ขั้นตอนวิธีการแบ่งกั้นเวียนเกิดแบบการปรับดุลด้วยตนเองสำหรับปัญหาการจำแนกประเภท

นายอาทิตย์ สกุลเมือง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรดุษฎีบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

SELF-BALANCING RECURSIVE PARTITIONING ALGORITHM FOR CLASSIFICATION PROBLEMS

Mr. Artit Sagoolmuang

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy Program in Applied Mathematics and Computational Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2019

Dissertation Title	SELF-BALANCING RECURSIVE PARTITIONING ALGORITHM FOR CLASSIFICATION PROBLEMS
By	Mr. Artit Sagoolmuang
Field of Study	Applied Mathematics and Computational Science
Dissertation Advisor	Assistant Professor Krung Sinapiromsaran, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

..... Dean of the Faculty of Science
(Professor Polkit Sangvanich, Ph.D.)

DISSERTATION COMMITTEE

..... Chairman
(Assistant Professor Boonyarit Intiyot, Ph.D)

..... Dissertation Advisor
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

..... Examiner
(Assistant Professor Kitiporn Plaimas, Ph.D)

..... Examiner
(Thap Panitanarak, Ph.D)

..... External Examiner
(Assistant Professor Chumphol Bunkhumpornpat, Ph.D.)

อาทิตย์ สกุลเมือง : ขั้นตอนวิธีการแบ่งกั้นเวียนเกิดแบบการปรับดุลด้วยตนเองสำหรับปัญหาการจำแนกประเภท. (SELF-BALANCING RECURSIVE PARTITIONING ALGORITHM FOR CLASSIFICATION PROBLEMS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร. กรุง สินอภิรมย์สราญ, 125 หน้า.

การสร้างตัวแบบการจำแนกประเภทที่มีประสิทธิภาพมีบทบาทสำคัญในระเบียบวิธีการค้นพบความรู้ในฐานข้อมูลในช่วงหลายปีที่ผ่านมา อย่างไรก็ตามประเด็นสำคัญที่ส่งผลกระทบต่อประสิทธิภาพในการจำแนกประเภทอย่างมีนัยสำคัญเรียกว่าปัญหาความไม่ดุลระหว่างกลุ่ม ซึ่งปรากฏในสถานการณ์โลกจริงจำนวนมาก ในวิทยานิพนธ์นี้นำเสนอตัวแบบการจำแนกประเภทที่สร้างขึ้นตามขั้นตอนวิธีการแบ่งกั้นเวียนเกิดที่ได้รับการปรับปรุงภายใต้แนวคิดของส่วนประกอบเอนโทรปีที่ถูกดัดแปลง เพื่อจัดการกับปัญหาการจำแนกประเภทโดยไม่คำนึงถึงสถานการณ์ความไม่ดุลระหว่างกลุ่ม ระเบียบวิธีสามรูปแบบถูกนำเสนอด้วยจุดมุ่งหมายที่ต่างกัน ระเบียบวิธีแรกถูกนำเสนอเพื่อจำแนกชุดข้อมูลที่ไม่ดุลทวิคลาส สำหรับลักษณะประจำเชิงตัวเลขเท่านั้น และจากนั้นวิธีดังกล่าวถูกเพิ่มสมรรถนะเพื่อให้จัดการกับกรณีหลายคลาสที่ปรากฏในระเบียบวิธีที่สอง สำหรับระเบียบวิธีที่สามสามารถทำงานกับชุดข้อมูลที่ประกอบด้วยลักษณะประจำเชิงตัวเลขและลักษณะประจำแบบเด็ดขาด ผลการทดลองกับทั้งชุดข้อมูลสังเคราะห์และชุดข้อมูลโลกจริง จากคลังข้อมูลยูซีไอแสดงให้เห็นว่าระเบียบวิธีที่นำเสนอเหล่านี้ให้ผลที่ดีกว่าวิธีการอื่นที่มีอยู่อย่างมีนัยสำคัญ

ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	ลายมือชื่อนิสิต	.....
		ลายมือชื่อ อ.ที่ปรึกษาหลัก	.....
สาขาวิชา	คณิตศาสตร์ประยุกต์และวิทยาการคณนา		
ปีการศึกษา	2562		

6072834623 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS : RECURSIVE PARTITIONING ALGORITHM / DECISION TREE / CLASSIFICATION PROBLEM / CLASS IMBALANCED PROBLEM

ARTIT SAGOOLMUANG : SELF-BALANCING RECURSIVE PARTITIONING ALGORITHM FOR CLASSIFICATION PROBLEMS. ADVISOR : ASST. PROF. KRUNG SINAPIROMSARAN, Ph.D., 125 pp.

Creating an effective classification model has been played an important role in knowledge discovery in a database methodology for the past several years. However, there is a critical issue that significantly affects the classification performance appearing in many real-world situations, which is called a class imbalanced problem. In this dissertation, a classification model built based on the recursive partitioning algorithm is improved under the concept of modified entropy components for handling a classification problem regardless of the class imbalanced situation. Three methodologies are introduced for achieving different purposes. The first methodology is presented to classify a binary-class imbalanced dataset dealing only with numeric attributes, and then it is enhanced to deal with the multi-class case in the second methodology. The third methodology is designed to work with a dataset consisting of both numeric attributes and categorical attributes. The experimental results on both synthetic datasets and real-world datasets from the UCI repository show that these proposed methodologies significantly outperform other existing methods.

Department	: Mathematics and Computer Science	Student's Signature	.....
		Advisor's Signature	.....
Field of Study	: Applied Mathematics and Computational Science		
Academic Year	: 2019		

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my dissertation advisor Assistant Professor Dr. Krung Sinapiromsaran for his salutary suggestions and encouragement for the time throughout my doctorate degrees. He did not just offer gainful guidance for this research, he also provided all kinds of valuable advice in my life.

Also, I am grateful to the rest of my dissertation committee consisting of Assistant Professor Dr. Boonyarit Intiyot, Assistant Professor Dr. Kitiporn Plaimas, Dr. Thap Panitanarak, and Assistant Professor Dr. Chumphol Bunkhumpornpat for their useful comments and suggestions on my research.

In addition, I would like to thank the Applied Mathematics and Computational Science (AMCS) Program in the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University for all resources I used during the duration of my education, and Science Achievement Scholarship of Thailand (SAST) for the financial support.

Finally, I am thankful to my family and my friends, especially Miss Chittima Chiamanusorn, Mr. Perasut Rungcharassang, Mr. Ampol Duangpan and everybody in the AMCS laboratory for their supporting and sharing useful knowledge.

CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Due to the rapid increase in the amount of information accumulated continuously, the automated or convenient tools for analyzing them are in great demand. Knowledge discovery in database (KDD) [1] is the non-trivial process of identifying novel, valid, understandable, and potentially useful patterns from a large and complex dataset. It involves tools from various fields of knowledge, such as mathematics, statistics, computer programming, database management, including specific domain expertise. The process of KDD is displayed in Figure 1.1, which consists of numerous steps to achieve the determined goals.

1. The *selection* step is the initial preparatory step aiming to define the target dataset. It relates to analyst's understanding of the application domains and the data domains, the availability of data, and then assembling all data into a single dataset along with all attributes that will be used in the next process.

2. The *preprocessing* step enhances the reliability and quality of the dataset. It covers the verification of completeness and redundancy, the treatment of the missing values, the detection of noises and outliers, etc.

3. The *transformation* step is the process to convert the dataset from one format/ structure into another format/structure depending on the goals. It consists of the feature selection, the feature extraction, the dimensionality reduction, and the attribute transformation.

4. The *data mining* step is the most important step of all processes in the KDD process aiming to discover or extract or "mine" formerly unknown patterns from the dataset that is provided from previous steps.

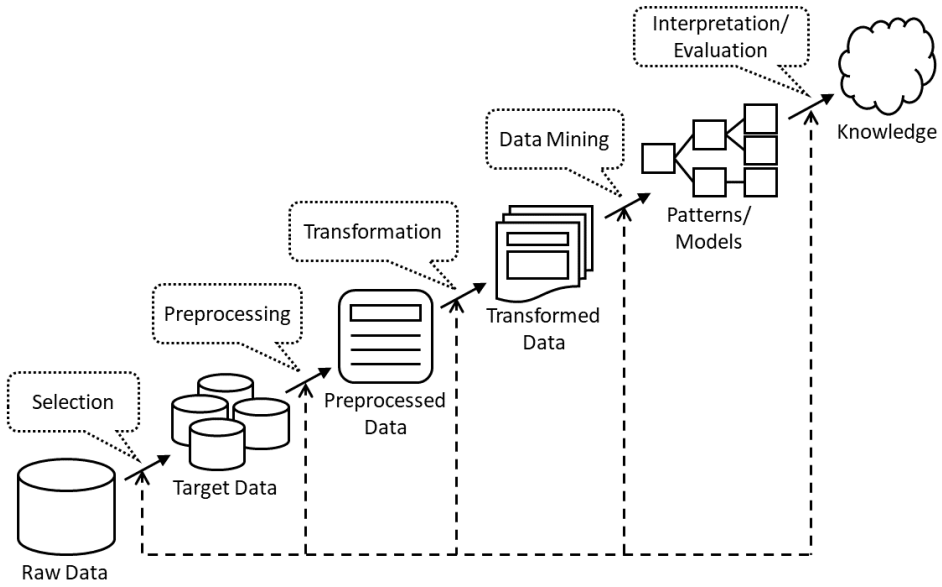


Figure 1.1: KDD process

5. The *interpretation/evaluation* step concentrates on the comprehensibility and usefulness of the derived patterns with respect to the goals. Then, the current usage and the overall feedbacks are deployed.

For the data mining task, there are a variety of methods that have been proposed for achieving different objectives. They can be roughly divided into two main categories that are descriptive methods and predictive methods.

The descriptive methods focus on transforming a dataset to make it easier in interpreting and understanding the nature for an analyst. It performs on a dataset without any prespecified provision nor dependent variable. The examples of methods in this group include the followings.

- The *summarization* involves the techniques to encounter the compact description of a dataset. The basic summarization methods usually relate to statistical techniques. For example, the mean, the median, and the mode are used for measuring the central tendency of each attribute, while the range, the interquartile range, the variance, and the standard deviation are used for measuring the dispersion of the numeric attribute.

- The *visualization* may be classified as a part of summarization aiming to inspect a dataset through graphical means. It is able to help identify relations, trends, and biases that are hidden in a dataset. Many techniques are used for visualizing a dataset covering the boxplot, the histogram, the scatter-plot matrices, and the 3D plot.

- The *association rule induction* is applied to discover the relationships or the frequent patterns among a set of data items in a large database. It is used in the market basket analysis [2], which looks for the combinations of items occurring together in the transactions frequently.

- The *clustering* concentrates on identifying groups of instances based on the structure of a dataset. Instances that have similar characteristics will be in the same cluster, while instances that have distinct characteristics will be in different clusters. The similarity for clustering is determined by various measures, such as the distance among the instances and the density around instances. The examples of clustering techniques include k-mean clustering [3], hierarchical clustering [4], and DBSCAN [5].

The predictive methods are oriented to build a model from the discovered relationship between a set of input attributes (independent variables) and target attributes (dependent variables). Generally, the models are constructed based on inductive learning from a sufficient number of labeled instances. For instance, the regression analysis [6] concerns with building the model to predict the target values of instances being in an infinite real-valued domain.

Additionally, classification is another significant predictive method that has been studied continuously from the past to the present. It performs on a dataset that its target domain is finite and discrete known as a class label. A classification algorithm will be responsible to learn from a training dataset, in which each class instance is labeled, to create a classification model or a classifier that can predict or classify the class of unlabeled instances. Many well-known classification models along with their algorithms

have been presented based on different concepts. For example, the k-nearest neighbors classifier (KNN) [7] uses the concept of similarity, the Naïve Bayes model [8] and the logistic regression [9] rely on the knowledge of probability and statistics, a decision tree [10] and decision rules [11] apply the recursive partitioning method based on properties of all attributes, a support vector machine (SVM) [12] and a linear discriminant model [13] relate to the linear separation, and an artificial neural network (ANN) [14] and a deep learning [15] simulates the work of the human brain.

This dissertation concentrates on the decision tree which has received attention due to its popularity both in the fields of education, business, including science and technology [16]. It is a tree-based classification model consisting of multiple connected nodes. Each non-leaf node represents a specific splitting condition for traversing to a child node. Each leaf node indicates a specific class for labeling instances reaching to that node. To induce a decision tree, the recursive partitioning algorithm is widely used. The set of instances in each non-leaf node is recursively partitioned until a child node contains all instances from the same class which is used for labelling the leaf node. Many studies recognize a decision tree as one of the most prevailing models to learn a dataset [17, 18, 19]. Especially for the latest in 2019 [20], it was voted to be the second-ranked of the top used methods in data science and machine learning, which is the most popular model for the classification task.

The success of the decision tree can be explained by various factors. First, a decision tree algorithm consumes small computational time to build a model. Moreover, it also spends negligible time in predicting the classes of unlabeled instances. Second, the mechanism of a decision tree is not complicated, which is able to be easily interpreted for humans. Third, a decision tree is robust. It is still effective even if a dataset contains some anomalies and missing values.

Nevertheless, similar to other methods, a decision tree algorithm usually faces the hassle of constructing a non-bias model from a dataset that has extremely unequal class distribution [21]. The built decision tree tends to classify most unlabeled instances to be

in the classes having a large number of instances (called majority classes), causing a lot of misclassification for instances in other classes having a tiny number of instances (called minority classes). This problem is known as the **class imbalanced problem** playing an important role in data mining over the past several years. It can be found abundantly in numerous real-world situations, in which the minority classes have normally been more focused on classifying correctly than the majority ones. For example, in disease diagnosis [22, 23, 24], there is a small number of patients compared with normal people, but they are important and must be discovered. Similarly in fraud detection [25, 26], the prediction of fraudulent transactions is more focused than non-fraudulent cases. Furthermore, the class imbalanced problem also appears in network intrusion detection [27], industrial systems monitoring [28], e-mail spam filtering [29], protein/DNA identification [30], target detection [31], activity recognition [32], sentiment analysis [33], text mining [34], video mining [35], and many other interesting issues [36]. Therefore, it is absolutely necessary to deal with this issue carefully.

Adjusting the decision tree algorithm for building a decision tree from an imbalanced dataset has been continuously introduced [37]. One approach handles the class imbalanced problem directly without changing the classification algorithm of the dataset such as the sampling method, another commonly used technique. Normally, the improvement of the decision tree algorithm usually focuses on modifying the splitting measures, which are used to select the splitting condition for each non-leaf node. Traditional splitting measures, especially the Shannon's entropy (SE) [38] using in the ID3 algorithm [10] and the C4.5 algorithm [39], have been improved with various concepts. Modifying the components of SE is the latest concept presented by Boonchuay and her colleagues in 2017 [40]. Their proposed splitting measure, the minority entropy or ME, offers significantly better performance than other measures, even if the original structure of the SE formula remains the same. However, it lacks the important properties of the decision tree being able to work with a multi-class dataset consisting of categorical attributes, which are the main issues leading to this dissertation.

1.1 Research Objectives

The objectives of this dissertation are to propose new recursive partitioning algorithms for building the decision trees being able to classify both binary-class and multi-class balanced and imbalanced datasets dealing with both numeric attributes and categorical attributes effectively. Moreover, an empirical, experimental and theoretical analysis of each proposed method are also demonstrated.

1.2 Proposed Methodologies

To achieve the objectives above, three methodologies are introduced in this dissertation.

- **Methodology I**: Decision tree induction for a binary-class imbalanced dataset dealing with numeric attributes

  The concept of modified entropy components is enhanced by applying the outlier detection to present the recursive partitioning algorithm for building the decision tree to classify a binary-class imbalanced dataset dealing only with numeric attributes.

- **Methodology II**: Decision tree induction for a multi-class imbalanced dataset dealing with numeric attributes

  The concept of modified entropy components is expanded to present the self-balancing recursive partitioning algorithm for building the decision tree to classify a multi-class imbalanced dataset dealing only numeric attributes.

- **Methodology III**: Decision tree induction for a multi-class imbalanced dataset dealing with numeric and categorical attributes

  The proposed self-balancing recursive partitioning algorithm is generalized from methodology II to classify a multi-class imbalanced dataset dealing with both numeric attributes and categorical attributes.

1.3 Dissertation Overview

The remainder of this dissertation is organized as follows. In the second chapter, a brief review of the required background knowledge, including some related works are shown. Then, the third, the fourth and the fifth chapters demonstrate the detail of methodologies I, II and III, respectively. They also state properties, examples, algorithms, and evaluations of each method as well. Finally, the findings of this dissertation are concluded in the sixth chapter along with their directions for further research.

CHAPTER II

BACKGROUND KNOWLEDGE AND RELATED WORKS

This chapter offers more details about the background knowledge that is necessary for understanding this dissertation, which covers classification, a decision tree, a class imbalanced problem, and a decision tree for an imbalance dataset. In addition, it also includes a review of some interesting related works.

2.1 Overview of Classification Process

Classification is a process to extract information by class characteristics from a dataset, which consists of two steps shown in Figure 2.1, i.e. the learning step (training phase) and the classifying step (testing phase). In the first step, the classification model is built from the training dataset (the set of labeled instances) using the classification algorithm. Then, in the second step, the classification model is used to predict the classes of unlabeled instances in the testing dataset with known class labels.

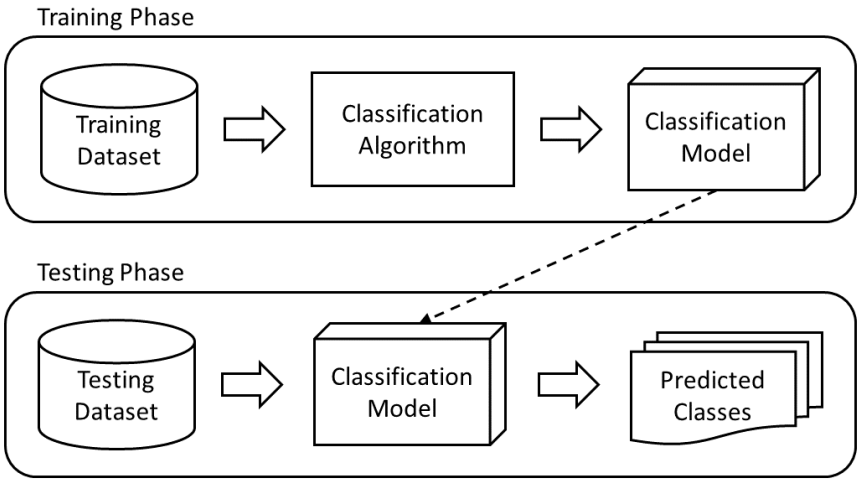


Figure 2.1: Classification process

To evaluate the performance of each classification model, it is applied to classify a labeled dataset and then measure differences between the predicted class and the actual class. However, the model is normally overfitted with a dataset that is used to build from the training dataset, so it tends to correctly predict most instances in that dataset while it may misclassify unseen instances in the testing dataset. Accordingly, the set of instances used in the training process and the testing process should not be the same. There are two methods that are widely used to split a dataset, i.e. the holdout method and cross-validation.

- For the *holdout* method, it directly divides a dataset into two subsets by the randomization.

  1. The *training dataset* is for building the classifier.
  2. The *testing dataset* is for assessing the classifier.

  Moreover, sometimes, a dataset may be separated into three subsets that are the training dataset, the testing dataset, and the validating dataset. For the validating dataset, it is used for parameter tuning that is normally applied before evaluating the model with the testing dataset.

- For *cross-validation*, it divides a dataset into k partitions, in which k is the specific parameter. Then, each partition is employed as the testing dataset and others are used for train. The process is repeated for all k partitions. Accordingly, any instance is treated as a testing instance only once during the evaluation. Moreover, there is a special case of cross-validation that takes only one instance for the testing dataset, called the *leave-one-out* method. It normally uses when the size of the dataset is small.

2.2 Training Dataset in Classification

Mathematically, the number of instances, the number of input attributes, and the number of classes in a training dataset are denoted by m, n, and p, which are indexed by

i, j, and k, respectively. Let $\mathbf{A} = \{A_j \mid j = 1, 2, ..., n\}$ be a set of input attributes, and y be a target attribute (class label). The input space is defined as a Cartesian product of all input attribute domains denoted by X, i.e. $X = dom(A_1) \times dom(A_2) \times ... \times dom(A_n)$. For the domain of the target attribute, it represents a set of classes denoted by C, i.e. $C = dom(y) = \{c_k \mid k = 1, 2, ..., p\}$. Then, define $\mathbf{D} = \{(\vec{x}_i, y_i) \mid i = 1, 2, ..., m\}$ where each instance is in $X \times C$ be a training dataset represented by Figure 2.2, in which $\vec{x}_i = \langle x_1^i, x_2^i, ..., x_n^i \rangle$. Therefore, $\mathbf{D}$ can be separated into p partitions according to the class label, i.e. $\mathbf{D} = \mathbf{D}_1 \cup \mathbf{D}_2 \cup ... \cup \mathbf{D}_p$, where $\mathbf{D}_k = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid y_i = c_k \text{ for } i = 1, 2, ..., m\}$ having size m_k, such that $m_1 + m_2 + ... + m_k = m$. Note that, dataset $\mathbf{D}$ will be called a binary-class dataset or a multi-class dataset, when there are two classes ($p = 2$) and more than two classes ($p > 2$), respectively. For a classification model, it is the function that maps the input space to the set of classes, i.e. $f : X \to C$.

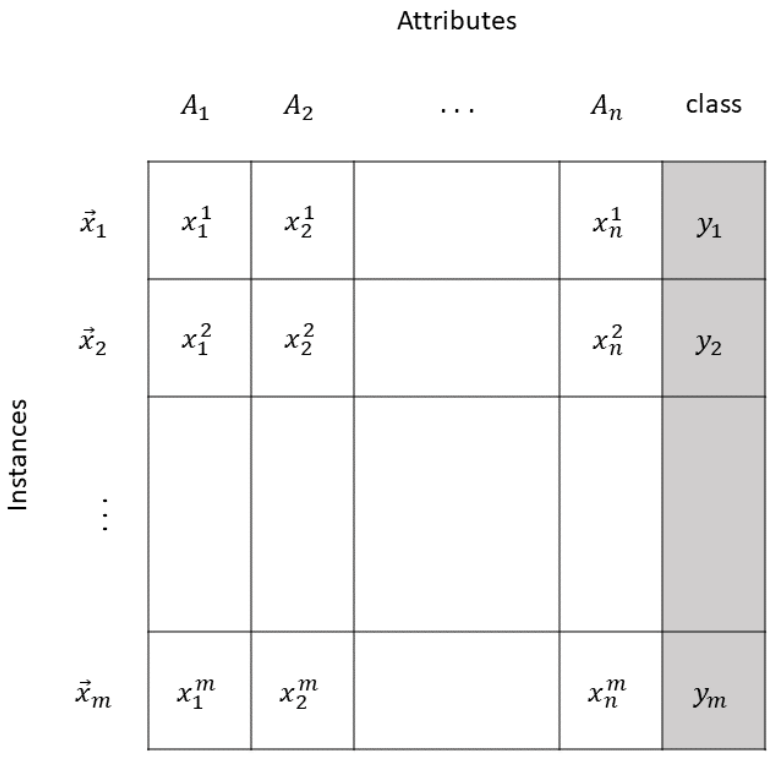


Figure 2.2: A structure of training dataset $\mathbf{D}$

2.3 Decision Tree

A decision tree is a type of the rule-based classification model. Graphically, it has a flowchart-like tree structure as shown in Figure 2.3 consisting of a set of nodes connected by edges or branches. The nodes can be divided into three types.

- The *root node* is the node that has no incoming edges with two or more outgoing edges. It represents the condition to select the next level child nodes for each evaluated instance.
- The *internal nodes* are the nodes that have exactly one incoming edge with two or more outgoing edges. They also represent the condition to select the next level child nodes for the evaluated instance reaching them.
- The *leaf nodes* or *leaves* are the nodes that have exactly one incoming edge with no outgoing edges. They represent the specific class to assign to the evaluated instance reaching them.

For each branch, it represents an outcome of the condition. An instance from the node at the beginning of the branch will be forwarded to the leaf node at the terminal of the path if it satisfied the path's outcome.

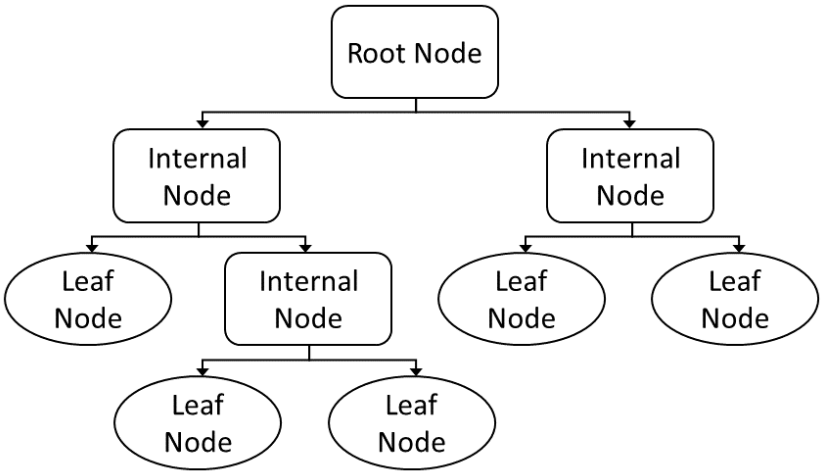
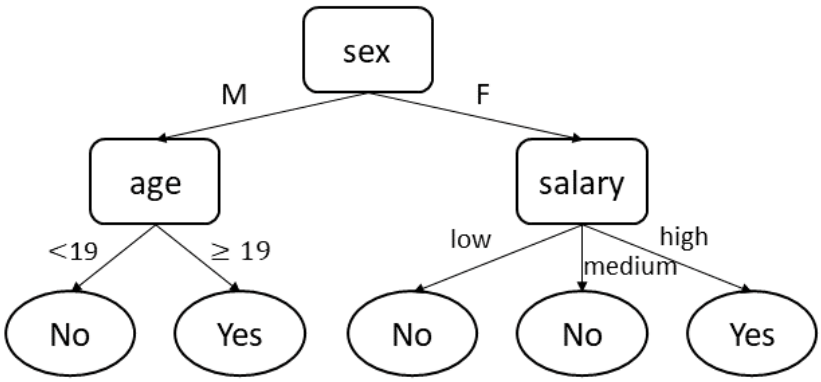
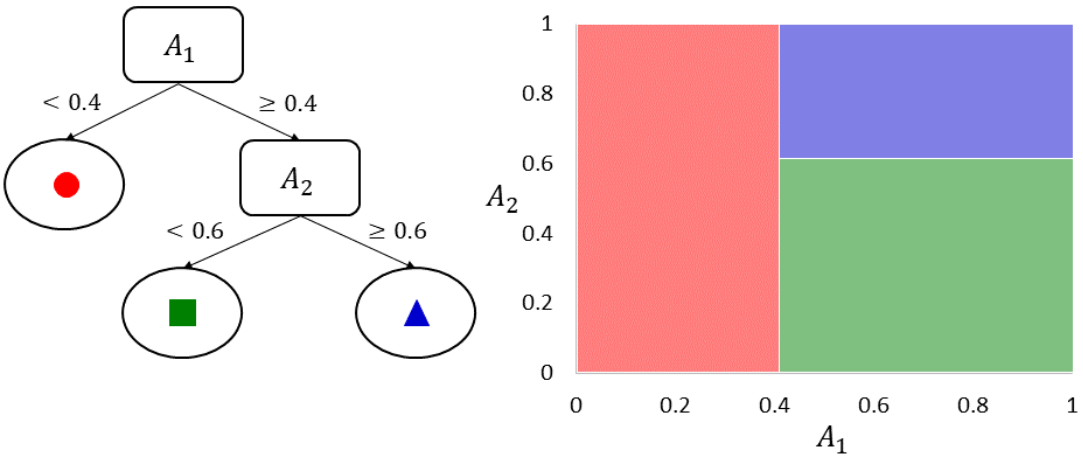


Figure 2.3: A structure of the decision tree

Two decision tree examples are shown in Figure 2.4(a) and Figure 2.4(b). For the first example, it predicts whether people are likely to purchase a car. Each non-leaf node represents the condition on the attribute, in which each outcome is shown in each branch. For each leaf node, it represents the class, either buy a car ("Yes") or not ("No"). In the case that all input attributes are continuous values, it gives the result as dividing the input space, as shown in the second example. The space $\mathbb{R}^2$ is divided into three parts for each class, i.e. the red-circle class, the green-square class, and the blue-triangle class.



(a)



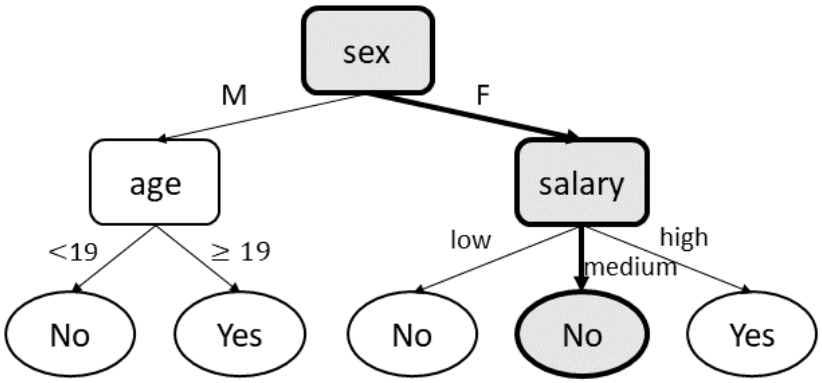
(b)

Figure 2.4: Two decision tree examples

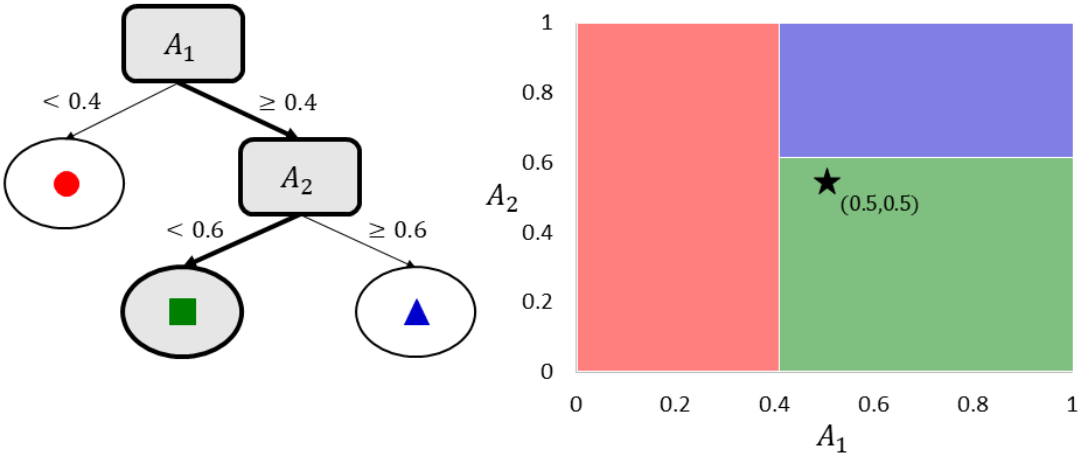
To classify an unlabeled instance, the decision tree evaluates this instance at the root node and moving through the internal nodes until the leaf node is encountered. For example, in Figure 2.5(a), thirty-two years old woman who has a medium salary is

predicted that she will not buy a car. While in Figure 2.5(b), the unlabeled instance which is represented by the star locating at (0.5,0.5) is classified to be the green-square class.

Gender	Salary	Age	Buy car?
Female	Medium	32	?



(a)



(b)

Figure 2.5: Classifying unlabeled instances using the decision trees

2.3.1 Recursive Partitioning Algorithm

A task of building a decision tree is usually called decision tree induction. It constructs the tree by learning from a class label of training instances. Most existing decision tree induction methods build the tree in the top-down fashion [41] using the recursive partitioning algorithm, which is derived from the concept learning system (CLS) [42].

The pseudocode of inducing the decision tree from training dataset $\mathbf{D}$ using the recursive partitioning algorithm is displayed by Algorithm 2.1. It starts with creating the root node for the entire training dataset. If all instances come from the same class, the leaf node is created corresponding to that class. Otherwise, dataset $\mathbf{D}$ will be separated into a certain number of partitions, denoted by q, using the selected splitting condition. Then, the algorithm is performed recursively on each partition until all instances come from the same class or meeting other stopping criteria such as the specified tree depth.

Algorithm 2.1: RecursivePartitioning($\mathbf{D}$)

Input: a set of instances $\mathbf{D}$
Output: a decision tree

```
• create the root node of the tree
if all instances in D are in the same class then
    return the leaf node with respect to that class
else
    • select the splitting condition
    • separate the dataset into q partitions corresponding to the
      outcomes of splitting condition: D = D^(1) ∪ D^(2) ∪ ... ∪ D^(q)
    • iterate for each partition: RecursivePartitioning(D^(l)) for
      l = 1, 2, ..., q
```

Traditionally, the condition for splitting a set of instances in each non-leaf node is selected based on a single attribute. The attribute that provides the optimal value of a specific splitting measure will be chosen. For each attribute, all possible scenarios to partition the dataset called candidates are all considered, which differ according to the type of attributes.

- *Categorical attribute*: The dataset will be divided into subsets based on the number of possible values of the categorical attribute, see Figure 2.6. Let A_j be a categorical attribute of training dataset $\mathbf{D}$ having q distinct values, $\{A_j^{(1)}, A_j^{(2)}, ..., A_j^{(q)}\}$. The outcomes of the splitting condition based on attribute A_j correspond directly to all of its values. A branch is generated for each value. For $l = 1, 2, ..., q$, partition $\mathbf{D}^{(l)}$ is the subset of instances in $\mathbf{D}$ that their values of attribute A_j is $A_j^{(l)}$, i.e. $\mathbf{D}^{(l)} = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid x_j^i = A_j^{(l)} \text{ for } i = 1, 2, ..., m\}$. Since all instances in each partition have the same value of attribute A_j, this attribute A_j will not be used for splitting in any child node.

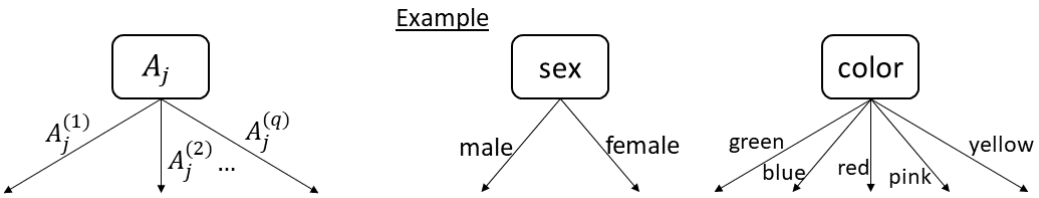


Figure 2.6: Partitioning the set of instances using the categorical attributes

- *Numeric attribute*: Due to the infinite possibility of values, partitioning the set of instances by a numeric attribute could not use all values as the outcomes of the splitting condition. In general, it usually uses a particular value called the splitting value to partition the dataset into two subsets ($q = 2$), see Figure 2.7. Let A_j be the numeric attribute of training dataset $\mathbf{D}$, and a_0 be the splitting value. Two branches are created for the values of attribute A_j that are less than a_0, and are greater than or equal to a_0. The left partition and the right partition are the subsets of instances in $\mathbf{D}$ defined as $\mathbf{D}^{(\text{left})} = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid x_j^i < a_0 \text{ for } i = 1, 2, ..., m\}$ and $\mathbf{D}^{(\text{right})} = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid x_j^i \geq a_0 \text{ for } i = 1, 2, ..., m\}$. To select value a_0, the greedy approach is applied to a set of instance values along attribute A_j, $\pi_j(\mathbf{D}) = \{\pi_j(\vec{x}_i) = x_j^i \mid (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, 2, ..., m\}$. The middle values between each pair of adjacent sequential values of $\pi_j(\mathbf{D})$, i.e. $\frac{x_j^{(i)} + x_j^{(i+1)}}{2}$ for $i = 1, 2, ..., m-1$, are all considered as the candidates of the splitting values, which is shown in Figure 2.8. The value that offers the best splitting measure is chosen to be the splitting value of attribute A_j.

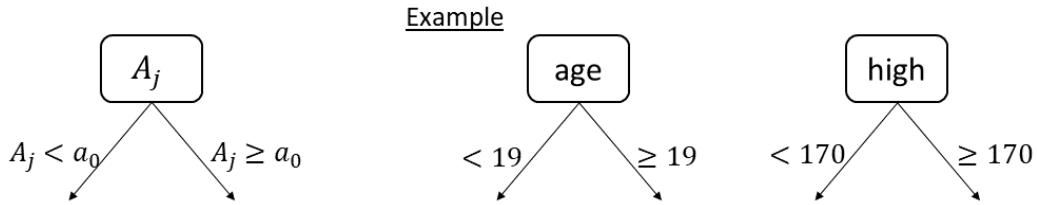


Figure 2.7: Partitioning the set of instances using numeric attributes

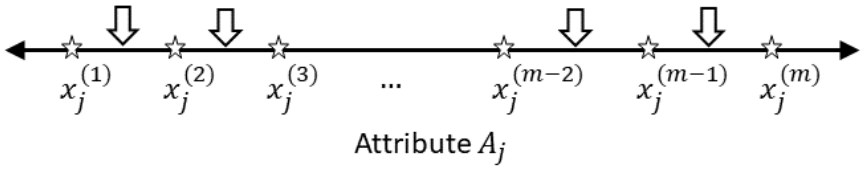


Figure 2.8: The candidates of the splitting value (displayed by each arrow) with respect to numeric attribute A_j

2.3.2 Splitting Measures

Many splitting measures have been proposed as the criteria for selecting the splitting condition at each non-leaf node in decision tree induction. Most of them are presented under the concept of measuring the impurity of dataset $\mathbf{D}$, such as the Gini index [43], the Shannon's entropy (or SE or entropy) [38] and the misclassification error, which are defined by (2.1), (2.2) and (2.3), respectively, in which $P_k(\mathbf{D})$ denotes the proportion of instances from class c_k that are contained in $\mathbf{D}$, i.e. $P_k(\mathbf{D}) = \frac{|\mathbf{D}_k|}{|\mathbf{D}|}$. Their highest value will occur when the dataset contains a similar number of instances from all classes, and they will have the lowest value approaching zero when there is a single class in the dataset. For example, the comparison of the Gini index, SE and the misclassification errors for the binary-class dataset is shown in Figure 2.9.

$$Gini(\mathbf{D}) = 1 - \sum_{k=1}^{p} (P_k(\mathbf{D}))^2 \tag{2.1}$$

$$Entropy(\mathbf{D}) = -\sum_{k=1}^{p} P_k(\mathbf{D}) \log_2 P_k(\mathbf{D}) \tag{2.2}$$

$$ClassError(\mathbf{D}) = 1 - \max_{k=\{1,2,\ldots,p\}} P_k(\mathbf{D}) \tag{2.3}$$

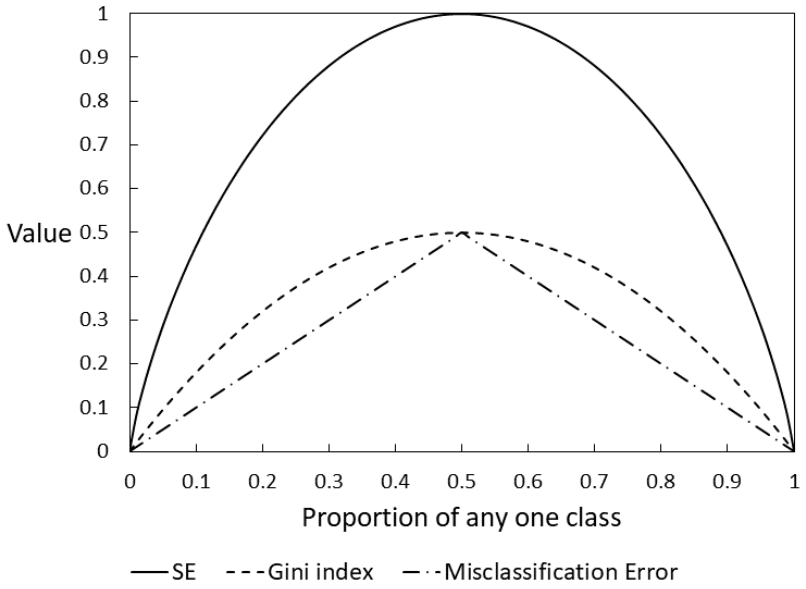


Figure 2.9: The comparison of each splitting measure for the binary-class dataset

In practice, the weighted average of the assigned weight values from a splitting measure calculated from each subset of instances after partitioning dataset $\mathbf{D}$ is used to compare for selecting the splitting condition, which is shown by (2.4), in which $SplitMeasure$ represents any splitting measure, and $c = \{\text{splitting attribute(s)} : \text{outcome } 1, ..., \text{outcome } q\}$ denotes a candidate of the splitting condition that partitions dataset $\mathbf{D}$ into q subsets, i.e. $\mathbf{D} = \mathbf{D}^{(1)} \cup \mathbf{D}^{(2)} \cup ... \cup \mathbf{D}^{(q)}$. For the impurity-based splitting measures mentioned above, the candidate that offers the lowest weighted average value is chosen to be the splitting condition.

$$SplitMeasure_c(\mathbf{D}) = \sum_{l=1}^{q} \frac{|\mathbf{D}^{(l)}|}{|\mathbf{D}|} SplitMeasure(\mathbf{D}^{(l)}) \tag{2.4}$$

Those splitting measures have been applied to use in many well-known decision tree algorithms. For the Gini index, it is directly used as the splitting measure in the classification and regression trees (CART) algorithm [44]. In 1986, Quinlan proposed the decision tree algorithm called the Iterative Dichotomiser 3 (ID3) algorithm [10] that employs the information gain to determine the splitting condition at each non-leaf node. It is the subtraction between the entropy of the entire dataset before splitting and the weighted average of the entropy of each partition after splitting, which is shown by (2.5). The candidate providing the highest information gain is chosen to be the splitting condi-

tion. However, the use of the information gain tends to favor the attribute having many distinct values. So, the decision algorithm known as the C4.5 algorithm is introduced to handle this issue. It uses the gain ratio, shown in (2.7), as the splitting measure that normalizes the information gain by dividing with the split information defined by (2.6), in which $P^{(l)}(\mathbf{D})$ denotes the proportion of instances in $D^{(l)}$ compared to entire dataset $\mathbf{D}$, i.e. $P^{(l)}(\mathbf{D}) = \frac{|\mathbf{D}^{(l)}|}{|\mathbf{D}|}$.

$$InfoGain_c(\mathbf{D}) = Entropy(\mathbf{D}) - Entropy_c(\mathbf{D}) \tag{2.5}$$

$$SplitInfo_c(\mathbf{D}) = -\sum_{l=1}^{q} P^{(l)}(\mathbf{D}) \log_2 P^{(l)}(\mathbf{D}) \tag{2.6}$$

$$GainRatio_c(\mathbf{D}) = \frac{InfoGain_c(\mathbf{D})}{SplitInfo_c(\mathbf{D})} \tag{2.7}$$

In addition, another interesting splitting measure called the distinct class based splitting measure (DCSM) [45] has been proposed recently. It is defined by (2.8), which is a combination of two concepts. First, considering the number of distinct classes in each partition $\mathbf{D}^{(l)}$ after splitting dataset $\mathbf{D}$, which is denoted by $C(\mathbf{D}^{(l)})$. The partition having a lesser number of distinct classes is purer and gives a low value, corresponding to the first term of the DCSM formula. Second, considering the proportion of instances from each class c_k in $D^{(l)}$ denoted by $P_k(\mathbf{D}^{(l)})$, where $P_k(\mathbf{D}^{(l)}) = \frac{|\mathbf{D}_k^{(l)}|}{|\mathbf{D}^{(l)}|}$. It decreases dramatically when there are many instances from one class approaching the total number of instances.

$$DCSM(\mathbf{D}^{(l)}) = C(\mathbf{D}^{(l)}) \exp\left(C(\mathbf{D}^{(l)})\right) \cdot \sum_{k=1}^{p} P_k(\mathbf{D}^{(l)}) \exp\left(\frac{C(\mathbf{D}^{(l)})}{p}\left(1 - (P_k(\mathbf{D}^{(l)}))^2\right)\right) \tag{2.8}$$

2.3.3 Examples of Inducing a Decision Tree

In this section, two training datasets (named **D1** and **D2**) are used as two examples to demonstrate decision tree induction. The first example (dataset **D1**) is represented by Table 2.1, which contains eight instances having three input attributes, i.e. Gender, Salary and Age denoted by G, S and A, respectively. It is the binary-class dataset that indicates whether the individual is likely to buy a car (Buy car? = "Yes") or not (Buy car? = "No").

Gender	Salary	Age	Buy car?
Male	High	18	No
Female	Medium	25	No
Female	Low	40	No
Female	Medium	48	No
Male	Medium	20	Yes
Male	Low	32	Yes
Male	High	54	Yes
Female	High	45	Yes

Table 2.1: Sample dataset **D1**

The recursive partitioning algorithm is performed on **D1**, which obtains the result as shown in Figure 2.10. Since the set of instances considered in the nodes at the top level and the second level are from different classes, they are partitioned into subsets forwarding to the child nodes using the selected attribute. The nodes at the bottom level would not continue the partitioning process because they contain instances from the same class.

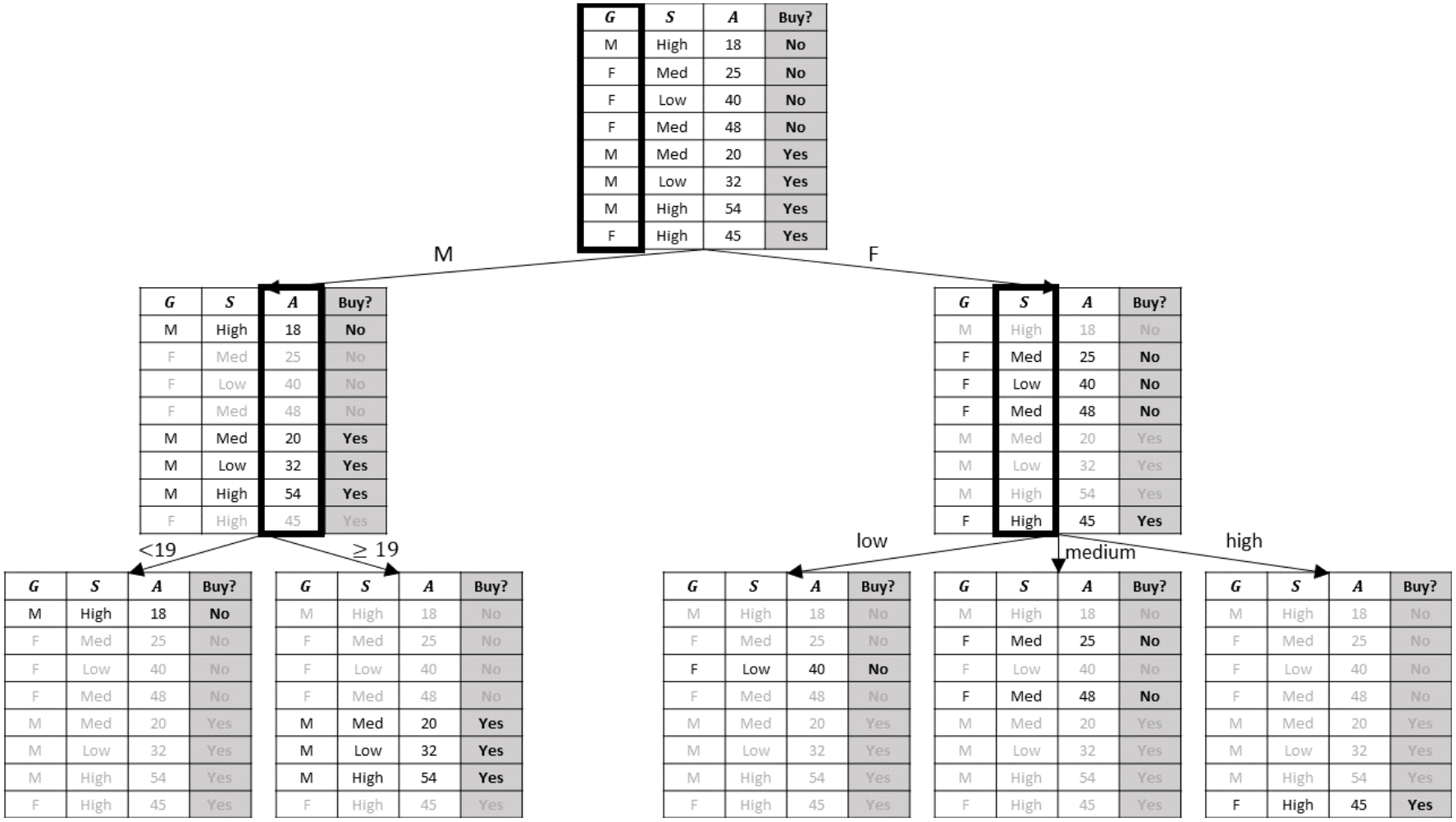


Figure 2.10: Performing the recursive partitioning on dataset **D1**

To select the splitting attribute at each non-leaf node, the Shannon's entropy (SE) is used as the splitting measure, which there will be a calculated demonstration for the root node as follows:

For entire dataset **D1**, three input attributes are considered.

1. Categorical attribute Gender has two distinct values, i.e. "Male" and "Female". Then, dataset **D1** will be divided into two subsets according to the splitting condition $c_1 = \{\text{Gender} : (1)\ \text{Gender} = \text{“Male”}, (2)\ \text{Gender} = \text{“Female”}\}$, where Gender is the splitting attribute, Gender = "Male" and Gender = "Female" are all possible splitting values.

   - There are four instances that their values of attribute Gender are "Male", in which three instances are from "Yes" class and one instance is from "No" class. Therefore,

   $$Entropy(\mathbf{D1}^{(1)}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.81128.$$

   - There are four instances that their values of attribute Gender are "Female", in which one instance is from "Yes" class and three instances are from "No" class. Therefore,

   $$Entropy(\mathbf{D1}^{(2)}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.81128.$$

So, the entropy after partitioning dataset **D1** by attribute Gender is equal to

$$\begin{aligned} Entropy_{c_1}(\mathbf{D1}) &= \frac{4}{8}\cdot Entropy(\mathbf{D1}^{(1)}) + \frac{4}{8}\cdot Entropy(\mathbf{D1}^{(2)}) \\ &= \frac{4}{8}\cdot 0.81128 + \frac{4}{8}\cdot 0.81128 \\ &= 0.81128. \end{aligned}$$

2. Categorical attribute Salary has three distinct values, i.e. “High”, “Medium” and “Low”, denoted by Hi, Md and Lo, respectively. Then, dataset **D1** will be divided in to three subsets according to the splitting condition $c_2 = \{\text{Salary} : (1)\ \text{Salary} = \text{“High”}, (2)\ \text{Salary} = \text{“Medium”}, (3)\ \text{Salary} = \text{“Low”}\}$, where Salary is the splitting attribute, Salary = “High”, Salary = “Medium” and, Salary = “Low” are all possible splitting values.

   - There are three instances that their values of attribute Salary are “High”, in which two instances are from “Yes” class and one instance is from “No” class. Therefore,

$$Entropy(\mathbf{D1}^{(1)}) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183.$$

   - There are three instances that their values of attribute Salary are “Medium”, in which one instance is from “Yes” class and two instances are from “No” class. Therefore,

$$Entropy(\mathbf{D1}^{(2)}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.9183.$$

   - There are two instances that their values of attribute Salary are “Low”, in which one instance is from “Yes” class and one instance is from “No” class. Therefore,

$$Entropy(\mathbf{D1}^{(3)}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1.$$

   So, the entropy after partitioning dataset **D1** by attribute Salary is equal to

$$\begin{aligned} Entropy_{c_2}(\mathbf{D1}) &= \frac{3}{8}\cdot Entropy(\mathbf{D1}^{(1)}) + \frac{3}{8}\cdot Entropy(\mathbf{D1}^{(2)}) \\ &\quad + \frac{2}{8}\cdot Entropy(\mathbf{D1}^{(3)}) \\ &= \frac{3}{8}\cdot 0.9183 + \frac{3}{8}\cdot 0.9183 + \frac{2}{8}\cdot 1 \\ &= 0.93872. \end{aligned}$$

3. Numeric attribute Age has eight distinct values that are ordered. Using the greedy approach, all middle values must be considered as the candidates of splitting value and computed their values of entropy as shown in Figure 2.11. The value that gives the lowest entropy will be the first one in the calculation here, which is 19. Then, dataset **D1** will be divided into 2 subsets according to the splitting condition $c_3 = \{\text{Age} : (1)\ \text{Age} < 19, (2)\ \text{Age} \geq 19\}$.

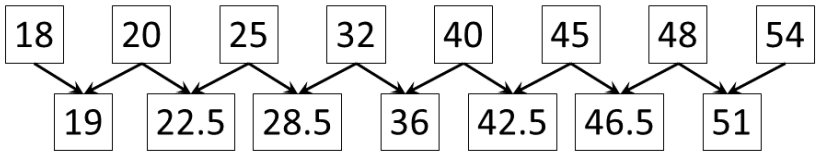


Figure 2.11: The values of attribute Age that are sorted (top row), and their middle values (bottom row)

- There is only one instance that its value of attribute Age less than 19, in which it is from "No" class. Therefore,

$$Entropy(\mathbf{D1}^{(1)}) = -\frac{1}{1}\log_2\frac{1}{1} = 0.$$

- There are seven instances that their values of attribute Age greater than or equal to 19, in which four instances are from "Yes" class and three instances are from "No" class. Therefore,

$$Entropy(\mathbf{D1}^{(2)}) = -\frac{4}{7}\log_2\frac{4}{7} - \frac{3}{7}\log_2\frac{3}{7} = 0.98523.$$

So, the entropy after partitioning dataset **D1** by splitting value 19 of attribute Age is equal to

$$\begin{aligned} Entropy_{c_3}(\mathbf{D1}) &= \frac{1}{8}\cdot Entropy(\mathbf{D1}^{(1)}) + \frac{7}{8}\cdot Entropy(\mathbf{D1}^{(2)}) \\ &= \frac{1}{8}\cdot 0 + \frac{7}{8}\cdot 0.98523 \\ &= 0.86207. \end{aligned}$$

It can be seen that attribute Gender provides the least value of entropy. Consequently, it is selected to be the splitting condition for partitioning entire dataset **D1**. Then, recursively perform the calculation steps above with each partition in each child node until they contain only instances from the same class.

For the second example, dataset **D2** is represented by Figure 2.12, which contains twelve instances having two numeric attributes. They are from three different classes that are the red-circle class, the blue-triangle class, and the green-square class.

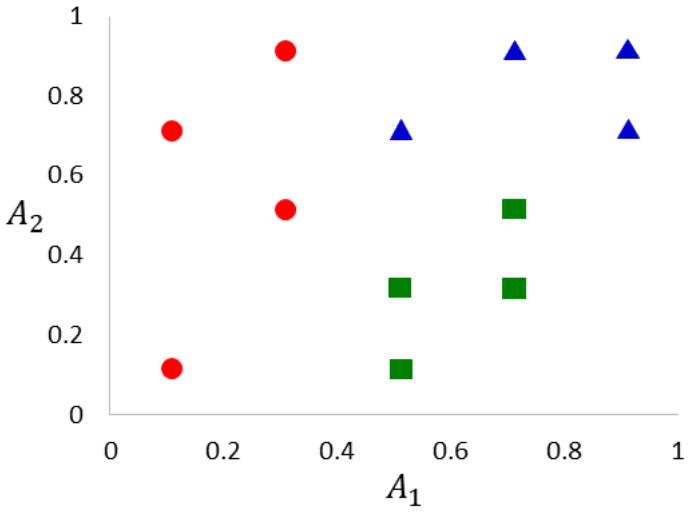


Figure 2.12: Sample dataset **D2**

The recursive partitioning algorithm is performed on **D2**, which obtains the result as shown in Figure 2.13. At each non-leaf node, the greedy approach is applied to select the splitting condition. All middle values between each pair of adjacent sequential distinct values of all attributes, which are indicated by dash lines, are all considered. Then, the best one providing the optimal splitting measure is chosen, which is indicated by the solid line. Finally, the leaf nodes display the regions of each class determined by the boundary of solid lines.

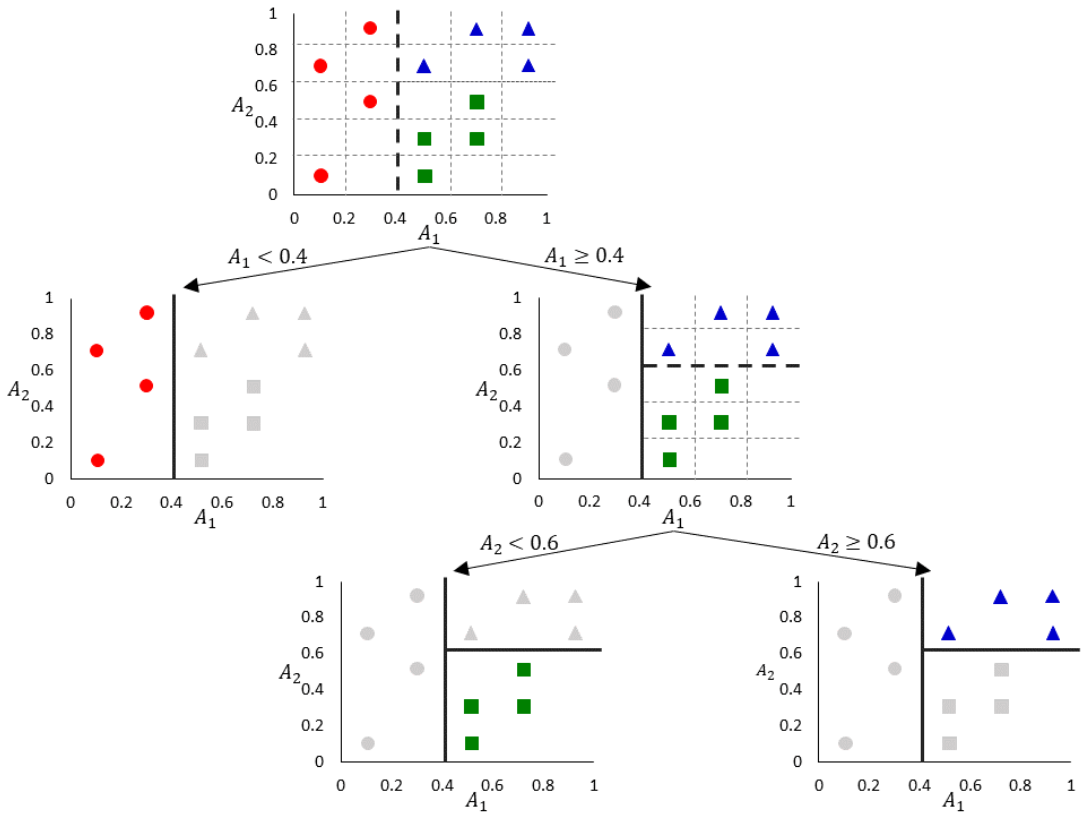


Figure 2.13: Performing the recursive partitioning on dataset **D2**

2.3.4 Oblique Decision Tree Algorithm

For a dataset consisting of only numeric attributes, the use of a single attribute to partition is quite effective but the distribution of real-world datasets usually deviates from the axis of each attribute, which is called the parallel axis. For example, the distribution of the dataset shown in Figure 2.14(a) lies onto the parallel axes that can easily partition with a single attribute as shown from the previous section, but it is not applicable for the dataset shown in Figure 2.14(b). The distribution of each class in that dataset does not arrange along any parallel axis, it is therefore difficult to separate with a single attribute.

In Figure 2.15, there are multiple iterations to separate the dataset using a single attribute causing a deep decision tree, if splitting is applied using multi-attributes then it will need less iterations. The splitting condition of a dataset containing only numeric attributes is defined by hyperplane $H : \vec{a} \cdot \vec{x} - a_0 = 0$, where $\vec{a} = \langle a_1, a_2, ..., a_n \rangle$ is the normal vector, $\vec{x} = \langle x_1, x_2, ..., x_n \rangle \in X$ is any vector in the input attribute domain,

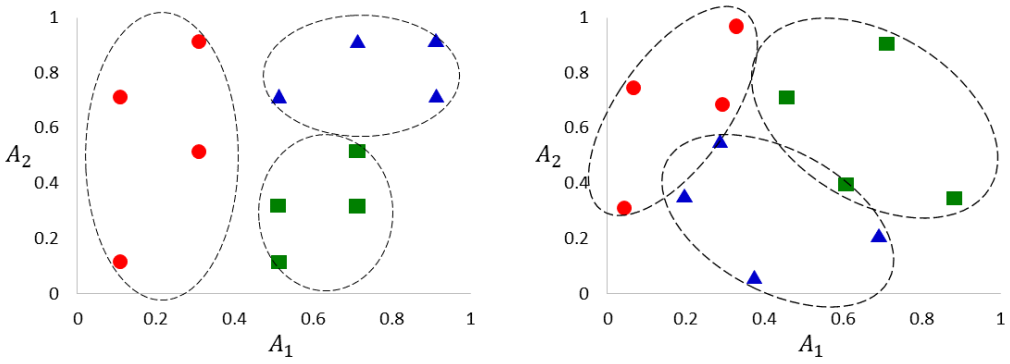


Figure 2.14: The distribution of each class in two sample datasets

and a_0 is the splitting value. In the partitioning process, a set of instances $\mathbf{D}$ will be divided into two partitions according to the splitting condition $c = \{\vec{a} = \langle a_1, a_2, ..., a_n \rangle : (1)\ \vec{a} \cdot \vec{x} - a_0 < 0, (2)\ \vec{a} \cdot \vec{x} - a_0 \geq 0\}$, i.e. the lower partition and the upper partition defined as $\mathbf{D}^{(1)} = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid \vec{a} \cdot \vec{x}_i - a_0 < 0 \text{ for } i = 1, 2, ..., m\}$ and $\mathbf{D}^{(2)} = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid \vec{a} \cdot \vec{x}_i - a_0 \geq 0 \text{ for } i = 1, 2, ..., m\}$. The hyperplane that all elements in $\vec{a}$ are zero except for a_j gives the same result as using single attribute A_j, which is called the axis-parallel hyperplane. In general, the hyperplane that all attributes of instances are used together is called the oblique hyperplane, and therefore the decision tree building from oblique hyperplanes is called the oblique decision tree.

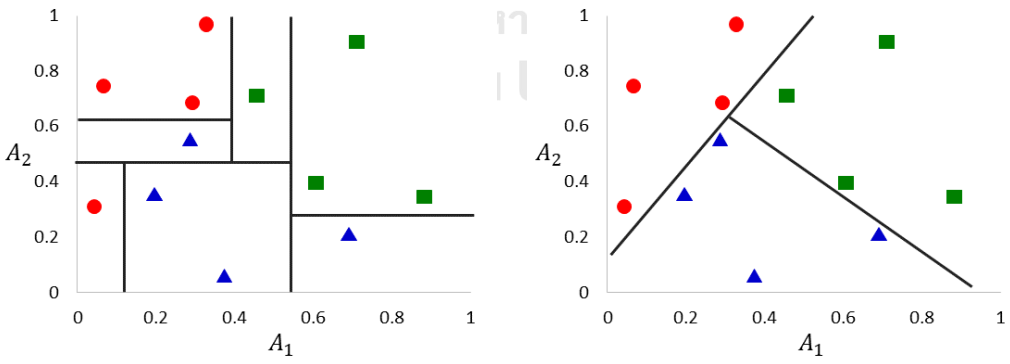


Figure 2.15: Partitioning the sample dataset using axis-parallel hyperplanes (a) and using oblique hyperplanes (b)

At each non-leaf node, however, there is an exponential number of different parti-

tioning when using oblique hyperplanes as the splitting condition, which is up to $2^n \cdot \binom{m}{n}$. So, it is impractical to explore all oblique hyperplanes using the greedy approach to discover the best one, which is proved to be the NP-hard problem [46]. Thus, other techniques are required for searching the (local) optimal hyperplane. In 1984, Breiman et al. proposed the linear combination version of the CART algorithm (CART-LC) [44] for inducing the oblique decision tree. It applies the deterministic hill-climbing algorithm for searching the best oblique hyperplane. Then, in 1993, simulated annealing decision tree (SADT) was presented by Heath et al [47]. It employs the randomization of the simulated annealing algorithm to find the best oblique hyperplane. The concepts of these two methodologies motivated the well-known oblique decision tree algorithm as oblique classifier 1 (OC1) [48] presented in 1994. It consists of three steps to search the best hyperplane.

- First, initial hyperplane $H: \ \vec{a} \cdot \vec{x} - a_0 = 0$ is determined by the best axis-parallel hyperplane which is obtained from the greedy approach like a traditional decision tree algorithm.

- Second, hyperplane H is adjusted along each attribute's axis by the deterministic hill-climbing method. For attribute A_j, coefficient a_j of $\vec{a}$ is treated as a variable, and other coefficients are treated as constants. The set of the candidate values of a_j allowing the hyperplane to pass through each instance $\vec{x}_i$ in dataset $\mathbf{D}$ is defined by $u_j(\mathbf{D})$ as follows:

$$u_j(\mathbf{D}) = \left\{ u_j(\vec{x}_i) = \frac{-\vec{a} \cdot \vec{x}_i + a_j \cdot x_j^i + a_0}{x_j^i} \;\middle|\; (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, ..., m \right\} \quad (2.9)$$

  Then, applying the greedy approach on $u_j(\mathbf{D})$ to determine the best value of a_j. For example, Figure 2.16(a) displays the mechanism of perturbing hyperplane H by the deterministic hill-climbing method with respect to attribute A_1, which all candidate hyperplanes are represented by dash lines.

- Third, hyperplane H from the previous step is perturbed to avoid the local solution using random hyperplane $R: \ \vec{r} \cdot \vec{x} - r_0 = 0$ with scale α, i.e. $H + \alpha R$. In which

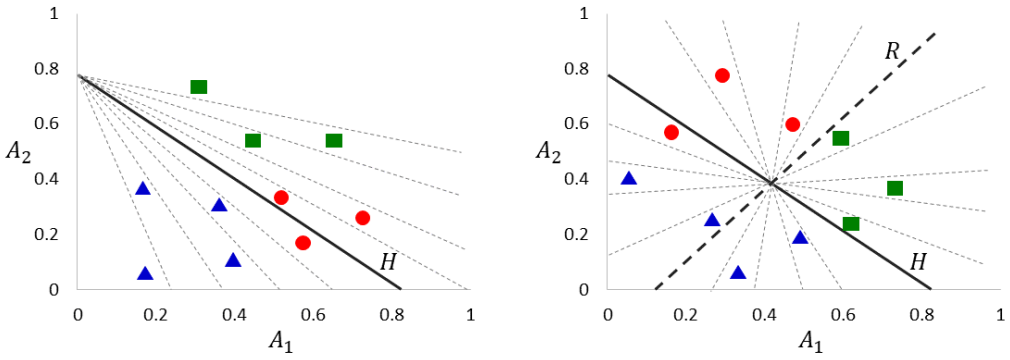


Figure 2.16: Applying the deterministic hill-climbing algorithm (a) and the randomization algorithm (b) to perturb the hyperplane

each component of $\vec{r} = \langle r_1, r_2, ..., r_n \rangle$ including r_0 is a random value obtained from the uniform distribution within an interval $[-1, 1]$. Then, value α is treated as a variable, and other coefficients of both hyperplane H and R are treated as constants. The set of the candidate values of α allowing the hyperplane to pass through the intersection of H and R, and each instance $\vec{x}_i$ in dataset $\mathbf{D}$ is defined by $v(\mathbf{D})$ as follows:

$$v(\mathbf{D}) = \left\{ v(\vec{x}_i) = \frac{-\vec{a} \cdot \vec{x}_i + a_0}{\vec{r} \cdot \vec{x}_i - r_0} \;\middle|\; (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, ..., m \right\} \tag{2.10}$$

Then, applying the greedy approach on $v(\mathbf{D})$ to discover the best value of α. For example, Figure 2.16(b) displays the mechanism of perturbing hyperplane H by the randomization algorithm with respect to random hyperplane R (solid-dash line), which all candidate hyperplanes are represented by dash lines.

The brief pseudocode of building Oblique Classifier 1 from $\mathbf{D}$ is displayed by Algorithm 2.2. In the process of finding the best hyperplane at each internal node, it starts by obtaining initial hyperplane H, which is the best axis-parallel hyperplane, from the greedy approach. Then, hyperplane H is perturbed using the deterministic hill-climbing algorithm and the randomization algorithm, respectively. If the hyperplane being adjusted gives the better value of a specific splitting measure, repeating the previous step. Otherwise, oblique hyperplane H is selected as the splitting condition.

Algorithm 2.2: OC1(**D**)

Input: a set of instances **D**
Output: a decision tree

- create the root node of the tree
- **if** all instances in **D** are in the same class **then**
  - **return** the leaf node with respect to that class
- **else**
  - `/* select the initial hyperplane */`
  - obtain the initial hyperplane H
  - `/* perturb the hyperplane */`
  - **do**
    - perturb H using the deterministic hill-climbing algorithm
    - perturb H using the randomization algorithm
  - **while** the hyperplane H has been improved;
  - obtain the best hyperplane H
  - `/* recursively partition the dataset */`
  - separate the dataset into 2 partitions corresponding to the outcomes of the hyperplane H: $\mathbf{D} = \mathbf{D}^{(\text{left})} \cup \mathbf{D}^{(\text{right})}$
  - iterate for each partition: OC1($\mathbf{D}^{(\text{left})}$) and OC1($\mathbf{D}^{(\text{right})}$)

Additionally, there have also been other algorithms to construct an oblique decision tree besides the use of the optimization techniques like CART-LC, SADT, and OC1. For instance, OC1-SA, OC1-GA, and OC1-ES [49] apply the existing heuristic arguments, which are the simulated annealing algorithm, the genetic algorithm, and the evolution strategy algorithm, respectively. For the algorithms based on the feature extraction techniques, Fisher's decision tree (FDT) [50] employs the linear discriminant analysis, while the Householder reflection is used in HHCART [51]. In addition, some new heuristic algorithms have been proposed such as the Cline algorithm [52] and the CARTopt algorithm [53].

2.4 Class Imbalanced Problem

Although the class imbalanced problem has received widespread attention in recent years, there is no clear definition of a dataset that is imbalanced. Technically, a dataset having unequal class distribution should be imbalanced. However, it is considered imbalanced when the number of instances in each class is significantly different [54].

2.4.1 Binary-class Imbalanced Problem

Traditionally, the class imbalanced problem is often related to a binary-class dataset that one class contains significantly more instances than another class, called the binary-class imbalanced dataset. The majority class is normally represented by the negative class (c_-), while the minority class, which will be focused for a classifier, is indicated by the positive class (c_+) shown in Figure 2.17.

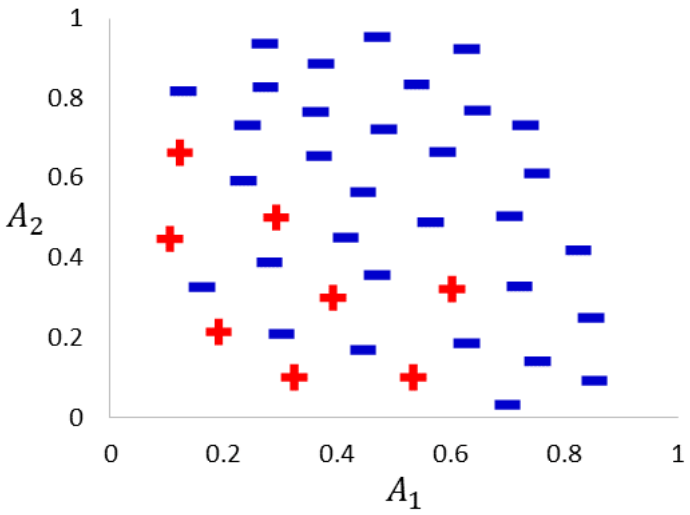


Figure 2.17: Binary-class imbalanced dataset

Mathematically, binary-class dataset $\mathbf{D}$ containing m_- instances from the majority class, and m_+ instances from the minority class is said to be imbalanced when m_- is much greater than m_+. To measure the degree of imbalanced for each binary-class dataset, the imbalanced ratio ($I.R.$) is used, which is defined by the ratio between the number of instances in the majority class and the minority class as shown in (2.11). For the problem of building the model to classify the binary-class imbalanced dataset, it is called the binary-class imbalanced problem. It is intended to predict the minority instances, while maintaining the correct classification of the majority instances.

$$I.R. = \frac{m_-}{m_+} \tag{2.11}$$

2.4.2 Multi-class Imbalanced Problem

The binary-class imbalanced problem is less complicated because it has only one clear primary goal of improving the performance of classifying the minority instances. In several real-world applications, however, a collected dataset may contain the instances from more than two classes having completely different proportions as shown in Figure 2.18, which is called the multi-class imbalanced dataset.

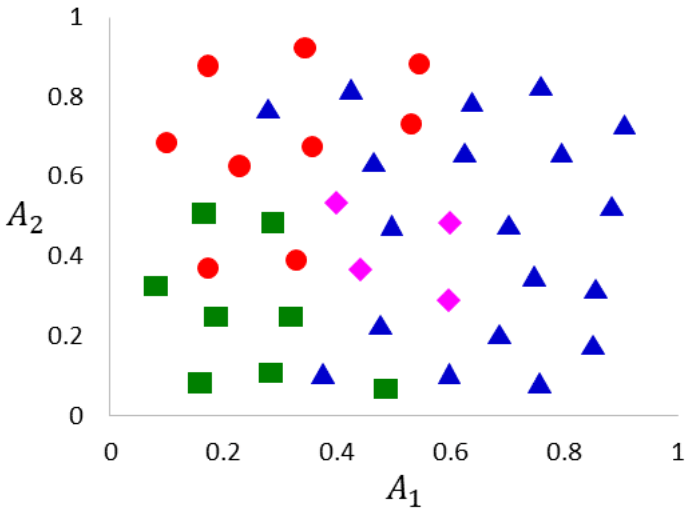


Figure 2.18: Multi-class imbalanced dataset

Mathematically, multi-class dataset $\mathbf{D}$ containing m instances from p classes c_k with size m_k for $k = 1, 2, ..., p$ is said to be imbalanced when there are at least two class c_{k_1} and c_{k_2} such that m_{k_1} is much greater than m_{k_2}. To measure the degree of imbalanced for a multi-class dataset, the imbalanced ratio ($I.R.$) is generally defined by the ratio between the number of instances in the largest class and the smallest class as shown in (2.12). For the problem of building the model to classify the multi-class imbalanced dataset, it is called the multi-class imbalanced problem. It is more complex and more difficult to handle than the binary case.

$$I.R. = \frac{\underset{k\in\{1,2,...,p\}}{max} m_k}{\underset{k\in\{1,2,...,p\}}{min} m_k} \tag{2.12}$$

The challenge of handling the multi-class imbalanced problem is to deal with two characteristics that are only found in this situation.

- First, it is the presence of the multi-majority classes and the multi-minority classes. [55]. A binary-class imbalanced dataset has only one majority class containing a large number of instances and one minority class containing a small number of instances. Nevertheless, a multi-class imbalanced dataset may have either multi-majority classes or multi-minority classes, which contain more or less instances than the average, respectively.

- Second, it is the flexibility to change the role of each class in different contexts [56]. On one hand, a class may be considered as the majority class compared to other classes. On the other hand, it may be considered as the minority class compared to other ones.

As shown in Figure 2.18, for example, the multi-class imbalanced dataset contains one majority class indicated by the blue-triangle class and three minority classes indicated by the red-circle class, the green-square class, and the pink-diamond class. Considering the red-circle class, it is the minority comparing with the blue-triangle class, but, at the same time, it can be considered as the majority comparing with two other classes. Similarly, although the blue-triangle class looks like the majority comparing with any other class explicitly, it immediately becomes the minority comparing with all remaining classes. Therefore, unlike the binary-class imbalanced problem, the multi-class imbalanced problem cannot specifically treat any particular class as the minority class. The performance of classifying the overall dataset may be easily lost, when aiming at just one class exclusively [57].

2.4.3 Class Imbalanced Methodologies

Many methodologies have been introduced for handling the class imbalanced problem, especially for the binary case. They can be categorized into four different techniques [54]:

1. *Data-level approach* is based on a sampling method performing on an original imbalanced dataset to re-balance the class distribution. They include the use of the over-sampling method [58, 59] and the under-sampling method [60, 61], which synthesizes the minority instances and eliminates the majority instances, respectively.

2. *Algorithm-level approach* focuses on modifying a classification algorithm to build a classifier that is suitable for an imbalanced dataset [62]. It also includes creating a new classifier to address this issue specifically. Their mechanisms incline to classify the instances to be the minority class instead of the majority one.

   This dissertation is interested to study the decision tree improvement for the class imbalanced problem. In recent years, many articles presented the decision tree algorithms for building the model to classify a binary-class imbalanced dataset particularly. They normally focus on introducing specific splitting measures under various concepts, which can be categorized into three groups as follows:

   - The concept of non-symmetric biases toward the minority class. The maximum value of splitting measures are shifted from the equal proportion of the minority class and the majority class to the one that is larger, indicating with parameter $\theta \in [0, 0.5)$. Accordingly, the value of these splitting measures will decrease when the proportion of the minority class increases from θ to 0.5, while the value of symmetric splitting measures such as the Shannon's entropy will increase. The most well-known non-symmetric splitting measure is the asymmetric entropy (AE) [63, 64] defined by (2.13). The comparison of values of AE and the Shannon's entropy (SE) is shown in Figure 2.19, in which parameter θ set to 0.2. The value of AE decreases when the proportion of the minority class is in the ranges from 0.2 to 1, but it is only the ranges from 0.5 to 1 for the entropy. Moreover, there are also other splitting measures in this category such as the off-centered entropy (OCE) [65, 66] and AECID [67].

$$AE(\mathbf{D}) = \frac{P_+(\mathbf{D})(1 - P_+(\mathbf{D}))}{(1 - 2\theta)P_+(\mathbf{D}) + \theta^2} \tag{2.13}$$

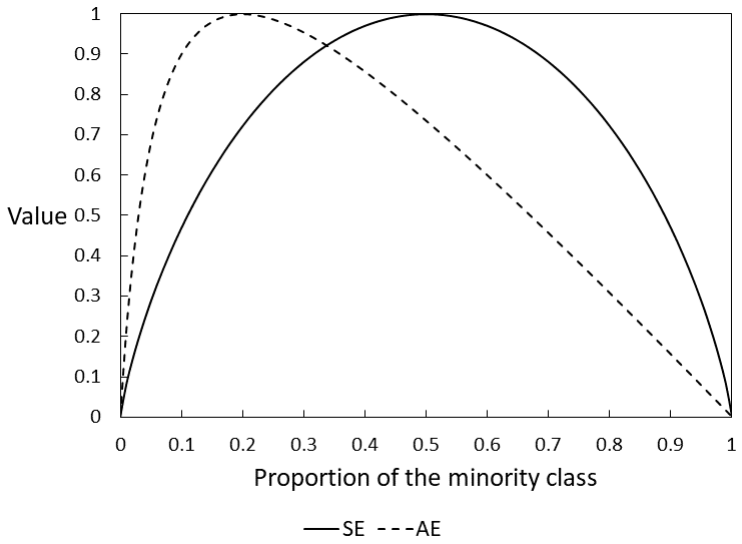


Figure 2.19: The values of the asymmetric entropy with $\theta = 0.2$ comparing with SE

- The concept of skew-insensitivity can be resistant to the different number of instances in the majority class and the minority class. It does not much affect the value of splitting measures in this group. The most well-known distance function that is used as a skew-insensitivity splitting measure is the Hellinger distance [68, 69], which is defined by (2.14), in which $P^{(l)}(\mathbf{D}_+) = \frac{|\mathbf{D}_+^{(l)}|}{|\mathbf{D}_+|}$ and $P^{(l)}(\mathbf{D}_-) = \frac{|\mathbf{D}_-^{(l)}|}{|\mathbf{D}_-|}$ are the proportion of the minority instances and the majority instances in partition l comparing with the total instances in their classes, respectively. That is, it measures the distance between the vector corresponding to the proportion of the minority instances in each partition, i.e. $P(\mathbf{D}_+) = \langle P^{(1)}(\mathbf{D}_+), P^{(2)}(\mathbf{D}_+), ..., P^{(q)}(\mathbf{D}_+)\rangle$, and that of the majority instances, i.e. $P(\mathbf{D}_-) = \langle P^{(1)}(\mathbf{D}_-), P^{(2)}(\mathbf{D}_-), ..., P^{(q)}(\mathbf{D}_-)\rangle$. So, the splitting condition that offers the greatest distances is selected. The decision tree algorithm building the model based on the Hellinger distance is called HDDT [70, 71], which is used to deal with the binary-class imbalanced problem. Furthermore, DKM [72, 73] is an another splitting measure that is skew-insensitive.

$$d_H(P(\mathbf{D}_+), P(\mathbf{D}_-)) = \sqrt{\sum_{l=1}^{q} \left(P^{(l)}(\mathbf{D}_+) - P^{(l)}(\mathbf{D}_-)\right)^2} \tag{2.14}$$

- The concept of modifying the components to calculate the Shannon's entropy (SE) to be inclined towards the minority class is lately introduced by Boonchuay et al. in 2017. They proposed the splitting measure named the minority entropy (ME) [40], which discards the majority instances that do not affect the partition from the minority class. For each attribute A_j, only the subset of instances within the range of the minority class is employed in consideration, which has a similar effect as the sampling approach. Mathematically, the minority range of dataset $\mathbf{D}$, denoted by $\phi_+(\pi_j(\mathbf{D}))$, is the range between the minimum value and the maximum value of the minority instances corresponding to A_j, i.e. $\phi_+(\pi_j(\mathbf{D})) = [\min \pi_j(\mathbf{D}_+), \max \pi_j(\mathbf{D}_+)]$ where $\pi_j(\mathbf{D}_+) = \{\pi_j(\vec{x}_i) = x_j^i \mid (\vec{x}_i, y_i) \in \mathbf{D}_+ \text{ for } i = 1, 2, ..., m\}$. Then, $\Phi_+(\pi_j(\mathbf{D}))$ implies the subset of instances within the minority range, which is defined by $\Phi_+(\pi_j(\mathbf{D})) = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid \pi_j(\vec{x}_i) = x_j^i \in \phi_+(\pi_j(\mathbf{D})) \text{ for } i = 1, 2, ..., m\}$. For example, the minority ranges and the subset of instances within them are demonstrated in Figure 2.20. Thus, the definition of the minority entropy according to attribute A_j is determined by (2.15) as follows:

$$ME_j(\mathbf{D}) = Entropy(\Phi_+(\pi_j(\mathbf{D}))) \tag{2.15}$$

3. *Cost-sensitive approach* involves the cost adjusted techniques [74]. The large misclassification cost is assigned to instances from the minority class, while the instances from the majority class receive the lower cost. The classification model is created biasing toward the class having a higher cost for minimizing the total costs.

4. *Ensemble-based approach* incorporates a technique mentioned above with the ensemble learning method [75], e.g. Bagging [76] and Boosting [77]. It has shown to be effective over a single classification model.

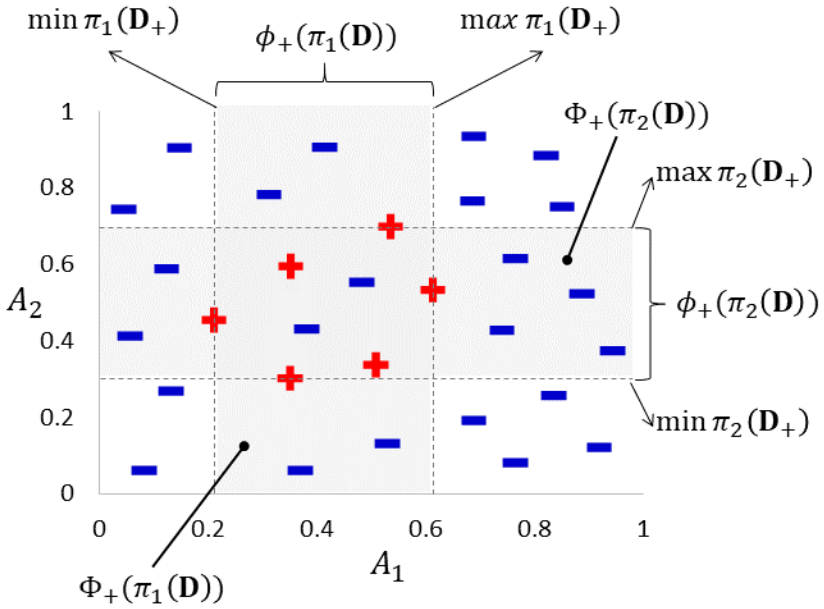


Figure 2.20: The example of minority ranges and the subset of instances within them

However, most methodologies are likely to involve only the binary-class case, which cannot be directly applied to the multi-class case. So, additional techniques are required to handle the multi-class imbalanced problem. A variety of methods have been introduced which are categorized into two main strategies [54]:

1. *Decomposition-based approach* decomposes a multi-class imbalanced dataset into a set of binary-class datasets. Then, a traditional technique, designed for the binary-class problem, can be applied to each generated dataset. A set of classifiers is therefore aggregated as the final model. Most common methods to decompose a multi-class dataset include the one-versus-one approach (OVO) [78] and the one-versus-all approach (OVA) [79]. For the p classes problem, it is decomposed into $\binom{p}{2}$ binary subproblems using the OVO scheme, which each subproblem for each pair of the classes. Differently, the OVA scheme decomposes the p classes problem into p binary subproblems which distinguishes one class from the remaining classes.

2. *Ad-hoc approach* directly learns a multi-class imbalanced dataset without employing the decomposition technique. It is believed that using an appropriate specific solution, rather than applying the traditional methods on the binary subproblems, is sufficient for solving the multi-class imbalanced problem effectively [80]. There

are many methods have been presented to deal with this issue using various techniques covering both the data-level approach, the algorithm-level approach, the cost-sensitive approach, and the ensemble-based approach.

Almost all decision tree algorithms discussed in the previous section are created to deal with a binary-class imbalanced problem using their proposed splitting measures, which cannot work with a dataset containing multiple classes. In 2012, however, Hoens et al. developed the HDDT algorithm as the improved decision tree algorithm, called the multi-class Hellinger distance decision tree (MC-HDDT) [81], which is able to work with multi-class datasets. It presents the application of the decomposition method like OVA. Instead of using it to decompose an entire problem into many subproblems, it is used as a part of selecting the splitting condition at each internal node. For p classes problem, each candidate of splitting condition will be considered via the Hellinger distance (2.14) up to p times rather than a single time. For each time, one class is treated as the positive class and the remaining classes are treated as the negative class. Then, the splitting condition that gives the largest distance is chosen regardless of which class is determined as the positive class.

2.4.4 Performance Measures and Statistical Test

To evaluate the efficiency of classifying an imbalanced dataset of each classification model, various performance measures are employed. In addition, a non-parametric statistical hypothesis test is also used to compare the different performance obtaining from each classifier.

Performance Measures

For the binary-class case, the performance measures are all related to the values in the confusion matrix as shown in Table 2.2.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 2.2: Confusion matrix

They specifically focus on measuring the effectiveness of classifying positive (minority) instances, which consist of the precision, the recall, and the F-measure [54], which are defined by (2.16), (2.17), and (2.18), respectively. The precision is the percentage of predicted minority instances that are correctly classified, and the recall is the percentage of actual minority instances that are correctly classified. For the F-measure, it presents the harmonic mean of the precision and the recall.

$$Precision = \frac{TP}{TP + FP} \tag{2.16}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.17}$$

$$F\text{–}measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.18}$$

where

- TP is the number of predicted positive instances that are correctly classified.
- FP is the number of predicted positive instances that are incorrectly classified.
- TN is the number of predicted negative instances that are correctly classified.
- FN is the number of predicted negative instances that are incorrectly classified.

For example, the results of classifying the binary-class dataset are presented by the confusion matrix in Table 2.3. So, the precision, the recall, and the F-measure are computed by (2.19), (2.20), and (2.21), respectively. For the precision which equals 0.6, it means that the instances that are predicted as the positive class have 60% chance that they are actually positive. For the recall which equals 0.75, it means that the instances that are actually positive have 75% chance that they are predicted as the positive class. Then, the harmonic mean between these two measures is represented by the F-measure which equals 0.67.

	Predicted Positive	Predicted Negative
Actual Positive	$TP = 15$	$FN = 5$
Actual Negative	$FP = 10$	$TN = 70$

Table 2.3: An example of the confusion matrix

$$Precision = \frac{15}{15+10} = 0.6 \tag{2.19}$$

$$Recall = \frac{15}{15+5} = 0.75 \tag{2.20}$$

$$F\text{–}measure = 2 \times \frac{0.6 \times 0.75}{0.6+0.75} = 0.67 \tag{2.21}$$

For the multi-class case, the performance measures are all expanded from the binary-class measures using the one-versus-all approach (OVA). The average values obtaining from each subclass is compiled [82]. So, all classes are treated equally which does not focus on any particular class. They consist of the macro-precision, the macro-recall, and the macro-F-measure, which are defined by (2.22), (2.23), and (2.24), respectively. The macro-precision is the average of the percentage of predicted instances in each class that are correctly classified, and the macro-recall is the average of the percentage of actual instances in each class that are correctly classified. For the macro-F-measure, it presents the harmonic mean of the macro-precision and the macro-recall. Moreover, the accuracy is also used in the multi-class case, defined by (2.25). It is the overall performance of classification computed by the percentage of all instances that are correctly classified.

$$Macro\text{–}Precision = \frac{1}{p}\sum_{k=1}^{p}\frac{TP_k}{TP_k + FP_k} \tag{2.22}$$

$$Macro\text{–}Recall = \frac{1}{p}\sum_{k=1}^{p}\frac{TP_k}{TP_k + FN_k} \tag{2.23}$$

$$Macro\text{–}F\text{–}measure = 2 \times \frac{Macro\text{–}Precision \times Macro\text{–}Recall}{Macro\text{–}Precision + Macro\text{–}Recall} \tag{2.24}$$

$$Accuracy = \frac{1}{m}\sum_{k=1}^{p} TP_k \tag{2.25}$$

where

- TP_k is the number of instances predicted as class c_k that are correctly classified.
- FP_k is the number of instances predicted as class c_k that are incorrectly classified.
- TN_k is the number of instances predicted as other classes that are correctly classified.
- FN_k is the number of instances predicted as other classes that are incorrectly classified.

For example, the results of classifying the three classes dataset are presented by Table 2.4. Then, when considering each class as the positive class, it is able to construct the confusion matrices corresponding to classes c_1, c_2 and c_3 by Tables 2.5, 2.6 and 2.7, respectively. So, the macro-precision, the macro-recall, the macro-F-measure, and the accuracy are computed by (2.26), (2.27), (2.28), and (2.29), respectively. For the macro-precision which equals 0.68, it is the average of the percentage of predicted instances in each class c_k that are correctly classified. For the macro-recall which equals 0.71, it is the average of the percentage of actual instances in each class c_k that are correctly classified. Then, the harmonic mean between these two measures is represented by the macro-F-measure which equals 0.69. For the accuracy which equals 0.7, it means that there is 70 % average chance that each instance is correctly classified.

	Predicted Class c_1	Predicted Class c_2	Predicted Class c_3
Actual Class c_1	15	3	2
Actual Class c_2	3	20	7
Actual Class c_3	8	7	35

Table 2.4: The results of classifying the three classes dataset

	Predicted Positive	Predicted Negative
Actual Positive	$TP_1 = 15$	$FN_1 = 5$
Actual Negative	$FP_1 = 11$	$TN_1 = 69$

Table 2.5: The confusion matrix coresponding to class c_1

	Predicted Positive	Predicted Negative
Actual Positive	$TP_2 = 20$	$FN_2 = 10$
Actual Negative	$FP_2 = 10$	$TN_2 = 60$

Table 2.6: The confusion matrix coresponding to class c_2

	Predicted Positive	Predicted Negative
Actual Positive	$TP_3 = 35$	$FN_3 = 15$
Actual Negative	$FP_3 = 9$	$TN_3 = 41$

Table 2.7: The confusion matrix coresponding to class c_3

$$Macro\text{–}Precision = \frac{1}{3} \times \left(\frac{15}{15+11} + \frac{20}{20+10} + \frac{35}{35+9} \right) = 0.68 \tag{2.26}$$

$$Macro\text{–}Recall = \frac{1}{3} \times \left(\frac{15}{15+5} + \frac{20}{20+10} + \frac{35}{35+15} \right) = 0.71 \tag{2.27}$$

$$Macro\text{–}F\text{–}measure = 2 \times \frac{0.68 \times 0.71}{0.68 + 0.71} = 0.69 \tag{2.28}$$

$$Accuracy = \frac{1}{100} \times (15 + 20 + 35) = 0.7 \tag{2.29}$$

Statistical Test

To compare the performance of the proposed method with the others, the Wilcoxon signed-rank test with a significance level (α) [83] is conducted. It is a non-parametric statistical hypothesis test used for comparing two related samples, which is more suitable than the pairwise t-test when the sample distribution can not assume to be a normal distribution.

For the two-tailed test, the null hypothesis (H_0), and the alternative hypothesis (H_1) are indicated as follows:

H_0: The performance of the proposed model and the compared model are no different.

H_1: The performance of the proposed model and the compared model are different.

For the one-tailed test, the null hypothesis (H_0), and the alternative hypothesis (H_1) are indicated as follows:

H_0: The performance of the proposed model is not greater than the compared model.

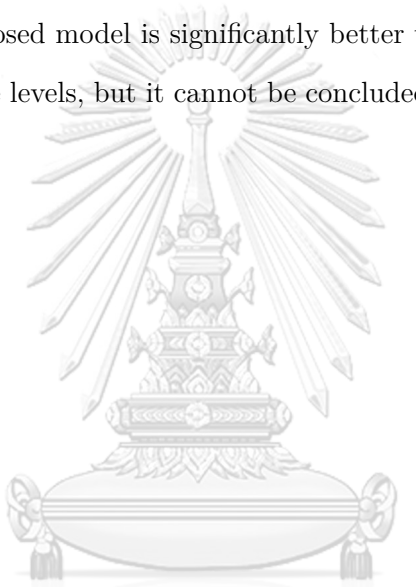
H_1: The performance of the proposed model is greater than the compared model.

The test starts with ranking the difference of the performance between the proposed model and the compared model according to datasets. Then, let R^+ and R^- be the sum of ranks corresponding to the cases that the proposed model is better and the cases that the compared model is better, respectively. So, the statistical value T is obtained from the minimum value between R^+ and R^- for the two-tailed test, and it is R^- for the one-tailed test. For a large number of datasets, statistics z defined by (2.30) is the standard score of the approximately normal distribution, in which N is the number of datasets.

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{2.30}$$

Accordingly, for the two-tailed test, a null-hypothesis can be rejected if z is smaller than -2.58, -1.96, -1.65 when $\alpha = 0.01$, 0.05, and 0.1, respectively. Also, for the one-tailed test, the null-hypothesis can be rejected if z is smaller than -2.33, -1.65, -1.28 when $\alpha = 0.01$, 0.05, and 0.1, respectively.

In practice, this dissertation uses the one-tailed Wilcoxon signed-rank test to compare the performance between the proposed model and the compared model. For example, conducting on 20 datasets, the sum of ranks for the cases that the compared model is better than the proposed model (R^-) is equal to 50. So, statistics z can be calculated as -2.05. That is, the proposed model is significantly better than the compared model with 90% and 95% confidence levels, but it cannot be concluded that it is significant at a 99% confidence level.



CHAPTER III

DECISION TREE INDUCTION FOR A BINARY-CLASS IMBALANCED NUMERIC DATASET

In this chapter, the splitting measure is introduced called the minority condensation entropy (MCE). It enhances the concept of modifying the entropy components by applying the outlier detection. MCE is used as the splitting condition in the recursive partitioning algorithm to build the decision tree called the minority condensation decision tree (MCDT). In addition, MCE is also employed in the next recursive partitioning algorithm to construct the oblique decision tree called the oblique minority condensation decision tree (OMCT).

The motivation of the methods presented in this chapter comes from the success of using the minority entropy (ME) to build a decision tree for handling the binary-class imbalanced numeric dataset. For each attribute, using the minority range directly to determine the region of the minority class shows the impressive results. However, if some minority instance values extremely deviate from the others, which are outliers [84], the minority range will be wider. For example, in Figure 3.1(a), there is one minority instance that has the value of attribute A_1 much higher than the others. It unnecessarily widens the minority range, which covers more majority instances. Moreover, the distribution of the real-world datasets usually does not lie on any parallel axes so many majority instances may not be discarded by the minority range as shown in Figure 3.1(b). The region of the minority class will be appropriately defined, which is able to significantly reduce the number of majority instances, if the minority range of the suitable axis is considered. So, the concept of ME will be more effective when inducing the decision tree using the oblique hyperplanes.

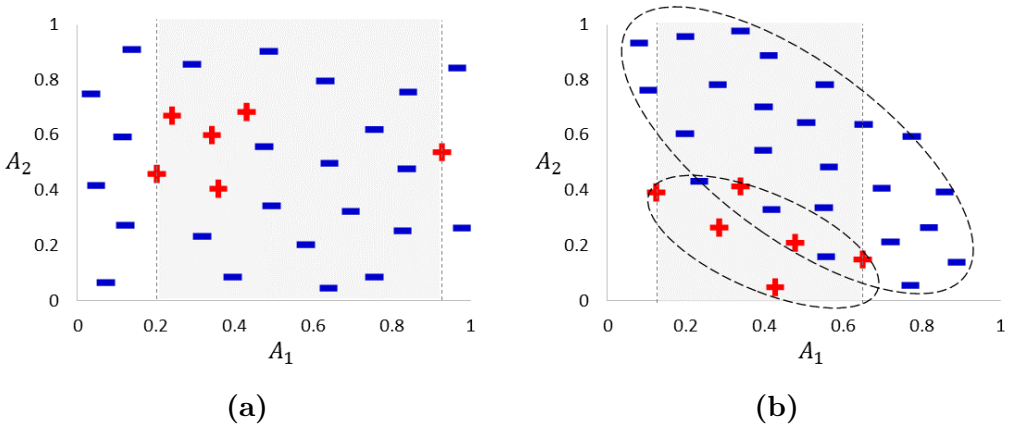


Figure 3.1: Limitations of using the minority range to determine the region of the minority class (a), and discard the majority instances (b)

For these reasons, therefore, an improvement of ME is proposed in this chapter. The interquartile range (IQR) rule is employed to the set of minority instance values for detecting the outliers. It defines the boundary that represents the range of acceptable values for the minority instances based on the Tukey's boxplot [85]. The lower inner fence is defined by the first quartile minus 1.5 times of IQR, while the upper inner fence is defined by the third quartile plus 1.5 times of IQR. So, only the minority instances having values within the inner fences are kept to calculate the minority range. For example, Figure 3.2 demonstrates the use of the IQR rule. The minority instance outside the inner fences is discarded in determining the minority range. It creates a smaller minority range and

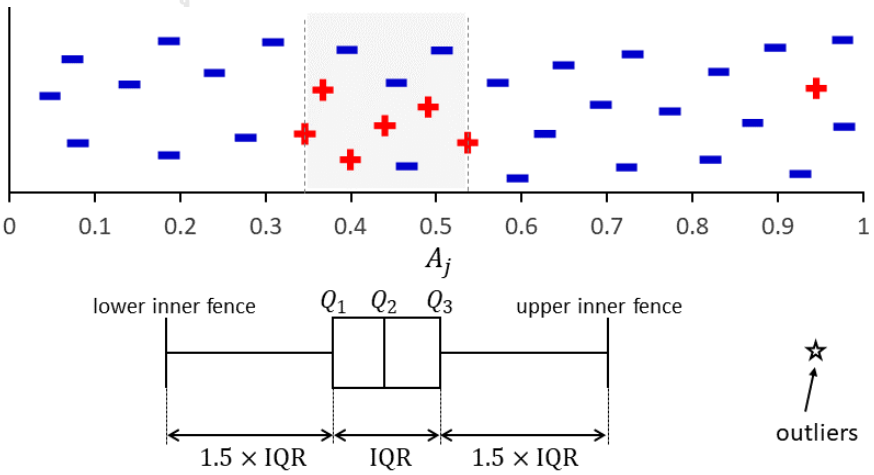


Figure 3.2: Applying the interquartile range rule to detect outliers before determining the minority range

greatly reduces the number of majority instances. Then, the set of instances within that range is considered, in which the minority class is more condensed. Accordingly, the entropy computed with that set is called the minority condensation entropy (MCE), and the decision tree built based on MCE is called the minority condensation decision tree (MCDT).

In addition, this chapter also presents the application of MCE to construct an oblique decision tree. It is used as the splitting measure to select the splitting value in each step of the OC1 algorithm, and then named a generated decision tree to be the oblique minority condensation decision tree (OMCT). At the beginning, MCE is employed in determining the initial hyperplane, which is the best axis-parallel hyperplane as used in MCDT. Then, for each attribute A_j, the deterministic hill-climbing method with MCE is used in finding the best value of a_j from set $u_j(\mathbf{D})$ (2.9), which is restricted to the region of the minority class as shown in Figure 3.3(a). Likewise, the randomization method with MCE is used in finding the best value of α from set $v(\mathbf{D})$ (2.10), which is restricted to the region of the minority class as shown in Figure 3.3(b).

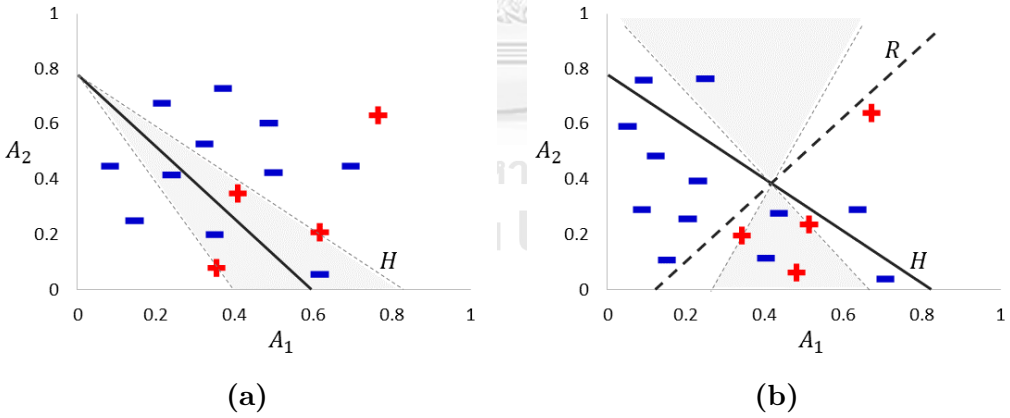


Figure 3.3: Applying the minority condensation entropy to the deterministic hill-climbing method (a), and the randomization method (b)

3.1 Proposed Methodologies

In this section, the mathematical definitions corresponding to the proposed methodologies are formally introduced. It also includes examples and the pseudocode for each proposed algorithm.

3.1.1 Minority Condensation Entropy

Initially, the minority condensation entropy or MCE is presented as a new splitting measure in accordance with the motivation discussed above. It is designed to work with binary-class imbalanced dataset $\mathbf{D} = \mathbf{D}_+ \cup \mathbf{D}_- = \{(\vec{x}_i, y_i) \mid i = 1, 2, ..., m\}$ consisting only of numeric attributes. Let s be a function that maps each instance $\vec{x}_i$ in dataset $\mathbf{D}$ to the set of real numbers, i.e. $s : \mathbf{D} \to \mathbb{R}$. For example with two-dimensional dataset $\mathbf{D} \subseteq \mathbb{R}^2$, define the instance values function as $s(\vec{x}_i) = s(x_1^i, x_2^i) = x_1^i + x_2^i$. Then, for any set of instance values $s(\mathbf{D}) = \{s(\vec{x}_i) \in \mathbb{R} \mid (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, 2, ..., m\}$ corresponding to $\mathbf{D}$, define $s^*(\mathbf{D})$ be the subset of $s(\mathbf{D})$ that ignores the outliers shown in (3.1) as follows:

$$s^*(\mathbf{D}) = \{s(\vec{x}_i) \in s(\mathbf{D}) \mid Q_1 - 1.5 * IQR \leq s(\vec{x}_i) \leq Q_3 + 1.5 * IQR \text{ for } i = 1, 2, ..., m\} \tag{3.1}$$

where Q_1 is the first quartile of $s(\mathbf{D})$, Q_3 is the third quartile of $s(\mathbf{D})$, and IQR is the interquatile range of $s(\mathbf{D})$ which is equal to $Q_3 - Q_1$.

So, the minority range that ignores the outliers of $s(\mathbf{D})$ is determined by the range between the minimum value and the maximum value of $s^*(\mathbf{D}_+)$, i.e. $\phi_+^*(s(\mathbf{D})) = [\min s^*(\mathbf{D}_+), \max s^*(\mathbf{D}_+)]$. Then, $\Phi_+^*(s(\mathbf{D}))$ implies the subset of instances within the minority range that ignores the outliers, $\Phi_+^*(s(\mathbf{D})) = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid s(\vec{x}_i) \in \phi_+^*(s(\mathbf{D})) \text{ for } i = 1, 2, ..., m\}$. Thus, the definition of the minority condensation entropy according to $s(\mathbf{D})$ is determined by (3.2) as follows:

$$MCE_s(\mathbf{D}) = Entropy(\Phi_+^*(s(\mathbf{D}))) \tag{3.2}$$

The brief pseudocode to compute MCE of dataset $\mathbf{D}$ with respect to a function to obtain the set of instance values s is displayed in Algorithm 3.1. The minority range that ignores the outliers is generated to limit the set of instances before calculating the entropy.

Algorithm 3.1: MCE($\mathbf{D}, s$)

Input: dataset $\mathbf{D}$, a function to obtain the instance values s
Output: the minority condensation entropy of $\mathbf{D}$ with respect to s

1 • generate the set of instance values corresponding to the minority class without the outliers: $s^*(\mathbf{D}_+)$
2 • create the minority range that ignores the outliers: $\phi^*_+(s(\mathbf{D}))$
3 • compute the subset of instances within the minority range that ignores the outliers: $\Phi^*_+(s(\mathbf{D}))$
4 **return** $Entropy(\Phi^*_+(s(\mathbf{D})))$

3.1.2 Minority Condensation Decision Tree

In this section, MCE is used to build a decision tree based on a single attribute. For each attribute A_j, the minority range that ignores the outliers of $\pi_j(\mathbf{D})$ is determined by the range between the minimum value and the maximum value of $\pi^*_j(\mathbf{D}_+)$, i.e. $\phi^*_+(\pi_j(\mathbf{D})) = \left[\min \pi^*_j(\mathbf{D}_+), \max \pi^*_j(\mathbf{D}_+)\right]$. Then, $\Phi^*_+(\pi_j(\mathbf{D}))$ implies the subset of instances within the minority range that ignores the outliers, $\Phi^*_+(\pi_j(\mathbf{D})) = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid \pi_j(\vec{x}_i) = x^i_j \in \phi^*_+(\pi_j(\mathbf{D})) \text{ for } i = 1, 2, ..., m\}$. For example, the minority range that ignores the outliers and the subset of instances within it are demonstrated in Figure 3.4. Thus, the minority condensation entropy according to attribute A_j is determined by (3.3).

$$MCE_{\pi_j}(\mathbf{D}) = Entropy(\Phi^*_+(\pi_j(\mathbf{D}))) \tag{3.3}$$

So, a decision tree from the recursive partitioning algorithm using MCE as the splitting measure is called the minority condensation decision tree or MCDT, and that algorithm is also called the MCDT algorithm. Its pseudocode is displayed in Algorithm 3.2. For each attribute A_j, the greedy approach is applied to select the best condition for splitting dataset $\mathbf{D}$ based on the value of MCE according to each candidate from $\pi_j(\mathbf{D})$.

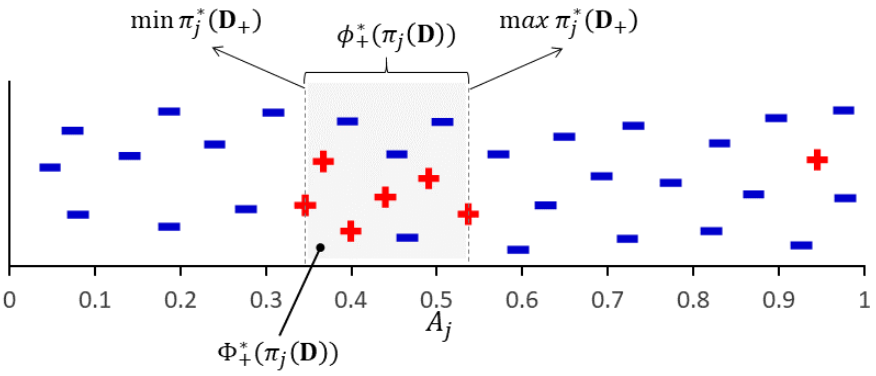


Figure 3.4: The example of the minority range that ignores the outliers and the subset of instances within them

Algorithm 3.2: MCDT($\mathbf{D}$)

Input: dataset $\mathbf{D}$
Output: a decision tree

- create the root node of the tree
- **if** all instances in $\mathbf{D}$ are in the same class **then**
  - **return** the leaf node with respect to that class
- **else**
  - `/* select the splitting condition */`
  - **for** each attribute A_j **do**
    - apply the greedy approach on $\pi_j(\mathbf{D})$ based on the minority condensation entropy via **Algorithm 3.1**
    - update the best splitting condition
  - obtain the best splitting condition
  - `/* recursively partition the dataset */`
  - separate the dataset into 2 partitions corresponding to the outcomes of the splitting condition: $\mathbf{D} = \mathbf{D}^{(\text{left})} \cup \mathbf{D}^{(\text{right})}$
  - iterate for each partition: MCDT($\mathbf{D}^{(\text{left})}$) and MCDT($\mathbf{D}^{(\text{right})}$)

3.1.3 Oblique Minority Condensation Decision Tree

In this section, MCE is used to build a decision tree based on multi-attribute. For binary-class dataset $\mathbf{D} = \mathbf{D}_+ \cup \mathbf{D} = \{(\vec{x}_i, y_i) \mid i = 1, 2, ..., m\}$, MCE is applied to each step of the OC1 algorithm (Algorithm 2.2) for building the oblique decision tree as follows:

- First, MCE is employed to obtain initial hyperplane H. For each attribute A_j, the greedy approach is used to find the best splitting value based on the minority

condensation entropy corresponding to the set of instance values $\pi_j(\mathbf{D})$. It has been determined by (3.3) from the previous section.

- Second, MCE is employed to the deterministic hill-climbing process for perturbing hyperplane H along each attribute A_j. The greedy approach is used to find the optimal value of a_j based on the minority condensation entropy corresponding to the set of instance values $u_j(\mathbf{D})$ (2.9).

  For each attribute A_j, the minority range that ignores the outliers of $u_j(\mathbf{D})$ is determined by the range between the minimum value and the maximum value of $u_j^*(\mathbf{D}_+)$, i.e. $\phi_+^*(u_j(\mathbf{D})) = \left[\min u_j^*(\mathbf{D}_+), \max u_j^*(\mathbf{D}_+)\right]$. Then, $\Phi_+^*(u_j(\mathbf{D}))$ implies the subset of instances within the minority range that ignores the outliers, $\Phi_+^*(u_j(\mathbf{D})) = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid u_j(\vec{x}_i) \in \phi_+^*(u_j(\mathbf{D})) \text{ for } i = 1, 2, ..., m\}$. Thus, the minority condensation entropy according to u_j is determined by (3.4) as follows:

$$MCE_{u_j}(\mathbf{D}) = Entropy(\Phi_+^*(u_j(\mathbf{D}))) \tag{3.4}$$

- Third, MCE is employed to the randomization process for perturbing hyperplane H by random hyperplane R. The greedy approach is used to find the optimal value of α based on the minority condensation entropy corresponding to the set of instance values $v(\mathbf{D})$ (2.10).

  The minority range that ignores the outliers of $v(\mathbf{D})$ is determined by the range between the minimum value and the maximum value of $v^*(\mathbf{D}_+)$, i.e. $\phi_+^*(v(\mathbf{D})) = [\min v^*(\mathbf{D}_+), \max v^*(\mathbf{D}_+)]$. Then, $\Phi_+^*(v(\mathbf{D}))$ implies the subset of instances within the minority range that ignores the outliers, $\Phi_+^*(v(\mathbf{D})) = \{(\vec{x}_i, y_i) \in \mathbf{D} \mid v(\vec{x}_i) \in \phi_+^*(v(\mathbf{D})) \text{ for } i = 1, 2, ..., m\}$. Thus, the minority condensation entropy according to v is determined by (3.5) as follows:

$$MCE_V(\mathbf{D}) = Entropy(\Phi_+^*(v(\mathbf{D}))) \tag{3.5}$$

So, an oblique decision tree from the recursive partitioning algorithm using MCE as the splitting measure is called the oblique minority condensation decision tree or OMCT, and that algorithm is also called the OMCT algorithm. Its pseudocode is displayed in Algorithm 3.3 consisting of three main steps discussed above to find the best oblique hyperplane H. For each step, the greedy approach is applied to select the best condition for splitting dataset $\mathbf{D}$ based on the value of MCE according to each candidate from $\pi_j(\mathbf{D})$, $u_j(\mathbf{D})$ and $v(\mathbf{D})$, respectively.

3.2 Experiments and Results

In order to evaluate the performance of inducing a decision tree that uses MCE as the splitting measure, two collections of experiments are conducted. The first collection employs the synthetic datasets varying their imbalanced ratios to show the effectiveness of the proposed splitting measure like MCE over the traditional splitting measure that is widely used like SE. For the second collection, the performance comparison in classifying the real-world datasets from the UCI repository of MCDT and OMCT with other methods is presented.

3.2.1 Experiments on Synthetic Datasets

In this section, an improvement of the Shannon's entropy (SE) to classify minority instances in the binary-class imbalanced datasets dealing with numeric attributes using the minority condensation entropy (MCE) is demonstrated in the experiments on the collections of synthetic datasets generated according to the following specifications:

- Each dataset contains 1000 instances consisting of 10 numeric attributes.
- Each instance is labeled as either the minority class or the majority class.
- For each attribute, an overlapping range of each class is randomly defined. Then, the uniform sampling is performed within that range.
- There are ten groups of experiments varying the percentages of minority instances from 5% to 50%.

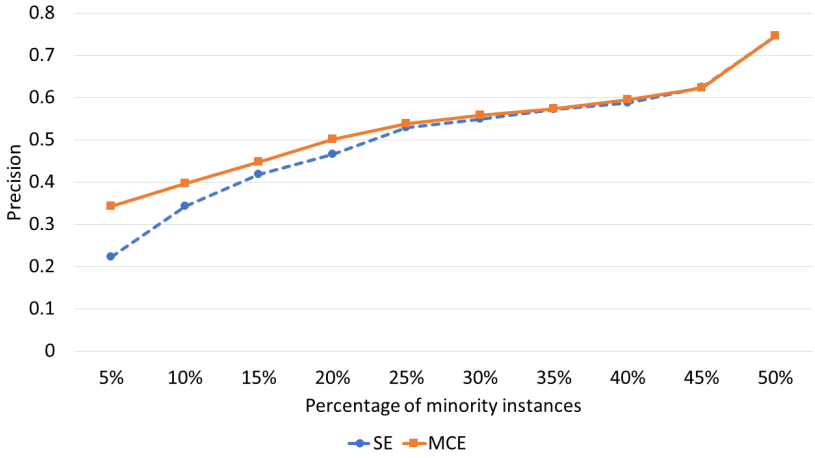
Algorithm 3.3: OMCT(**D**)

Input: dataset **D**
Output: a decision tree

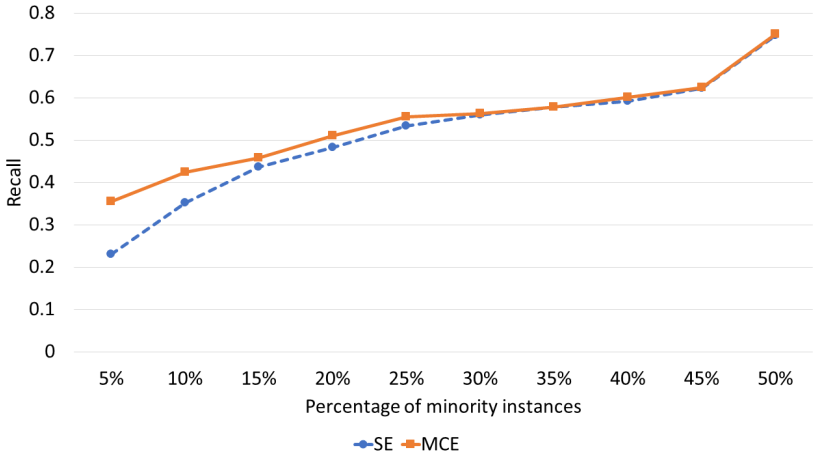
- • create the root node of the tree
- **if** all instances in **D** are in the same class **then**
  - **return** the leaf node with respect to that class
- **else**
  - `/* select the initial hyperplane */`
  - **for** each attribute A_j **do**
    - • apply the greedy approach on $\pi_j(\mathbf{D})$ based on the minority condensation entropy via **Algorithm 3.1**
    - • update the best splitting condition
  - • obtain the best splitting condition as the initial hyperplane $H : \vec{a} \cdot \vec{x} - a_0 = 0$
  - `/* perturb the hyperplane */`
  - **do**
    - `/* apply the deterministic hill-climbing algorithm */`
    - **for** each attribute A_j **do**
      - • compute the set of the candidate values of a_j: $u_j(\mathbf{D})$ (2.9)
      - • apply the greedy approach on $u_j(\mathbf{D})$ based on the minority condensation entropy (MCE) via **Algorithm 3.1**
      - • update the best value of a_j
    - • obtain the adjusted hyperplane H
    - `/* apply the randomization algorithm */`
    - • random a hyperplane $R : \vec{r} \cdot \vec{x} - r_0 = 0$
    - • compute the set of the candidate values of α: $v(\mathbf{D})$ (2.10)
    - • apply the greedy approach on $v(\mathbf{D})$ based on the minority condensation entropy (MCE) via **Algorithm 3.1**
    - • obtain the adjusted hyperplane $H : H + \alpha R$
  - **while** hyperplane H has been improved;
  - • obtain the best hyperplane H
  - `/* recursively partition the dataset */`
  - • separate the dataset into 2 partitions corresponding to the outcomes of the hyperplane H: $\mathbf{D} = \mathbf{D}^{(\text{left})} \cup \mathbf{D}^{(\text{right})}$
  - • iterate for each partition: MCDT($\mathbf{D}^{(\text{left})}$) and MCDT($\mathbf{D}^{(\text{right})}$)

- For each group, ten datasets are synthesized and then applying five-fold cross-validation ten times to evaluate the performance of each method.

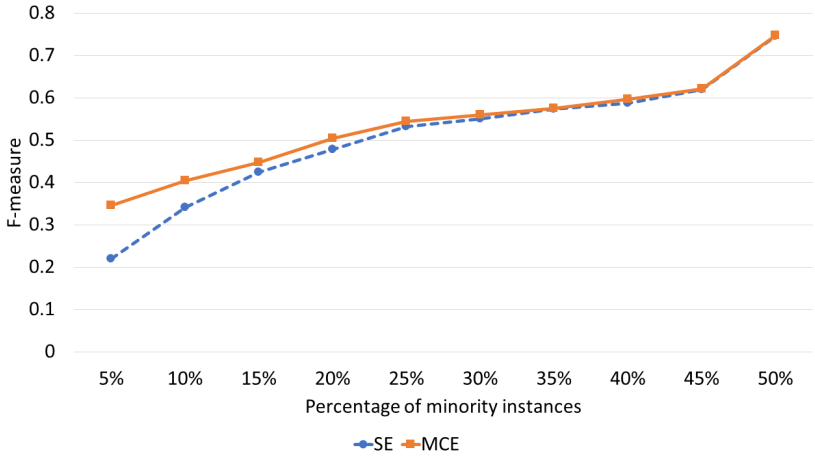
Accordingly, the average results of SE and MCE are compared via the precision (2.16), the recall (2.17), and the F-measure (2.18) with respect to the minority class and



(a) comparing by the precision

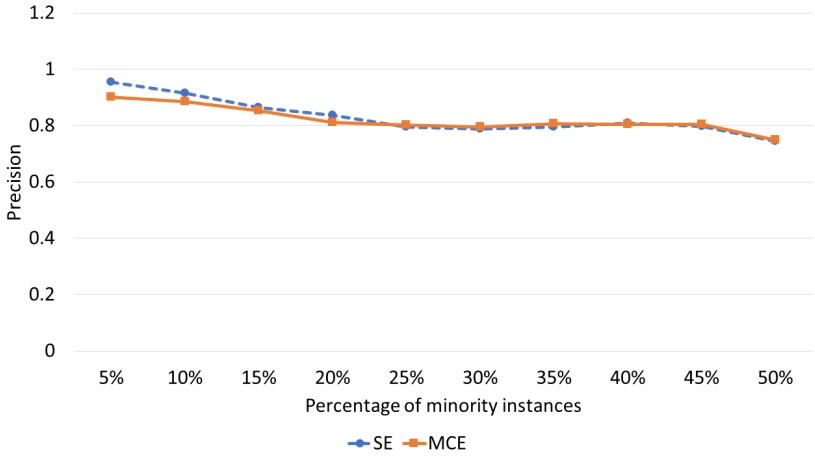


(b) comparing by the recall

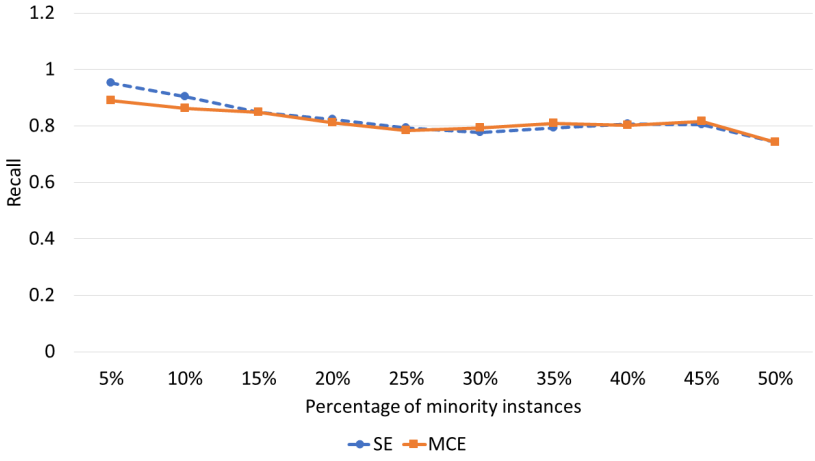


(c) comparing by the F-measure

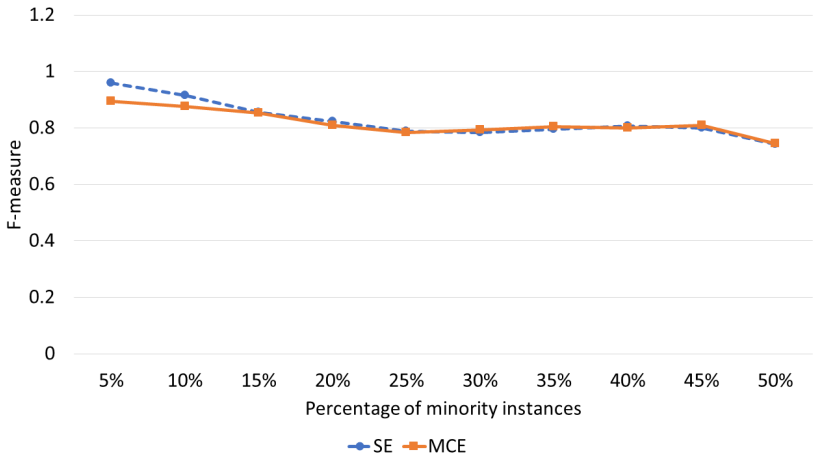
Figure 3.5: The experimental results with respect to the minority class on the synthetic binary-class datasets varying the percentage of minority instances



(a) comparing by the precision



(b) comparing by the recall



(c) comparing by the F-measure

Figure 3.6: The experimental results with respect to the majority class on the synthetic binary-class datasets varying the percentage of minority instances

the majority class displaying in Figure 3.5 and Figure 3.6, respectively.

For the results corresponding to the minority class, their results are similar in all performance measures, which ostensibly increase when the percentage of minority instances increases. Nonetheless, MCE significantly outperforms SE when the number of instances in the minority class is tiny. For example, the performance of SE is only about 0.2 when there are 5 % of minority instances, while MCE offers the performance up to 0.35. Then their values will approach the same value when a dataset is more balanced, which are equal to 0.75 when the dataset is completely balanced. It is because SE tends to focus on the class having a large number of instances, while MCE tries to make them balanced before considering. Therefore, MCE is better than SE when there is a large difference between the number of instances in each class. These results evidently confirm that SE is not suitable to deal with the binary-class imbalanced problem, which is able to be captured using MCE.

Differently, considering the results based on the majority class, all performance measures gradually decrease when a dataset is more balanced. That is because of the decrease in the percentage of majority instances. However, due to the effort to reduce the bias toward the majority class of MCE, it gives a slightly worse result according to the majority class compared to SE in the case of extreme imbalance. For example, the performance of SE is about 0.95 when there are only 5 % of minority instances, while MCE offers the performance around 0.9. Their values will approach to the same value when a dataset is more balanced, which are equal to 0.75 when the dataset is completely balanced.

3.2.2 Experiments on Real-world Datasets

To demonstrate the effectiveness of MCE on a general dataset, the decision trees built based on MCE like MCDT and OMCT are evaluated with the experiments on real-world datasets in this section. Their results are compared to those of seven other decision trees. The first two models are the decision trees from two well-known algorithms, i.e. CART [44] and C.45 [39]. Additionally, the decision tree built based on DCSM [45] is

used as well. The others are three state-of-the-art models designed for a binary-class imbalanced dataset consisting of the decision trees built based on AE [63, 64] and ME [40], including with HDDT [70, 71]. Lastly, the oblique decision tree that is induced with the most famous algorithm like OC1 [48] is also used in the comparison.

Datasets

For the datasets used in the experiments, they are collected from the UCI repository [86] having a total of 23 datasets with the selected classes to be treated as the binary-class imbalanced problems. Their attributes are all numeric, which are summarized in Table 3.1. They are sorted in the ascending order by the imbalanced ratio (I.R.), equivalent to the ascending order of the percentage of instances in the majority class (% Maj.) and the descending order of the percentage of instances in the minority class (% Min.). The first two columns represent the index and the name of each dataset. For the third column and the fourth column, they indicate the number of instances (# Inst.) and the number of attributes (# Att.), respectively. Particularly, the classes determining as the majority class (Maj. Cl.) and the minority class (Min. Cl.) are shown in the next two columns. In order to evaluate the performance of each decision tree, the five-fold cross-validation is employed to divide the dataset into the training set and the testing set, which is repeated 20 times. That is, there are up to one hundred experiments performed on each dataset.

Results

Accordingly, the average results of each decision tree are compared via the precision (2.16), the recall (2.17), the F-measure (2.18), and the number of leaf nodes displaying in four tables for each measure. In each table, a comparison of performance according to each dataset is represented in each row. Furthermore, the rank of each method corresponding to each dataset is shown in the parentheses, emphasizing the best rank in bold. Summarily, the bottom row of each table represents the average rank of each method.

No.	Dataset	# Inst.	# Att.	Maj. Cl.	Min. Cl.	% Maj.	% Min.	I.R.
1	OpticDigits	1108	64	“0”	“8”	50	50	1
2	PenDigits	2110	16	“8”	“5”	50	50	1
3	Vowel	180	10	“9”	“0”	50	50	1
4	Ionosphere	351	34	“g”	“b”	64.1	35.9	1.79
5	Pima	768	8	“0”	“1”	65.1	34.9	1.87
6	Wine	178	13	The rest	“3”	73.03	26.97	2.71
7	Haberman	306	3	“1”	“2”	73.53	26.47	2.78
8	StatlogVehicle	846	18	The rest	“bus”	74.23	25.77	2.88
9	Parkinsons	195	22	“1”	“0”	75.38	24.62	3.06
10	StatlogShuttle	58000	9	“1”	The rest	78.6	21.4	3.67
11	LibrasMovement	360	90	The rest	“1”, “2”, “3”	80	20	4
12	StatlogImage	2310	19	The rest	“5”	85.71	14.29	6
13	BreastTissue	106	9	The rest	“fad”	85.85	14.15	6.07
14	NewThyroid	215	5	The rest	“3”	86.05	13.95	6.17
15	Fertility	100	9	“N”	“O”	88	12	7.33
16	Ecoli	336	7	The rest	“imU”	89.58	10.42	8.6
17	StatlogLandsat	6435	36	The rest	“4”	90.27	9.73	9.28
18	Glass	214	9	The rest	“5”	93.93	6.07	15.46
19	PageBlocks	5473	10	The rest	“2”	93.99	6.01	15.64
20	Winequality-red	1599	11	The rest	“3”, “4”	96.06	3.94	24.38
21	Winequality-white	4898	11	The rest	“3”, “4”	96.26	3.74	25.77
22	Letter	20000	16	The rest	“H”	96.33	3.67	26.25
23	Yeast	1484	8	The rest	“VAC”	97.98	2.02	48.47

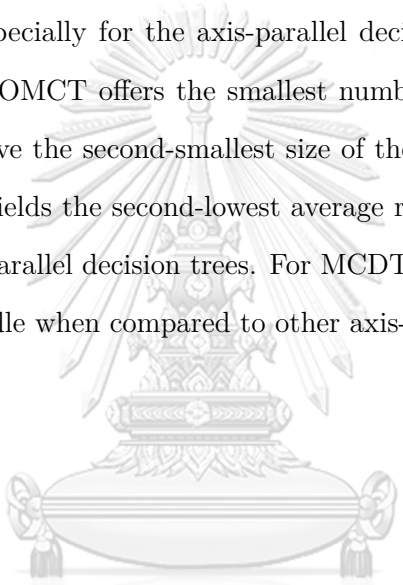
Table 3.1: The characteristics of the experimental real-world binary-class datasets

In Table 3.2, the performance comparison according to the precision of each method is shown. DCSM yields the lowest average rank at 4.13, which is not much different from MCDT and OMCT. They yield the third-lowest average rank and the second-lowest average rank at 4.3 and 4.22, respectively. There are up to 10 out of 23 datasets that OMCT offers the highest or the second-highest precision, which is more than DCSM that has only 9 datasets. For MCDT, it provides the highest precision in 3 datasets which are not less than other methods except for DCSM and OMCT.

In Table 3.3, the performance comparison according to the recall of each method is shown. OMCT yields the lowest average rank at 3.43, which is much different from CART that yields the second-lowest average rank at 4.43. There are up to 12 out of 23 datasets that OMCT offers the highest or the second-highest recall. For MCDT, it yields the third-lowest average rank at 4.57 having 3 datasets that it offers the second-highest recall which is equal to CART. However, there are 4 datasets that MCDT offers the highest recall, while CART has not.

In Table 3.4, the performance comparison according to the F-measure of each method is shown. OMCT yields the lowest average rank at 3.78, which is much different from other methods. There are up to 11 out of 23 datasets that OMCT offers the highest or the second-highest F-measure. For MCDT, it yields the second-lowest average rank at 4.13 that is the best rank when considering only the axis-parallel decision trees. There are 5 datasets that MCDT offers the highest F-measure which is equal to OMCT.

In Table 3.5, the tree size comparison according to the number of leaf nodes of each method is shown. OMCT yields the lowest average rank at 1.57, which is much different from other methods, especially for the axis-parallel decision trees. There are up to 15 out of 23 datasets that OMCT offers the smallest number of leaf nodes, while 6 out of 8 remaining datasets have the second-smallest size of the tree. Besides, another oblique decision tree like OC1 yields the second-lowest average rank at 2.13, which is also much different from the axis-parallel decision trees. For MCDT, it receives the average rank at 5.7, which is in the middle when compared to other axis-parallel decision trees.



No.	Dataset	Axis-parallel Decision Tree							Oblique DT	
		CART	C.45	DCSM	AE	HDDT	ME	MCDT	OC1	OMCT
1	OpticDigits	0.9893 (3)	0.9876 (4)	0.9824 (8)	0.9867 (5)	0.9802 (9)	0.9827 (7)	0.9854 (6)	**0.9946 (1)**	0.9937 (2)
2	PenDigits	0.9856 (8)	0.9865 (6)	0.9891 (3)	0.9858 (7)	0.9915 (2)	0.9880 (4)	**0.9936 (1)**	0.9868 (5)	0.9855 (9)
3	Vowel	0.9716 (5)	0.9716 (5)	**0.9924 (1)**	0.9716 (5)	0.9742 (3)	0.9716 (5)	0.9918 (2)	0.9716 (4)	0.9641 (9)
4	Ionosphere	0.8719 (5)	**0.9272 (1)**	0.8726 (4)	0.8467 (8)	0.8533 (7)	0.8782 (2)	0.8774 (3)	0.8588 (6)	0.8308 (9)
5	Pima	0.5516 (9)	0.5705 (3)	0.5619 (8)	0.5650 (6)	0.5833 (2)	0.5665 (5)	**0.5865 (1)**	0.5669 (4)	0.5621 (7)
6	Wine	0.9429 (8)	0.9397 (9)	0.9461 (7)	**0.9804 (1)**	0.9597 (2)	0.9570 (3)	0.9532 (4)	0.9488 (6)	0.9491 (5)
7	Haberman	0.3468 (7)	0.3648 (3)	0.3665 (2)	0.3577 (5)	0.3477 (6)	0.3415 (8)	0.3283 (9)	0.3596 (4)	**0.3670 (1)**
8	StatlogVehicle	0.9201 (6)	0.8995 (9)	**0.9357 (1)**	0.9040 (8)	0.9227 (4)	0.9283 (3)	0.9290 (2)	0.9219 (5)	0.9133 (7)
9	Parkinsons	0.7254 (5)	0.6412 (9)	0.6724 (8)	0.7223 (6)	0.6746 (7)	0.7810 (3)	**0.7932 (1)**	0.7623 (4)	0.7883 (2)
10	StatlogShuttle	0.9998 (6)	0.9994 (9)	**0.9999 (1)**	0.9998 (8)	0.9999 (3)	0.9998 (6)	0.9998 (5)	0.9999 (4)	0.9999 (2)
11	LibrasMovement	0.7768 (6)	0.6085 (9)	0.7990 (3)	0.7698 (7)	**0.8325 (1)**	0.7941 (4)	0.7941 (4)	0.7694 (8)	0.8016 (2)
12	StatlogImage	0.8997 (2)	0.8516 (9)	0.8932 (4)	0.8914 (5)	**0.9054 (1)**	0.8747 (8)	0.8777 (7)	0.8856 (6)	0.8995 (3)
13	BreastTissue	0.3664 (5)	0.3818 (4)	0.3933 (2)	0.2667 (9)	0.3910 (3)	0.3160 (7)	0.3160 (7)	0.3511 (6)	**0.3968 (1)**
14	NewThyroid	**0.9127 (1)**	0.9092 (4)	0.8607 (9)	0.8898 (7)	0.8696 (8)	0.9056 (5)	0.9056 (5)	0.9101 (2)	0.9101 (2)
15	Fertility	0.1412 (8)	0.1462 (6)	0.2158 (2)	0.1292 (9)	0.1417 (7)	0.1562 (5)	0.1808 (4)	0.1963 (3)	**0.2875 (1)**
16	Ecoli	0.6314 (3)	0.5101 (9)	0.6182 (7)	0.6269 (4)	0.6417 (2)	0.6206 (5)	0.6206 (5)	**0.6607 (1)**	0.6182 (8)
17	StatlogLandsat	0.5333 (8)	0.5214 (9)	0.5713 (2)	**0.5761 (1)**	0.5581 (6)	0.5587 (5)	0.5596 (4)	0.5555 (7)	0.5646 (3)
18	Glass	0.6383 (8)	0.6375 (9)	0.6608 (6)	0.6908 (3)	0.6908 (3)	**0.7317 (1)**	0.6451 (7)	0.6917 (2)	0.6883 (5)
19	PageBlocks	0.8594 (8)	0.8515 (9)	0.8841 (3)	0.8676 (6)	0.8760 (4)	0.8653 (7)	0.8731 (5)	0.8845 (2)	**0.8922 (1)**
20	Winequality-red	0.1840 (3)	**0.2292 (1)**	0.1520 (9)	0.1524 (8)	0.1725 (5)	0.1697 (6)	0.1840 (4)	0.2088 (2)	0.1681 (7)
21	Winequality-white	0.3062 (9)	0.3167 (6)	**0.3469 (1)**	0.3457 (2)	0.3154 (7)	0.3234 (5)	0.3406 (3)	0.3110 (8)	0.3322 (4)
22	Letter	0.7570 (9)	0.7769 (7)	**0.8118 (1)**	0.8067 (3)	0.7866 (6)	0.8114 (2)	0.8045 (4)	0.7754 (8)	0.8038 (5)
23	Yeast	0.0596 (5)	0.0543 (8)	0.0836 (3)	**0.1050 (1)**	0.0358 (9)	0.0551 (7)	0.0554 (6)	0.0727 (4)	0.1000 (2)
	average rank	5.96	6.43	4.13	5.39	4.65	4.91	4.3	4.43	4.22

Table 3.2: The experimental results on the real-world binary-class datasets comparing by the precision

No.	Dataset	Axis-parallel Decision Tree							Oblique DT	
		CART	C.45	DCSM	AE	HDDT	ME	MCDT	OC1	OMCT
1	OpticDigits	0.9896 (5)	0.9923 (3)	0.9869 (9)	0.9869 (8)	0.9896 (4)	0.9887 (6)	0.9878 (7)	**0.9932 (1)**	0.9923 (2)
2	PenDigits	0.9884 (4)	0.9839 (8)	0.9874 (7)	0.9839 (8)	**0.9927 (1)**	0.9910 (3)	0.9924 (2)	0.9879 (6)	0.9882 (5)
3	Vowel	0.9722 (5)	0.9722 (5)	**0.9889 (1)**	0.9722 (5)	0.9806 (3)	0.9722 (5)	0.9583 (9)	0.9722 (4)	0.9889 (2)
4	Ionosphere	0.8351 (4)	**0.9185 (1)**	0.8032 (8)	0.7815 (9)	0.8054 (7)	0.8153 (6)	0.8172 (5)	0.8389 (3)	0.8430 (2)
5	Pima	0.5597 (6)	0.5748 (3)	0.5459 (8)	0.5140 (9)	0.5589 (7)	0.5699 (4)	**0.5783 (1)**	0.5672 (5)	0.5764 (2)
6	Wine	0.9422 (3)	0.9483 (2)	0.9422 (3)	0.9372 (6)	0.9422 (3)	0.9067 (9)	0.9322 (7)	0.9317 (8)	**0.9628 (1)**
7	Haberman	0.3482 (4)	0.3461 (5)	0.3445 (6)	0.3243 (9)	0.3382 (7)	0.3544 (3)	0.3325 (8)	0.3664 (2)	**0.3700 (1)**
8	StatlogVehicle	0.9071 (7)	0.9026 (8)	0.9211 (4)	0.8990 (9)	**0.9326 (1)**	0.9302 (3)	0.9313 (2)	0.9174 (5)	0.9152 (6)
9	Parkinsons	0.7594 (3)	0.7017 (8)	0.7133 (6)	0.7056 (7)	0.6972 (9)	0.7167 (5)	**0.7794 (1)**	0.7556 (4)	0.7600 (2)
10	StatlogShuttle	0.9998 (2)	0.9997 (6)	0.9995 (9)	0.9997 (4)	**0.9998 (1)**	0.9997 (5)	0.9998 (3)	0.9996 (8)	0.9996 (7)
11	LibrasMovement	0.6710 (5)	0.5948 (9)	0.6402 (8)	0.6760 (2)	0.6674 (6)	0.6750 (3)	0.6750 (3)	0.6579 (7)	**0.7062 (1)**
12	StatlogImage	0.8659 (8)	0.8583 (9)	**0.8939 (1)**	0.8803 (4)	0.8924 (2)	0.8720 (7)	0.8727 (6)	0.8795 (5)	0.8864 (3)
13	BreastTissue	0.3333 (4)	**0.4000 (1)**	0.3000 (7)	0.3167 (6)	0.3333 (4)	0.3000 (7)	0.3000 (7)	0.3333 (3)	0.3833 (2)
14	NewThyroid	0.8750 (4)	0.9083 (3)	0.8333 (7)	0.8083 (9)	0.8333 (7)	0.8750 (4)	0.8750 (4)	**0.9167 (1)**	**0.9167 (1)**
15	Fertility	0.2250 (4)	0.2250 (4)	**0.3083 (1)**	0.1500 (9)	0.2000 (8)	0.2500 (3)	0.2583 (2)	0.2083 (6)	0.2083 (6)
16	Ecoli	0.5714 (5)	0.5571 (8)	0.6000 (2)	0.5357 (9)	**0.6071 (1)**	0.5643 (6)	0.5643 (6)	0.5714 (3)	0.5714 (3)
17	StatlogLandsat	0.5543 (9)	0.5603 (7)	0.5727 (3)	0.5584 (8)	0.5683 (5)	0.5795 (2)	**0.5875 (1)**	0.5699 (4)	0.5655 (6)
18	Glass	0.6667 (2)	0.6500 (3)	0.6167 (7)	0.6250 (6)	0.6417 (4)	0.6417 (4)	**0.6917 (1)**	0.5833 (9)	0.6167 (8)
19	PageBlocks	0.8670 (2)	0.8517 (7)	0.8501 (8)	0.8464 (9)	**0.8715 (1)**	0.8631 (4)	0.8555 (6)	0.8555 (5)	0.8639 (3)
20	Winequality-red	0.1808 (3)	**0.2356 (1)**	0.1410 (8)	0.1378 (9)	0.1458 (7)	0.1490 (6)	0.1497 (5)	0.1821 (2)	0.1513 (4)
21	Winequality-white	0.3184 (3)	**0.3361 (1)**	0.2868 (9)	0.3143 (4)	0.3010 (6)	0.3089 (5)	0.3006 (7)	0.2923 (8)	0.3238 (2)
22	Letter	0.7779 (7)	0.8041 (5)	0.8082 (3)	0.8123 (2)	0.8042 (4)	**0.8232 (1)**	0.7991 (6)	0.7425 (9)	0.7674 (8)
23	Yeast	0.0917 (3)	0.0750 (6)	0.0917 (3)	**0.1250 (1)**	0.0500 (9)	0.0750 (6)	0.0750 (6)	0.0833 (5)	0.1083 (2)
	average rank	4.43	4.91	5.57	6.61	4.65	4.65	4.57	4.91	3.43

Table 3.3: The experimental results on the real-world binary-class datasets comparing by the recall

No.	Dataset	Axis-parallel Decision Tree							Oblique DT	
		CART	C.45	DCSM	AE	HDDT	ME	MCDT	OC1	OMCT
1	OpticDigits	0.9894 (4)	0.9898 (3)	0.9844 (9)	0.9867 (5)	0.9847 (8)	0.9856 (7)	0.9865 (6)	**0.9939 (1)**	0.9930 (2)
2	PenDigits	0.9870 (6)	0.9852 (8)	0.9883 (4)	0.9848 (9)	0.9921 (2)	0.9895 (3)	**0.9930 (1)**	0.9873 (5)	0.9867 (7)
3	Vowel	0.9707 (5)	0.9707 (5)	**0.9903 (1)**	0.9707 (5)	0.9763 (2)	0.9707 (5)	0.9750 (3)	0.9707 (9)	0.9749 (4)
4	Ionosphere	0.8849 (2)	**0.9386 (1)**	0.8730 (6)	0.8547 (9)	0.8654 (8)	0.8786 (5)	0.8787 (4)	0.8794 (3)	0.8703 (7)
5	Pima	0.6561 (8)	0.6691 (3)	0.6585 (7)	0.6523 (9)	0.6737 (2)	0.6669 (4)	**0.6788 (1)**	0.6658 (5)	0.6651 (6)
6	Wine	0.9574 (7)	0.9576 (6)	0.9590 (5)	**0.9691 (1)**	0.9639 (3)	0.9508 (9)	0.9602 (4)	0.9558 (8)	0.9669 (2)
7	Haberman	0.5527 (6)	0.5586 (4)	0.5630 (2)	0.5546 (5)	0.5503 (8)	0.5506 (7)	0.5408 (9)	0.5629 (3)	**0.5665 (1)**
8	StatlogVehicle	0.9411 (7)	0.9329 (8)	0.9514 (2)	0.9327 (9)	0.9505 (4)	0.9512 (3)	**0.9519 (1)**	0.9452 (5)	0.9414 (6)
9	Parkinsons	0.8206 (5)	0.7734 (9)	0.7878 (7)	0.8020 (6)	0.7847 (8)	0.8271 (4)	**0.8557 (1)**	0.8343 (3)	0.8408 (2)
10	StatlogShuttle	0.9999 (3)	0.9997 (9)	0.9998 (8)	0.9998 (4)	**0.9999 (1)**	0.9998 (5)	0.9999 (2)	0.9998 (7)	0.9998 (6)
11	LibrasMovement	0.8242 (5)	0.7440 (9)	0.8166 (8)	0.8240 (6)	0.8400 (2)	0.8315 (3)	0.8315 (3)	0.8169 (7)	**0.8428 (1)**
12	StatlogImage	0.9311 (6)	0.9149 (9)	0.9375 (2)	0.9331 (4)	**0.9406 (1)**	0.9257 (8)	0.9268 (7)	0.9312 (5)	0.9371 (3)
13	BreastTissue	0.6096 (5)	0.6318 (2)	0.6104 (4)	0.5820 (9)	0.6135 (3)	0.5890 (7)	0.5890 (7)	0.6046 (6)	**0.6344 (1)**
14	NewThyroid	0.9356 (4)	0.9437 (3)	0.9058 (8)	0.9050 (9)	0.9080 (7)	0.9333 (5)	0.9333 (5)	**0.9455 (1)**	**0.9455 (1)**
15	Fertility	0.5198 (8)	0.5241 (6)	**0.5628 (1)**	0.5137 (9)	0.5224 (7)	0.5343 (5)	0.5408 (4)	0.5428 (3)	0.5544 (2)
16	Ecoli	0.7695 (3)	0.7266 (9)	0.7694 (4)	0.7542 (8)	**0.7743 (1)**	0.7598 (6)	0.7598 (6)	0.7698 (2)	0.7607 (5)
17	StatlogLandsat	0.7460 (8)	0.7439 (9)	0.7625 (2)	0.7601 (4)	0.7573 (6)	0.7605 (3)	**0.7627 (1)**	0.7568 (7)	0.7586 (5)
18	Glass	0.8038 (2)	0.7986 (6)	0.7912 (7)	0.8031 (3)	0.8013 (4)	**0.8120 (1)**	0.8010 (5)	0.7811 (9)	0.7874 (8)
19	PageBlocks	0.9266 (7)	0.9202 (9)	0.9286 (4)	0.9233 (8)	0.9322 (2)	0.9273 (6)	0.9274 (5)	0.9299 (3)	**0.9344 (1)**
20	Winequality-red	0.5721 (3)	**0.5981 (1)**	0.5549 (8)	0.5529 (9)	0.5618 (5)	0.5612 (7)	0.5651 (4)	0.5791 (2)	0.5617 (6)
21	Winequality-white	0.6417 (7)	0.6486 (3)	0.6440 (5)	**0.6505 (1)**	0.6398 (8)	0.6439 (6)	0.6460 (4)	0.6364 (9)	0.6501 (2)
22	Letter	0.8789 (8)	0.8908 (6)	0.9011 (2)	0.9008 (3)	0.8934 (5)	**0.9049 (1)**	0.8969 (4)	0.8741 (9)	0.8884 (7)
23	Yeast	0.5238 (5)	0.5199 (6)	0.5340 (3)	**0.5401 (1)**	0.5095 (9)	0.5196 (7)	0.5193 (8)	0.5286 (4)	0.5380 (2)
	average rank	5.39	5.83	4.74	5.91	4.61	5.09	4.13	5.04	3.78

Table 3.4: The experimental results on the real-world binary-class datasets comparing by the F-measure

No.	Dataset	Axis-parallel Decision Tree							Oblique DT	
		CART	**C.45**	**DCSM**	**AE**	**HDDT**	**ME**	**MCDT**	**OC1**	**OMCT**
1	OpticDigits	8.60 (7)	8.45 (6)	7.30 (3)	8.70 (8)	8.10 (4)	8.30 (5)	10.30 (9)	**2.85 (1)**	3.15 (2)
2	PenDigits	24.50 (7)	28.50 (9)	17.50 (4)	25.50 (8)	15.30 (3)	17.60 (5)	19.25 (6)	13.40 (2)	**13.00 (1)**
3	Vowel	4.60 (4)	4.60 (4)	**3.80 (1)**	4.60 (4)	4.40 (3)	4.60 (4)	5.30 (9)	4.60 (4)	4.30 (2)
4	Ionosphere	20.25 (7)	21.15 (8)	17.25 (4)	22.30 (9)	17.35 (6)	17.20 (3)	17.25 (4)	**15.65 (1)**	16.65 (2)
5	Pima	110.95 (5)	149.85 (9)	112.10 (6)	125.20 (8)	107.15 (4)	103.90 (3)	114.20 (7)	73.50 (2)	**72.20 (1)**
6	Wine	4.75 (7)	4.75 (7)	3.75 (4)	3.70 (2)	**3.65 (1)**	4.15 (5)	5.35 (9)	3.70 (2)	4.15 (5)
7	Haberman	80.00 (5)	87.75 (9)	80.20 (6)	82.30 (8)	81.45 (7)	79.30 (3)	79.80 (4)	56.35 (2)	**53.80 (1)**
8	StatlogVehicle	26.35 (7)	34.55 (9)	20.05 (3)	27.70 (8)	20.55 (4)	22.35 (5)	22.65 (6)	18.15 (2)	**17.85 (1)**
9	Parkinsons	13.20 (6)	18.20 (9)	12.30 (4)	15.15 (8)	12.75 (5)	**11.80 (1)**	14.95 (7)	11.85 (3)	**11.80 (1)**
10	StatlogShuttle	22.10 (8)	29.65 (9)	12.45 (2)	19.40 (7)	**11.80 (1)**	16.65 (6)	15.25 (3)	16.10 (5)	15.85 (4)
11	LibrasMovement	21.65 (7)	39.50 (9)	17.65 (6)	21.80 (8)	**15.50 (1)**	17.60 (4)	17.60 (4)	16.60 (3)	15.95 (2)
12	StatlogImage	53.75 (7)	69.95 (9)	41.65 (2)	57.00 (8)	43.40 (4)	47.60 (5)	49.20 (6)	42.35 (3)	**41.50 (1)**
13	BreastTissue	10.05 (3)	10.60 (8)	10.25 (5)	12.20 (9)	10.15 (4)	10.30 (6)	10.30 (6)	9.10 (2)	**8.95 (1)**
14	NewThyroid	5.55 (5)	6.20 (9)	5.80 (6)	6.05 (8)	5.95 (7)	5.45 (3)	5.45 (3)	**4.05 (1)**	4.20 (2)
15	Fertility	14.95 (4)	15.35 (7)	15.55 (8)	15.95 (9)	14.85 (3)	15.05 (6)	14.95 (4)	**8.05 (1)**	**8.05 (1)**
16	Ecoli	19.35 (7)	22.20 (9)	19.00 (6)	20.60 (8)	18.40 (5)	18.20 (3)	18.30 (4)	16.25 (2)	**16.05 (1)**
17	StatlogLandsat	253.15 (7)	367.35 (9)	229.85 (6)	259.75 (8)	216.10 (3)	217.10 (5)	217.00 (4)	171.20 (2)	**170.20 (1)**
18	Glass	6.70 (8)	7.30 (9)	6.10 (3)	6.50 (7)	6.10 (3)	6.10 (3)	6.20 (6)	6.00 (2)	**5.95 (1)**
19	PageBlocks	66.95 (5)	77.55 (9)	75.40 (8)	69.00 (6)	63.20 (4)	63.15 (3)	74.90 (7)	44.85 (2)	**43.95 (1)**
20	Winequality-red	63.80 (6)	74.05 (8)	85.30 (9)	62.30 (5)	57.55 (4)	55.60 (3)	66.95 (7)	46.05 (2)	**44.90 (1)**
21	Winequality-white	161.25 (6)	189.50 (8)	201.70 (9)	159.80 (5)	149.95 (4)	144.45 (3)	167.35 (7)	119.85 (2)	**118.70 (1)**
22	Letter	296.25 (8)	301.50 (9)	272.75 (7)	253.10 (4)	255.30 (5)	232.05 (3)	266.75 (6)	157.05 (2)	**150.65 (1)**
23	Yeast	51.90 (7)	55.85 (8)	64.50 (9)	50.65 (6)	49.85 (5)	48.80 (3)	48.80 (3)	**37.90 (1)**	38.45 (2)
	average rank	6.22	8.26	5.26	7	3.91	3.91	5.7	2.13	1.57

Table 3.5: The experimental results on the real-world binary-class datasets comparing by the number of leaf nodes

Discussions

Graphically, the bar chart representing the comparison of results by the average rank corresponding to each measure is shown in Figure 3.7, in which a lower value indicates a better rank. Moreover, the results of the Wilcoxon signed-rank test that compares the performance of MCDT and OMCT with other decision trees are shown in Tables 3.6 and 3.7, respectively. The testing results including the statistical value and the p-value for each comparison is shown in each row, in which the symbol checkmark denotes that the proposed method is significantly better than another method with the $(1-\alpha)100\%$ confidence level. In addition, the testing results are also indicated in Figure 3.7 using the specific symbols for each confidence level.

The experimental results confirm that the conventional decision trees like CART and C4.5 are not suitable to deal with the binary-class imbalanced problem, especially in terms of precision. Their results are significantly less than MDCT with the 95% confidence level the same as ME. It means that they predict a lot of majority instances to be the minority class. For ME, this happens because the range of minority class is unnecessarily wide due to the appearance of outliers, which does not happen in MCDT. On the other hand, although DCSM shows the impressive results in terms of the precision, it provides an unacceptable value of the recall. Likewise, the recall of AE is also unsatisfied that is less than MDCT with a confidence level at 95%. It means that they focus on classifying correct minority instances excessively causing the boundary of partitioning to overfit the minority class. So, there are a small number of actual minority instances that are correctly classified. Importantly, MCDT outperforms other axis-parallel decision trees in terms of the F-measure, which is significantly better than AE and ME with the 95% confidence level, due to their poor values of the recall and the precision, respectively.

For OMCT, it also significantly outperforms the conventional decision trees like CART and DCSM in terms of the precision with the 95% and 99% confidence level, respectively. Moreover, OMCT gives a better value of the recall than MCDT including other methods. Its result is significantly superior to CART, DCSM, HDDT, and ME with

the 95% confidence level, and up to 99% for AE and OC1. That is, the use of the oblique hyperplane as the splitting measure is able to increase the percentage of actual minority instances that are correctly classified. This happens because using all attributes together to determine the boundary of the minority class can avoid the overfitting phenomenon that appears when considering only one attribute. Thus, due to the impressive results of OMCT in both terms of precision and recall, it exactly shows the improvement of the F-measure comparing with all other methods, which is significantly better than CART, C4.5, ME, and OC1 with the 95% confidence level, and up to 99% for AE. Importantly, the results demonstrate that the sizes of oblique decision trees like OC1 and OMCT are obviously smaller than that of axis-parallel decision trees. From the statistical test, OMCT significantly contains a less number of leaf nodes than CART, C4.5, DCSM, AE, HDDT, and ME with up to 99% confidence level. The reason is due to the flexibility of using the oblique hyperplanes to deal with a dataset having various distributions. In addition, the ability to determine the boundary of the minority class using the minority condensation technique makes OMCT significantly small which is smaller than OC1 with the 99% confidence level.

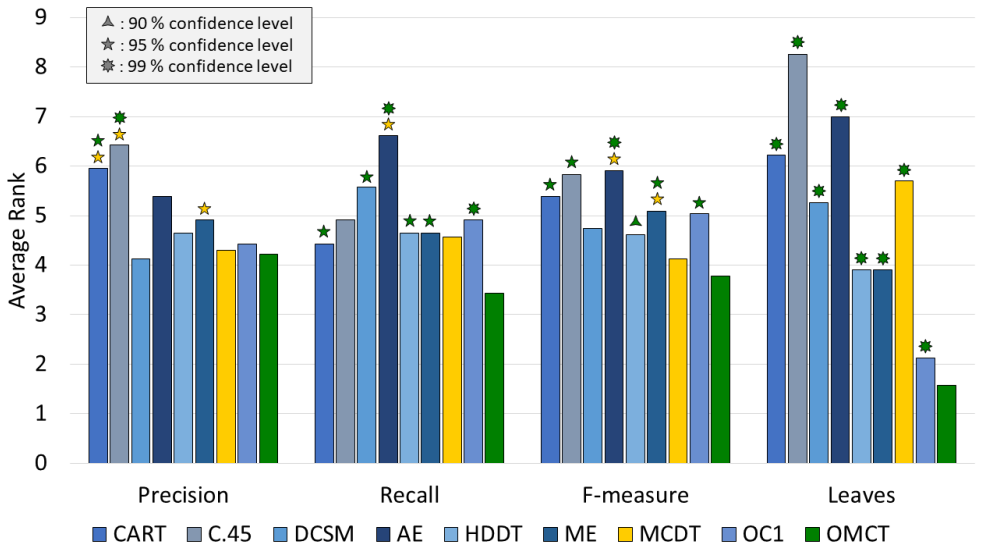


Figure 3.7: The comparison of the experimental results on the real-world binary-class datasets with respect to the average rank

Performance Measures	Decision Tree	α			Statistics	p-value
		0.1	**0.05**	**0.01**		
Precision	CART	✓	✓	-	78	0.034008
	C4.5	✓	✓	-	80	0.038860
	DCSM	-	-	-	170	0.834793
	AE	-	-	-	104	0.150543
	HDDT	-	-	-	118	0.271495
	ME	✓	✓	-	54	0.049479
Recall	CART	-	-	-	120	0.416433
	C4.5	-	-	-	130	0.545234
	DCSM	-	-	-	102	0.213188
	AE	✓	✓	-	76	0.029659
	HDDT	-	-	-	121	0.302559
	ME	-	-	-	64.5	0.180184
F-measure	CART	-	-	-	113	0.223516
	C4.5	-	-	-	97	0.106196
	DCSM	-	-	-	125	0.346276
	AE	✓	✓	-	68	0.016625
	HDDT	-	-	-	109	0.188879
	ME	✓	✓	-	53	0.045498

Table 3.6: The statistical results based on the Wilcoxon signed-rank test comparing MCDT against other decision trees

Performance Measures	Decision Tree	α			Statistics	p-value
		0.1	0.05	0.01		
Precision	CART	✓	✓	-	64	0.012202
	C4.5	✓	✓	✓	53	0.004865
	DCSM	-	-	-	102	0.136772
	AE	-	-	-	100	0.123888
	HDDT	-	-	-	113	0.223516
	ME	-	-	-	101	0.130220
	MCDT	-	-	-	114	0.232708
	OC1	-	-	-	91	0.124552
Recall	CART	✓	✓	-	76.5	0.030698
	C4.5	-	-	-	114.5	0.237369
	DCSM	✓	✓	-	79	0.036368
	AE	✓	✓	✓	33	0.000703
	HDDT	✓	✓	-	78	0.034008
	ME	✓	✓	-	78	0.034008
	MCDT	-	-	-	98	0.111879
	OC1	✓	✓	✓	29	0.002275
F-measure	CART	✓	✓	-	62	0.010402
	C4.5	✓	✓	-	70	0.019310
	DCSM	-	-	-	102	0.136772
	AE	✓	✓	✓	45	0.002338
	HDDT	✓	-	-	84	0.050253
	ME	✓	✓	-	74	0.025794
	MCDT	-	-	-	103	0.143546
	OC1	✓	✓	-	59	0.014210
The number of leaves	CART	✓	✓	✓	0	0.000014
	C4.5	✓	✓	✓	0	0.000014
	DCSM	✓	✓	✓	19.5	0.000156
	AE	✓	✓	✓	2	0.000018
	HDDT	✓	✓	✓	20	0.000166
	ME	✓	✓	✓	45	0.000030
	MCDT	✓	✓	✓	2	0.000018
	OC1	✓	✓	✓	74	0.007114

Table 3.7: The statistical results based on the Wilcoxon signed-rank test comparing OMCT against other decision trees

CHAPTER IV

DECISION TREE INDUCTION FOR A MULTI-CLASS IMBALANCED NUMERIC DATASET

In this chapter, another splitting measure is introduced called the individually weighted entropy (IWE). It adapts the concept of modifying the entropy components according to a set of instance weights. Moreover, this chapter also presents two weighting functions to deal with the multi-class imbalanced problem. They are called the class-overlapping weighting function and the class-balancing weighting function, denoted by α and μ, respectively. Then, IWE according to the set of instance weights obtained from the composition of functions α and μ, which has been specifically named the class-overlapping-balancing entropy (OBE), is used as the splitting condition in the recursive partitioning algorithm to build the decision tree called the self-balancing decision tree (SBDT).

The motivation of the methods presented in this chapter comes from the success of using the minority condensation entropy (MCE) to build a decision tree for handling the binary-class imbalanced numeric dataset. For each attribute, determining a subset of instances based on the specific range of minority instance values to calculate the entropy gives the impressive results. However, it cannot be directly applied to a multi-class imbalanced dataset. The existence of the multi-minority classes and the multi-majority classes creates a lot of overlapping class ranges. For example, in Figure 4.1, there are three minority classes that their ranges cover different subsets of the majority instances, which makes it difficult to split. On the one hand, discarding all instances outside the range of one minority class causes a very small amount of instances in consideration. On the other hand, using the ranges of all minority classes cause a coverage of all majority

instances in the dataset. So, the concept of MCE that determines whether to discard each majority instance or not, by comparing its value with the minority range, is not appropriate for this situation.

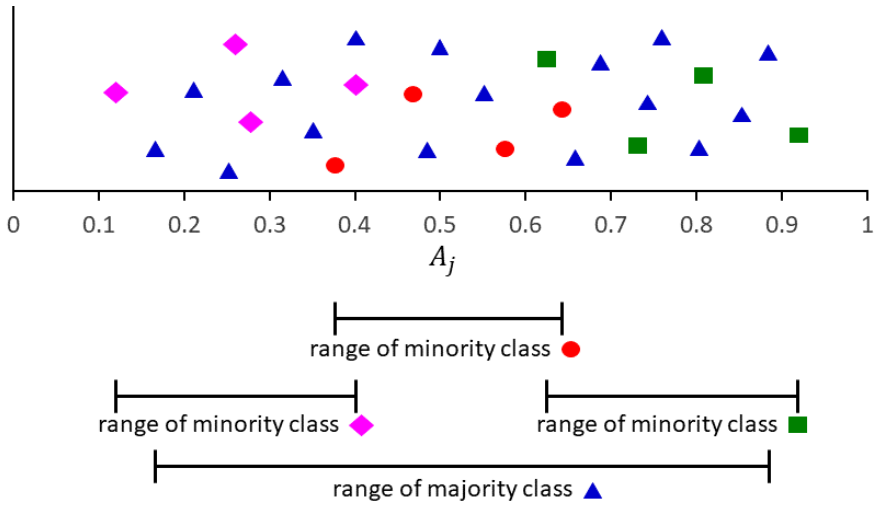


Figure 4.1: The range of instance values from each class according to attribute A_j

For these reasons, an improvement of MCE is proposed in this chapter. Assigning the continuous weighted value to each instance is introduced for a multi-class imbalanced dataset, instead of assigning it either 0 or 1 similar to the mechanism of MCE. Nonetheless, two important concepts of MCE are still used as based assumptions in determining the weights as follows:

1. First, the weights are assigned according to the range of each class.

   Initially, the instances detected as the outliers in their class by IQR rule are not used in determining the class range, and their weights are all set to be 0. For other instances, they are considered in which class ranges cover their positions, which have two issues to discuss:

   - *The number of overlapping classes*

     An instance that is located at the position of overlapping of more classes must have less weight than an instance from the same class that is located at

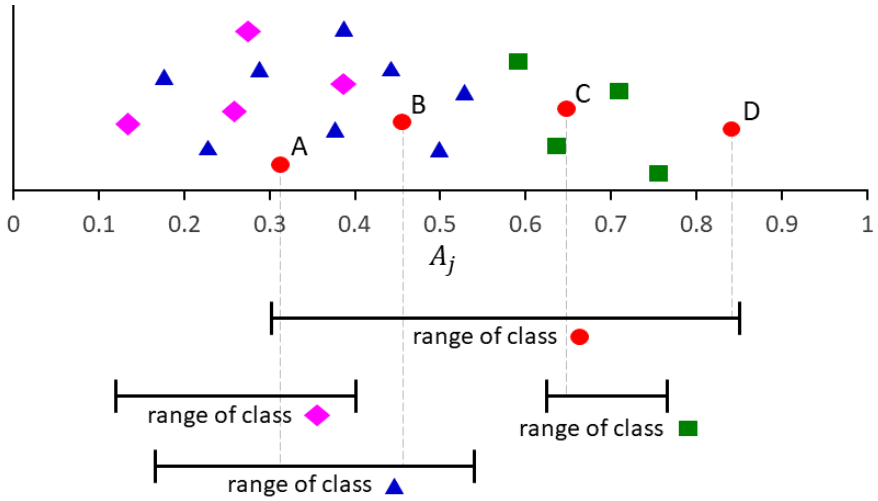


Figure 4.2: The distribution of the sample dataset with respect to numeric attribute A_j for assigning the weighted value to each instance from the red-circle class according to that attribute

the position of overlapping of fewer classes. That is because it represents the region of the class more clearly, not an area that is shared with many other classes. For example, in Figure 4.2, instances A, B, C and D are located in the overlapping ranges of three classes, two classes, two classes, and one class, respectively. So, instance A must have the smallest weight, while instance D must have the largest weight.

- *The effect of the class size*

  The instance that is located at the position of overlapping of the bigger class must have less weight than the instance from the same class that is located at the position of overlapping of the smaller class. That is because it represents the region of the class more clearly, not an area that is shared with other classes having a lot of instances. For example, in Figure 4.2, instances B and C are both located in the overlapping ranges of two classes. Explicitly, they are in the range of the red-circle class which is their class. However, another class that covers the position of instance B is the blue-triangle class having eight instances, while another class that covers the position of instance C is the green-square class having four instances. So, instance B must have a smaller weight than instance C.

2. Second, the total weight with respect to each class must be more balanced.

   After assigning the weighted values to all instances based on their positions, a set of instance weights with respect to the instances from each class is normalized. They are made to have the same total weight equal to $1/p$, where p is the number of all classes, for balancing between the classes.

Then, the entropy computed according to that set of instance weights is employed to build the decision tree for classifying multi-class imbalanced datasets, which is called the self-balancing decision tree (SBDT).

4.1 Proposed Methodologies

In this section, the mathematical definitions corresponding to the proposed methodologies are formally introduced. It also includes related properties, examples, and the pseudocode for each proposed algorithm.

4.1.1 Individually Weighted Entropy

Initially, the individually weighted entropy or IWE is presented as a new splitting measure to support applying the set of instance weights discussed above. It adjusts the components of calculating the Shannon's entropy (SE). Originally, the components of SE comprise the proportion of instances in each class c_k denoted by P_k, as shown in (2.2). It treats all instances equally, in which each one is counted as one. For MCE and ME, the meaning of the components is not changed, it just modifies the set of instances used in consideration. In this section, therefore, an adjusted definition of each class's proportion is presented to be consistent with the set of instance weights. The summation of the weights corresponding to the instances in each class c_k divided by the total weight is used as the components of calculating the entropy, which is denoted by Γ_k.

IWE is defined according to a set of instance weights $\mathbf{W} = \{(w_i, y_i) \mid i = 1, 2, ..., m\}$ by (4.1), in which each instance weight is in $\mathbb{R} \times C$. It is able to deal with dataset $\mathbf{D}$ depending on the assignment of weighted value to each instance.

$$IWE(\mathbf{W}) = -\sum_{k=1}^{p} \Gamma_k(\mathbf{W}) \log_2 \Gamma_k(\mathbf{W}) \tag{4.1}$$

where

- $\Gamma_k(\mathbf{W})$ is the proportion of weights corresponding to class c_k, i.e. $\Gamma_k(\mathbf{W}) = \frac{\sum_{(w_i,y_i)\in\mathbf{W}_k} w_i}{\sum_{(w_i,y_i)\in\mathbf{W}} w_i}$.
- $\mathbf{W}_k$ is the set of instance weights corresponding to class c_k, i.e. $\mathbf{W}_k = \{(w_i, y_i) \in \mathbf{W} \mid y_i = c_k \text{ for } i = 1, 2, ..., m\}$.

The brief pseudocode to compute IWE with respect to a set of instance weights $\mathbf{W}$ is displayed in Algorithm 4.1. For each class, the proportion of weights is computed to be the components of calculating the entropy.

Algorithm 4.1: IWE($\mathbf{W}$)

Input: a set of instance weights $\mathbf{W}$
Output: the individually weighted entropy according to $\mathbf{W}$

```
• let iwe = 0
for each class c_k do
    • compute the proportion of weights corresponding to class c_k: Γ_k(W)
    • iwe += Γ_k(W) log_2 Γ_k(W)
return iwe
```


4.1.2 Class-overlapping Weighting Function

In this section, the first proposed weighting function is presented in accordance with the first assumption mentioned above, which is called the class-overlapping weighting function denoted by α. It is defined with respect to the instance value $s(\vec{x}_i) \in s(\mathbf{D})$ by (4.2) as follows:

$$\alpha(s(\vec{x}_i)) = \left(\sum_{k=1}^{p} \mathbb{1}(s(\vec{x}_i) \in \phi_k^*(s(\mathbf{D}))) \cdot \log_2(m_k) \right)^{-1} \tag{4.2}$$

where $\mathbb{1}(\omega)$ is the indicator function, 1 if the condition ω is true, 0 otherwise.

The weight corresponding to each instance is inversely proportional to the total number of instances in all classes that their ranges without the outliers cover the instance's position, which has been scaled down by the logarithmic function to decrease the extreme difference between each case.

For example, consider Figure 4.2, the calculation of the class-overlapping weight for each instance in the red-circle class is demonstrated as follows:

- Instance A locates in the overlapping range of three classes: the red-circle class, the pink-diamond class, and the blue-triangle class, which contain 4 instances, 4 instances, and 8 instances, respectively. Thus,

$$\alpha(\pi_j(\mathrm{A})) = \frac{1}{\log_2(4) + \log_2(4) + \log_2(8)} = \frac{1}{2+2+3} = \frac{1}{7}.$$

- Instance B locates in the overlapping range of two classes: the red-circle class and the blue-triangle class, which contain 4 instances and 8 instances, respectively. Thus,

$$\alpha(\pi_j(\mathrm{B})) = \frac{1}{\log_2(4) + \log_2(8)} = \frac{1}{2+3} = \frac{1}{5}.$$

- Instance C locates in the overlapping range of two classes: the red-circle class and the green-square class, which contain 4 instances and 4 instances, respectively. Thus,

$$\alpha(\pi_j(\mathrm{C})) = \frac{1}{\log_2(4) + \log_2(4)} = \frac{1}{2+2} = \frac{1}{4}.$$

- Instance D locates only in the range of one class: the red-circle class, which contains 4 instances. Thus,

$$\alpha(\pi_j(\mathrm{D})) = \frac{1}{\log_2(4)} = \frac{1}{2}.$$

That is, instance A receives the smallest weight, while instance D receives the largest weight, corresponding to the first issue considered in the first assumption. Moreover, instance B receives a smaller weight than instance C, corresponding to the second issue considered in the first assumption.

Theoretically, two properties corresponding to those issues of determining the weighted values to the instances in $\mathbf{D}$ with respect to $s(\mathbf{D})$ using the class-overlapping weighting function are presented in Theorem 4.1 and Theorem 4.2 as follows:

Theorem 4.1. For a set of instance values $s(\mathbf{D})$, let $C_a \subseteq C$ and $C_b \subseteq C$ be the sets of classes that their ranges without the outliers cover the position of instance values $s(\vec{x}_a) \in s(\mathbf{D})$ and $s(\vec{x}_b) \in s(\mathbf{D})$, respectively. If $C_a \subset C_b$, then $\alpha(s(\vec{x}_a)) > \alpha(s(\vec{x}_b))$.

Proof. Let $M_a = \{m_k \mid c_k \in C_a \text{ for } k = 1, 2, ..., p\}$ and $M_b = \{m_k \mid c_k \in C_b \text{ for } k = 1, 2, ..., p\}$ be the sets containing the number of instances from each class in C_a and C_b, respectively. Since $C_a \subset C_b$, M_a is a subset of M_b, which M_b can be partitioned as $M_b = M_a \cup M_r$ where $M_r = M_b \setminus M_a$. Hence,

$$\prod_{m \in M_a} m < \prod_{m \in M_a} m + \prod_{m \in M_r} m = \prod_{m \in M_b} m$$

$$\log_2 \left(\prod_{m \in M_a} m \right) < \log_2 \left(\prod_{m \in M_b} m \right)$$

$$\sum_{m \in M_a} \log_2 m < \sum_{m \in M_b} \log_2 m$$

$$\left(\sum_{m \in M_a} \log_2 m \right)^{-1} > \left(\sum_{m \in M_b} \log_2 m \right)^{-1}$$

$$\alpha(s(\vec{x}_a)) > \alpha(s(\vec{x}_b)).$$

□

Theorem 4.2. For a set of instance values $s(\mathbf{D})$, let $C_a \subseteq C$ and $C_b \subseteq C$ be the sets of classes that their ranges without the outliers cover the position of instance values $s(\vec{x}_a) \in s(\mathbf{D})$ and $s(\vec{x}_b) \in s(\mathbf{D})$, respectively. If $C_a \setminus C_b = \{c_{k_a}\}$ and $C_b \setminus C_a = \{c_{k_b}\}$ s.t. $m_{k_a} < m_{k_b}$, then $\alpha(s(\vec{x}_a)) > \alpha(s(\vec{x}_b))$.

Proof. Let $M_a = \{m_k \mid c_k \in C_a \text{ for } k = 1, 2, ..., p\}$ and $M_b = \{m_k \mid c_k \in C_b \text{ for } k = 1, 2, ..., p\}$ be the sets containing the number of instances from each class in C_a and C_b, respectively. Since $C_a \setminus C_b = \{c_{k_a}\}$, so $M_a \setminus M_b = \{m_{k_a}\}$, then M_a can be transformed

as $M_a = (M_a \cap M_b) \cup \{m_{k_a}\}$. Likewise, $M_b = (M_a \cap M_b) \cup \{m_{k_b}\}$. Hence,

$$
\begin{aligned}
m_{k_a} &< m_{k_b} \\
m_{k_a} \times \left(\prod_{m \in M_a \cap M_b} m \right) &< m_{k_b} \times \left(\prod_{m \in M_a \cap M_b} m \right) \\
\prod_{m \in (M_a \cap M_b) \cup \{m_{k_a}\}} m &< \prod_{m \in (M_a \cap M_b) \cup \{m_{k_b}\}} m \\
\prod_{m \in M_a} m &< \prod_{m \in M_b} m \\
\log_2 \left(\prod_{m \in M_a} m \right) &< \log_2 \left(\prod_{m \in M_b} m \right) \\
\sum_{m \in M_a} \log_2 m &< \sum_{m \in M_b} \log_2 m \\
\left(\sum_{m \in M_a} \log_2 m \right)^{-1} &> \left(\sum_{m \in M_b} \log_2 m \right)^{-1} \\
\alpha(s(\vec{x}_a)) &> \alpha(s(\vec{x}_b)).
\end{aligned}
$$

□

Then, a set of class-overlapping weights received from a set of instance values $s(\mathbf{D})$ is determined by (4.3) as follows:

$$\alpha(s(\mathbf{D})) = \{(\alpha(s(\vec{x}_i)), y_i) \mid (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, 2, ..., m\} \tag{4.3}$$

In addition, the pseudocode to obtain a set of class-overlapping weights with respect to a set of instance values $s(\mathbf{D})$ is displayed in Algorithm 4.2. The range of instance values without the outliers corresponding to each class is computed for considering the instances that their values locate there. Then, the weighted values of those instances are updated, and then returning as their inverses.

Algorithm 4.2: OverlappingWeight($\mathbf{D}, s$)

Input: dataset $\mathbf{D}$, a function to obtain the instance values s
Output: a set of class-overlapping weights with respect to $s(\mathbf{D})$

```
• let w_i = 0 for i = 1, 2, ..., m
for each class c_k do
    • generate the range of instance values without the outliers
    corresponding to class c_k: φ*_k(s(D))
    for each value s(x_i) do
        if min(φ*_k(s(D))) ≤ s(x_i) ≤ max(φ*_k(s(D))) then
            • w_i += log_2(m_k)
return {(1/w_i, y_i) | (x_i, y_i) ∈ D for i = 1, ..., m}
```

4.1.3 Class-balancing Weighting Function

In this section, the second proposed weighting function is presented in accordance with the second assumption mentioned above, which is called the class-balancing weighting function denoted by μ. It is defined with respect to the instance weight $(w_i, y_i) \in \mathbf{W}$ by (4.4) as follows:

$$\mu(w_i, y_i) = \frac{1}{p} \cdot \frac{w_i}{\sum\limits_{w_l \in W_{y_i}} w_l} \tag{4.4}$$

The weight corresponding to each instance is normalized by its class. The total weight with respect to each class is balanced, which is set to $1/p$ to make the total weight of a whole dataset having p distinct classes equals to 1. It is discussed in Theorem 4.3, which corresponds to the second assumption.

Theorem 4.3. For a set of instance weights $\mathbf{W}$, the summation of class-balancing weights of $\mathbf{W}$ with respect to each class is equal to $1/p$, where p is the number of all classes.

Proof. Let $\mathbf{W}_k \subset \mathbf{W}$ be the set of instance weights corresponding to each class c_k of size m_k. So, class-balancing weight for each $(w_i, y_i) \in W_k$ is equal to $\frac{1}{p} \cdot \frac{w_i}{\sum\limits_{w_l \in \mathbf{W}_k} w_l}$. Hence,

the summation of class-balancing weights with respect to class c_k is equal to

$$\sum_{w_i \in \mathbf{W}_k} \left(\frac{1}{p} \cdot \frac{w_i}{\sum\limits_{w_l \in \mathbf{W}_k} w_l} \right) = \left(\frac{1}{p} \cdot \frac{1}{\sum\limits_{w_l \in \mathbf{W}_k} w_l} \right) \cdot \sum_{w_i \in \mathbf{W}_k} w_i = \frac{1}{p} \cdot \frac{\sum\limits_{w_i \in \mathbf{W}_k} w_i}{\sum\limits_{w_l \in \mathbf{W}_k} w_l} = \frac{1}{p}$$

□

For example, class-overlapping weights of the instances from the red-circle class in Figure 4.2 obtained in the previous section will be normalized by their total weight, i.e. $\frac{1}{7} + \frac{1}{5} + \frac{1}{4} + \frac{1}{2} = 1.09$. Consequently, by using the class-balancing weighting function, the weighted values that are normalized of instances A, B, C, and D are determined as $\frac{1}{4} \cdot \frac{1}{7 \times 1.09} = 0.0325$, $\frac{1}{4} \cdot \frac{1}{5 \times 1.09} = 0.045$, $\frac{1}{4} \cdot \frac{1}{4 \times 1.09} = 0.0575$, and $\frac{1}{4} \cdot \frac{1}{2 \times 1.09} = 0.115$, respectively. So, their total weight is equal to $0.0325 + 0.045 + 0.0575 + 0.115 = 0.25 = \frac{1}{4}$, corresponding to the second assumption.

Then, a set of class-balancing weights received from a set of instance weights $\mathbf{W}$ is determined by (4.5) as follows:

$$\mu(\mathbf{W}) = \{(\mu(w_i, y_i), y_i) \mid (w_i, y_i) \in \mathbf{W} \text{ for } i = 1, 2, ..., m\} \tag{4.5}$$

In addition, the pseudocode to obtain a set of class-balancing weights with respect to a set of instance weights $\mathbf{W}$ is displayed in Algorithm 4.3. The total weight corresponding to each class is computed for normalizing the instances in that class.

Algorithm 4.3: BalancingWeight($\mathbf{W}$)

Input: a set of instance values $\mathbf{W}$
Output: a set of class-balancing weights with respect to $\mathbf{W}$

```
for each class c_k do
    • generate the set of instance weights corresponding to class c_k: W_k
    • let Σ_k be the summation of the weighted values in W_k
return {(w_i/Σ_i, y_i) | (w_i, y_i) ∈ W for i = 1, ..., m}
```

4.1.4 Self-balancing Decision Tree

Then, in this section, a decision tree based on a single attribute is constructed from the set of instance weights received from applying the composition of functions α and μ to dataset $\mathbf{D}$. It is called a set of class-overlapping-balancing weights, which is defined according to each attribute A_j by (4.6) as follows:

$$OBW_{\pi_j}(\mathbf{D}) = \mu(\alpha(\pi_j(\mathbf{D}))) \tag{4.6}$$

Thus, the individually weighted entropy corresponding to a set of class-overlapping-balancing weights is also called the class-overlapping-balancing entropy (OBE) defined by (4.7) as follows:

$$OBE_{\pi_j}(\mathbf{D}) = IWE\left(OBW_{\pi_j}(\mathbf{D})\right) \tag{4.7}$$

Since $OBW_{\pi_j}(\mathbf{D})$ contains a balanced proportion of instance weights corresponding to each class, $OBE_{\pi_j}(\mathbf{D})$ always equals to one. In a recursive partitioning algorithm, however, the class-overlapping-balancing entropy is applied to each subset of instance weights for each partition after splitting shown in (4.8) and (4.9). In which the proportion of each class varies according to the splitting condition.

$$OBE_{\pi_j}(\mathbf{D}^{(\text{left})}) = IWE\left(OBW^{(\text{left})}_{\pi_j}(\mathbf{D})\right) \tag{4.8}$$

$$OBE_{\pi_j}(\mathbf{D}^{(\text{right})}) = IWE\left(OBW^{(\text{right})}_{\pi_j}(\mathbf{D})\right) \tag{4.9}$$

So, a decision tree from the recursive partitioning algorithm using OBE as the splitting measure is called the self-balancing decision tree or SBDT, and that algorithm is also called the SBDT algorithm. Its pseudocode is displayed in Algorithm 4.4. For each

attribute A_j, it starts by obtaining a set of class-overlapping weights. Then, the greedy approach is applied to select the best condition for splitting dataset $\mathbf{D}$ based on the value of OBE according to each candidate from $\pi_j(\mathbf{D})$.

Algorithm 4.4: SBDT($\mathbf{D}$)

Input: dataset $\mathbf{D}$
Output: a decision tree

- create the root node of the tree
- **if** all instances in $\mathbf{D}$ are in the same class **then**
  - **return** the leaf node with respect to that class
- **else**
  - `/* select the splitting condition */`
  - **for** each attribute A_j **do**
    - receive a set of class-overlapping weights with respect to $\pi_j(\mathbf{D})$ via **Algorithm 4.2**: $\alpha(\pi_j(\mathbf{D}))$ = OverlappingWeight($\mathbf{D}$, π_j)
    - receive a set of class-overlapping-balancing weights with respect to $\pi_j(\mathbf{D})$ via **Algorithm 4.3**: $OBW_{\pi_j}(\mathbf{D})$ = BalancingWeight($\alpha(\pi_j(\mathbf{D}))$)
    - apply the greedy approach on $\pi_j(\mathbf{D})$ based on the individually weighted entropy with respect to $OBW_{\pi_j}(\mathbf{D})$ via **Algorithm 4.1**
    - update the best splitting condition
  - obtain the best splitting condition
  - `/* recursively partition the dataset */`
  - separate the dataset into 2 partitions corresponding to the outcomes of the splitting condition: $\mathbf{D} = \mathbf{D}^{(\text{left})} \cup \mathbf{D}^{(\text{right})}$
  - iterate for each partition: SBDT($\mathbf{D}^{(\text{left})}$) and SBDT($\mathbf{D}^{(\text{right})}$)

4.2 Experiments and Results

In order to evaluate the performance of inducing a decision tree that uses OBE as the splitting measure, two collections of experiments are conducted. The first collection employs the synthetic datasets varying their imbalanced ratios, to show the effectiveness of OBE over SE. For the second collection, the performance comparison in classifying the real-world datasets from the UCI repository of SBDT with other methods is presented.

4.2.1 Experiments on Synthetic Datasets

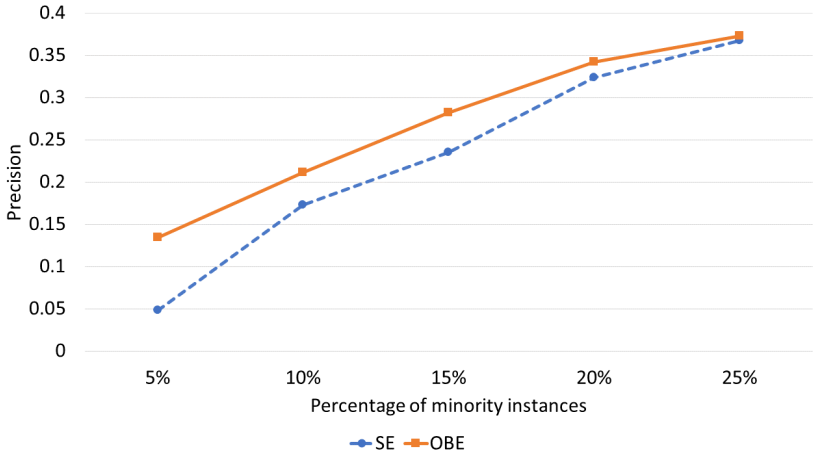
In this section, an improvement of the Shannon's entropy (SE) to classify minority

instances in the multi-class imbalanced datasets dealing with numeric attributes using the overlapping-balancing entropy (OBE) is demonstrated in the experiments on the collections of synthetic datasets generated according to the following specifications:

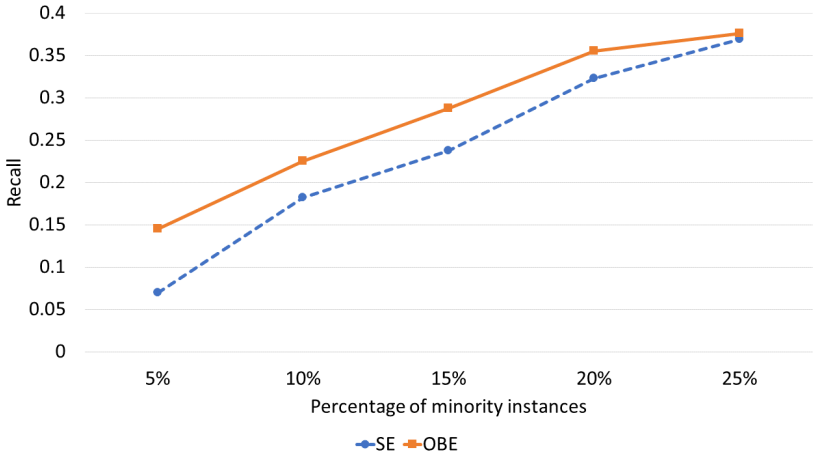
- Each dataset contains 1000 instances consisting of 10 numeric attributes.
- Each instance is labeled as one of four determined classes as follows:
  - The minority class (containing the instances not exceed 25%)
  - The majority class (containing the instances not less than 25%)
  - The third class (containing exactly 25% of all instances)
  - The fourth class (containing exactly 25% of all instances)
- For each attribute, an overlapping range of each class is randomly defined. Then, the uniform sampling is performed within that range.
- There are five groups of experiments varying the percentages of minority instances from 5% to 25%.
- For each group, ten datasets are synthesized and then applying five-fold cross-validation ten times to evaluate the performance of each method.

Accordingly, the average results of SE and OBE are compared via the precision (2.16), the recall (2.17), and the F-measure (2.18) with respect to the minority class and the majority class displaying in Figure 4.3 and Figure 4.4, respectively.

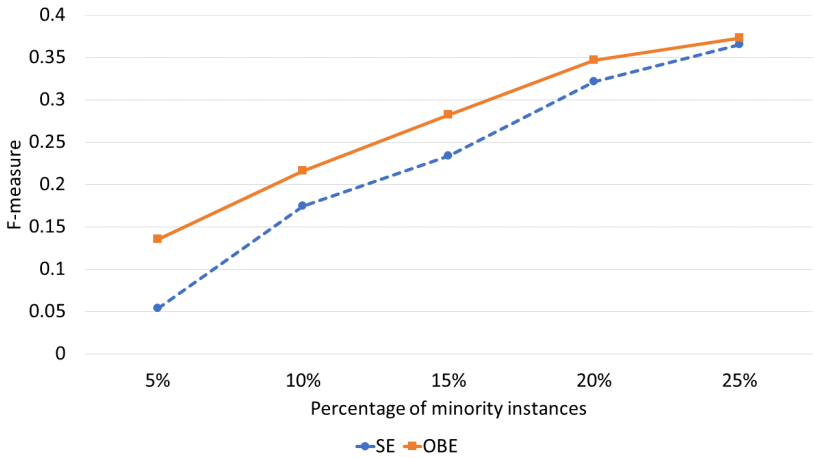
For the results corresponding to the minority class, their results are similar in all performance measures, which ostensibly increase when the percentage of minority instances increases. Nonetheless, OBE significantly outperforms SE when the number of instances in the minority class is tiny. For example, the performance of SE is only about 0.05 when there are 5 % of minority instances, while OBE offers the performance up to 0.15. Then their values will approach to be equal when a dataset is more balanced. For the case that the dataset is completely balanced, the performance of SE and OBE are



(a) comparing by the precision

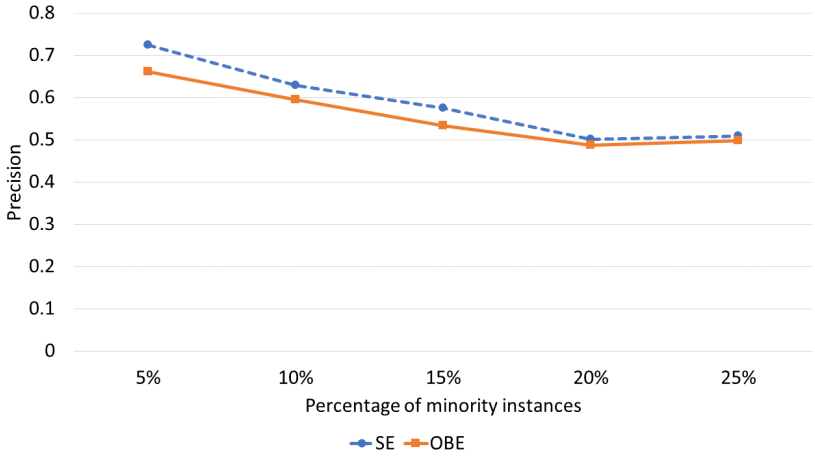


(b) comparing by the recall

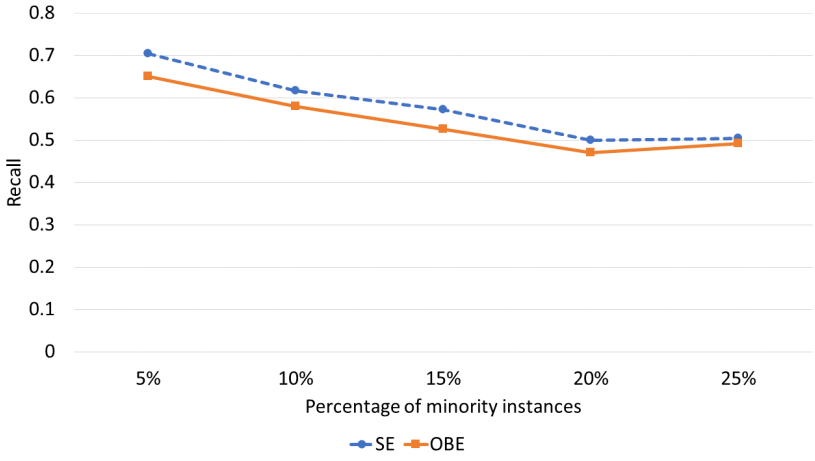


(c) comparing by the F-measure

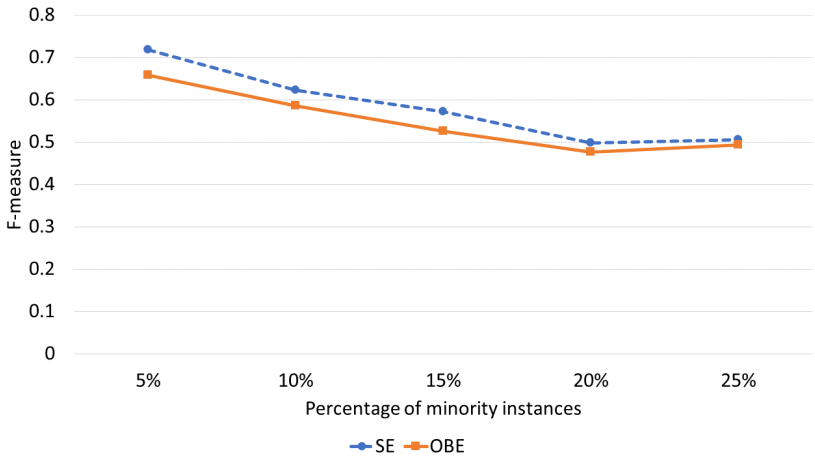
Figure 4.3: The experimental results with respect to the minority class on the synthetic multi-class datasets varying the percentage of minority instances



(a) comparing by the precision



(b) comparing by the recall



(c) comparing by the F-measure

Figure 4.4: The experimental results with respect to the majority class on the synthetic multi-class datasets varying the percentage of minority instances

almost the same at 0.35 due to the overlapping of each class range in each attribute. These results evidently confirm that SE is not suitable to deal with the multi-class imbalanced problem, which is able to be captured using OBE.

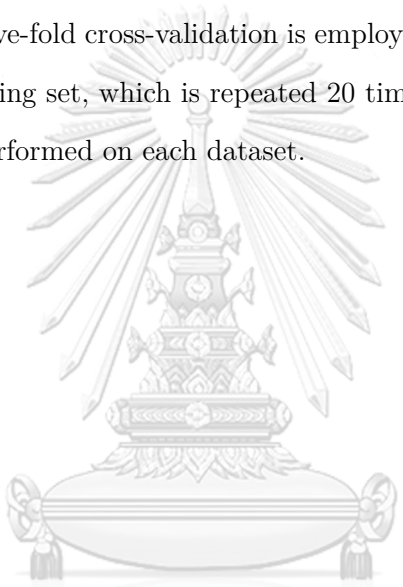
Differently, considering the results based on the majority class, all performance measures gradually decrease when a dataset is more balanced. That is because of the decrease in the percentage of majority instances. However, due to the effort to reduce the bias toward the majority class of OBE, it gives a slightly worse result according to the majority class compared to SE in the case of extreme imbalance. For example, the performance of SE is about 0.7 when there are only 5 % of minority instances, while OBE offers the performance around 0.65. Their values will approach to the same value when a dataset is more balanced, which are equal to 0.5 when the dataset is completely balanced.

4.2.2 Experiments on Real-world Datasets

To demonstrate the effectiveness of OBE on general datasets, the decision tree built based on OBE like SBDT is evaluated with the experiments on real-world datasets in this section. Its results are compared to those of four other decision trees. The first two models are the decision trees from the well-known algorithms, i.e. CART [44] and C. 45 [39]. Additionally, the decision tree built based on DCSM [45] is used as well. The other is the state-of-the-art decision tree designed for a multi-class imbalanced dataset like MC-HDDT [81].

Datasets

For the datasets used in the experiments, they are collected from the UCI repository [86] having a total of 23 datasets that their attributes are all numeric, which are summarized in Table 4.1. They are sorted in the ascending order by the imbalanced ratio (I.R.), corresponding to the percentage of instances in each class (% Cl.). The first two columns represent the index and the name of each dataset. For the next three columns, they indicate the number of instances (# Inst.), the number of attributes (# Att.), and the number of classes (# Cl.), respectively. In order to evaluate the performance of each decision tree, the five-fold cross-validation is employed to divide the dataset into the training set and the testing set, which is repeated 20 times. That is, there are up to one hundred experiments performed on each dataset.



No.	Dataset	# Inst.	# Att.	# Cl.	% Cl.	I.R.
1	LibrasMovement	360	90	15	6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67 / 6.67	1
2	StatlogImage	2310	19	7	14.29 / 14.29 / 14.29 / 14.29 / 14.29 / 14.29 / 14.29	1
3	Vowel	990	10	11	9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09 / 9.09	1
4	OpticDigits	5620	64	10	10.18 / 10.16 / 10.11 / 10.07 / 10.0 / 9.93 / 9.93 / 9.91 / 9.86 / 9.86	1.03
5	PenDigits	10992	16	10	10.41 / 10.41 / 10.4 / 10.4 / 10.39 / 9.61 / 9.6 / 9.6 / 9.6 / 9.6	1.08
6	StatlogVehicle	846	18	4	25.77 / 25.65 / 25.06 / 23.52	1.1
7	Letter	20000	16	26	4.07 / 4.03 / 4.01 / 3.98 / 3.96 / 3.94 / 3.94 / 3.93 / 3.92 / 3.92 / 3.88 / 3.86 / 3.84 / 3.83 / 3.82 / 3.8 / 3.79 / 3.78 / 3.76 / 3.76 / 3.74 / 3.74 / 3.7 / 3.68 / 3.67 / 3.67	1.11
8	Wine	178	13	3	39.89 / 33.15 / 26.97	1.48
9	BreastTissue	106	9	6	20.75 / 19.81 / 16.98 / 15.09 / 14.15 / 13.21	1.57
10	Ionosphere	351	34	2	64.1 / 35.9	1.79
11	Pima	768	8	2	65.1 / 34.9	1.87
12	StatlogLandsat	6435	36	6	23.82 / 23.43 / 21.1 / 10.99 / 10.92 / 9.73	2.45
13	Haberman	306	3	2	73.53 / 26.47	2.78
14	Parkinsons	195	22	2	75.38 / 24.62	3.06
15	NewThyroid	215	5	3	69.77 / 16.28 / 13.95	5
16	Fertility	100	9	2	88.0 / 12.0	7.33
17	Glass	214	9	6	35.51 / 32.71 / 13.55 / 7.94 / 6.07 / 4.21	8.44
18	Ecoli	332	7	6	43.07 / 23.19 / 15.66 / 10.54 / 6.02 / 1.51	28.6
19	Winequality-red	1599	11	6	42.59 / 39.9 / 12.45 / 3.31 / 1.13 / 0.63	68.1
20	Yeast	1484	8	10	31.2 / 28.91 / 16.44 / 10.98 / 3.44 / 2.96 / 2.36 / 2.02 / 1.35 / 0.34	92.6
21	PageBlocks	5473	10	5	89.77 / 6.01 / 2.1 / 1.61 / 0.51	175.46
22	Winequality-white	4898	11	7	44.88 / 29.75 / 17.97 / 3.57 / 3.33 / 0.41 / 0.1	439.6
23	StatlogShuttle	58000	9	7	78.6 / 15.35 / 5.63 / 0.29 / 0.09 / 0.02 / 0.02	4558.6

Table 4.1: The characteristics of the experimental real-world multi-class datasets

Results

Accordingly, the average results of each decision tree are compared via the macro-precision (2.22), the macro-recall (2.23), the macro-F-measure (2.24), and the accuracy (2.25) displaying in four tables for each measure. In each table, a comparison of performance for each dataset is represented in each row. Furthermore, the rank of each method corresponding to each dataset is shown in parentheses, emphasizing the best rank in bold. Summarily, the bottom row of each table represents the average rank of each method.

In Table 4.2, the performance comparison according to the macro-precision of each method is shown. SBDT yields the lowest average rank at 2, which is much different from other methods. There are up to 15 out of 23 datasets that SBDT offers the highest or the second-highest macro-precision. For MC-HDDT, it yields the third-lowest average rank at 3, which is worse than DCSM having the average rank at 2.83.

In Table 4.3, the performance comparison according to the macro-recall of each method is shown. SBDT yields the lowest average rank at 1.65, which is much different from other methods. There are up to 12 out of 23 datasets that SBDT offers the highest macro-recall, while 7 out of 11 remaining datasets have the second-best performance. For the others, they receive a similar average rank that is all equal to or greater than 3.

In Table 4.4, the performance comparison according to the macro-F-measure of each method is shown. SBDT yields the lowest average rank at 1.74, which is much different from other methods. There are up to 12 out of 23 datasets that SBDT offers the highest macro-F-measure, while 5 out of 11 remaining datasets have the second-best performance. For MC-HDDT, it yields the third-lowest average rank at 3.09, which is worse than DCSM having the average rank at 2.87.

In Table 4.5, the performance comparison according to the accuracy of each method is shown. SBDT yields the lowest average rank at 2.04, which is much different from other methods. There are up to 15 out of 23 datasets that SBDT offers the highest or the second-highest accuracy. For MC-HDDT, it yields the second-lowest average rank at 2.91, which is close to DCSM and CART having the average rank at 3.04 and 3.13, respectively.

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**SBDT**
1	LibrasMovement	**0.7266 (1)**	0.6562 (5)	0.6825 (3)	0.6713 (4)	0.7058 (2)
2	StatlogImage	0.9628 (4)	0.9620 (5)	0.9655 (2)	**0.9674 (1)**	0.9653 (3)
3	Vowel	0.8047 (2)	0.7687 (5)	0.8022 (3)	0.7987 (4)	**0.8102 (1)**
4	OpticDigits	0.8982 (3)	0.8868 (5)	0.9016 (2)	0.8881 (4)	**0.9065 (1)**
5	PenDigits	0.9611 (4)	0.9617 (3)	0.9623 (2)	0.9608 (5)	**0.9655 (1)**
6	StatlogVehicle	0.6972 (5)	0.7052 (4)	0.7242 (3)	**0.7331 (1)**	0.7281 (2)
7	Letter	0.8699 (2)	0.8602 (4)	0.8651 (3)	0.8408 (5)	**0.8742 (1)**
8	Wine	0.9107 (5)	0.9317 (3)	0.9250 (4)	**0.9484 (1)**	0.9379 (2)
9	BreastTissue	0.6589 (4)	**0.6856 (1)**	0.6674 (3)	0.6551 (5)	0.6779 (2)
10	Ionosphere	0.8846 (5)	**0.9374 (1)**	0.8882 (4)	0.8930 (2)	0.8912 (3)
11	Pima	0.6571 (5)	0.6710 (3)	0.6609 (4)	0.6763 (2)	**0.6783 (1)**
12	StatlogLandsat	0.8305 (2)	0.8093 (5)	**0.8340 (1)**	0.8302 (3)	0.8297 (4)
13	Haberman	0.5550 (4)	0.5650 (2)	**0.5675 (1)**	0.5550 (5)	0.5609 (3)
14	Parkinsons	0.8299 (2)	0.7658 (5)	0.7983 (4)	0.8186 (3)	**0.8361 (1)**
15	NewThyroid	0.9191 (4)	0.9158 (5)	0.9327 (2)	0.9277 (3)	**0.9455 (1)**
16	Fertility	0.5161 (4)	0.5197 (3)	**0.5593 (1)**	0.5159 (5)	0.5572 (2)
17	Glass	0.6655 (4)	0.6432 (5)	0.6892 (3)	0.7194 (2)	**0.7197 (1)**
18	Ecoli	0.7421 (5)	**0.7894 (1)**	0.7546 (4)	0.7891 (2)	0.7840 (3)
19	Winequality-red	**0.3749 (1)**	0.3414 (5)	0.3547 (3)	0.3539 (4)	0.3672 (2)
20	Yeast	0.4053 (4)	0.4506 (2)	0.3878 (5)	**0.4654 (1)**	0.4336 (3)
21	PageBlocks	0.8047 (4)	0.7908 (5)	**0.8304 (1)**	0.8295 (2)	0.8246 (3)
22	Winequality-white	0.3649 (5)	0.3678 (4)	0.3750 (2)	0.3710 (3)	**0.3848 (1)**
23	StatlogShuttle	**0.9896 (1)**	0.9714 (4)	0.9692 (5)	0.9826 (2)	0.9723 (3)
	average rank	3.48	3.7	2.83	3	2

Table 4.2: The experimental results on the real-world multi-class datasets comparing by the macro-precision

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**SBDT**
1	LibrasMovement	**0.6990 (1)**	0.6407 (4)	0.6532 (3)	0.6375 (5)	0.6877 (2)
2	StatlogImage	0.9619 (4)	0.9616 (5)	0.9649 (2)	**0.9666 (1)**	0.9647 (3)
3	Vowel	0.7975 (2)	0.7619 (5)	0.7932 (3)	0.7896 (4)	**0.8013 (1)**
4	OpticDigits	0.8973 (3)	0.8861 (5)	0.9010 (2)	0.8874 (4)	**0.9055 (1)**
5	PenDigits	0.9607 (4)	0.9615 (3)	0.9622 (2)	0.9604 (5)	**0.9654 (1)**
6	StatlogVehicle	0.6999 (5)	0.7002 (4)	0.7250 (3)	**0.7340 (1)**	0.7280 (2)
7	Letter	0.8689 (2)	0.8592 (4)	0.8639 (3)	0.8395 (5)	**0.8729 (1)**
8	Wine	0.9077 (5)	0.9305 (3)	0.9216 (4)	**0.9466 (1)**	0.9348 (2)
9	BreastTissue	0.6276 (5)	0.6606 (2)	0.6550 (3)	0.6346 (4)	**0.6696 (1)**
10	Ionosphere	0.8820 (2)	**0.9374 (1)**	0.8768 (5)	0.8804 (4)	0.8810 (3)
11	Pima	0.6574 (5)	0.6704 (3)	0.6579 (4)	0.6724 (2)	**0.6751 (1)**
12	StatlogLandsat	0.8299 (3)	0.8108 (5)	**0.8306 (1)**	0.8289 (4)	0.8300 (2)
13	Haberman	0.5530 (4)	0.5586 (3)	**0.5634 (1)**	0.5508 (5)	0.5607 (2)
14	Parkinsons	**0.8248 (1)**	0.7688 (5)	0.7886 (4)	0.8175 (3)	0.8241 (2)
15	NewThyroid	0.9008 (5)	0.9158 (2)	0.9090 (3)	0.9024 (4)	**0.9194 (1)**
16	Fertility	0.5431 (4)	0.5502 (3)	0.5903 (2)	0.5404 (5)	**0.6069 (1)**
17	Glass	0.6579 (5)	0.6671 (3)	0.6581 (4)	0.6890 (2)	**0.6936 (1)**
18	Ecoli	0.7236 (5)	**0.7830 (1)**	0.7319 (4)	0.7803 (2)	0.7665 (3)
19	Winequality-red	**0.3660 (1)**	0.3417 (5)	0.3550 (3)	0.3454 (4)	0.3605 (2)
20	Yeast	0.3948 (4)	0.4516 (2)	0.3882 (5)	**0.4780 (1)**	0.4368 (3)
21	PageBlocks	0.8095 (4)	0.8008 (5)	0.8186 (3)	0.8233 (2)	**0.8256 (1)**
22	Winequality-white	0.3675 (5)	0.3750 (2)	0.3720 (3)	0.3704 (4)	**0.3943 (1)**
23	StatlogShuttle	0.9652 (4)	0.9672 (3)	0.9787 (2)	0.9652 (5)	**0.9913 (1)**
	average rank	3.61	3.39	3	3.35	1.65

Table 4.3: The experimental results on the real-world multi-class datasets comparing by the macro-recall

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**SBDT**
1	LibrasMovement	**0.6940 (1)**	0.6303 (5)	0.6458 (3)	0.6322 (4)	0.6784 (2)
2	StatlogImage	0.9619 (4)	0.9615 (5)	0.9649 (2)	**0.9666 (1)**	0.9647 (3)
3	Vowel	0.7967 (2)	0.7596 (5)	0.7930 (3)	0.7891 (4)	**0.8000 (1)**
4	OpticDigits	0.8973 (3)	0.8860 (5)	0.9009 (2)	0.8873 (4)	**0.9056 (1)**
5	PenDigits	0.9608 (4)	0.9615 (3)	0.9621 (2)	0.9605 (5)	**0.9653 (1)**
6	StatlogVehicle	0.6965 (5)	0.7011 (4)	0.7231 (3)	**0.7316 (1)**	0.7268 (2)
7	Letter	0.8690 (2)	0.8593 (4)	0.8640 (3)	0.8396 (5)	**0.8731 (1)**
8	Wine	0.9058 (5)	0.9267 (3)	0.9197 (4)	**0.9448 (1)**	0.9319 (2)
9	BreastTissue	0.6181 (5)	0.6487 (2)	0.6406 (3)	0.6210 (4)	**0.6581 (1)**
10	Ionosphere	0.8825 (4)	**0.9366 (1)**	0.8809 (5)	0.8846 (2)	0.8842 (3)
11	Pima	0.6561 (5)	0.6691 (3)	0.6585 (4)	0.6737 (2)	**0.6758 (1)**
12	StatlogLandsat	0.8299 (2)	0.8097 (5)	**0.8320 (1)**	0.8292 (4)	0.8295 (3)
13	Haberman	0.5527 (4)	0.5586 (3)	**0.5630 (1)**	0.5503 (5)	0.5588 (2)
14	Parkinsons	0.8238 (2)	0.7630 (5)	0.7884 (4)	0.8127 (3)	**0.8243 (1)**
15	NewThyroid	0.9033 (5)	0.9103 (3)	0.9149 (2)	0.9086 (4)	**0.9266 (1)**
16	Fertility	0.5198 (5)	0.5241 (3)	0.5628 (2)	0.5224 (4)	**0.5668 (1)**
17	Glass	0.6430 (4)	0.6407 (5)	0.6492 (3)	0.6782 (2)	**0.6866 (1)**
18	Ecoli	0.7207 (5)	**0.7687 (1)**	0.7245 (4)	0.7663 (2)	0.7623 (3)
19	Winequality-red	**0.3639 (1)**	0.3376 (5)	0.3516 (3)	0.3437 (4)	0.3592 (2)
20	Yeast	0.3945 (4)	0.4425 (2)	0.3843 (5)	**0.4644 (1)**	0.4287 (3)
21	PageBlocks	0.8029 (4)	0.7911 (5)	0.8198 (2)	**0.8202 (1)**	0.8189 (3)
22	Winequality-white	0.3652 (5)	0.3699 (3)	0.3724 (2)	0.3695 (4)	**0.3868 (1)**
23	StatlogShuttle	0.9721 (2)	0.9633 (5)	0.9691 (3)	0.9675 (4)	**0.9800 (1)**
	average rank	3.61	3.7	2.87	3.09	1.74

Table 4.4: The experimental results on the real-world multi-class datasets comparing by the macro-F-measure

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**SBDT**
1	LibrasMovement	**0.6990 (1)**	0.6407 (4)	0.6532 (3)	0.6375 (5)	0.6877 (2)
2	StatlogImage	0.9619 (4)	0.9616 (5)	0.9649 (2)	**0.9666 (1)**	0.9647 (3)
3	Vowel	0.7975 (2)	0.7619 (5)	0.7932 (3)	0.7896 (4)	**0.8013 (1)**
4	OpticDigits	0.8972 (3)	0.8860 (5)	0.9008 (2)	0.8874 (4)	**0.9054 (1)**
5	PenDigits	0.9607 (4)	0.9614 (3)	0.9622 (2)	0.9605 (5)	**0.9653 (1)**
6	StatlogVehicle	0.6978 (5)	0.6985 (4)	0.7228 (3)	**0.7319 (1)**	0.7261 (2)
7	Letter	0.8692 (2)	0.8595 (4)	0.8642 (3)	0.8398 (5)	**0.8732 (1)**
8	Wine	0.9047 (5)	0.9270 (3)	0.9186 (4)	**0.9439 (1)**	0.9312 (2)
9	BreastTissue	0.6441 (5)	0.6727 (2)	0.6676 (3)	0.6473 (4)	**0.6790 (1)**
10	Ionosphere	0.8924 (4)	**0.9416 (1)**	0.8917 (5)	0.8953 (2)	0.8945 (3)
11	Pima	0.6868 (5)	0.6992 (3)	0.6917 (4)	0.7067 (2)	**0.7080 (1)**
12	StatlogLandsat	0.8539 (2)	0.8361 (5)	**0.8566 (1)**	0.8532 (3)	0.8529 (4)
13	Haberman	0.6495 (5)	0.6586 (2)	**0.6667 (1)**	0.6510 (4)	0.6536 (3)
14	Parkinsons	**0.8717 (1)**	0.8229 (5)	0.8454 (4)	0.8602 (3)	0.8709 (2)
15	NewThyroid	0.9279 (5)	0.9314 (4)	0.9384 (2)	0.9337 (3)	**0.9477 (1)**
16	Fertility	0.7837 (5)	0.7962 (4)	0.8036 (2)	0.7986 (3)	**0.8059 (1)**
17	Glass	**0.7014 (1)**	0.6743 (5)	0.6745 (4)	0.6807 (3)	0.6923 (2)
18	Ecoli	0.8132 (2)	**0.8147 (1)**	0.7987 (5)	0.8052 (3)	0.8049 (4)
19	Winequality-red	**0.6154 (1)**	0.5936 (5)	0.6116 (2)	0.6083 (4)	0.6104 (3)
20	Yeast	**0.5115 (1)**	0.4921 (4)	0.4866 (5)	0.4945 (2)	0.4934 (3)
21	PageBlocks	0.9612 (4)	0.9599 (5)	0.9637 (2)	**0.9644 (1)**	0.9634 (3)
22	Winequality-white	0.6020 (2)	0.6001 (5)	0.6007 (4)	0.6016 (3)	**0.6061 (1)**
23	StatlogShuttle	0.9998 (3)	0.9997 (5)	0.9998 (4)	**0.9999 (1)**	0.9999 (2)
	average rank	3.13	3.87	3.04	2.91	2.04

Table 4.5: The experimental results on the real-world multi-class datasets comparing by the accuracy

Discussions

Graphically, the bar chart representing the comparison of results by the average rank corresponding to each measure is shown in Figure 4.5, in which a lower value indicates a better rank. Moreover, the results of the Wilcoxon signed-rank test that compares the performance of SBDT with other decision trees are shown in Table 4.6. The testing results including the statistical value and the p-value for each comparison is shown in each row, in which the symbol checkmark denotes that the proposed method is significantly better than another method with the $(1 - \alpha)100\%$ confidence level. In addition, the testing results are also indicated in Figure 4.5 using the specific symbols for each confidence level.

Similarly to the binary-class case, the conventional decision trees like CART and C4.5 are also not suitable to deal with the multi-class imbalanced problem. They show the worst performances in both terms of the macro-precision and the macro-recall, which make an unacceptable value of the macro-F-measure as well. These happen because there are a lot of incorrect classified instances in each class. Especially, the same number of misclassified instances related to the smaller class will have more effect than the larger class, due to their ratio of classified instances. So, the decision tree algorithms that are designed without considering the class imbalanced problem will give the poor results in terms of those performance measures. Nonetheless, DCSM gave the good performance, although it was not proposed to deal with the class imbalanced problem directly. This happens because of the mechanism of DCSM that considers the number of distinct classes in each partition after splitting. It still gives importance to all classes regardless of how small they are. Therefore, its performance in classifying a class imbalanced dataset is impressive, which is better than the decision tree presented for handling the multi-class imbalanced problem like MC-HDDT. Although it obviously outperforms CART and C4.5, MC-HDDT provides disappointing results, especially in terms of macro-recall. Applying the one-versus-all (OVA) approach in the process of selecting the splitting condition at each internal node causes excessive focus on any class. So, there is the overfitting phenomenon in MC-HDDT, which does not happen to SBDT. Assigning the class-balancing

weight to the instances in each class causes all classes to be considered equally, not focusing too much on any specific class. It is able to avoid the overfitting problem. Thus, SBDT gives a significantly higher value of the macro-recall than all other methods with a confidence level up to 99%. Moreover, applying the class-overlapping weight causes a larger weighted value for an instance located in the position that represents its class more clearly. Then, each class's region is defined not to overlap in the other class regions. Thus, SBDT gives a significantly higher value of the macro-precision than MC-HDDT with a confidence level at 95%, and up to 99% for the others. Consequently, SBDT exactly shows the significant improvement of the macro-F-measure comparing to all methods with up to 99% confidence level. In terms of accuracy, finally, SBDT also gives the best result superior to other methods, which is significantly better than C4.5, DCSM, and MC-HDDT with up to 99% confidence level, and 95% for CART. That is, despite considering the percentage of all instances that are correctly classified regardless of their class, SBDT also outperforms the others.

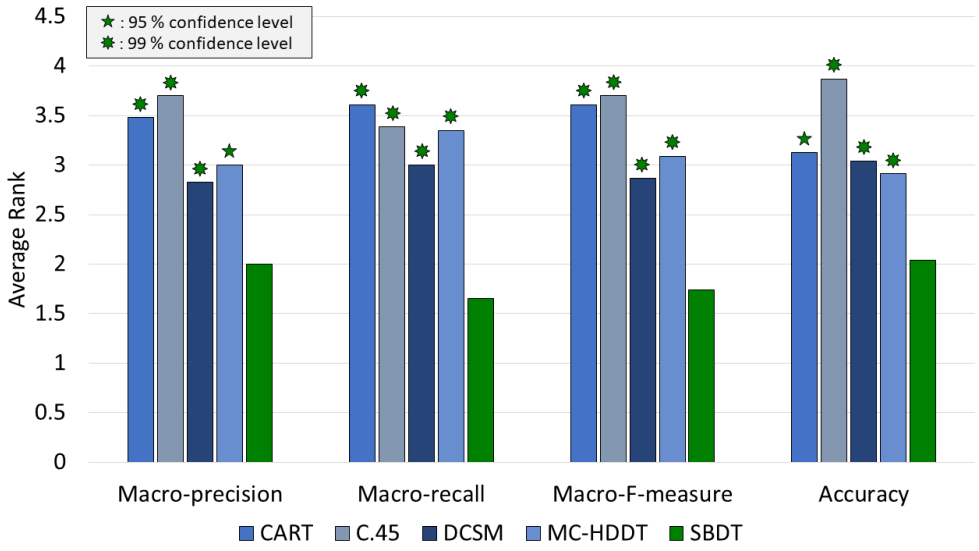


Figure 4.5: The comparison of the experimental results on the real-world multi-class datasets with respect to the average rank

Performance Measures	Decision Tree	α			Statistics	p-value
		0.1	**0.05**	**0.01**		
Precision	CART	✓	✓	✓	36	0.000960
	C4.5	✓	✓	✓	48	0.003097
	DCSM	✓	✓	✓	29	0.000458
	MC-HDDT	✓	✓	-	76	0.029666
Recall	CART	✓	✓	✓	24	0.000263
	C4.5	✓	✓	✓	41	0.001588
	DCSM	✓	✓	✓	6	0.000030
	MC-HDDT	✓	✓	✓	57	0.006877
F-measure	CART	✓	✓	✓	21	0.000186
	C4.5	✓	✓	✓	36	0.000960
	DCSM	✓	✓	✓	14	0.000081
	MC-HDDT	✓	✓	✓	56	0.006315
Accuracy	CART	✓	✓	-	79	0.036368
	C4.5	✓	✓	✓	41	0.001588
	DCSM	✓	✓	✓	37	0.001063
	MC-HDDT	✓	✓	✓	60	0.008837

Table 4.6: The statistical results based on the Wilcoxon signed-rank test comparing SBDT against other decision trees

CHAPTER V

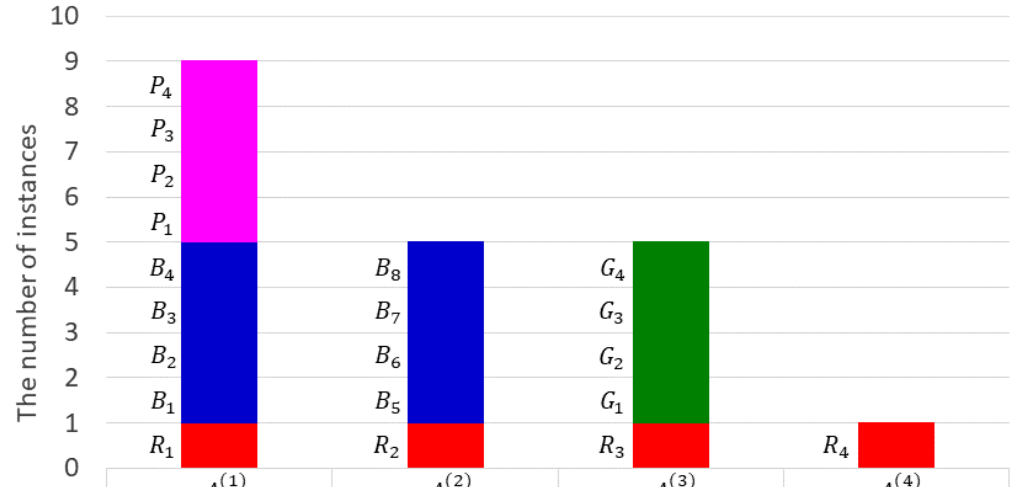
DECISION TREE INDUCTION FOR A MULTI-CLASS IMBALANCED DATASET

In this chapter, another weighting function is introduced to deal with a multi-class imbalanced dataset consisting of categorical attributes. It is called the class-intermingle weighting function, denoted by β. Then, IWE according to the set of instance weights obtained from the composition of functions β and μ, which has been specifically named the class-intermingle-balancing entropy (IBE), is used as the splitting condition in the recursive partitioning algorithm to build the generalization of SBDT called the generalized self-balancing decision tree (GSBT).

The motivation of the methods presented in this chapter comes from the success of using the class-overlapping-balancing entropy (OBE) to build the self-balancing decision tree (SBDT) for handling the multi-class imbalanced numeric dataset. For each attribute, assigning weights based on the overlapping of the instance values in each class gives the impressive results. However, it can only be done with numeric attributes. For categorical attributes, there is no well-defined range. Thus, it cannot define the region that the values of each class are overlapping. That is, the previously proposed methods, which consider the range of each class, lose the basic property of the decision tree to deal with the datasets consisting of categorical attributes.

For these reasons, an improvement of the class-overlapping weighting function (α) is proposed in this chapter. The difference between the series of classes that mix in each attribute value is used for assigning the weighted value to each instance, instead of using the overlapping of class ranges that cannot apply to the categorical attribute. The equivalent concept similar to that of function α can still be employed in determining the weights in this situation. Initially, the instances that their values are different from the

others in the same class are specified as the outliers, and their weights are all set to be 0. For other instances, the weights are assigned based on the assumption according to the class of instances, which have two issues to discuss:



	$A_j^{(1)}$	$A_j^{(2)}$	$A_j^{(3)}$	$A_j^{(4)}$
Class ■	0	0	4	0
Class ◆	4	0	0	0
Class ▲	4	4	0	0
Class ●	1	1	1	1

Figure 5.1: The distribution of the sample dataset with respect to categorical attribute A_j having 4 different values for assigning the weighted value to each instance from the red-circle class according to that attribute

- *The number of distinct classes sharing a specific value*

  An instance that its categorical value shares among instances from a large number of classes should be assigned smaller weight than the one that its value shares among lesser classes. For example, in Figure 5.1, there are instances at the possible values of the categorical attribute as $A_j^{(1)}$ sharing among three classes, $A_j^{(2)}$ sharing among two classes, $A_j^{(3)}$ sharing among two classes, and $A_j^{(4)}$ sharing no class, while instances from the red-circle class are instances R_1, R_2, R_3 and R_4, distributed evenly. So, instance R_1 must have the smallest weight, while instance R_4 must have the largest weight.

- *The class size effect*

  An instance that its categorical value shares with another instance from a larger class size should be assigned a smaller weight than the one that its categorical value shares with another instance from a lesser class size. For example, in Figure 5.1, there are instances from two classes of instance R_2 at $A_j^{(2)}$ and instance R_3 at $A_j^{(3)}$. Explicitly, these values have only a single instance from the red-circle class. However, another class that mixes with instance R_2 is the blue-triangle class having 8 instances in the whole blue-triangle class, while another class that mixes with instance R_3 is the green-square class having 4 instances in the whole green-square class. So, instance R_2 must have a smaller weight than instance R_3.

In addition, a set of weights corresponding to instances from each class is normalized. They are made to have the same total weights using the class-balancing weighting function μ (4.4) for balancing between classes. Then, the entropy computed according to that set of instance weights is employed to build the decision tree for classifying multi-class imbalanced datasets consisting of categorical attributes, which is called the generalized self-balancing decision tree (GSBT).

5.1 Proposed Methodologies

In this section, the mathematical definitions corresponding to the proposed methodologies are formally introduced. It also includes related properties, examples, and the pseudocode for each proposed algorithm.

5.1.1 Class-intermingle Weighting Function

Initially, the third proposed weighting function is presented in accordance with the assumption mentioned above, which is called the class-intermingle weighting function denoted by β. For any set of instance categorical values $t(\mathbf{D}) = \{t(\vec{x}_i) \in dom(t) \mid (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, 2, ..., m\}$, where $dom(t) = \{\tau^{(l)} \mid l = 1, 2, ..., q\}$ is the domain of t having q distinct values, the class-intermingle weighting function is defined with respect to instance value $t(\vec{x}_i) \in t(\mathbf{D})$ by (5.1) as follows:

$$\beta(t(\vec{x}_i)) = \left(\sum_{k=1}^{p} \mathbb{1}(|t^{(t(\vec{x}_i))}(\mathbf{D}_k)| \geq \eta) \cdot \log_2(m_k) \right)^{-1} \tag{5.1}$$

where

- $\mathbb{1}(\omega)$ is the indicator function, 1 if the condition ω is true, 0 otherwise
- $t^{(l)}(\mathbf{D}_k)$ is the set of instance categorical values with respect to class c_k that their values are all $\tau^{(l)}$, i.e. $t^{(l)}(\mathbf{D}_k) = \{t(\vec{x}_i) \in t(\mathbf{D}_k) \mid (\vec{x}_i, y_i) \in \mathbf{D}_k \text{ and } t(\vec{x}_i) = \tau^{(l)} \text{ for } i = 1, 2, ..., m\}$
- η is the outlier parameter indicating the minimum number of instances in each class that there are required for each value, which is set to be 1 as default.

The weight corresponding to each instance is inversely proportional to the total number of instances in all classes containing at least η instances that their values are the same as the instance considered, which has been scaled down by the logarithmic function.

Consider the example in Figure 5.1, the calculation of the class-intermingle weight for each instance in the red-circle class is demonstrated as follows:

- There are instances from three classes having the same value as instance R_1: the red-circle class, the pink-diamond class, and the blue-triangle class, which contain 4 total instances, 4 total instances, and 8 total instances, respectively. Thus,

$$\beta(\pi_j(R_1)) = \frac{1}{\log_2(4) + \log_2(4) + \log_2(8)} = \frac{1}{2+2+3} = \frac{1}{7}.$$

- There are instances from two classes having the same value as instance R_2: the red-circle class and the blue-triangle class, which contain 4 total instances and 8 total instances, respectively. Thus,

$$\beta(\pi_j(R_2)) = \frac{1}{\log_2(4) + \log_2(8)} = \frac{1}{2+3} = \frac{1}{5}.$$

- There are instances from two classes having the same value as instance R_3: the red-circle class and the green-square class, which contain 4 total instances and 4 total instances, respectively. Thus,

$$\beta(\pi_j(R_3)) = \frac{1}{\log_2(4) + \log_2(4)} = \frac{1}{2+2} = \frac{1}{4}.$$

- There are only instances from one class having the same value as instance R_4: the red-circle class, which contains 4 total instances. Thus,

$$\beta(\pi_j(R_4)) = \frac{1}{\log_2(4)} = \frac{1}{2}.$$

That is, instance R_1 receives the smallest weight, while instance R_4 receives the largest weight, corresponding to the first issue considered in the assumption. Moreover, instance R_2 receives a smaller weight than instance R_3, corresponding to the second issue considered in the assumption.

Theoretically, two properties corresponding to those issues of determining the weighted values to the instances in $\mathbf{D}$ with respect to $t(\mathbf{D})$ using the class-intermingle weighting function are presented in Theorem 5.1 and Theorem 5.2 as follows:

Theorem 5.1. For a set of instance categorical values $t(\mathbf{D})$, let $C_a \subseteq C$ and $C_b \subseteq C$ be the sets of classes having a greater number of instances than a specific outlier parameter, which their values are the same as instance values $t(\vec{x}_a) \in t(\mathbf{D})$ and $t(\vec{x}_b) \in t(\mathbf{D})$, respectively. If $C_a \subset C_b$, then $\beta(t(\vec{x}_a)) > \beta(t(\vec{x}_b))$.

Proof. Let $M_a = \{m_k \mid c_k \in C_a \text{ for } k = 1, 2, ..., p\}$ and $M_b = \{m_k \mid c_k \in C_b \text{ for } k = 1, 2, ..., p\}$ be the sets containing the number of instances from each class in C_a and C_b, respectively. Since $C_a \subset C_b$, M_a is a subset of M_b, which M_b can be partitioned as

$M_b = M_a \cup M_r$ where $M_r = M_b \setminus M_a$. Hence,

$$\prod_{m \in M_a} m < \prod_{m \in M_a} m + \prod_{m \in M_r} m = \prod_{m \in M_b} m$$

$$\log_2 \left(\prod_{m \in M_a} m \right) < \log_2 \left(\prod_{m \in M_b} m \right)$$

$$\sum_{m \in M_a} \log_2 m < \sum_{m \in M_b} \log_2 m$$

$$\left(\sum_{m \in M_a} \log_2 m \right)^{-1} > \left(\sum_{m \in M_b} \log_2 m \right)^{-1}$$

$$\beta(t(\vec{x}_a)) > \beta(t(\vec{x}_b)).$$

□

Theorem 5.2. For a set of instance categorical values $t(\mathbf{D})$, let $C_a \subseteq C$ and $C_b \subseteq C$ be the sets of classes having a greater number of instances than a specific outlier parameter, which their values are the same as instance values $t(\vec{x}_a) \in t(\mathbf{D})$ and $t(\vec{x}_b) \in t(\mathbf{D})$, respectively. If $C_a \setminus C_b = \{c_{k_a}\}$ and $C_b \setminus C_a = \{c_{k_b}\}$ s.t. $m_{k_a} < m_{k_b}$, then $\beta(t(\vec{x}_a)) > \beta(t(\vec{x}_b))$.

Proof. Let $M_a = \{m_k \mid c_k \in C_a \text{ for } k = 1, 2, ..., p\}$ and $M_b = \{m_k \mid c_k \in C_b \text{ for } k = 1, 2, ..., p\}$ be the sets containing the number of instances from each class in C_a and C_b, respectively. Since $C_a \setminus C_b = \{c_{k_a}\}$, so $M_a \setminus M_b = \{m_{k_a}\}$, then M_a can be transformed as $M_a = (M_a \cap M_b) \cup \{m_{k_a}\}$. Likewise, $M_b = (M_a \cap M_b) \cup \{m_{k_b}\}$. Hence,

$$m_{k_a} < m_{k_b}$$

$$m_{k_a} \times \left(\prod_{m \in M_a \cap M_b} m \right) < m_{k_b} \times \left(\prod_{m \in M_a \cap M_b} m \right)$$

$$\prod_{m \in (M_a \cap M_b) \cup \{m_{k_a}\}} m < \prod_{m \in (M_a \cap M_b) \cup \{m_{k_b}\}} m$$

$$\prod_{m \in M_a} m < \prod_{m \in M_b} m$$

$$\log_2 \left(\prod_{m \in M_a} m \right) < \log_2 \left(\prod_{m \in M_b} m \right)$$

$$\sum_{m \in M_a} \log_2 m < \sum_{m \in M_b} \log_2 m$$

$$\left(\sum_{m \in M_a} \log_2 m \right)^{-1} > \left(\sum_{m \in M_b} \log_2 m \right)^{-1}$$

$$\beta(t(\vec{x}_a)) > \beta(t(\vec{x}_b)).$$

□

Then, a set of class-intermingle weights received from a set of instance categorical values $t(\mathbf{D})$ is determined by (5.2) as follows:

$$\beta(t(\mathbf{D})) = \{(\beta(t(\vec{x}_i)), y_i) \mid (\vec{x}_i, y_i) \in \mathbf{D} \text{ for } i = 1, 2, ..., m\} \tag{5.2}$$

In addition, the pseudocode to obtain a set of class-intermingle weights with respect to a set of instance categorical values $t(\mathbf{D})$ is displayed in Algorithm 5.1. For each value $\tau^{(l)}$, the number of instances in each class having value $\tau^{(l)}$ is counted. If it is not less than the outlier parameter, the weight corresponding to $\tau^{(l)}$ is updated. For each instance, therefore, the inverse of the weight corresponding to its value is assigned.

Algorithm 5.1: IntermingleWeight($\mathbf{D}, t, \eta$)

Input: dataset $\mathbf{D}$, a function to obtain the instance categorical values t, an outlier parameter η

Output: a set of class-intermingle weights with respect to $t(\mathbf{D})$

```
• let w^(l) = 0 for l = 1, 2, ..., q
for each value τ^(l) do
    for each class c_k do
        • generate the set of instance categorical values with respect to
        class c_k that their values are all τ^(l): t^(l)(D_k)
        if |t^(l)(D_k)| ≥ η then
            • w^(l) += log_2(m_k)
for each value t(x_i) do
    • w_i = w^(t(x_i))
return {(1/w_i, y_i) | (x_i, y_i) ∈ D for i = 1, ..., m}
```

5.1.2 Generalized Self-balancing Decision Tree

In this section, a decision tree based on a single attribute, which is a generalization of SBDT, is presented to deal with a categorical attribute by considering the set of instance weights from applying the composition of functions β and μ to dataset $\mathbf{D}$. It is called a set of class-intermingle-balancing weights, which is defined according to each categorical attribute A_j by (5.3) as follows:

$$IBW_{\pi_j}(\mathbf{D}) = \mu(\beta(\pi_j(\mathbf{D}))) \tag{5.3}$$

Thus, the individually weighted entropy corresponding to a set of class-intermingle-balancing weights is also called the class-intermingle-balancing entropy (IBE) defined by (5.4) as follows:

$$IBE_{\pi_j}(\mathbf{D}) = IWE\left(IBW_{\pi_j}(\mathbf{D})\right) \tag{5.4}$$

Since $IBW_{\pi_j}(\mathbf{D})$ contains a balanced proportion of instance weights corresponding to each class, $IBE_{\pi_j}(\mathbf{D})$ always equals to one. In a recursive partitioning algorithm, however, the class-intermingle-balancing entropy is applied to each subset of instance weights for each partition after splitting shown in (5.5). In which the proportion of each class varies according to the splitting condition.

$$IBE_{\pi_j}(\mathbf{D}^{(l)}) = IWE\left(IBW^{(l)}_{\pi_j}(\mathbf{D})\right) \tag{5.5}$$

So, a decision tree from the recursive partitioning algorithm using both OBE and IBE as the splitting measures is called the generalized self-balancing decision tree or GSBT, and that algorithm is also called the GSBT algorithm. Its pseudocode is displayed in Algorithm 5.2. For each attribute A_j, it starts by obtaining either a set of class-overlapping weights or a set of class-intermingle weights depending on the attribute's type. Then, the greedy approach is applied to select the best condition for splitting dataset $\mathbf{D}$ based on the value of OBE/IBE according to each candidate from $\pi_j(\mathbf{D})$.

5.2 Experiments and Results

In order to evaluate the performance of inducing a decision tree that uses IBE as the splitting measure, two collections of experiments are conducted. The first collection employs the synthetic datasets varying their imbalanced ratios, to show the effectiveness of IBE over SE. For the second collection, the performance comparison in classifying the real-world datasets from the UCI repository of GSBT with other methods is presented.

5.2.1 Experiments on Synthetic Datasets

In this section, an improvement of the Shannon's entropy (SE) to classify minority instances in the multi-class imbalanced datasets dealing with categorical attributes using the class-intermingle-balancing entropy (IBE) is demonstrated in the experiments on the

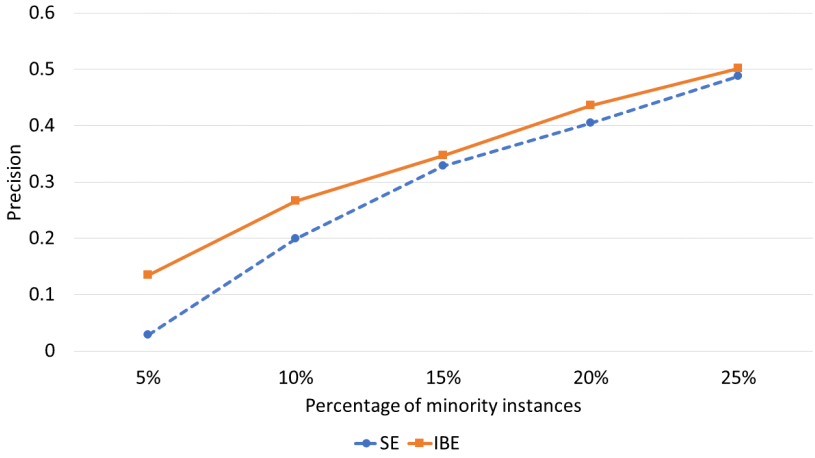
Algorithm 5.2: GSBT($\mathbf{D}, \eta$)

Input: dataset $\mathbf{D}$, an outlier parameter η
Output: a decision tree

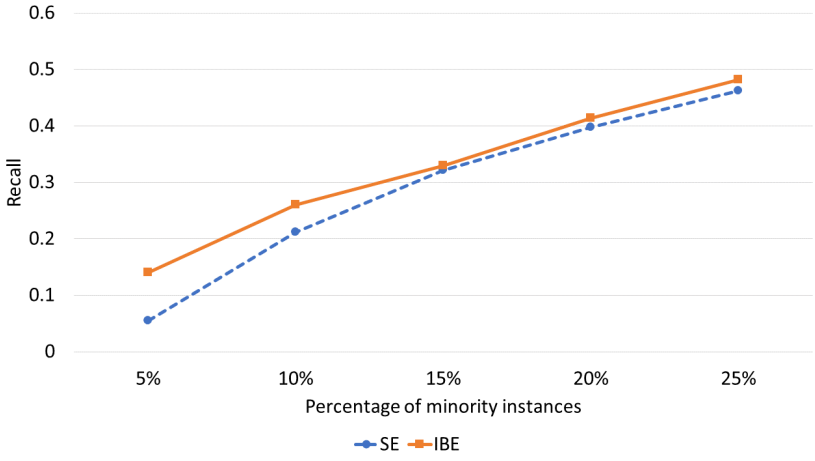
- • create the root node of the tree
- **if** all instances in $\mathbf{D}$ are in the same class **then**
  - **return** the leaf node with respect to that class
- **else**
  - `/* select the splitting condition */`
  - **for** each attribute A_j **do**
    - **if** A_j is numeric attribute **then**
      - • receive a set of class-overlapping weights with respect to $\pi_j(\mathbf{D})$ via **Algorithm 4.2**: W = OverlappingWeight($\mathbf{D}$, π_j)
    - **else**
      - • receive a set of class-intermingle weights with respect to $\pi_j(\mathbf{D})$ via **Algorithm 5.1**: W = IntermingleWeight($\mathbf{D}$, π_j, η)
    - • balance a set of weights via **Algorithm 4.3**: BW = BalancingWeight(W)
    - • apply the greedy approach on $\pi_j(\mathbf{D})$ based on the individually weighted entropy with respect to BW via **Algorithm 4.1**
    - • update the best splitting condition
  - • obtain the best splitting condition
  - `/* recursively partition the dataset */`
  - • separate the dataset into q partitions corresponding to the outcomes of the splitting condition: $\mathbf{D} = \mathbf{D}^{(1)} \cup \mathbf{D}^{(2)} \cup ... \cup \mathbf{D}^{(q)}$
  - • iterate for each partition: GSBT($\mathbf{D}^{(l)}, \eta$) for $l = 1, 2, ..., q$

collections of synthetic datasets generated according to the following specifications:

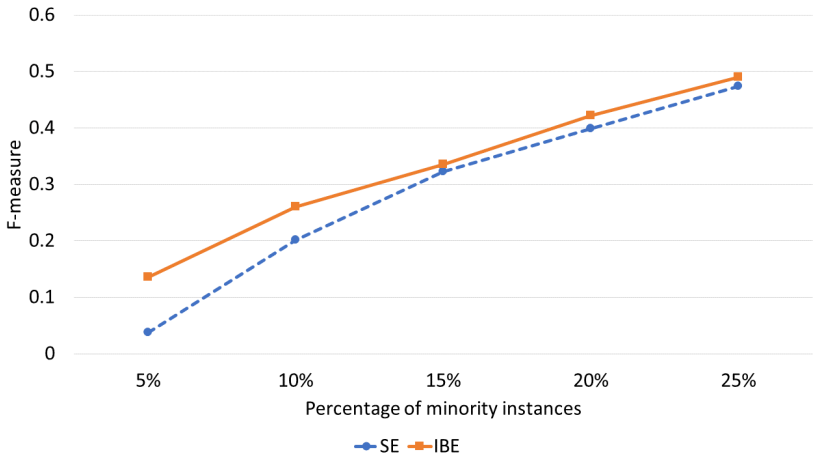
- Each dataset contains 1000 instances consisting of 10 categorical attributes with 4 distinct values for each.
- Each instance is labeled as one of four determined classes as follows:
  - The minority class (containing the instances not exceed 25%)
  - The majority class (containing the instances not less than 25%)
  - The third class (containing exactly 25% of all instances)
  - The fourth class (containing exactly 25% of all instances)
- For each attribute, two out of four values are randomly selected for each class.



(a) comparing by the precision

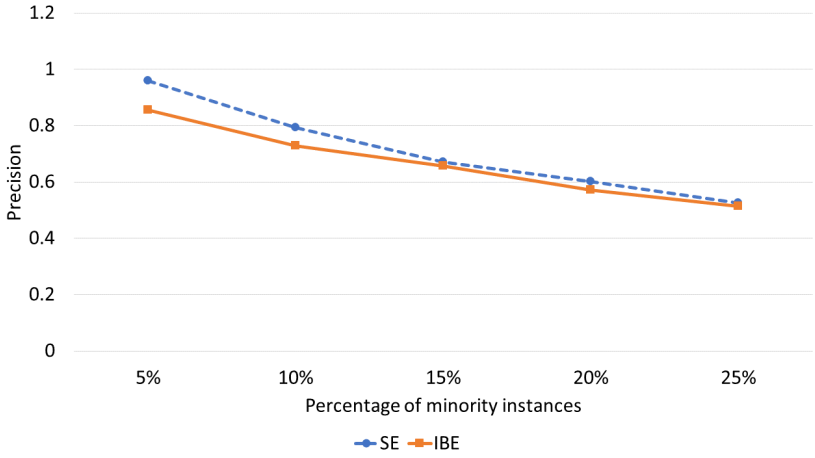


(b) comparing by the recall

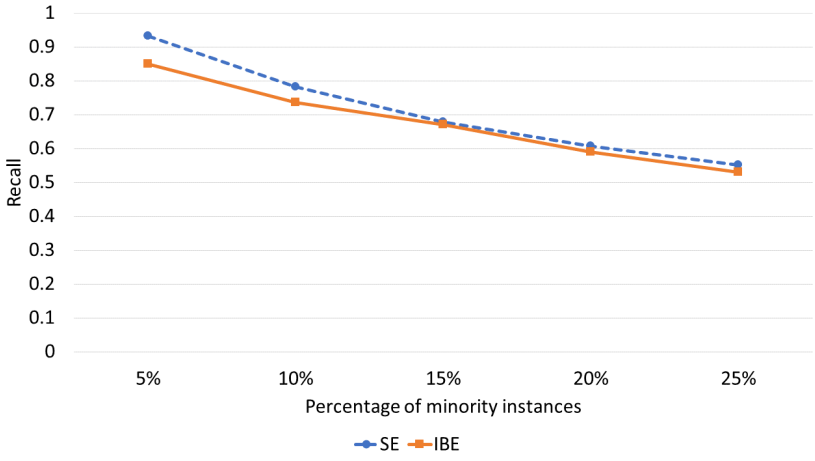


(c) comparing by the F-measure

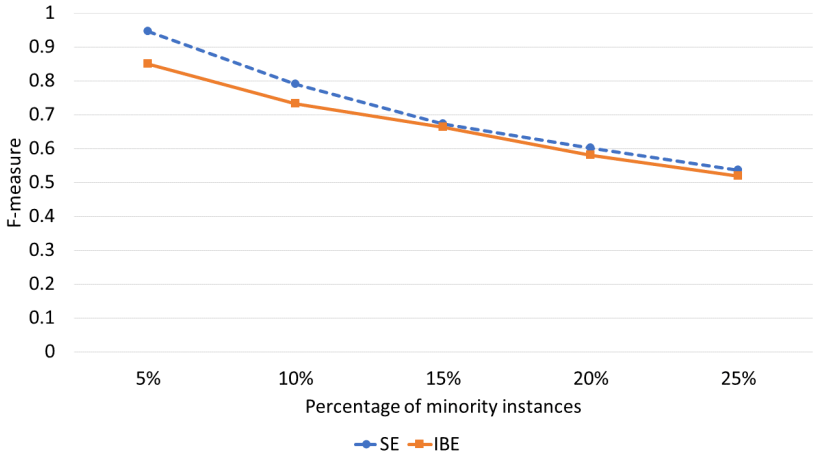
Figure 5.2: The experimental results with respect to the minority class on the synthetic multi-class datasets dealing with categorical attributes varying the percentage of minority instances



(a) comparing by the precision



(b) comparing by the recall



(c) comparing by the F-measure

Figure 5.3: The experimental results with respect to the majority class on the synthetic multi-class datasets dealing with categorical attributes varying the percentage of minority instances

Then, the discrete uniform sampling is performed on these values.

- There are five groups of experiments varying the percentages of minority instances from 5% to 25%.
- For each group, ten datasets are synthesized and then applying five-fold cross-validation ten times to evaluate the performance of each method.

Accordingly, the average results of SE and IBE are compared via the precision (2.16), the recall (2.17), and the F-measure (2.18) with respect to the minority class and the majority class displaying in Figure 5.2 and Figure 5.3, respectively.

For the results corresponding to the minority class, their results are similar in all performance measures, which ostensibly increase when the percentage of minority instances increases. Nonetheless, IBE significantly outperforms SE when the number of instances in the minority class is tiny. For example, the performance of SE is only about 0.05 when there are 5 % of minority instances, while IBE offers the performance up to 0.15. Then their values will approach to be equal when a dataset is more balanced. For the case that the dataset is completely balanced, the performance of SE and IBE are almost the same at 0.5 due to the sharing of attribute values for each class in each attribute. These results evidently confirm that SE is not suitable to deal with the multi-class imbalanced problem having categorical attributes, which is able to be captured using IBE.

Differently, considering the results based on the majority class, all performance measures gradually decrease when a dataset is more balanced. That is because of the decrease in the percentage of majority instances. However, due to the effort to reduce the bias toward the majority class of IBE, it gives a slightly worse result according to the majority class compared to SE in the case of extreme imbalance. For example, the performance of SE is about 0.95 when there are only 5 % of minority instances, while IBE offers the performance around 0.85. Their values will approach to the same value when a dataset is more balanced, which are equal to 0.5 when the dataset is balanced.

5.2.2 Experiments on Real-world Datasets

To demonstrate the effectiveness of IBE on general datasets, the decision tree built based on IBE like GSBT is evaluated on real-world datasets in this section. Its results are compared to those of four other decision trees. The first two models are the decision trees from two well-known algorithms, i.e. CART [44] and C.45 [39]. Additionally, the decision tree built based on DCSM [45] is used as well. The other is the state-of-the-art decision tree designed for a multi-class imbalanced dataset like MC-HDDT [81].

Datasets

For the datasets used in the experiments, they are collected from the UCI repository [86] having a total of 14 datasets, which are summarized in Table 5.1. They are sorted in the ascending order by the imbalanced ratio (I.R.), corresponding to the percentage of instances in each class (% Cl.). The first two columns represent the index and the name of each dataset. The next four columns indicate the number of instances (# Inst.), the number of categorical attributes (# CatAtt.), the number of numeric attributes (# NumAtt.), and the number of classes (# Cl.), respectively. In order to evaluate the performance of each decision tree, the five-fold cross-validation is employed to divide the dataset into the training set and the testing set, which is repeated 20 times. That is, there are up to one hundred experiments performed on each dataset.

No.	Dataset	# Inst.	# CatAtt.	# NumAtt.	# Cl.	% Cl.	I.R.
1	TAEvaluation	151	4	1	3	34.44 / 33.11 / 32.45	1.06
2	Chess(KRvsKP)	3196	36	0	2	52.22 / 47.78	1.09
3	HeartDisease	270	8	5	2	55.56 / 44.44	1.25
4	Lymphography	142	18	0	2	57.04 / 42.96	1.33
5	Hayes-roth	132	4	0	3	38.64 / 38.64 / 22.73	1.7
6	Tic-tac-toe	958	9	0	2	65.34 / 34.66	1.89
7	Contraceptive	1473	8	1	3	42.7 / 34.69 / 22.61	1.89
8	GermanCredit	1000	13	7	2	70.0 / 30.0	2.33
9	Balance-scale	625	4	0	3	46.08 / 46.08 / 7.84	5.88
10	Zoo	97	16	0	6	42.27 / 20.62 / 13.4 / 10.31 / 8.25 / 5.15	8.2
11	Nursery	12958	8	0	4	33.34 / 32.92 / 31.21 / 2.53	13.17
12	CarEvaluation	1728	6	0	4	70.02 / 22.22 / 3.99 / 3.76	18.62
13	Abalone	4168	1	7	21	16.53 / 15.21 / 13.63 / 11.68 / 9.38 / 6.41 / 6.21 / 4.87 / 3.02 / 2.76 / 2.47 / 1.61 / 1.39 / 1.37 / 1.01 / 0.77 / 0.62 / 0.36 / 0.34 / 0.22 / 0.14	114.83
14	Chess(KRvsK)	28056	6	0	18	16.23 / 14.95 / 12.82 / 10.17 / 9.97 / 7.72 / 7.08 / 6.1 / 5.11 / 2.43 / 2.11 / 1.68 / 1.39 / 0.88 / 0.71 / 0.29 / 0.28 / 0.1	168.63

Table 5.1: The characteristics of the experimental real-world multi-class datasets consisting of both categorical attributes and numeric attributes

Results

Accordingly, the average results of each decision tree are compared via the macro-precision (2.22), the macro-recall (2.23), the macro-F-measure (2.24), and the accuracy (2.25) displaying in four tables for each measure. In each table, a comparison of performance according to each dataset is represented in each row. Furthermore, the rank of each method corresponding to each dataset is shown in the parentheses, emphasizing the best rank in bold. Summarily, the bottom row of each table represents the average rank of each method.

In Table 5.2, the performance comparison according to the macro-precision of each method is shown. GSBT yields the lowest average rank at 2.57, in which there are up to 7 out of 14 datasets that GSBT offers the highest or the second-highest macro-precision. For CART, C4.5, and MC-HDDT, they have the same average rank at 2.71, while DCSM yields the worst average rank at 3.21.

In Table 5.3, the performance comparison according to the macro-recall of each method is shown. GSBT yields the lowest average rank at 2.14, which is much different from other methods. There are up to 8 out of 14 datasets that GSBT obtains the highest macro-recall. For CART, C4.5, and MC-HDDT, they receive a similar average rank that is much lower than DCSM.

In Table 5.4, the performance comparison according to the macro-F-measure of each method is shown. GSBT yields the lowest average rank at 2.29, in which there are up to 7 out of 14 datasets that GSBT offers the highest macro-F-measure. For CART, C4.5, and MC-HDDT, they receive a similar average rank that is much lower than DCSM.

In Table 5.5, the performance comparison according to the accuracy of each method is shown. Both GSBT and C4.5 yield the lowest average rank at 2.57, which is close to CART and MC-HDDT having the same average rank at 2.71. However, there are up to 6 datasets that GSBT offers the highest accuracy, which is more than other methods. For DCSM, it yields the worst average rank at 3.29.

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**GSBT**
1	TAEvaluation	0.5866 (3)	0.5866 (3)	0.5866 (3)	0.5996 (2)	**0.6234 (1)**
2	Chess(KRvsKP)	0.9940 (3)	0.9951 (2)	0.9928 (5)	**0.9952 (1)**	0.9934 (4)
3	HeartDisease	0.5852 (3)	0.5852 (3)	0.5852 (3)	0.7444 (2)	**0.7710 (1)**
4	Lymphography	0.7831 (4)	0.8062 (2)	**0.8098 (1)**	0.7768 (5)	0.7860 (3)
5	Hayes-roth	0.7704 (3)	0.7646 (4)	0.7721 (2)	**0.7812 (1)**	0.7645 (5)
6	Tic-tac-toe	0.8392 (3)	0.8327 (5)	0.8353 (4)	0.8530 (2)	**0.8642 (1)**
7	Contraceptive	0.4409 (3)	**0.4424 (1)**	0.4389 (4)	0.4388 (5)	0.4422 (2)
8	GermanCredit	0.4944 (2)	0.4944 (2)	0.4944 (2)	0.4944 (2)	**0.6258 (1)**
9	Balance-scale	**0.4777 (1)**	0.4749 (4)	0.4705 (5)	0.4754 (3)	0.4770 (2)
10	Zoo	0.9209 (2)	**0.9225 (1)**	0.9002 (4)	0.8812 (5)	0.9097 (3)
11	Nursery	0.9564 (5)	**0.9584 (1)**	0.9573 (2)	0.9570 (3)	0.9564 (4)
12	CarEvaluation	0.8387 (2)	0.8323 (3)	0.7952 (4)	**0.8538 (1)**	0.7918 (5)
13	Abalone	0.0972 (2)	0.0972 (2)	0.0972 (2)	0.0963 (5)	**0.1175 (1)**
14	Chess(KRvsK)	0.5753 (2)	0.4936 (5)	0.4965 (4)	**0.6357 (1)**	0.5626 (3)
	average rank	2.71	2.71	3.21	2.71	2.57

Table 5.2: The experimental results on the real-world multi-class datasets dealing with both categorical attributes and numeric attributes comparing by the macro-precision

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**GSBT**
1	TAEvaluation	0.5823 (3)	0.5823 (3)	0.5823 (3)	0.5970 (2)	**0.6188 (1)**
2	Chess(KRvsKP)	0.9938 (3)	0.9949 (2)	0.9925 (5)	**0.9951 (1)**	0.9931 (4)
3	HeartDisease	0.5642 (3)	0.5642 (3)	0.5642 (3)	0.7285 (2)	**0.7533 (1)**
4	Lymphography	0.7698 (4)	0.7979 (2)	**0.8006 (1)**	0.7605 (5)	0.7727 (3)
5	Hayes-roth	0.6725 (3)	0.6635 (5)	0.6859 (2)	**0.7132 (1)**	0.6715 (4)
6	Tic-tac-toe	0.8384 (3)	0.8320 (5)	0.8336 (4)	0.8496 (2)	**0.8615 (1)**
7	Contraceptive	0.4409 (3)	0.4422 (2)	0.4394 (4)	0.4386 (5)	**0.4434 (1)**
8	GermanCredit	0.4989 (2)	0.4989 (2)	0.4989 (2)	0.4989 (2)	**0.6155 (1)**
9	Balance-scale	0.4736 (2)	**0.4781 (1)**	0.4699 (5)	0.4716 (4)	0.4727 (3)
10	Zoo	0.9167 (2)	0.9167 (2)	0.8979 (5)	0.9083 (4)	**0.9250 (1)**
11	Nursery	0.9556 (5)	0.9565 (2)	0.9559 (3)	0.9557 (4)	**0.9565 (1)**
12	CarEvaluation	0.8423 (2)	0.8241 (3)	0.8195 (4)	**0.8634 (1)**	0.7870 (5)
13	Abalone	0.0745 (2)	0.0745 (2)	0.0745 (2)	0.0741 (5)	**0.1128 (1)**
14	Chess(KRvsK)	0.5386 (2)	0.4802 (4)	0.4731 (5)	**0.5975 (1)**	0.5280 (3)
	average rank	2.79	2.71	3.43	2.79	2.14

Table 5.3: The experimental results on the real-world multi-class datasets dealing with both categorical attributes and numeric attributes comparing by the macro-recall

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**GSBT**
1	TAEvaluation	0.5788 (3)	0.5788 (3)	0.5788 (3)	0.5931 (2)	**0.6100 (1)**
2	Chess(KRvsKP)	0.9939 (3)	0.9950 (2)	0.9926 (5)	**0.9951 (1)**	0.9933 (4)
3	HeartDisease	0.5486 (3)	0.5486 (3)	0.5486 (3)	0.7287 (2)	**0.7540 (1)**
4	Lymphography	0.7688 (4)	0.7952 (2)	**0.7989 (1)**	0.7596 (5)	0.7695 (3)
5	Hayes-roth	0.6900 (3)	0.6816 (5)	0.7019 (2)	**0.7275 (1)**	0.6864 (4)
6	Tic-tac-toe	0.8384 (3)	0.8320 (5)	0.8340 (4)	0.8509 (2)	**0.8623 (1)**
7	Contraceptive	0.4394 (3)	0.4407 (2)	0.4375 (5)	0.4377 (4)	**0.4418 (1)**
8	GermanCredit	0.4340 (2)	0.4340 (2)	0.4340 (2)	0.4340 (2)	**0.6190 (1)**
9	Balance-scale	0.4735 (2)	**0.4744 (1)**	0.4678 (5)	0.4710 (4)	0.4727 (3)
10	Zoo	0.9073 (3)	0.9089 (2)	0.8889 (5)	0.8892 (4)	**0.9106 (1)**
11	Nursery	0.9557 (5)	**0.9572 (1)**	0.9563 (2)	0.9561 (4)	0.9561 (3)
12	CarEvaluation	0.8366 (2)	0.8240 (3)	0.8039 (4)	**0.8545 (1)**	0.7861 (5)
13	Abalone	0.0761 (2)	0.0761 (2)	0.0761 (2)	0.0755 (5)	**0.1082 (1)**
14	Chess(KRvsK)	0.5505 (2)	0.4806 (4)	0.4781 (5)	**0.6107 (1)**	0.5380 (3)
	average rank	2.86	2.64	3.43	2.71	2.29

Table 5.4: The experimental results on the real-world multi-class datasets dealing with both categorical attributes and numeric attributes comparing by the macro-F-measure

No.	Dataset	Decision Tree				
		CART	**C.45**	**DCSM**	**MC-HDDT**	**GSBT**
1	TAEvaluation	0.5833 (3)	0.5833 (3)	0.5833 (3)	0.5982 (2)	**0.6194 (1)**
2	Chess(KRvsKP)	0.9939 (3)	0.9950 (2)	0.9926 (5)	**0.9951 (1)**	0.9933 (4)
3	HeartDisease	0.5898 (3)	0.5898 (3)	0.5898 (3)	0.7380 (2)	**0.7630 (1)**
4	Lymphography	0.7763 (3)	0.8008 (2)	**0.8044 (1)**	0.7674 (5)	0.7761 (4)
5	Hayes-roth	0.6933 (4)	0.6857 (5)	0.7047 (2)	**0.7221 (1)**	0.6952 (3)
6	Tic-tac-toe	0.8537 (3)	0.8479 (5)	0.8498 (4)	0.8656 (2)	**0.8758 (1)**
7	Contraceptive	0.4673 (2)	**0.4674 (1)**	0.4659 (3)	0.4632 (5)	0.4642 (4)
8	GermanCredit	0.6858 (2)	0.6858 (2)	0.6858 (2)	0.6858 (2)	**0.6928 (1)**
9	Balance-scale	0.6548 (2)	**0.6609 (1)**	0.6496 (5)	0.6519 (4)	0.6535 (3)
10	Zoo	0.9462 (3)	0.9462 (3)	0.9356 (5)	0.9663 (2)	**0.9683 (1)**
11	Nursery	0.9847 (4)	**0.9852 (1)**	0.9850 (2)	0.9846 (5)	0.9850 (3)
12	CarEvaluation	0.9348 (2)	0.9326 (3)	0.8947 (5)	**0.9391 (1)**	0.9200 (4)
13	Abalone	0.1564 (2)	0.1564 (2)	0.1564 (2)	0.1555 (5)	**0.1980 (1)**
14	Chess(KRvsK)	0.5760 (2)	0.5325 (3)	0.5283 (4)	**0.5991 (1)**	0.5104 (5)
	average rank	2.71	2.57	3.29	2.71	2.57

Table 5.5: The experimental results on the real-world multi-class datasets dealing with both categorical attributes and numeric attributes comparing by the accuracy

Discussions

Graphically, the bar chart representing the result comparison by the average rank corresponding to each measure is shown in Figure 5.4, in which a lower value indicates a better rank. Moreover, the results of the Wilcoxon signed-rank test that compares the performance of GSBT with other decision trees are shown in Table 5.6. The testing results including the statistical value and the p-value for each comparison is shown in each row, in which the symbol checkmark denotes that the proposed method is significantly better than another method with the $(1 - \alpha)100\%$ confidence level. In addition, the testing results are also indicated in Figure 5.4 using the specific symbols for each confidence level.

For a dataset consisting of both numeric attributes and categorical attributes, the conventional decision trees like CART and C4.5 obtain satisfying results in terms of the macro-precision, which are similar to MC-HDDT and not much less than GSBT. For DCSM, however, it gives the low macro-precision different from the others, which is significantly less than GSBT with the 95% confidence level. This happens because there are a lot of partitions when splitting a dataset by categorical attributes, so there are not many instances in each partition. Nevertheless, DCSM still focuses on the number of distinct classes in those partitions, regardless of the tiny number of instances in each class. Therefore, it gives too much importance to an attribute value having a very small number of instances from the considered class, which does not happen in GSBT. Applying the class-intermingle weight causes a larger weighted value for an instance that its attribute value has more instances from its class. Then, the significance of each attribute value according to each class is properly determined, which does not focus on unnecessary values that may be more related to other classes. In terms of the macro-recall, GSBT ostensibly outperforms all other methods, which is significantly better than C4.5 and DCSM with the 95% confidence level. It is due to the assignment of the class-balancing weight to the instances in each class. It causes all classes to be considered equally, not focusing too much on any specific class, which is able to avoid the overfitting problem. Consequently, from the results of both the macro-precision and the macro-recall, GSBT exactly gives

the improvement of the macro-F-measure comparing against all other methods, which is superior to DCSM with the 95% confidence level. Finally, GSBT also provides the best accuracy, which is equal to C4.5 and not much different from CART and MC-HDDT. While DCSM still gives unacceptable results for this performance measure as well. C4.5 gets impressive results on the dataset consisting of categorical attributes because it is originally designed to deal with a dataset consisting of an attribute with a lot of values, which can be done by normalizing the information gain using the split information.

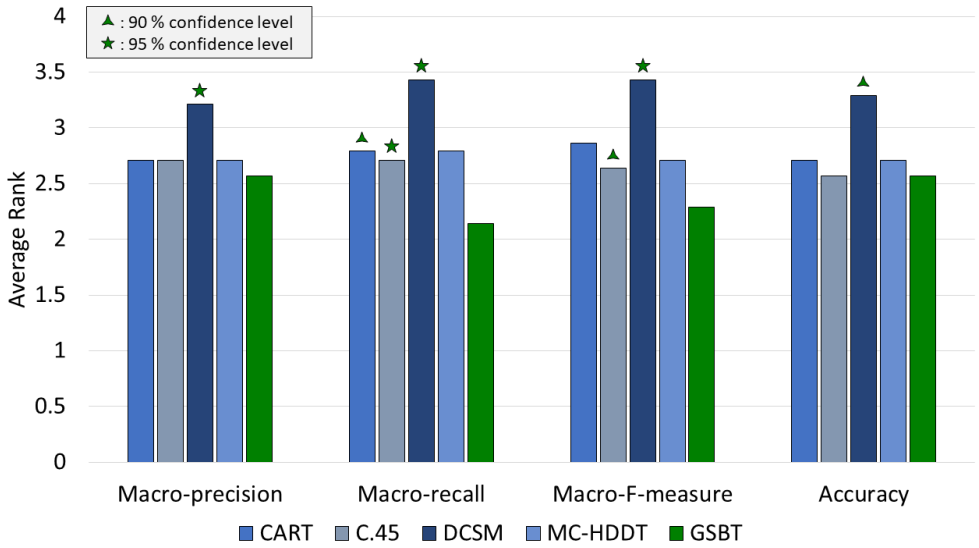


Figure 5.4: The comparison of the real-world multi-class datasets consisting of both categorical attributes and numeric attributes with respect to the average rank

Performance Measures	Decision Tree	α			Statistics	p-value
		0.1	0.05	0.01		
Precision	CART	-	-	-	38	0.181343
	C4.5	-	-	-	34	0.122747
	DCSM	✓	✓	-	21	0.023995
	MC-HDDT	-	-	-	36	0.150145
Recall	CART	✓	-	-	28	0.062021
	C4.5	✓	✓	-	24	0.036797
	DCSM	✓	✓	-	22	0.027766
	MC-HDDT	-	-	-	39	0.198363
F-measure	CART	-	-	-	33	0.110449
	C4.5	✓	-	-	27	0.054711
	DCSM	✓	✓	-	21	0.023995
	MC-HDDT	-	-	-	39	0.198363
Accuracy	CART	-	-	-	35	0.135974
	C4.5	-	-	-	36	0.150145
	DCSM	✓	-	-	27	0.054711
	MC-HDDT	-	-	-	39	0.198363

Table 5.6: The statistical results based on the Wilcoxon signed-rank test comparing GSBT against other decision trees

CHAPTER VI

CONCLUSIONS AND FUTURE WORKS

All proposed methodologies, including their motivations and evaluations, are summarized in this chapter. Besides, the directions to expand this research in the future are also revealed.

6.1 Conclusions

Due to the occurrence of the class imbalanced problem that significantly affects the classification performance, numerous techniques have been continuously introduced to deal with this problem. In this dissertation, therefore, the enhancement of building a classification model based on the recursive partitioning algorithm is proposed to handle the classification problem regardless of the class imbalanced situation. The concept of modified entropy components used in the existing method like the minority entropy (ME) is improved and applied as the basis of the proposed methodologies. They are divided into three works according to their purposes, which is consistently extending the previous methods.

For the first methodology, it presents decision tree induction for a binary-class imbalanced dataset dealing with numeric attributes. The improvement of ME is proposed by fine-tuning the minority range via the outlier detection. Then, the set of instances lying into the range of minority class that ignores the outliers is used to be the components for calculating the entropy, in which the received value is called the minority condensation entropy (MCE). It is used as the splitting measure to build both axis-parallel decision tree and oblique decision tree, which are named the minority condensation decision tree (MCDT) and the oblique minority condensation decision tree (OMCT), respectively. From the experiments on synthetic datasets, MCE shows the ostensible improvement of the performance to classify an imbalanced dataset comparing with the original entropy.

Moreover, when evaluated with the real-world datasets, MCDT gives the best results in terms of F-measure comparing with other axis-parallel decision trees, which is significantly better than ME. For OMCT, it presents the impressive results that notably superior to other decision trees in all performance measures, especially for the recall. Furthermore, it also significantly provides the smallest size of the tree due to the use of oblique hyperplane as the splitting condition.

For the second methodology, it presents decision tree induction for a multi-class imbalanced dataset dealing with numeric attributes. The components to calculate the entropy is adjusted, in which the summation of the weights corresponding to the instances in each class is used instead of the proportion of instances in each class. The entropy computed based on these components is called the individually weighted entropy (IWE). It can be applied in various ways depending on the assignment of weighted value to each instance. So, there are two weighting functions are introduced in this methodology consisting of the class-overlapping weighting function (α) and the class-balancing weighting function (μ). The first one is presented for assigning the weighted value to each instance according to the overlapping range of each class at its position. While the second one is responsible for balancing the total weight corresponding to each class. Then, IWE that is computed with respect to the set of instance weights received from the composition of function α and μ is called the class-overlapping-balancing entropy (OBE). It is used as the splitting measure to build the self-balancing decision tree (SBDT). From the experiments on synthetic datasets, OBE shows the ostensible improvement of the performance to classify an imbalanced dataset comparing with the original entropy. Moreover, when evaluated with the real-world datasets, SBDT significantly outperforms other decision trees in all performance measures, i.e. the macro-precision, the macro-recall, the macro-F-measure, and the accuracy.

For the third methodology, it presents decision tree induction for a multi-class imbalanced dataset dealing with numeric and categorical attributes. The concept of class-overlapping weight is extended to deal with a categorical attribute via the class-intermingle weighting function (β). It is presented for assigning the weighted value to

each instance according to the mixture of different classes in its value. Then, IWE that is computed with respect to the set of instance weights received from the composition of function β and μ is called the class-intermingle-balancing entropy (IBE). Both OBE and IBE are therefore used as the splitting measures, depending on the type of each attribute, to build the generalized self-balancing decision tree (GSBT). From the experiments on synthetic datasets, IBE shows the ostensible improvement of the performance to classify an imbalanced dataset comparing with the original entropy. Moreover, when evaluated with the real-world datasets, GSBT offers better results than other decision trees in all terms of performance measures, especially for the macro-recall.

6.2 Future Works

Although each proposed methodology is highly successful in handling the class imbalanced problem, there is considerable room for future work in the line of this research. Firstly, other weighting functions with respect to IWE may be introduced to enhance the performance of the decision tree in classifying a dataset facing the class imbalanced situation including other issues. Secondly, the performance of each decision tree presented in this dissertation is able to be improved with the ensemble learning method, in which multiple decision trees are combined as a random forest classifier. Lastly, the concept of assigning weighted values to the instances in a dataset can be embedded in other classification models, such as SVM and deep learning, to help them deal with class imbalanced issue.

REFERENCES

[1] O. Maimon and L. Rokach, *Data mining and knowledge discovery handbook*. Springer, 2005.

[2] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets, "Using association rules for product assortment decisions: A case study," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 254–260, 1999.

[3] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[4] L. Kaufmann and P. J. Rousseeuw, "Finding groups in data: an introduction to cluster analysis," *New York: Jonh Wiley*, 1990.

[5] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[6] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson, R. L. Tatham, *et al.*, *Multivariate data analysis*, vol. 5. Prentice hall Upper Saddle River, NJ, 1998.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[8] H. Zhang, "Exploring conditions for the optimality of naive bayes," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 02, pp. 183–198, 2005.

[9] R. E. Wright, *Logistic regression.* American Psychological Association, 1995.

[10] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[11] J. R. Quinlan, "Simplifying decision trees," *International journal of man-machine studies*, vol. 27, no. 3, pp. 221–234, 1987.

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

[13] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[14] I. V. Tetko, D. J. Livingstone, and A. I. Luik, "Neural network studies. 1. comparison of overfitting and overtraining," *Journal of chemical information and computer sciences*, vol. 35, no. 5, pp. 826–833, 1995.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[16] K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan, *Data mining: a knowledge discovery approach*. Springer Science & Business Media, 2007.

[17] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[18] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[19] R. C. Barros, M. P. Basgalupp, A. C. De Carvalho, and A. A. Freitas, "A survey of evolutionary algorithms for decision-tree induction," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 3, pp. 291–312, 2011.

[20] M. Mayo and KDnuggets, "Top data science and machine learning methods used in 2018, 2019." `https://www.kdnuggets.com/2019/04/top-data-science-machine-learning-methods-2018-2019.html/`, 2019. [Online; accessed 29-April-2019].

[21] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information sciences*, vol. 250, pp. 113–141, 2013.

[22] B. Krawczyk, M. Galar, Ł. Jeleń, and F. Herrera, "Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy," *Applied Soft Computing*, vol. 38, pp. 714–726, 2016.

[23] S.-H. Bae and K.-J. Yoon, "Polyp detection via imbalanced learning and discriminative feature learning," *IEEE transactions on medical imaging*, vol. 34, no. 11, pp. 2379–2393, 2015.

[24] N. H. Vo and Y. Won, "Classification of unbalanced medical data with weighted regularized least squares," in *2007 Frontiers in the Convergence of Bioscience and Information Technologies*, pp. 347–352, IEEE, 2007.

[25] Y. Sahin, S. Bulkan, and E. Duman, "A cost-sensitive decision tree approach for fraud detection," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5916–5923, 2013.

[26] W. Wei, J. Li, L. Cao, Y. Ou, and J. Chen, "Effective detection of sophisticated online banking fraud on extremely imbalanced data," *World Wide Web*, vol. 16, no. 4, pp. 449–475, 2013.

[27] S. Hajian, J. Domingo-Ferrer, and A. Martinez-Balleste, "Discrimination prevention in data mining for intrusion and crime detection," in *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp. 47–54, IEEE, 2011.

[28] E. Ramentol, I. Gondres, S. Lajes, R. Bello, Y. Caballero, C. Cornelis, and F. Herrera, "Fuzzy-rough imbalanced learning for the diagnosis of high voltage circuit breaker maintenance: The smote-frst-2t algorithm," *Engineering Applications of Artificial Intelligence*, vol. 48, pp. 134–139, 2016.

[29] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, no. 1, pp. 63–92, 2008.

[30] L. Song, D. Li, X. Zeng, Y. Wu, L. Guo, and Q. Zou, "ndna-prot: identification of dna-binding proteins based on unbalanced classification," *BMC bioinformatics*, vol. 15, no. 1, pp. 1–10, 2014.

[31] S. Razakarivony and F. Jurie, "Vehicle detection in aerial imagery: A small target detection benchmark," *Journal of Visual Communication and Image Representation*, vol. 34, pp. 187–203, 2016.

[32] X. Gao, Z. Chen, S. Tang, Y. Zhang, and J. Li, "Adaptive weighted imbalance learning with application to abnormal activity recognition," *Neurocomputing*, vol. 173, pp. 1927–1935, 2016.

[33] R. Xu, T. Chen, Y. Xia, Q. Lu, B. Liu, and X. Wang, "Word embedding composition for data imbalances in sentiment and emotion classification," *Cognitive Computation*, vol. 7, no. 2, pp. 226–240, 2015.

[34] T. Munkhdalai, O.-E. Namsrai, and K. H. Ryu, "Self-training in significance space of support vectors for imbalanced biomedical event data," *BMC bioinformatics*, vol. 16, no. S7, pp. 1–8, 2015.

[35] Z. Gao, L.-f. Zhang, M.-y. Chen, A. Hauptmann, H. Zhang, and A.-N. Cai, "Enhanced and hierarchical structure algorithm for data imbalance problem in semantic extraction under

massive video dataset," *Multimedia tools and applications*, vol. 68, no. 3, pp. 641–657, 2014.

[36] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[37] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[38] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[39] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[40] K. Boonchuay, K. Sinapiromsaran, and C. Lursinsap, "Decision tree induction based on minority entropy for the class imbalance problem," *Pattern Analysis and Applications*, vol. 20, no. 3, pp. 769–782, 2017.

[41] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005.

[42] E. B. Hunt, J. Marin, and P. J. Stone, *Experiments in induction*. Academic press, 1966.

[43] C. W. Gini, "Variability and mutability, contribution to the study of statistical distributions and relations. studi cconomico-giuridici della r. universita de cagliari (1912). reviewed in: Light, rj, margolin, bh: An analysis of variance for categorical data," *J. American Statistical Association*, vol. 66, pp. 534–544, 1971.

[44] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[45] B. Chandra, R. Kothari, and P. Paul, "A new node splitting measure for decision tree construction," *Pattern Recognition*, vol. 43, no. 8, pp. 2725–2731, 2010.

[46] D. G. Heath, *A geometric framework for machine learning*. Johns Hopkins University, 1993.

[47] D. Heath, S. Kasif, and S. Salzberg, "Induction of oblique decision trees," in *IJCAI*, vol. 1993, pp. 1002–1007, 1993.

[48] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of artificial intelligence research*, vol. 2, pp. 1–32, 1994.

[49] E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 1, pp. 54–68, 2003.

[50] A. López-Chau, J. Cervantes, L. López-García, and F. G. Lamont, "Fisher's decision tree," *Expert Systems with Applications*, vol. 40, no. 16, pp. 6283–6291, 2013.

[51] D. Wickramarachchi, B. Robertson, M. Reale, C. Price, and J. Brown, "Hhcart: An oblique decision tree," *Computational Statistics & Data Analysis*, vol. 96, pp. 12–23, 2016.

[52] M. F. Amasyali and O. Ersoy, "Cline: A new decision-tree family," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 356–363, 2008.

[53] B. Robertson, C. Price, and M. Reale, "Cartopt: a random search method for nonsmooth unconstrained optimization," *Computational Optimization and Applications*, vol. 56, no. 2, pp. 291–315, 2013.

[54] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from imbalanced data sets*. Springer, 2018.

[55] S. Wang, L. L. Minku, and X. Yao, "Dealing with multiple classes in online class imbalance learning.," in *IJCAI*, pp. 2118–2124, 2016.

[56] S. Wang, Z. Li, W. Chao, and Q. Cao, "Applying adaptive over-sampling technique based on data density and cost-sensitive svm to imbalanced learning," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2012.

[57] A. FernáNdez, V. LóPez, M. Galar, M. J. Del Jesus, and F. Herrera, "Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches," *Knowledge-based systems*, vol. 42, pp. 97–110, 2013.

[58] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[59] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 475–482, Springer, 2009.

[60] I. Tomek *et al.*, "Two modifications of cnn.," *IEEE Transactions on Systems, Man andCybernetics*, vol. 6, pp. 769—772, 1976.

[61] C. Bunkhumpornpat and K. Sinapiromsaran, "Dbmute: density-based majority under-sampling technique," *Knowledge and Information Systems*, vol. 50, no. 3, pp. 827–850, 2017.

[62] V. Ganganwar, "An overview of classification algorithms for imbalanced datasets," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 4, pp. 42–47, 2012.

[63] S. Marcellin, D. A. Zighed, and G. Ritschard, "An asymmetric entropy measure for decision trees," in *11th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2006*, 2006.

[64] D. A. Zighed, G. Ritschard, and S. Marcellin, "Asymmetric and sample size sensitive entropy measures for supervised learning," in *Advances in intelligent information systems*, pp. 27–42, Springer, 2010.

[65] P. Lenca, S. Lallich, T.-N. Do, and N.-K. Pham, "A comparison of different off-centered entropies to deal with class imbalance for decision trees," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 634–643, Springer, 2008.

[66] P. Lenca, S. Lallich, and B. Vaillant, "Construction of an off-centered entropy for the supervised learning of imbalanced classes: some first results," *Communications in Statistics —Theory and Methods*, vol. 39, no. 3, pp. 493–507, 2010.

[67] R. Guermazi, I. Chaabane, and M. Hammami, "Aecid: asymmetric entropy for classifying imbalanced data," *Information Sciences*, vol. 467, pp. 373–397, 2018.

[68] T. Kailath, "The divergence and bhattacharyya distance measures in signal selection," *IEEE transactions on communication technology*, vol. 15, no. 1, pp. 52–60, 1967.

[69] C. R. Rao, "A review of canonical coordinates and an alternative to correspondence analysis using hellinger distance," *Qüestiió: quaderns d'estadística i investigació operativa*, vol. 19, no. 1, 1995.

[70] D. A. Cieslak and N. V. Chawla, "Learning decision trees for unbalanced data," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 241–256, Springer, 2008.

[71] D. A. Cieslak, T. R. Hoens, N. V. Chawla, and W. P. Kegelmeyer, "Hellinger distance decision trees are robust and skew-insensitive," *Data Mining and Knowledge Discovery*, vol. 24, no. 1, pp. 136–158, 2012.

[72] T. Dietterich, M. Kearns, and Y. Mansour, "Applying the weak learning framework to understand and improve c4. 5," in *ICML*, pp. 96–104, 1996.

[73] C. Drummond and R. C. Holte, "Exploiting the cost (in) sensitivity of decision tree splitting criteria," in *ICML*, vol. 1, 2000.

[74] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-sensitive learning methods for imbalanced data," in *The 2010 International joint conference on neural networks (IJCNN)*, pp. 1–8, IEEE, 2010.

[75] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[76] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[77] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm," in *icml*, vol. 96, pp. 148–156, Citeseer, 1996.

[78] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," in *Advances in neural information processing systems*, pp. 507–513, 1998.

[79] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of machine learning research*, vol. 5, no. Jan, pp. 101–141, 2004.

[80] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1119–1130, 2012.

[81] T. R. Hoens, Q. Qian, N. V. Chawla, and Z.-H. Zhou, "Building decision trees for the multi-class imbalance problem," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 122–134, Springer, 2012.

[82] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[83] G. W. Corder and D. I. Foreman, *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons, 2014.

[84] D. M. Hawkins, *Identification of outliers*, vol. 11. Springer, 1980.

[85] J. W. Tukey, *Exploratory data analysis*, vol. 2. Reading, Mass., 1977.

[86] C. L. Blake and C. J. Merz, "Uci repository of machine learning databases, 1998," 1998.

BIOGRAPHY

Name Mr. Artit Sagoolmuang

Date of Birth January 10, 1993

Place of Birth Samut Songkhram, Thailand

Educations B.Sc. (Mathematics) (First-class Honours), Kasetsart University, 2014

M.Sc. (Applied Mathematics and Computational Science), Chulalongkorn University, 2016

Scholarships Science Achievement Scholarship of Thailand (SAST)

Publications

- A. Sagoolmuang and K. Sinapiromsaran, "Median-difference window subseries score for contextual anomaly on time series," in *The 8th International Conference of Information and Communication Technology for Embedded Systems 2017 (IC-ICTES)*, pp. 92-97, IEEE, 2017.
- A. Sagoolmuang and K. Sinapiromsaran, "Oblique decision tree algorithm with minority entropy for class imbalanced problem," in *The 33d International Technical Conference on Circuits/Systems, Computers and Communications 2018 (ITC-CSCC)*, pp. 669-672, 2018.
- A. Sagoolmuang and K. Sinapiromsaran, "Oblique decision tree algorithm with minority condensation for class imbalanced problem," *Engineering Journal*, vol. 24, no. 1, pp. 221-237, 2020.
- A. Sagoolmuang and K. Sinapiromsaran, "Decision tree algorithm with class overlapping-balancing entropy for class imbalanced problem," *International Journal of Machine Learning and Computing* , vol. 10, no. 3, pp. 444-451, 2020.