitive learning, which are selecting a single global or multiple thresholds. We provided an extensive empirical study of existing techniques on real-world multi-label datasets. Experimental results demonstrate that the global and label-wise cost-driven methods have the best results considering global and multiple thresholds, respectively. Additionally, we concluded that tuning a global threshold with respect to per label cost is not significantly worse than a separate threshold per label.

Furthermore, some of the thresholding approaches discussed for binary classification such as label-wise rate driven may not be appropriate for multi-label data, which has highly imbalanced class labels. We recommend using the cost-driven threshold choice method either globally or per label, as it obtains the best performance. It is significantly better than some approaches such as fixed and rate-driven threshold choice methods.

More research on this topic needs to be undertaken to clarify the association between label misclassification costs, loss and threshold choice methods. Further work should investigate the influence when misclassification costs for some labels change while others remain constant across contexts.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis we have presented several novel solutions for relevant problems in machine learning. We have concentrated our work during this thesis on decision tree learning. In this last chapter, we summarise the work achieved in this thesis and describe potential areas for future work.

6.1 Thesis Summary

This thesis extended the classical decision tree algorithms and addressed two main shortcomings within these algorithms: *context changes* and handling *multiple labels* at leaves. To begin with, this thesis proposed the Versatile Model (**VM**) that handles distribution variations between training and deployment contexts. More specifically, we studied a specific type of context changes known as dataset shift where the covariates have different distributions. The VM is a decision tree model that adopts multiple strategies in order to handle different kinds of shift. Empirical results on both synthetic shift and real dataset shift show the strong performance of the proposed model. This indicates that adopting the proposed strategies, the percentiles, and linear thresholds, within the standard decision tree algorithms can be a great enhancement to make them more robust to any potential context changes.

Towards the second issue, we have developed decision tree algorithms that support the prediction of multiple labels, taking into consideration their relationships locally. In this context, we proposed two novel algorithms, namely, **LaCova** and **LaCovaC**. The key idea of **LaCova** is to use a label covariance matrix at every node of the tree to decide how to split labels. In addition to internal nodes that split the data horizontally (dividing the set of instances) based on the selected attributes, **LaCova** introduces the second kind of node for splitting the label space vertically. At deployment, all vertical splits outgoing edges should be tested to collect predictions about the entire label set. Furthermore, we proposed a novel splitting criterion designed to multi-label data, which rewards splits that reduce either variance or

covariance among labels.

The proposed algorithm, which we called **LaCovaC**, is further enhanced by incorporating label clusters. It computes the label correlation matrix at each node of the tree in order to identify label correlations and then cluster them locally. Vertical splits can handle dependant label sets not only independent labels as in **LaCova**. Our empirical evaluation on 13 benchmark datasets and seven common evaluation metrics for multi-label classification shows that **LaCovaC** has strong performance across all these metrics.

Finally, we contributed to the area of multi-label cost-sensitive learning as well as threshold choice methods. We explored another type of variation across contexts, which is changing label misclassification costs. We provided better understanding towards existing solutions and proposed novel approaches. An important aspect of our study is to investigate two options in this context, which are selecting a single global or multiple thresholds. We provided an extensive empirical study of existing techniques on real-world multi-label datasets. We used logistic regression algorithm as a base classifier instead of decision tree algorithm as this algorithm produces better calibrated probabilities than decision trees. Experimental results demonstrate that tuning a global threshold with respect to per label cost is not significantly worse than a separate threshold per label.

6.2 Future Directions

This work opens up many avenues for future work. We summarise the key points below.

6.2.1 VM for Regression

The versatile model (VM) for the classification task has clearly shown competitive results by comparisons with other methods. Another research direction is to extend the experiments in Chapter 3 to use the SubClass Re-estimation (SCRE) dataset shift method, which was by far the best performing data shift method in the experiments reported in [148].

Moreover, on the basis of the promising findings presented in Chapter 3, it is encouraging to adapt the VM to other predictive problems, such as regression. One possibility is to assume that the deployment data is partially labelled and utilise this knowledge in the VM. We could learn the distribution parameters of the target using some labelled data such as the mean and standard deviation parameters.

6.2.2 VM as a Multi-label Classifier

The VM addresses the dataset shift in the context of single-label classification. One possible future direction is to extend the versatile model to the multi-label setting. The versatile model has adopted three types of thresholds within the decision tree framework: original threshold, linear threshold, and percentile for non-linear shifts. It would be interesting to study the percentile in the multi-label setting

where multiple labels are present. Percentiles can be based on the marginal distributions (for independent labels), the joint distributions (for dependent labels), or both.

Additionally, in a multi-label decision tree, a leaf node can represent two kinds of label distribution: marginal and/or joint distribution. In the former, a leaf holds a probability vector of all labels that belong to this node in the training set, a label is applied when the probability is higher or equal to a certain threshold, while in the latter, the probability of all possible sets of labels will be included. When an unseen instance needs to be classified by a decision tree, it is given the majority label set of the region it falls into.

Multi-label classification is used widely for automatic tagging. Tags are usually annotated by human experts, which can be subjective to different opinions. The problem with such task is the presence or the absence of particular tags across contexts. It is interesting to investigate how to adapt the given tree model to the new context. One possibility is to consider a semantic and logical relationship among tags such as conjunction, negation and disjunction relationships.

6.2.3 **LaCova** as Meta-classifier and Ensembles

Another direction is that **LaCova** and **LaCovaC** algorithms can also be used as a meta-classifier so that any single-label classifier can be applied within this framework.

Moreover, we are interested in incorporating ensemble models with these algorithms and compare it with other ensemble algorithms. Furthermore, standard decision tree algorithms, in general, have high variance. This means that the performance of the algorithm can be influenced by small changes in the training dataset. As **LaCova** and **LaCovaC** are both decision tree algorithms, they are likely to have large variance as well. To overcome the large variance problem associated with the proposed algorithms, we could use bootstrap aggregates as in random forests.

6.2.4 More Aspects of **LaCovaC**

LaCovaC used the joint label distribution for the final prediction, while we can predict the marginal distribution at leaves, instead. One aspect that may be worth to investigate is when to interleave between those decisions. Interestingly, this can be used to optimise the performance with regards to the evaluation metric of interest.

Moreover, we plan to investigate the parameter configuration for **LaCovaC**, for example, a stopping criterion for the clustering algorithm. Moreover, it would be interesting to investigate alternate clustering approaches, such as spectral clustering.

Finally, using the significant threshold on label correlations balances the sample size and the strength of the correlation, such that both low correlations on a large sample size and high correlations on a small sample are detected as significant. However, finding an even better balance between the effect size and sample size is an interesting future research direction.

6.2.5 Thresholding for Multi-label Classification

One future research direction related to Chapter 5 could be to do experiments using a more advanced multi-label approach which takes label correlations into account; rather than just the binary relevance approach, which ignores label correlations.

Bibliography

[1] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 3–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1566770.1566773

[2] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*, 2010, pp. 667–685.

[3] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, 2014. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2013.39

[4] L. Rokach and O. Maimon, "Decision trees," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer US, 2005, pp. 165–192. [Online]. Available: http://dx.doi.org/10.1007/0-387-25465-X_9

[5] ——, *Data Mining with Decision Trees: Theory and Applications*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2008.

[6] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10462-011-9272-4

[7] J. Hernández-Orallo, A. M. Usó, R. B. C. Prudêncio, M. Kull, P. A. Flach, C. F. Ahmed, and N. Lachiche, "Reframing in context: A systematic approach for model reuse in machine learning," *AI Communication*, vol. 29, no. 5, pp. 551–566, 2016. [Online]. Available: http://dx.doi.org/10.3233/AIC-160705

[8] E. Gibaja and S. Ventura, "A tutorial on multilabel learning," *ACM Computing Surveys*, vol. 47, no. 3, pp. 52:1–52:38, April 2015. [Online]. Available: http://doi.acm.org/10.1145/2716262

[9] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD09)*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 254–269. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04174-7_17

[10] K. Dembczyński, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains." in *Proceedings of the International Conference on Machine Learning (ICML10)*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 279–286. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2010.html#DembczynskiCH10

[11] 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*, ser. KDD10. New York, NY, USA: ACM, 2010, pp. 999–1008. [Online]. Available: http://doi.acm.org/10.1145/1835804.1835930

[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. [Online]. Available: http://www.springerlink.com/content/g7577131654jk0v7/

[13] E. Montañes, R. Senge, J. Barranquero, J. Ramón Quevedo, J. José del Coz, and E. Hüllermeier, "Dependent binary relevance models for multi-label classification," *Pattern Recogn.*, vol. 47, no. 3, pp. 1494–1508, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2013.09.029

[14] M. S. Sorower, "A literature survey on algorithms for multi-label learning," Oregon State University, Tech. Rep., 2010.

[15] R. Al-Otaibi, "Phd thesis: Learning versatile decision trees towards context-awareness and multi-label classification," https://github.com/ralotaibi/PhD_thesis, 2016.

[16] R. Al-Otaibi, R. B.C. Prudêncio, M. Kull, and P. Flach, "Versatile decision trees for learning over multiple contexts," in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD15)*, ser. Lecture Notes in Computer Science, A. Appice, P. P. Rodrigues, V. Santos Costa, C. Soares, J. Gama, and A. Jorge, Eds. Cham: Springer International Publishing, September 2015, pp. 184–199. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23528-8_12

[17] ——, "A context-aware model based on versatile decision trees," February 2016, to appear.

[18] R. Al-Otaibi, M. Kull, and P. Flach, "Declaratively capturing local label correlations with multi-label trees," in *Proceedings of the 22nd Biennial European Conference on Artificial Intelligence (ECAI2016), Including Prestigious Applications of Intelligent Systems*

(PAIS-2016)., ser. Frontiers in Artificial Intelligence and Applications, G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, Eds., vol. 285. IOS press, 29 August- 2 September 2016, pp. 1467 – 1475. [Online]. Available: http://ebooks.iospress.com/volumearticle/44904

[19] ——, "Lacova: A tree-based multi-label classifier using label covariance as splitting criterion," in *Proceedings of the IEEE 13th International Conference on Machine Learning and Application (ICMLA-2014)*. IEEE, December 2014, pp. 74–79.

[20] ——, "Multi-label classification by label clustering based on covariance," in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECML-PKDD15 Doctoral Consortium*, ser. Aalto University publication series SCIENCE + TECHNOLOGY, 12/2015, J. Hollmén and P. Papapetrou, Eds. Aalto University, September 2015, pp. 43–52. [Online]. Available: https://aaltodoc.aalto.fi/handle/123456789/18224

[21] ——, "Hybrid multi-label decision trees for classification," in *Proceedings of the 8th Saudi Students Conference in the UK*, N. Alford and J. Fréchet, Eds. Imperial College Press, 2016, pp. 323–334.

[22] R. Al-Otaibi, P. Flach, and M. Kull, "Multi-label classification: A comparative study on threshold selection methods," in *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD 2014*, September 2014. [Online]. Available: http://users.dsic.upv.es/~flip/LMCE2014/Papers/lmce2014_submission_11.pdf

[23] R. Al-Otaibi, N. Jin, T. Wilcox, and P. Flach, "Feature construction and calibration for clustering daily load curves from smart meter data," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 645–654, April 2016.

[24] R. Xu and D. Wunsch, II, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005. [Online]. Available: http://dx.doi.org/10.1109/TNN.2005.845141

[25] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.

[26] J. Read, C. Bielza, and P. Larrañaga, "Multi-dimensional classification with super-classes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, pp. 1720–1733, July 2014.

[27] D. W. Hosmer and S. Lemeshow, *Applied logistic regression*, ser. Wiley series in probability and statistics. New York, Chichester, Weinheim: John Wiley & Sons, Inc. A Wiley-Interscience Publication, 2000. [Online]. Available: http://opac.inria.fr/record=b1128889

[28] R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Online]. Available: http://dx.doi.org/10.1023/A:1022643204877

[29] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93. [Online]. Available: http://dx.doi.org/10.1023/B:AMAI.0000018580.96245.c6

[30] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers – a survey," 2002.

[31] C. E. Brodley and P. E. Utgoff, "Multivariate versus univariate decision trees," Amherst, MA, USA, Tech. Rep., 1992.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Exploration Newsletter*, vol. 11, no. 1, pp. 10–18, November 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

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

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

[35] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/

[36] R. C. Barros, A. C. P. L. F. de Carvalho, and A. A. Freitas, *Automatic Design of Decision-Tree Induction Algorithms*, ser. Springer Briefs in Computer Science. Cham: Springer International Publishing, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14231-9

[37] K. Brinker, J. Fürnkranz, and E. Hüllermeier, "A unified model for multilabel classification and ranking," in *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI06)*. Amsterdam, The Netherlands, The Netherlands: IOS Press, September 2006, pp. 489–493. [Online]. Available: http://dl.acm.org/citation.cfm?id=1567016.1567123

[38] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*. Springer, 2010, pp. 667–685.

[39] J. Read, "Scalable multi-label classification," Ph.D. dissertation, University of Waikato, 2010.

[40] E. Loza Mencía and J. Fürnkranz, "Efficient multilabel classification algorithms for large-scale problems in the legal domain," in *Semantic Processing of Legal Texts – Where the Language of Law Meets the Law of Language*, 1st ed., ser. Lecture Notes in Artificial Intelligence, E. Francesconi, S. Montemagni, W. Peters, and

D. Tiscornia, Eds. Springer-Verlag, May 2010, vol. 6036, pp. 192–215. [Online]. Available: http://www.ke.tu-darmstadt.de/publications/papers/loza10eurlex.pdf

[41] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking," *Machine Learning*, vol. 73, no. 2, pp. 133–153, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10994-008-5064-8

[42] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *Proceedings of the 18th European Conference on Machine Learning*, ser. ECML07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 406–417. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74958-5_38

[43] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proceedings of The International Conference on Data Mining (ICDM09)*. IEEE Computer Society, 2008, pp. 995–1000. [Online]. Available: http://dblp.uni-trier.de/db/conf/icdm/icdm2008.html#ReadPH08

[44] J. H. Zaragoza, L. E. Sucar, E. F. Morales, C. Bielza, and P. Larrañaga, "Bayesian chain classifiers for multidimensional classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI11)*, ser. IJCAI11. AAAI Press, 2011, pp. 2192–2197. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-365

[45] J. Read, L. Martino, and D. Luengo, "Efficient monte carlo methods for multi-dimensional learning with classifier chains," *Pattern Recognition*, vol. 47, no. 3, pp. 1535–1546, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2013.10.006

[46] P. N. da Silva, E. C. Gonçalves, A. Plastino, and A. A. Freitas, "Distinct chains for different instances: An effective strategy for multi-label classifier chains," in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD14)*, ser. Lecture Notes in Computer Science, T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, Eds. Springer Berlin Heidelberg, 2014, vol. 8725, pp. 453–468. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44851-9_29

[47] E. C. Gonçalves, A. Plastino, and A. A. Freitas, "Simpler is better: A novel genetic algorithm to induce compact multi-label chain classifiers," in *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO15. New York, NY, USA: ACM, 2015, pp. 559–566. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754650

[48] E. Alvares Cherman, J. Metz, and M. Monard, "A Simple Approach to Incorporate Label Dependency in Multi-label Classification," in *Advances in Soft Computing*, ser. Lecture Notes in Computer Science, G. Sidorov, A. Hernández Aguirre, and C. Reyes García, Eds. Springer Berlin / Heidelberg, 2010, vol. 6438, ch. 3, pp. 33–43. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16773-73

[49] S. Huang and Z. Zhou, "Multi-label learning by exploiting label correlations locally," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4950

[50] K. Dembczyński, A. Jachnik, W. Kotlowski, W. Waegeman, and E. Hüllermeier, "Optimizing the f-measure in multi-label classification: Plug-in rule approach versus structured loss minimization," in *Proceedings of the 30th International Conference on Machine Learning (ICML13)*, S. Dasgupta and D. McAllester, Eds., vol. 28. JMLR Workshop and Conference Proceedings, May 2013, pp. 1130–1138. [Online]. Available: http://jmlr.org/proceedings/papers/v28/dembczynski13.pdf

[51] Y. Guo and D. Schuurmans, "Multi-label classification with output kernels." in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD13)*, ser. Lecture Notes in Computer Science, H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezn, Eds., vol. 8189. Springer, 2013, pp. 417–432. [Online]. Available: http://dblp.uni-trier.de/db/conf/pkdd/pkdd2013-2.html#GuoS13

[52] J. Langford, T. Zhang, D. J. Hsu, and S. M. Kakade, "Multi-label prediction via compressed sensing," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 772–780. [Online]. Available: http://papers.nips.cc/paper/3824-multi-label-prediction-via-compressed-sensing.pdf

[53] F. Tai and H.-T. Lin, "Multilabel classification with principal label space transformation," *Neural Comput.*, vol. 24, no. 9, pp. 2508–2542, September 2012. [Online]. Available: http://dx.doi.org/10.1162/NECO_a_00320

[54] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, ser. PKDD01. London, UK, UK: Springer-Verlag, 2001, pp. 42–53. [Online]. Available: http://dl.acm.org/citation.cfm?id=645805.670013

[55] F. De Comité, R. Gilleron, and M. Tommasi, "Learning multi-label alternating decision trees from texts and data," in *Proceedings of the 3rd International Conference on Machine Learning and Data Mining in Pattern Recognition*, ser. MLDM03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 35–49. [Online]. Available: http://dl.acm.org/citation.cfm?id=1759548.1759554

[56] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, December 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1007614523901

[57] D. Gjorgjevikj, G. Madjarov, and S. Džeroski, "Hybrid decision tree architecture utilizing local svms for efficient multi-label learning," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 27, no. 7, p. 1351004, 2013.

[58] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, pp. 2038–2048, 2007.

[59] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels," in *ECML-PKDD 2008 Workshop on Mining Multidimensional Data*, 2008. [Online]. Available: http://mlkd.csd.auth.gr/publication_details.asp?publicationID=276

[60] 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, April 2013. [Online]. Available: http://dx.doi.org/10.1007/s10994-012-5312-9

[61] P. E. Greenwood and M. S. Nikulin, *A Guide to Chi-Squared Testing*. New York, NY: Wiley, 1996.

[62] C. Silla, Jr. and A. A. Freitas, "A survey of hierarchical classification across different application domains," *Data Mining Knowledge Discovery*, vol. 22, no. 1-2, pp. 31–72, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10618-010-0175-9

[63] A. Clare and R. D. King, "Predicting gene function in saccharomyces cerevisiae," *Bioinformatics*, vol. 19, pp. 42–49, 2003.

[64] H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare, "Decision trees for hierarchical multilabel classification: A case study in functional genomics," in *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD06)*, ser. Lecture Notes in Artificial Intelligence, vol. 4213. Springer, 2006, pp. 18–29, uRL: http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=42302.

[65] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Machine Learning*, vol. 73, no. 2, pp. 185–214, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10994-008-5077-3

[66] W. Bi and J. T. Kwok, "Multi-label classification on tree-and dag-structured hierarchies," in *Proceedings of the 28th International Conference on Machine Learning (ICML11)*, L. Getoor and T. Scheffer, Eds., 2011, pp. 17–24.

[67] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni, "Incremental algorithms for hierarchical classification," *Journal of Machine Learning Research*, vol. 7, pp. 31–54, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248549

[68] J. Rousu, C. Saunders, S. Szedmák, and J. Shawe-Taylor, "Kernel-based learning of hierarchical multilabel classification models." *Journal of Machine Learning Research*, vol. 7, pp. 1601–1626, 2006.

[69] W. Bi and J. T. Kwok, "Hierarchical multilabel classification with minimum bayes risk," in *Proceedings of 12th International Conference on Data Mining (ICDM12)*, 2012, pp. 101–110.

[70] O. Luaces, J. Díez, J. Barranquero, J. José del Coz, and A. Bahamonde, "Binary relevance efficacy for multilabel classification," *Progress in AI*, vol. 1, no. 4, pp. 303–313, 2012.

[71] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, Sep. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2012.03.004

[72] Y. Yang, "An evaluation of statistical approaches to text categorization," *Journal of Information Retrieval*, vol. 1, pp. 67–88, 1999.

[73] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," in *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2004, pp. 22–30.

[74] N. Ghamrawi and A. McCallum, "Collective multi-label classification," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, ser. CIKM05. New York, NY, USA: ACM, 2005, pp. 195–200. [Online]. Available: http://doi.acm.org/10.1145/1099554.1099591

[75] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Machine Learning*, 1999, pp. 297–336. [Online]. Available: http://dx.doi.org/10.1023/A:1007614523901

[76] J. A. Fernandes, J. A. Lozano, I. n. Inza, X. Irigoien, A. Pérez, and J. D. Rodríguez, "Supervised pre-processing approaches in multiple class variables classification for fish recruitment forecasting," *Environmental Modelling and Software*, vol. 40, pp. 245 – 254, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815212002472

[77] E. Hadavandi, J. Shahrabi, and Y. Hayashi, "SPMoE: a novel subspace-projected mixture of experts model for multi-target regression problems," *Soft Computing*, pp. 1–19, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00500-015-1623-7

[78] L. Rokach, A. Schclar, and E. Itach, "Ensemble methods for multi-label classification," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7507–7523, Nov. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2014.06.015

[79] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 12, pp. 2411–2414, July 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021078

[80] F. C. Bernardini, R. Barbosa da Silva, and E. M. Meza, "Analyzing the influence of cardinality and density characteristics on multi-label learning," 2013. [Online]. Available: http://www.professores.uff.br/fcbernardini/papers/Bernardini_Barbosa_Meza-BRACIS2013.pdf

[81] P. P. da Gama, F. C. Bernardini, and B. Zadrozny, "Rb: A new method for constructing multi-label classifiers based on random selection and bagging," *Journal of the Brazilian Computational Intelligence Society-Learning and Nonlinear Models*, vol. 11, pp. 26–47, 2013.

[82] F. Charte and D. Charte, "Working with multilabel datasets in R: The mldr package," *The R Journal*, vol. 7, no. 2, pp. 149–162, dec 2015. [Online]. Available: http://journal.r-project.org/archive/2015-2/charte-charte.pdf

[83] F. Charte, A. J. R. Rivas, M. J. del Jesús, and F. Herrera, "Concurrence among imbalanced labels and its influence on multilabel resampling algorithms," in *Proceedings of the 9th International Conference on Hybrid Artificial Intelligence Systems, HAIS*, 2014, pp. 110–121. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07617-1_10

[84] P. Duygulu, K. Barnard, J. de Freitas, and D. Forsyth, "Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary," in *Computer Vision, ECCV 2002*, ser. Lecture Notes in Computer Science, 2002, vol. 2353, pp. 97–112.

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

[86] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multilabel text classification for automated tag suggestion," in *Proceedings of ECML PKDD08 Discovery Challenge*, 2008, pp. 75–83.

[87] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Proceedings of the 17th European Conference on Machine Learning, ECML04*, 2004, pp. 217–226.

[88] K. Crammer, M. Dredze, K. Ganchev, P. P. Talukdar, and S. Carroll, "Automatic code assignment to medical text," in *Proceedings Workshop on Biological, Translational, and Clinical Language Processing, BioNLP07*, 2007, pp. 129–136.

[89] S. Diplaris, G. Tsoumakas, P. Mitkas, and I. Vlahavas, "Protein classification with multiple algorithms," in *Proceedings of the 10th Panhellenic Conference on Informatics, PCI05*, 2005, pp. 448–456.

[90] F. Briggs, B. Lakshminarayanan, L. Neal, X. Z. Fern, R. Raich, S. J. K. Hadley, A. S. Hadley, and M. G. Betts, "Acoustic classification of multiple simultaneous bird species: A multi-instance multi-label approach," *The Journal of the Acoustical Society of America*, vol. 131, no. 6, pp. 4640–4650, 2012.

[91] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in Neural Information Processing Systems*, vol. 14, 2001, pp. 681–687.

[92] E. C. Gonçalves, A. Plastino, and A. A. Freitas, "A genetic algorithm for optimizing the label ordering in multi-label classifier chains," in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2013, pp. 469–476.

[93] A. Wieczorkowska, P. Synak, and Z. Raś, "Multi-label classification of emotions in music," in *Intelligent Information Processing and Web Mining*, 2006, vol. 35, ch. 30, pp. 307–315.

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

[95] C. X. Ling and V. S. Sheng, *Cost-Sensitive Learning and the Class Imbalance Problem*. Springer, December 2008, pp. 869–875. [Online]. Available: http://cling.csd.uwo.ca/papers/cost_sensitive.pdf

[96] C. Elkan, "The foundations of cost-sensitive learning," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 973–978.

[97] H.-T. Lin, "Cost-sensitive classification: Status and beyond," in *Proceedings of Workshop Machine Learning Research in Taiwan: Challenges and Directions*, 2010.

[98] Z.-H. Zhou and X.-Y. Liu, "On multi-class cost-sensitive learning," in *Proceedings of the 21st National Conference on Artificial intelligence*. AAAI Press, 2006, pp. 567–572. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597538.1597630

[99] J. Li, X. Li, and X. Yao, "Cost-sensitive classification with genetic programming," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 3. IEEE Press, 2005, pp. 2114–2121. [Online]. Available: http://www.cs.bham.ac.uk/~xin/papers/LiLiYaoCEC05.pdf

[100] W. Fan and S. J. Stolfo, "Adacost: misclassification cost-sensitive boosting," in *Proceedings of the 16th International Conference on Machine Learning (ICML99)*. Morgan Kaufmann, 1999, pp. 97–105.

[101] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM03)*. IEEE Computer Society, 2003, pp. 435–. [Online]. Available: http://dl.acm.org/citation.cfm?id=951949.952181

[102] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research (JAIR)*, vol. 16, no. 1, pp. 321–357, June 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[103] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD99)*. ACM Press, 1999, pp. 155–164.

[104] R. Santos-Rodríguez, A. Guerrero-Curieses, R. Alaiz-Rodríguez, and J. Cid-Sueiro, "Cost-sensitive learning based on bregman divergences," *Machine Learning*, vol. 76, no. 2, pp. 271–285, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10994-009-5132-8

[105] H.-Y. Lo, J.-C. Wang, H.-M. Wang, and S.-D. Lin, "Cost-sensitive multi-label learning for audio tag annotation and retrieval." *IEEE Transactions on Multimedia*, vol. 13, no. 3, pp. 518–529, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/tmm/tmm13.html#LoWWL11

[106] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[107] S. S. Choi, S. H. Cha, and C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

[108] A. K. Patidar, J. Agrawal, and N. Mishra, "Analysis of different similarity measure functions and their impacts on shared nearest neighbor clustering approach," *International Journal of Computer Applications*, pp. 1–5, 2012.

[109] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, June 2010. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2009.09.011

[110] P. Berkhin, *Grouping Multidimensional Data: Recent Advances in Clustering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ch. A Survey of Clustering Data Mining Techniques, pp. 25–71. [Online]. Available: http://dx.doi.org/10.1007/3-540-28349-8_2

[111] Z. Ghahramani, "Unsupervised learning," in *Advanced Lectures on Machine Learning*. Springer-Verlag, 2004, pp. 72–112.

[112] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: data mining, inference, and prediction*, 2nd ed., ser. Springer series in statistics. New York: Springer, 2009. [Online]. Available: http://opac.inria.fr/record=b1127878

[113] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[114] H.-S. Park, J. seok Lee, and C.-H. Jun, "A k-means-like algorithm for k-medoids clustering and its performance," 2009. [Online]. Available: http://nichol.as/papers/Hae-sang/AK-means-likeAlgorithmforK-medoidsClusteringandItsPerformance.pdf

[115] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283494

[116] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, MArch 2012. [Online]. Available: http://dx.doi.org/10.14778/2180912.2180915

[117] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert System with Applications*, vol. 36, no. 2, pp. 3336–3341, March 2009. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2008.01.039

[118] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers and Geosciences*, vol. 10, no. 2, pp. 191 – 203, 1984. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0098300484900207

[119] T. C. Havens, J. C. Bezdek, C. Leckie, L. O. Hall, and M. Palaniswami, "Fuzzy c-means algorithms for very large data," *IEEE Transactions on Fuzzy Systems*, vol. 20, no. 6, pp. 1130–1146, December 2012.

[120] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, September 1999. [Online]. Available: http://doi.acm.org/10.1145/331499.331504

[121] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[122] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335388

[123] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, June 2014. [Online]. Available: http://dx.doi.org/10.1126/science.1242072

[124] X. Xu, M. Ester, H. P. Kriegel, and J. Sander, "A distribution-based clustering algorithm for mining in large spatial databases," in *Proceedings of the 14th International Conference on Data Engineering*, Febraury 1998, pp. 324–331.

[125] H. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[126] S. E. Schaeffer, "Survey: Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, August 2007. [Online]. Available: http://dx.doi.org/10.1016/j.cosrev.2007.05.001

[127] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, December 2007. [Online]. Available: http://dx.doi.org/10.1007/s11222-007-9033-z

[128] C. Chrysouli and A. Tefas, "Spectral clustering and semi-supervised learning using evolving similarity graphs," *Applied Soft Computing*, vol. 34, pp. 625 – 637, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494615003269

[129] J. G. Moreno-Torres, T. Raeder, R. Aláiz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2011.06.019

[130] M. Kull and P. Flach, "Patterns of dataset shift," in *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD 2014*, Nancy, France, September 2014. [Online]. Available: http://users.dsic.upv.es/~flip/LMCE2014/Papers/lmce2014_submission_10.pdf

[131] F. J. Massey, "The Kolmogorov-Smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951. [Online]. Available: http://www.jstor.org/stable/2280095

[132] C. Elkan, "The foundations of cost-sensitive learning," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 973–978.

[133] A. J. Storkey, "When training and test sets are different: Characterising learning transfer," in *Dataset Shift in Machine Learning*, S. L. Eds Candela, Sugiyama, Ed. MIT Press, 2009, ch. 1, pp. 3–28.

[134] B. Zadrozny, "Learning and evaluating classifiers under sample selection bias," in *Proceedings of the International Conference on Machine Learning (ICML04)*, ser. ICML04, New York, NY, USA, 2004, pp. 903–910. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015425

[135] J. G. Moreno-Torres, X. Llorí, D. E. Goldberg, and R. Bhargava, "Repairing fractures between data using genetic programming-based feature extraction: A case study in cancer diagnosis," *Information Sciences*, vol. 222, pp. 805–823, Feb. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2010.09.018

[136] S. Bickel, M. Brückner, and T. Scheffer, "Discriminative learning under covariate shift," *Journal of Machine Learning Research*, vol. 10, pp. 2137–2155, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755858

[137] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, *Covariate Shift by Kernel Mean Matching*. Cambridge, MA, USA: The MIT Press, 2009, vol. 5, pp. 131–160.

[138] M. Sugiyama, M. Krauledat, and K.-R. Müller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, pp. 985–1005, Dec. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1314498.1390324

[139] C. F. Ahmed, N. Lachiche, C. Charnay, and A. Braud, "Reframing continuous input attributes," in *Proceedings of IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov 2014, pp. 31–38.

[140] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[141] J. R. Koza, M. J. Streeter, and M. A. Keane, "Routine high-return human-competitive automated problem-solving by means of genetic programming," *Inf. Sci.*, vol. 178, no. 23, pp. 4434–4452, December 2008. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2008.07.028

[142] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[143] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, July 2009. [Online]. Available: http://arxiv.org/abs/0907.4728

[144] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework." *Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/mvl/mvl17.html#Alcala-FdezFLDG11

[145] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

[146] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: A survey," *Knowledge-Based Systems*, vol. 80, no. C, pp. 14–23, May 2015. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2015.01.010

[147] W. S. Chu, F. D. l. Torre, and J. F. Cohn, "Selective transfer machine for personalized facial expression analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 3, pp. 529–545, March 2017.

[148] J. G. Moreno-Torres, T. Raeder, R. Aláiz-Rodríguez, N. V. Chawla, and F. Herrera, "Tackling dataset shift in classification: Benchmarks and methods," http://sci2s.ugr.es/dataset-shift, accessed: 2015-03-30.

[149] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore, "Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records," *BioMed research international*, vol. 2014, p. 781670, 2014. [Online]. Available: http://europepmc.org/articles/PMC3996476

[150] L. Rokach, "Decision forest: Twenty years of research," *Information Fusion*, vol. 27, pp. 111–125, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253515000561

[151] J. Wroughton and T. Cole, "Distinguishing between binomial, hypergeometric and negative binomial distributions," *Journal of Statistics Education*, vol. 21, no. 1, pp. 1–15, 2013.

[152] A. H. Cheetham and J. E. Hazel, "Binary (presence-absence) similarity coefficients," *Journal of Paleontology*, vol. 43, no. 5, pp. 1130–1136, 1969. [Online]. Available: http://dx.doi.org/10.2307/1302424

[153] M. J. Warrens, "On association coefficients for 2x2 tables and properties that do not depend on the marginal distributions," *Psychometrika*, vol. 73, no. 4, pp. 777–789, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11336-008-9070-3

[154] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, no. 3, pp. 199–215. [Online]. Available: http://dx.doi.org/10.1023/A:1024099825458

[155] J. Read and J. Hollmén, *A Deep Interpretation of Classifier Chains*. Cham: Springer International Publishing, 2014, pp. 251–262. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12571-8_22

[156] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: treating targets as inputs," *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10994-016-5546-z

[157] J. Read and J. Hollmén, "Multi-label classification using labels as hidden nodes," 2015. [Online]. Available: https://arxiv.org/abs/1503.09022

[158] M. Ioannou, G. Sakkas, G. Tsoumakas, and I. Vlahavas, "Obtaining bipartitions from score vectors for multi-label classification," in *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI10)*, Oct. 2010, pp. 409–416. [Online]. Available: http://dx.doi.org/10.1109/ictai.2010.65

[159] I. Pillai, G. Fumera, and F. Roli, "Threshold optimisation for multi-label classifiers," *Pattern Recognition*, vol. 46, no. 7, pp. 2055–2065, July 2013. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2013.01.012

[160] R.-E. Fan and C.-J. Lin, "A study on threshold selection for multi-label classification," National Taiwan University, Tech. Rep., 2007. [Online]. Available: http://www.csie.ntu.edu.tw/~{}cjlin/papers/threshold.pdf

[161] L. Tang, S. Rajan, and V. K. Narayanan, "Large scale multi-label classification via metalabeler," in *Proceedings of the 18th International Conference on World Wide Web*, ser. WWW09. New York, NY, USA: ACM, 2009, pp. 211–220. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526738

[162] X. Zhang and B. G. Hu, "A new strategy of cost-free learning in the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 2872–2885, Dec 2014.

[163] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown," in *Knowledge Discovery and Data Mining*, 2001, pp. 204–213. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.8060

[164] J. Hernández-Orallo, P. Flach, and C. Ferri, "A unified view of performance metrics: Translating threshold choice into expected classification loss," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2813–2869, Oct. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503332

[165] P. A. Flach, "ROC analysis," in *Encyclopedia of Machine Learning*, 2010, pp. 869–875. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-30164-8_733

[166] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML01)*, ser. ICML01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 609–616. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655658

[167] P. Flach, "Classification in context:adapting to changes in class and cost distribution," in *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD 2014*, Nancy, France, September 2014. [Online]. Available: http://users.dsic.upv.es/~flip/LMCE2014/Papers/lmce2014_submission_18.pdf

[168] E. A. Freeman and G. G. Moisen, "A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa,"

Ecological Modelling, vol. 217, no. 1-2, pp. 48 – 58, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304380008002275

[169] C. Drummond and R. C. Holte, "Cost curves: An improved method for visualizing classifier performance." *Machine Learning*, vol. 65, no. 1, pp. 95–130, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/ml/ml65.html#DrummondH06

[170] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22Nd International Conference on Machine Learning (ICML05)*, ser. ICML05, L. D. Raedt and S. Wrobel, Eds. New York, NY, USA: ACM, 2005, pp. 625–632. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102430

[171] Y. Yang, "A study of thresholding strategies for text categorization," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR01. New York, NY, USA: ACM, 2001, pp. 137–145. [Online]. Available: http://doi.acm.org/10.1145/383952.383975

[172] C. Largeron, C. Moulin, and M. Géry, "Mcut: A thresholding strategy for multi-label classification." in *Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis*, ser. Lecture Notes in Computer Science, J. Hollmén, F. Klawonn, and A. Tucker, Eds., vol. 7619. Springer, 2012, pp. 172–183. [Online]. Available: http://dblp.uni-trier.de/db/conf/ida/ida2012.html#LargeronMG12

[173] N. Cesa-Bianchi, M. Re, and G. Valentini, "Synergy of multi-label hierarchical ensembles, data fusion, and cost-sensitive methods for gene functional inference," *Machine Learning*, vol. 88, no. 1, pp. 209–241, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10994-011-5271-6

[174] J. Hernández-Orallo, P. Flach, and C. Ferri, "Brier curves: a new cost-based visualisation of classifier performance." in *Proceedings of the 28th International Conference on Machine Learning (ICML11*, ser. ICML11, L. Getoor and T. Scheffer, Eds. Omnipress, 2011, pp. 585–592. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2011.html#Hernandez-OralloFR11

[175] P. Flach, J. Hernández-Orallo, and C. Ferri, "A coherent interpretation of auc as a measure of aggregated classification performance," in *Proceedings of the 28th International Conference on Machine Learning (ICML11)*, L. Getoor and T. Scheffer, Eds., 2011.

APPENDIX A

APPENDIX: DETAILED EXPERIMENTAL RESULTS

This appendix is devoted to more experimental results. It includes supplementary table and cost curves for multi-label threshold choice methods that have been studied in Chapter 5.

A.1 Supplementary Tables

Table A.1 shows the empirical loss for 13 datasets using the cost-driven uniform (CDU) threshold choice method. We selected one label randomly and used its misclassification cost as a threshold for other labels. It demonstrates that the overall expected loss can be expressed in terms of both Brier score and MAE. The expected loss for the selected label will be equal to Brier score, whereas for other labels will be equal to the MAE.

A.2 Cost Curves of Multi-label Threshold Choice Methods

Figure A.1 compares the cost curves of the global methods for nine datasets in two cases: shared and per-label costs. The global method assigns only a single threshold for all labels at deployment. Figure A.2 shows the cost-driven uniform approach (CDU), which selects one label randomly and uses its cost as a threshold for other labels. The x-axis represents the selected label cost and the y-axis is the loss averaged over all labels. The pink curve is the cost-driven (CD) curve as all labels have the same cost. Whereas Figure A.3 shows the cost curves by applying multiple threshold methods, label-wise and instance-wise. In all figures, we used logistic regression algorithm as a base classifier. Figures A.4, A.5 and A.6 repeat the same approaches on the same datasets but using J48 as a base classifier.

Table A.1: Empirical loss of the cost-driven uniform (CDU) method over multiple datasets and a range of misclassification costs. l_j, G_j, BS_j, $\frac{G}{q-1}$, $\frac{MAE}{q-1}$ and q denote the selected label, selected label loss, the selected label Brier score, the loss averaged over all other labels, the Mean Absolute Error averaged over all other labels and the number of labels, respectively.

Dataset	l_j	G_j	BS_j	$\frac{G}{q-1}$	$\frac{MAE}{q-1}$
Corel5k	73	0.025	0.028	0.029	0.016
Cal500	164	0.043	0.047	0.206	0.200
Bibtex	27	0.008	0.009	0.014	0.019
Language log	67	0	0	0.035	0.026
Enron	28	0.006	0.006	0.071	0.072
Medical	12	0.011	0.012	0.034	0.033
Genbase	5	0.036	0.065	0.020	0.043
Slashdot	20	0.027	0.030	0.058	0.062
Birds	7	0.058	0.057	0.063	0.085
Yeast	12	0.186	0.182	0.272	0.285
Flags	5	0.167	0.164	0.360	0.385
Emotions	1	0.139	0.140	0.286	0.342
Scene	6	0.092	0.092	0.152	0.153

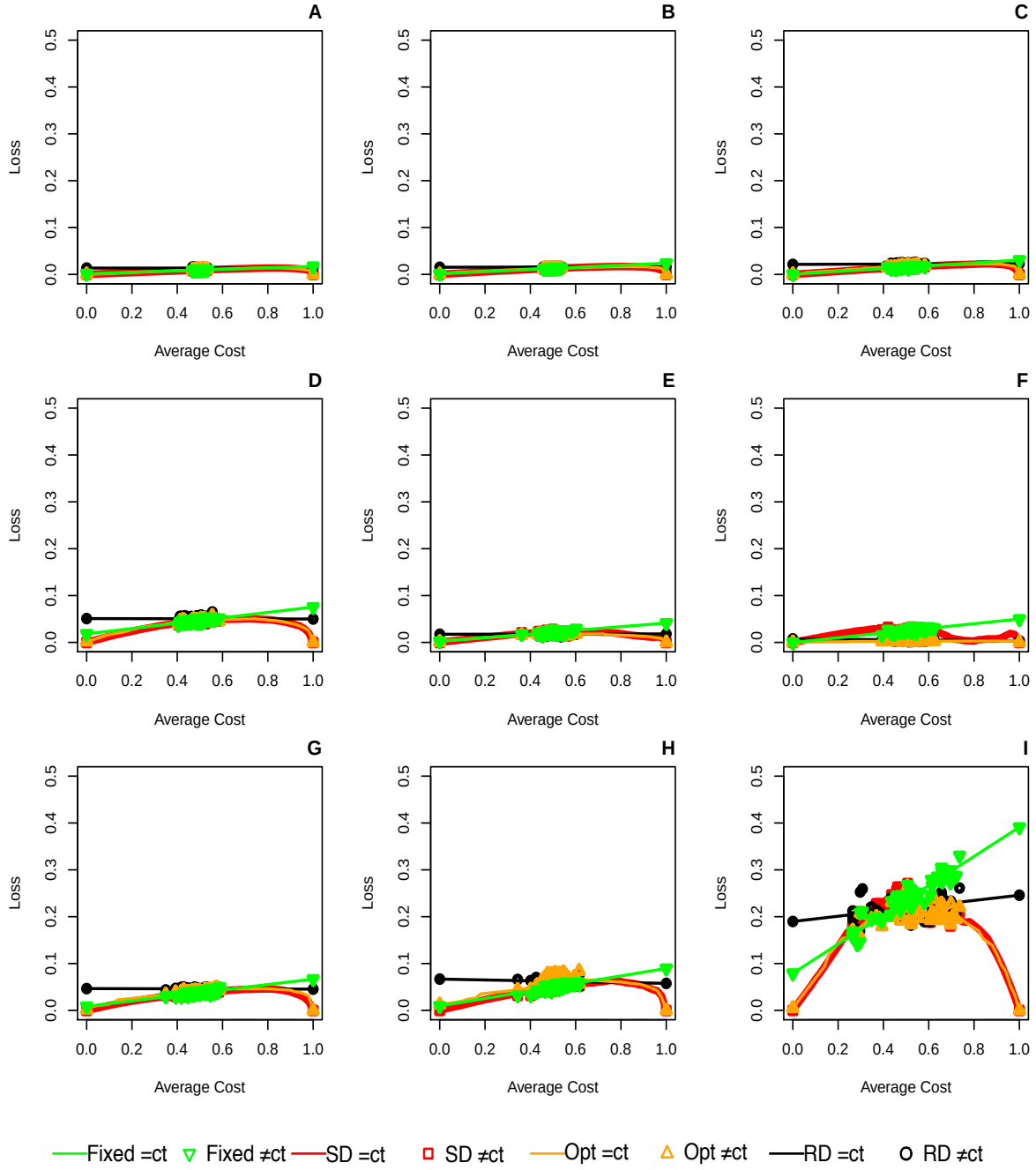


Figure A.1: Cost curves for global threshold methods using shared costs ($= ct$) and per-label cost ($\neq ct$). Logistic regression is the base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.

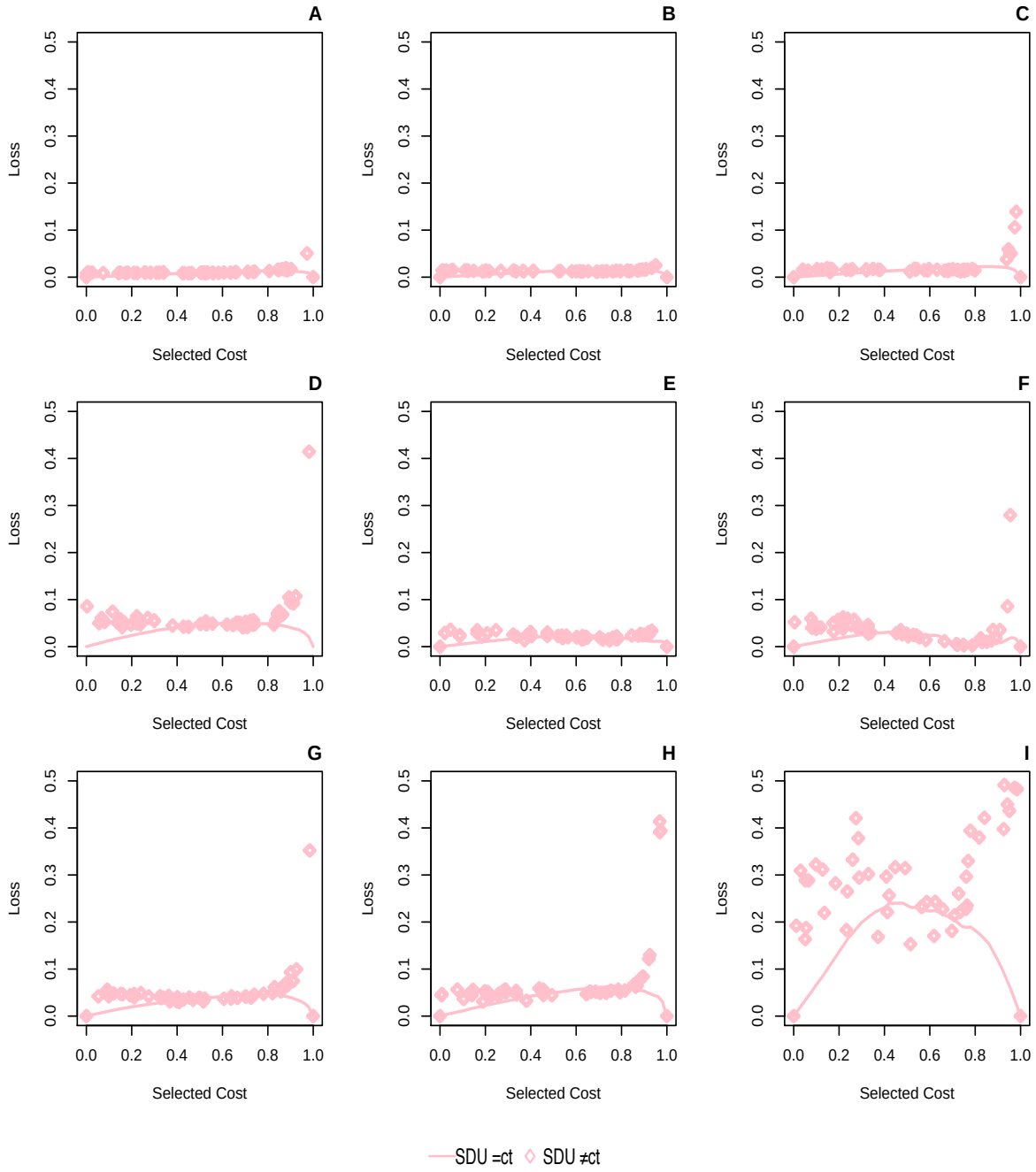


Figure A.2: Cost curves for the cost-driven uniform (CDU) method where the x-axis represents the selected label cost which has been used as a threshold for all labels. Logistic regression is the base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.

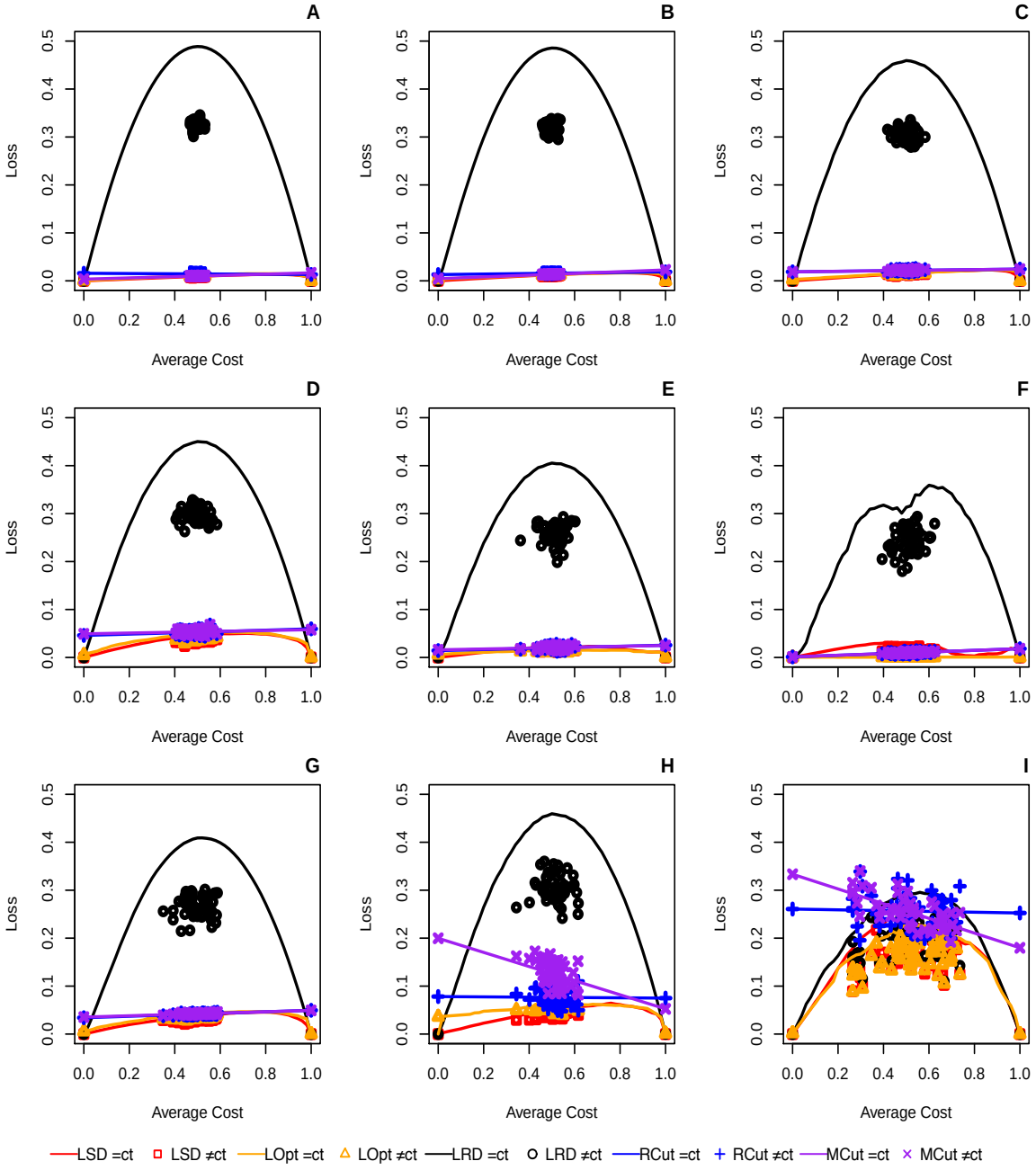


Figure A.3: Cost curves for multiple threshold methods using shared cost ($= ct$) and per-label cost ($\neq ct$). Logistic regression is used as a base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.

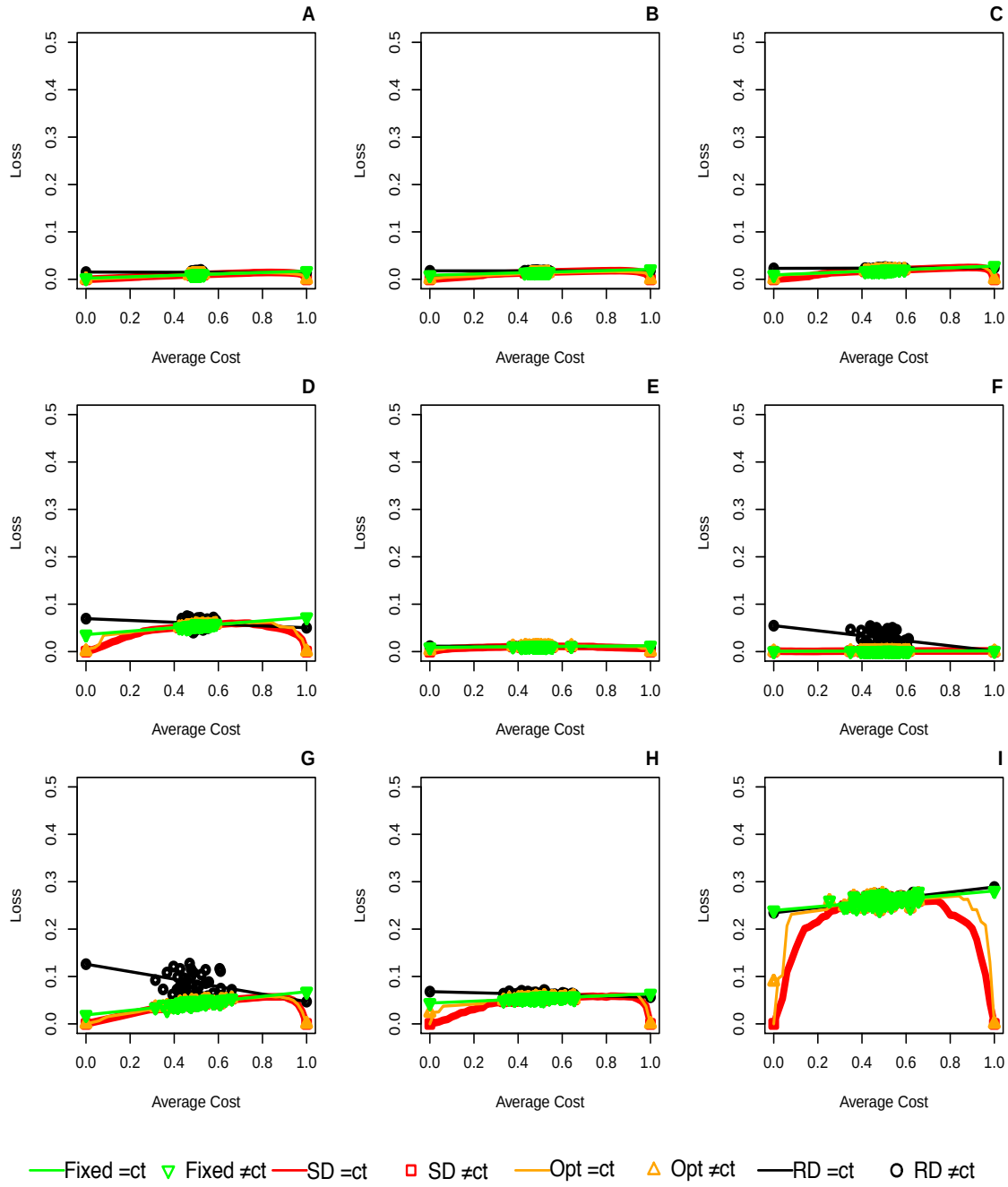


Figure A.4: Cost curves for global threshold methods using shared costs ($= ct$) and per-label cost ($\neq ct$). J48 is the base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.

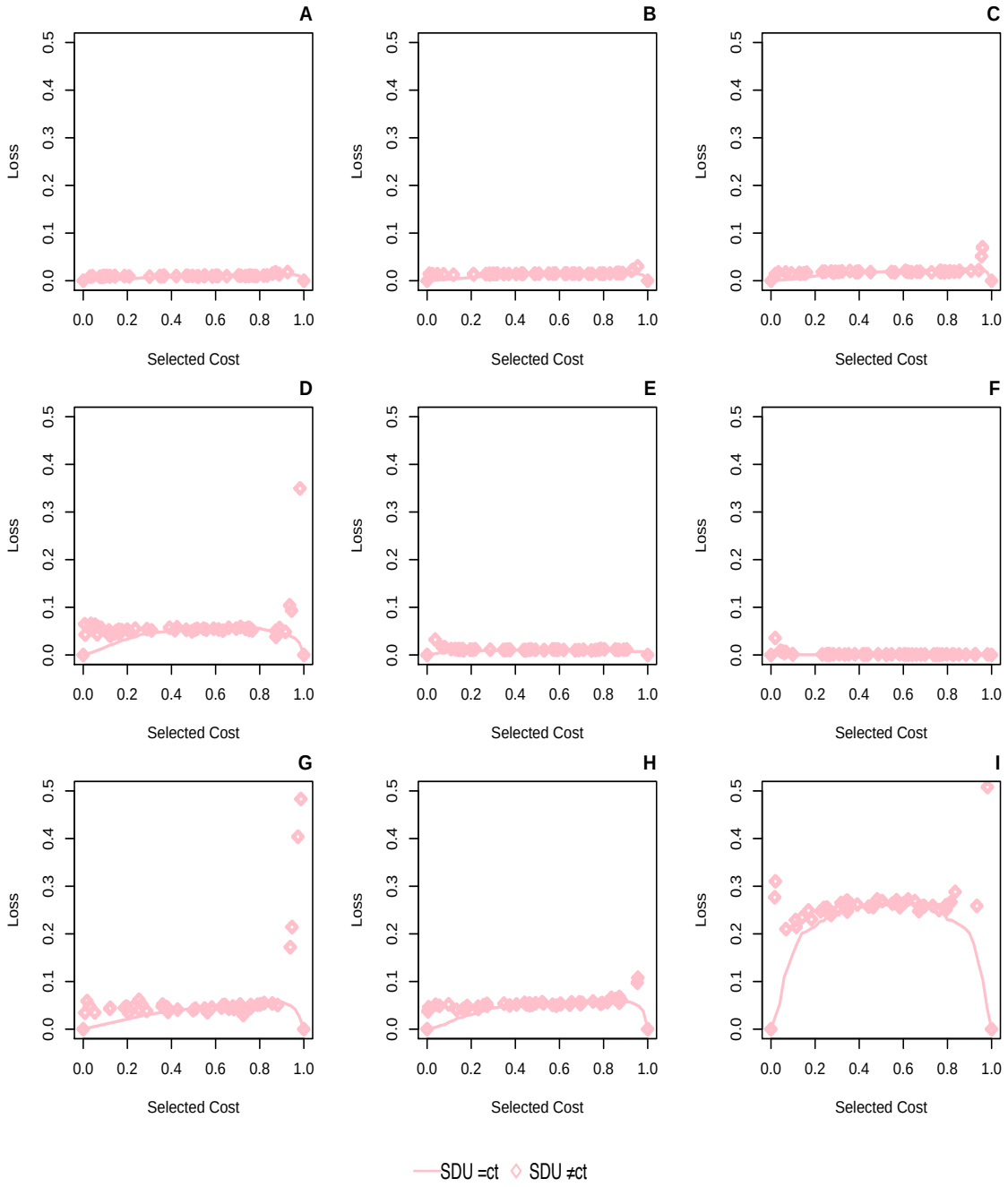


Figure A.5: Cost curves for the cost-driven uniform (CDU) method where the x-axis represents the selected label cost which has been used as a threshold for all labels. J48 is the base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.

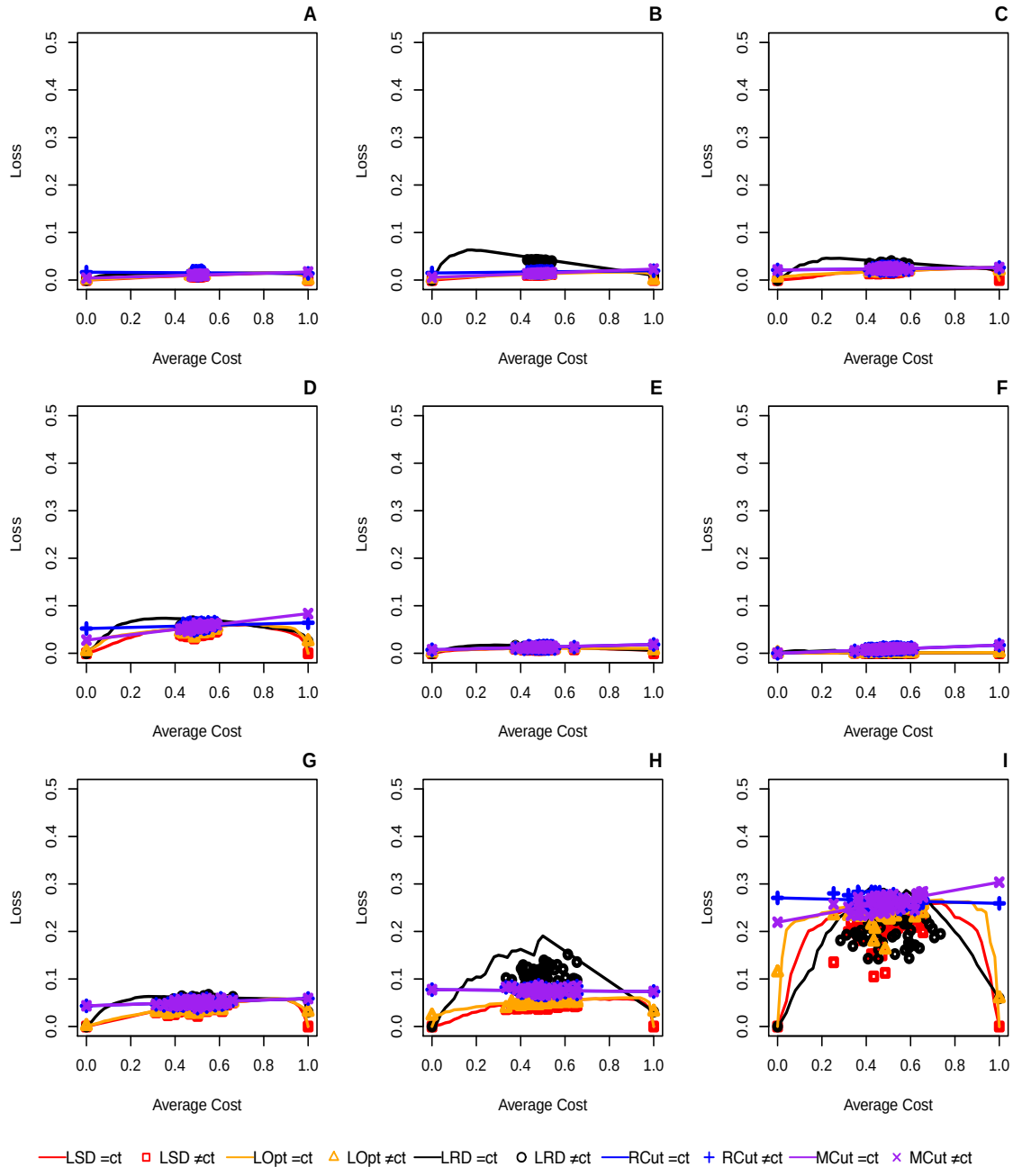


Figure A.6: Cost curves for multiple threshold methods using shared cost ($= ct$) and per-label cost ($\neq ct$). J48 is used as a base classifier, the datasets are: A) Corel5k B) Bibtex C) Language log D) Enron E) Medical F) Genbase G) Slashdot H) Birds I) Emotions.