SUPERVISED OPTIMAL DECISION MACHINE LEARNING APPROACH TO CLASS- AND METHOD-LEVEL DATA PREPROCESSING TOWARDS EFFECTIVE SOFTWARE DEFECT PREDICTION

EBUBEOGU AMARACHUKWU FELIX

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2020

SUPERVISED OPTIMAL DECISION MACHINE LEARNING APPROACH TO CLASS- AND METHOD-LEVEL DATA PREPROCESSING TOWARDS EFFECTIVE SOFTWARE DEFECT PREDICTION

EBUBEOGU AMARACHUKWU FELIX

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2020

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: EBUBEOGU AMARACHUKWU FELIX

Registration/Matric No.: WHA150025

Name of Degree: DOCTOR OF PHILOSOPHY

Title of Project Paper/Research Report/Dissertation/Thesis ("Supervised Optimal Decision Machine Learning Approach to Class- and Method-Level Data Preprocessing Towards Effective Software Defect Prediction")

Field of Study: SOFTWARE ENGINEERING

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:
Designation:

SUPERVISED OPTIMAL DECISION MACHINE LEARNING APPROACH TO CLASS- AND METHOD-LEVEL DATA PREPROCESSING TOWARDS EFFECTIVE SOFTWARE DEFECT PREDICTION

ABSTRACT

Software defect prediction provides actionable outputs to software teams while contributing to industrial success. Therefore, predicting the number of defects in a new version of software at both the class and method levels is an important goal of defect prediction studies to assist software teams in optimizing their test efforts towards improving software quality. However, despite remarkable achievements in defect prediction, the quality of the data applied in defect prediction studies has been a major concern, with related quality issues leading to numerous contradictory findings in machine learning research. In addition, a demonstrated approach for predicting the number of defects in a new software version is lacking. Therefore, efforts are required to demonstrate how class- and method-level defect prediction can be achieved for a new software version and to develop an approach for preprocessing the highly imbalanced class- and method-level data available for software defect prediction. To address these issues, first, a data preprocessing framework is proposed to overcome some of the challenges associated with typical software datasets, for instance, irrelevant and redundant features. A machine-learning-driven, supervised optimal decision procedure is followed in the development of this data preprocessing framework, resulting in a prime advantage of bias-free method- and class-level datasets. Second, a method of predicting the number of software defects in an upcoming product release is proposed using predictor variables derived from the defect acceleration observed based on the existing software defects, namely, the defect density, defect velocity and defect introduction time.

The number of defects in the current version of a software product is characterized by this defect acceleration; hence, these derived predictor variables can be used to construct regression models to predict the number of software defects in a new version. An experiment conducted on 69 open-source ELFF Java projects, containing 131,034 classes and 289,132 methods, as well as on the NASA datasets, which contain 10 different Java and C++ projects with 22,838 classes, is reported. To evaluate the effectiveness of the proposed framework for data preprocessing, the average classification performances of six selected state-of-the-art classifiers before and after data preprocessing are investigated and compared across multiple projects with data imbalances between the defective and defect-free classes. For both the class and method levels, these selected state-of-the-art classifiers, namely, naïve Bayes, logistic regression, neural network, K-nearest neighbors, support vector machine and random forest classifiers, achieve noteworthy performance when applied to preprocessed datasets. Moreover, for the ELFF projects, the results at the class and method levels respectively show correlation coefficients of 61% and 60% for the defect density, -11% and -4% for the defect introduction time, and 94% and 93% for the defect velocity (consistent results are also obtained for the NASA datasets, as presented in the results section). The proposed approach can serve as a blueprint for program testing to enhance the effectiveness of software development activities.

Keywords: Machine learning, software defect, defect prediction, data preprocessing, defect velocity.

PENDEKATAN PEMBELAJARAN MESIN KEPUTUSAN OPTIMUM YANG DIAWASI KEPADA TAHAP KELAS- DAN KAEDAH PRA PEMPROSESAN DATA KE ARAH RAMALAN KECACATAN PERISIAN YANG BERKESAN

ABSTRAK

Ramalan kecacatan perisian memberikan output tindakan yang boleh dilakukan kepada pasukan perisian sambil menyumbang kepada kejayaan industri. Oleh itu, meramalkan bilangan kecacatan dalam versi perisian baru di kedua-dua peringkat kelas dan kaedah adalah matlamat penting bagi kajian ramalan kecacatan untuk membantu pasukan perisian dalam mengoptimumkan usaha ujian mereka ke arah meningkatkan kualiti perisian. Walau bagaimanapun, disebalik pencapaian yang luar biasa dalam ramalan kecacatan, kualiti data yang digunakan dalam kajian ramalan kecacatan telah menjadi kebimbangan utama, dengan isu kualiti yang berkaitan yang membawa kepada banyak penemuan bertentangan dalam penyelidikan pembelajaran mesin. Di samping itu, pendekatan yang ditunjukkan untuk meramalkan bilangan kecacatan dalam versi perisian baru adalah kurang. Oleh itu, usaha diperlukan untuk menunjukkan bagaimana ramalan kecacatan tahap kelas- dan kaedah- boleh dicapai untuk versi perisian baru dan untuk membangunkan pendekatan untuk memproses data kelas- yang sangat tidak seimbang dan tahap-kaedah yang tersedia untuk ramalan kecacatan perisian. Untuk menangani isu-isu ini, pertama, rangka kerja pra pemprosesan data dicadangkan untuk mengatasi beberapa cabaran yang berkaitan dengan dataset perisian tipikal, contohnya, ciri-ciri tidak relevan dan berlebihan. Prosedur keputusan optimum yang diawasi oleh mesin dan pembelajaran diikuti dalam pembangunan kerangka kerja pra pemprosesan data ini, menghasilkan kelebihan utama bagi kaedah bebas bias dan tahap kelas datasets. Kedua, satu kaedah untuk meramal bilangan kecacatan

perisian dalam pelepasan produk yang akan datang dicadangkan menggunakan pemboleh ubah ramalan yang diperoleh daripada pecutan kecacatan yang diperhatikan berdasarkan kecacatan perisian yang sedia ada, iaitu, kepadatan kecacatan, kecacatan halaju dan masa pengenalan kecacatan. Bilangan cacat dalam versi produk perisian semasa dicirikan oleh pecutan kecacatan ini; Oleh itu, pembolehubah peramal yang diperolehi ini boleh digunakan untuk membina model regresi untuk meramalkan bilangan kecacatan perisian dalam versi baru. Eksperimen yang dijalankan pada 69 projek sumber terbuka ELFF Java, mengandungi 131,034 kelas dan 289,132 kaedah, serta pada kumpulan NASA dataset, yang mengandungi 10 projek Java dan C ++ yang berbeza dengan 22,838 kelas, dilaporkan. Untuk menilai keberkesanan rangka kerja yang dicadangkan bagi pra pemprosesan data, purata prestasi pengelasan untuk enam pengelas terkini yang terpilih sebelum dan selepas pengolahan pra pemprosesan data telah disiasat dan dibandingkan di pelbagai projek dengan ketidakseimbangan data antara kelas-kelas yang rosak dan cacat. Bagi kedua-dua peringkat kelas dan kaedah, pengelas terkini yang dipilih, iaitu, naive Bayes, logistic regression, neural network, K-nearest neighbors, support vector machine and random forest, mencapai prestasi yang patut diberi perhatian apabila digunakan untuk dataset yang diproses terlebih dahulu. Selain itu, untuk projek ELFF, keputusan di peringkat kelas dan kaedah masing-masing menunjukkan koefisien korelasi sebanyak 61% dan 60% untuk kepadatan kecacatan, -11% dan -4% untuk masa pengenalan kecacatan, dan 94% dan 93% untuk halaju kecacatan (hasil yang konsisten juga diperolehi untuk dataset NASA, seperti yang dibentangkan di bahagian hasil). Pendekatan yang dicadangkan ini boleh dijadikan sebagai pelan tindakan bagi ujian program untuk meningkatkan keberkesanan aktiviti pembangunan perisian.

Kata kunci: Mesin pembelajaran, kecacatan perisian, ramalan kecacatan, pra pemprosesan data, halaju kecacatan.

ACKNOWLEDGEMENTS

I wish to express my gratitude to Professor Sai Peck Lee, my research supervisor, for her kind advice and thoughtful comments throughout this work. I also wish to express my gratitude to the technician at the research lab who ensured that all resources were in place at the beginning of this work. My sincere thanks go to the faculty dean, the deputy dean, the head of the software engineering department and other nonacademic staff for their support and encouragement throughout this research work. Finally, I wish to express my special appreciation for my family, friends and well-wishers who have supported me all the way through this work. To you all, I will forever remain grateful. Glory to almighty God, who gave me the opportunity and wisdom to complete this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

CA : Classification Accuracy
FN : False Negatives
FP : False Positives
GAUS : Genetic Algorithm-based Under-Sampling
IBL : Instance-Based Learner
IS : Information Score
JIT : Just-In-Time
KNN : K-Nearest-Neighbors
LR : Logistic Regression
MCC : Matthews Correlation Coefficient
NASA : National Aeronautics and Space Administration
RBF : Radial Basis Function
RF : Random Forest
SDLC : Software Development Life Cycle
SVM : Support Vector Machine
TN : True Negatives
TP : True Positives
UCI : University of California Irvine
UML : Unified Modeling Language
VFI : Voting Feature Intervals

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

Software defect prediction involves building models to optimize performance standards using historical data. Most often, these prediction models may not completely capture the patterns present in the data, but useful estimates can nevertheless be obtained using these models. Although these approximations may not provide all of the necessary information, they can still aid in decision-making. In the context of software development, such data can be acquired from available software repositories, such as the National Aeronautics and Space Administration (NASA) and University of California Irvine (UCI) repositories (Dheeru & Karra Taniskidou, 2017). Further actions such as preprocessing should then be applied to these datasets to ensure that they are free from bias before they are used to build prediction models. By applying such prediction models during software testing, software companies can obtain a substantial amount of information regarding the number of defects in an existing software product and investigate the factors that influence the number of these defects. In the near future, if a new version of the existing software product is needed, it would be desirable to apply the information obtained from the existing software product to assist the software team in estimating the possible number of defects in the new version. However, to achieve this objective, a reliable and efficient means of preprocessing the data to achieve greater computing efficiency and better problem solving is needed, not only in the software engineering community but also in other domains.

Almost all aspects of current economic growth (for example, banking, social media, agriculture, health care, education, and transportation) involve the use of software tools that rely on data and models. These software products depend on models that must be trained using reliable datasets and are in high demand to aid organizations in their business activities. Therefore, it is important to develop a means of adequately preprocessing

the data used to train these models to achieve better model performance and overcome some of the identified issues associated with the existing datasets. Some of these issues include class imbalance, data heterogeneity, high skewness, privacy concerns, irrelevant and redundant features, continuous data, collinearity among metrics, missing values and noise in the data. If these datasets are not properly preprocessed, the prediction models will produce misleading results at both the class and method levels of the software. Therefore, this research attempts to address these inconsistencies associated with existing datasets through an optimal decision framework that is capable of making datasets suitable for use in defect prediction studies. An optimal decision in this context is a decision that can lead to an optimal result in every phase of the proposed data preprocessing framework. Furthermore, this study attempts to apply the preprocessed data obtained via the proposed optimal decision framework in training models that can predict the numbers of defects in a new version of a software product at both the class and method levels. Based on the optimal decisions made, optimal derived variables, namely, the defect density, defect velocity and defect introduction time, are identified during the implementation of the proposed data preprocessing technique. These derived variables are found to have some correlation with the number of defects in a software program.

Predicting the number of defects in software at both the class and method levels can help to ensure that software testing receives the necessary attention within the software engineering community. This is because defect prediction outcomes provide a software team with an idea of the possible number of defects in an upcoming product release prior to testing and thus can assist in the proper management of the available resources. For this purpose, considerable efforts are still required in demonstrating how to predict the number of software defects in a new product and in investigating the factors that influence the number of defects in software at both the class and method levels. To this end,

defect prediction provides advance information (i.e., prior information about the future) concerning the possible number of defects that are likely to be present in a new software product.

Information obtained from an existing software product can assist the software project team in identifying defect-prone modules in the software system. In terms of cost, such information can help the team to properly allocate the available resources to ensure their effective utilization in quality assurance activities in an effort to deliver a high-quality product to the end user. During software testing, additional effort may be required to identify and correct the errors found in a software system, and such effort will incur an additional cost. In such a case, the testing team may be faced with a lack of resources for carrying out the required tasks. Therefore, it would be helpful to have an idea of the possible number of defects likely to be present in a new version of a software product before testing begins. Such prior knowledge on the possible number of defects can play an essential role in assisting a software team in developing a reliable software product. Through software defect prediction, the team can correctly identify the most defect-prone software components and focus on reducing the number of defects likely to be present in those components. Although conducting a software defect prediction study with the aim of developing a reliable prediction model may require considerable amounts of time, energy, and money as well as skilled research personnel, defect prediction remains one of the most important means of improving software quality. To support the reliability and effectiveness of defect prediction in software engineering, a reliable software defect prediction technique is needed within the machine learning community.

To address this need for reliable software defect prediction models, previous studies have made significant contributions to defect prediction. However, despite these contributions, some controversies and limitations still exist regarding the existing defect prediction models.

For instance, the quality of the historical data applied in defect prediction studies has been a major concern, and related quality issues have led to numerous contradictory findings in the field of machine learning (Xu et al., 2016). Consequently, the existing literature lacks the following: (1) an approach for properly preprocessing the datasets applied in machine learning studies and (2) a means of predicting the number of defects in a new version of a software product. To address these gaps, this research focuses on (1) the quality of the data applied in machine learning studies and (2) the prediction of the number of defects present in a new version of a software product. The first phase of this research involves investigating the literature to uncover existing issues that affect the quality of the data applied in supervised machine learning studies; based on the findings, a framework is proposed that offers a means of preprocessing both the class- and method-level datasets applied in such studies. The second phase of this research also involves investigating the literature to uncover means of predicting the number of software defects in a new software version. On this basis, a technique is proposed for predicting the numbers of class- and method-level defects using derived attributes, such as the defect density, defect velocity and defect introduction time, obtained from the characteristics of the existing software product at the class and method levels. The proposed solution is also applicable to module level defects in a software program. Thus, this research addresses the need for a reliable approach to resolve the data quality issues in defect prediction studies and to obtain prior knowledge on the possible number of defects in a new software release. Such an approach will assist software companies in decision-making. More specifically, it will assist software testing teams in concentrating their efforts on the components with the highest predicted numbers of defects, thereby saving resources during software testing while producing a high-quality product with few defects, with the goal of remaining within the specified budget and schedule constraints while delivering a reliable product to the end user. In the

Software Development Life Cycle (SDLC), the testing phase typically consumes almost half of the software budget. Therefore, the proposed approach can also be helpful for reducing costs during the development of a new version of a software product, especially during the software testing phase because of the expensive nature of this phase, which requires more resources than other phases in the SDLC (Xu et al., 2016). The proposed approach can serve as a blueprint for program testing to enhance the effectiveness of software development activities.

The remainder of this section introduces the motivation for the research, the research background, the problem statement, the research objectives, details on the proposed solution and a statement on the research significance. Finally, the outline of the whole thesis is presented.

1.1 Background

Often, defects in software systems can cost various organizations enormous amounts of money and time in addition to causing disruption in human lives. These impacts can even destroy the entire prospects of an organization by hindering its operations and compromising its future. Many organizations have been greatly affected by financial loss as a result of software defects. For instance, in 2004 and 2005, software defects caused the UK Inland Revenue to issue tax-credit overpayments amounting to $3.45 billion. Between 2003 and 2004, the AT&T wireless service faced software defect problems that led to a revenue loss of $100 million. The United States Internal Revenue Service suffered an enormous loss of $4 billion in 1997 as a result of software defects. Furthermore, in 1994, the United States Federal Aviation Administration faced a loss of $2.6 billion linked to its advanced automation system. Also in 1994, Chemical Bank customers suffered a total loss of $15 million to their bank accounts as a result of software error (Charette, 2005). Such problems can be avoided by applying an effective defect prediction approach to improve

the quality of software products, thereby protecting organizations from financial losses and safeguarding the future of their businesses. However, developing a defect-free software system is a difficult task because it requires considerable effort, time, and money as well as a skilled software team. Although no software product is 100% free of defects, the number of defects in a software system can be minimized to ensure that the system will stand the test of time when it is deployed for its operational purpose. One of the means to achieve a reliable software product with few or almost no defects is through defect prediction. Defect prediction can provide a software team with prior information on the possible number of defects likely to be present in a new version of a software product. Such prior knowledge, if available, can enable the software team to restructure quality assurance procedures towards reducing or even preventing the predicted defects.

Thus, software defect prediction helps to ensure that testing and debugging remain in a fast-track mode by providing advance information on the number of faults that are likely to be found in a new program. As discussed above, both stakeholders and software companies may spend substantial resources repairing the damage caused by defects in software products. The need for software defect prediction in software engineering is driven by the importance of the proper use of available resources during software testing to ensure that quality software products are delivered to users. If the developer fails to address a problem early in the software development process, then that problem can become more complicated, with a corresponding increase in cost in later phases. Conversely, if the complexity is kept low, then the software can be more easily understood and modified during its life cycle (Parthipan et al., 2014).

To ensure that software stakeholders make good decisions about future programs and properly allocate resources to software projects, a prediction model must provide managers with practical and actionable outputs (Caglayan et al., 2015). Currently, software

companies apply a series of previously proposed techniques or custom approaches for predicting software defects. However, software companies continue to face unexpected faults that could have been revealed during the initial stages of development if suitable prediction practices had been implemented. Therefore, detecting fault-prone software components as early as possible can enable software experts to remain focused and to direct their resources towards addressing possible issues that may arise in a software system that is yet to be developed (Ma & Cukic, 2007). The metrics used early in the SDLC, such as the expected project size and speed, can play a significant role in project management by helping to reduce the defect density of a software product; specifically, these metrics can be used to determine whether increased quality monitoring is necessary during development. They can also be used in the planning of verification and validation activities if properly applied (Jiang et al., 2007). Furthermore, it is essential to consider the cost and benefits of predicting the number of software defects before program testing starts. If the quantitative cost of a prediction is not assessed first, then poor prediction results may be obtained. Furthermore, inappropriate resource allocation strategies can significantly increase the testing effort (Monden et al., 2013).

1.2 Problem statement

In line with the information provided in the previous section, this section presents the problems this research attempts to address. Notably, several previous empirical studies on software defect prediction have been conducted, for instance, by Brecher et al. (1996), Karayiannis et al. (1999) and Ghunaim & Dichter (2019). These studies have proposed defect prediction models with a focus on binary defect classification. However, despite the remarkable achievements reported in machine learning studies, the quality of the data applied in defect prediction studies has been a major concern, and related quality issues have led to numerous contradictory findings in machine learning studies (Xu et al., 2016).

In addition, a demonstrated approach for predicting the possible numbers of defects in software at both the class and method levels is lacking. Consequently, software companies still face challenges posed by software defects despite the remarkable achievements made in defect prediction studies. Thus, considerable efforts are still needed to develop an acceptable data preprocessing approach and a prediction model for software engineering that can address the challenges highlighted above. Moreover, it has yet to be demonstrated how optimal real-time metrics such as the defect velocity can be used not only to construct models to predict the possible number of software defects in a new product release but also to aid in the preprocessing of datasets applied in machine learning studies.

Obviously, studies on predicting the number of possible software defects in a new software version are lacking. In addition, the existing data preprocessing techniques, as reported, for instance, by Ryu et al. (2015), D. Zhang et al. (2015) and Bae & Yoon (2015), have not fully addressed the key data-related issues encountered in machine learning studies. Hence, an applicable data preprocessing framework as well as software defect prediction models are needed in the machine learning community to aid in the decision-making process. To address this need, this research presents a data preprocessing framework and proposes a method of predicting the number of software defects in a new software version. Finally, to ensure fair and accurate results in machine learning studies, this research presents a suitable data preprocessing approach to enable the accurate identification of the defects present in a dataset before the application of learning algorithms. The proper preprocessing of imbalanced datasets can assist in ascertaining the effects of data inconsistencies on classifier performance for the defect-free (majority) class, the defective (minority) class, or both (Chawla et al., 2004). Existing studies on defect prediction have been faced with various challenges and consequently have yet to demonstrate an acceptable approach for properly preprocessing the data applied in defect prediction studies. In addition,

this research also aims to show that the preprocessing of data is an essential phase of machine learning research. Specifically, data preprocessing should be carried out before the construction of learning algorithms to obtain reliable input datasets (Haixiang et al., 2017). Therefore, it is also important to gain an understanding of some of the issues associated with the existing datasets and propose means to address these data-related issues. These issues are discussed in Chapter 2. To clarify the above research problem, several research questions (RQs) have been formulated, as discussed in section 1.4.

1.3 Motivation

This section presents the motivation for this research work and a set of facts supporting this motivation. Data preprocessing is said to be the most challenging task in defect prediction studies and is a time-consuming exercise (Munková et al., 2013; F. Rahman et al., 2013; Gray et al., 2011). It is also an essential part of a defect prediction study (Agrawal & Menzies, 2018). Therefore, the importance of data preprocessing underscores the need for techniques that can ensure the reliability of the datasets used in defect prediction studies. Usually, the exact number of defective modules in an existing dataset is difficult to accurately determine, but through data preprocessing, accurate reporting of the details regarding the number of defects and inconsistencies in the data can be recorded. Consequently, researchers are encouraged to spend sufficient time on data preprocessing to ensure that their studies yield unbiased outcomes. Researchers also need to report in more detail the steps they follow when preprocessing the datasets used in their studies; such detailed reports may lead to knowledge discovery (Shepperd et al., 2013).

This study is motivated by the need to create a step-by-step and easy, but practical, optimal decision-making procedure for data preprocessing. The goal is to ensure that datasets are accurately cleaned before they are applied in machine learning studies. The proposed approach can be applied to various datasets used by different organizations and for

different purposes. First, the proposed approach ensures that every dataset undergoes the same preprocessing stages before it can be considered unbiased. The proposed approach includes a filter-based feature selection phase, in which only relevant features are selected from among the features present in the datasets. This phase ensures that redundant features are eliminated and that only features with high predictive power are selected. The predictive power of each selected feature is determined by applying a feature scoring method. Second, each dataset is visualized to determine the number of outliers it contains. Outliers are automatically removed from each dataset using widgets provided in Orange 2.7 (Demšar et al., 2013). This ensures that only clusterable data are used throughout the experiment. Thereafter, a unique identifier is assigned to each inlier in each dataset for further preprocessing; both target classes in the datasets are then thoroughly preprocessed with an input space created to recapture unreported faults. Via this proposed approach, a researcher can properly ensure that the datasets applied in a machine learning study are free from bias. In addition, a proper evaluation of the average performances of various classifiers on imbalanced data can be conducted.

The proposed decision-based data preprocessing approach is expected to enable some learning algorithms to independently and accurately learn from properly preprocessed imbalanced data while still generating suitable classification results, thus allowing them to maintain their average performances. In addition, these classifiers should be able to make optimal classification decisions based on training conducted using the preprocessed data and thus produce optimal outputs. Again, unlike for several previously reported studies, the motivation for this study is not to improve model performance. Rather, this research is motivated by the need to ensure that learning models are trained using reliable datasets to avoid bias and misclassification in defect prediction studies. If not addressed, misclassification often leads to consequences that incur high costs (Chawla,

2009; Van Hulse et al., 2007). To reduce costs while achieving unbiased results in a defect prediction study, it is advisable to apply clearly specified procedures such as those of the approach proposed by Doppa et al. (2014). Various data preprocessing methods and frameworks have been proposed to address imbalance issues, as reported by Saleem et al. (2014), Beckmann et al. (2011), Iliou et al. (2015), Gray et al. (2012), and Shepperd et al. (2013). In this regard, the researcher would like to acknowledge the contributions made by the existing frameworks that have been proposed to solve data imbalance problems, such as the framework of Menzies et al., the framework of Lessmann et al. (2008) and the framework of Q. Song et al. (2010). However, Wahono (2015) takes a contrary view, arguing that these frameworks produce misleading findings while addressing the issues associated with data inconsistencies. In addition, none of these studies has clearly demonstrated how to accurately pinpoint and record the identifiers of defective modules or the number of outliers present in a dataset or how to address the inconsistencies in the existing datasets. This research offers such an approach based on an optimal decision framework that can be replicated by other researchers. By so doing, better results can be achieved. A new approach is needed for data preprocessing to achieve greater computing efficiency and improve problem solving in organizations. When properly applied, the proposed approach is expected not only to overcome the challenges faced in assessing the average performances of learning algorithms but also to reduce the level of bias in the data applied in machine learning studies (Stefanowski, 2016).

In this study, it is hypothesized that when the proposed approach is used, not all learning algorithms will suffer significant degradation in performance when applied to data that are imbalanced at both the class and method levels. Hence, some classification algorithms are expected to be able to maintain their average performance while learning from imbalanced data by virtue of a decision-making approach that can enable these

algorithms to independently and accurately learn from imbalanced data such that they can generate suitable classification results. In addition, these classifiers should be able to make optimal classification decisions such that they will produce optimal outputs.

To reduce costs while achieving unbiased results in a defect prediction study, it is advisable to apply clearly specified procedures when performing such a study (Doppa et al., 2014). As mentioned earlier, it is necessary to predict the possible number of defects likely to be present in a new software product before testing such a product. To achieve this objective, the data applied in such a prediction study require proper cleaning to enable learning algorithms to generate accurate predictions, with no bias due to the quality of the data used in training and validating the prediction models.

At present, there is no generally accepted data preprocessing framework or technique for addressing the current data-related issues facing the machine learning community. In addition, studies on predicting the potential number of software defects likely to be present in a new version of a software product are lacking. Consequently, software companies continue to suffer from the effects of software defects despite the remarkable achievements made in defect prediction.

In this study, it is hypothesized that the choice of the metrics used for prediction also influences the prediction results. Based on this hypothesis, an optimal decision procedure is proposed to carefully select certain metrics that are mathematically derived based on specific relationships with the numbers of defects present at the class and method levels in a software program. These metrics are defined as follows: the defect density g is the ratio of the number of defects to the project size, and the defect velocity v is the rate of change in the defect position with respect to time t. Here, the time represents the defect introduction time. These metrics, which have not previously been applied in research on software defect prediction, are chosen as predictor variables. As reported by Felix &

Lee (2017b), the defect density of a software project depends on the rate at which defects occur, and these derived variables can be used to produce optimal results due to their correlation with the numbers of defects in software at both the class and method levels, which further indicates that the number of defects is a function of the defect acceleration. Another reason for selecting these metrics is to produce cost-effective practical outputs of software defect prediction for managers and development teams. Given the chosen metrics, a simple modeling technique is needed to evaluate their influence on the number of defects in an upcoming product release and to confirm their optimal output. As noted earlier, no previous study has considered these predictor variables for predicting the number of defects in a software product. Therefore, this study aims to investigate how the derived metrics selected through the proposed optimal decision procedure can be applied in software defect prediction during the software development process. The proposed modeling technique is based on the relationship between the number of defects and the defect density as a function of the defect acceleration. This technique is further explained in Chapters 3 and 4. These carefully selected metrics (for example, the defect velocity) can influence the outcome of any prediction study because these metrics show a strong positive correlation with the number of defects, thus making them optimal variables. Such careful selection has the potential to improve prediction outcomes (Laradji et al., 2015). Therefore, it is hypothesized that these carefully selected variables will enable significant and optimal outcomes in predicting the number of defects in a new product release as well as in cleaning both class- and method-level datasets applied in supervised machine learning studies.

Finally, the motivation for this research can be summarized as follows:

1. The need to gain an understanding and in-depth knowledge of the existing defect prediction models used in supervised machine learning studies.
2. The need to contribute to the existing data preprocessing techniques to enhance the

quality of the data applied in defect prediction studies.

3. The need to determine the average accuracy of the learning algorithms applied in defect prediction.

4. The need to investigate the influence of the chosen derived variables, namely, the defect velocity, defect density and defect introduction time, and their correlations with the number of defects in a software product at the class and method levels.

5. The need to apply the information obtained from a current software product to predict the number of software defects in a new version of that software product.

1.4 Research questions

Now that the problem that this study is attempting to address has been described, the following research questions are formulated as follows.

RQ1. How do researchers perform and report the techniques applied during data preprocessing?

RQ1.1 Do the reported data preprocessing techniques satisfactorily address data inconsistency issues?

RQ1.2 Does a generally accepted data preprocessing technique exist?

RQ1.3 To what extent do the existing data preprocessing techniques offer suitable solutions for data preprocessing?

RQ2. How is the performance of existing prediction models?

RQ2.1 Do classifiers exhibit degradation in their average performance as a result of imbalanced data?

RQ2.2 Does data imbalance result in unfair and inaccurate evaluation outcomes?

RQ2.3 Do classification algorithms learn independently from imbalanced data?

RQ3. How do supervised classification algorithms perform on average when applied to raw and preprocessed imbalanced data?

RQ3.1 Do certain supervised classification algorithms outperform others on average when applied to imbalanced data?

RQ3.2 How does class imbalance result in biased outcomes from supervised learning algorithms?

RQ3.3 To what extent do supervised classification algorithms maintain their average information entropy when applied to imbalanced data?

RQ4. How do the derived optimal predictor variables aid in data preprocessing and influence the number of defects in a software project?

RQ4.1 How does the defect velocity impact the number of software defects?

RQ4.2 Does the defect density influence the number of software defects?

RQ4.3 Does the defect introduction time impact the number of software defects?

To provide answers to the above research questions, the objectives of this research are formulated in the following section. The formulated research questions are aligned with the research objectives, which are explicitly defined in the next section.

1.5 Research objectives

It is very important to provide the software engineering community with a cost-effective but efficient framework that can enhance the quality of the data applied in various supervised machine learning studies. Such a framework could also serve as a guide for researchers when carrying out data preprocessing. In addition, it is important to consider numerous evaluation metrics when determining the performance of the learning algorithms applied in machine learning studies. This is because only a few metrics may not provide a sufficiently detailed assessment of the algorithm performance. Learning algorithms can be properly assessed through an optimal decision approach that accounts for numerous performance metrics. The results thus obtained can also serve as an acceptable measure of the average classifier performance, unlike an assessment using fewer evolution metrics.

There is also a need to identify an ideal optimal variable that can assist both researchers and software practitioners in software testing. Such an optimal variable, when derived, should be capable of revealing the rate at which software defects occur in a software project. As presented above, the existing research gives rise to numerous research questions that need to be addressed in the current study. Therefore, this study focuses on providing solutions that attempt to address these research questions as well as the problems at hand. The objectives of this research are summarized as follows:

RO1. To investigate existing issues associated with the highly imbalanced and erroneous class- and method-level datasets applied in supervised machine learning.

RO2. To investigate the performance of existing prediction models while identifying the existing approaches and techniques applied during data preprocessing to address the identified data-related issues at the class and method levels.

RO3. To propose a framework for preprocessing as well as a technique for predicting the numbers of class- and method-level defects in a new software version using derived optimal variables.

RO4. To evaluate the proposed data preprocessing framework as well as the class- and method-level defect prediction technique based on the accuracy, precision, standard error, information score, recall and entropy (level of uncertainty).

1.6 Mapping of research objectives and questions

To achieve these research objectives, the research questions are further clarified as follows:

RQ1. One of the objectives of this research is to propose a cost-effective and efficient framework for improving the quality of the data applied in machine learning studies. Therefore, it is necessary to investigate how other researchers perform data preprocessing

and how they report their approaches. There is a need to properly aggregate all clear evidence on the methods and techniques applied when preprocessing the data used in machine learning studies. Therefore, RQ1 is formulated to probe this evidence and ensure proper reporting of the aggregated findings. The results achieved by investigating the available evidence regarding the applied data preprocessing techniques and methods can lead to meaningful answers to this research question.

RQ1.1. Data quality issues within the machine learning community have attracted considerable attention. Although the existing research studies have proposed various techniques that can be implemented to address issues of data quality and inconsistency, these issues ultimately remain unsolved. Consequently, the outcomes of several research studies have been challenged as a result of the inconsistent nature of the data applied; it can thus be inferred that as a result of data quality issues, the results presented by most research studies are questionable. Therefore, it is necessary to investigate whether the reported data preprocessing techniques satisfactorily address data inconsistency issues. Accordingly, RQ1.1 is formulated to investigate data inconsistency issues and possible measures to address them.

RQ1.2. Obviously, there is an urgent need for an accepted data preprocessing framework and techniques that can serve as a benchmark for data cleaning in machine learning studies. Such a framework and techniques could also serve as a blueprint for the preprocessing of data within a machine learning environment. Despite the remarkable achievements that have been accomplished in machine learning studies, a generally accepted data preprocessing framework and related techniques are lacking. One objective of this research is to address this lack. In line with this objective, RQ1.2 is formulated to properly investigate the existing literature to uncover whether any generally accepted data preprocessing framework or technique exists.

RQ1.3. This research question focuses on identifying all necessary and available data preprocessing techniques in the literature and the corresponding data issues they address. This research question is expected to aid in the collection of a convenient set of data preprocessing techniques that can be applied to address data issues while performing data cleaning.

RQ2. Another objective of this research is to investigate the performance of the existing prediction models. This research question is formulated to assist in gaining a broader understanding of the work being carried out by other researchers with regard to the performance and accuracy of the existing defect prediction models. After a proper investigation of the above issues, this research question will enable us to draw logical conclusions regarding the performance and accuracy of the existing learning algorithms. RQ2 is divided into three subquestions: RQ2.1, RQ2.2, and RQ2.3.

RQ2.1. This research question is formulated to drive an in-depth investigation of the characteristics of the learning algorithms investigated as part of RQ2. Furthermore, RQ2.1 enables the accurate assessment of the average performances of these learning algorithms when applied to imbalanced data. The results achieved in the course of this investigation can yield logical conclusions regarding the average performances of the learning algorithms applied in various machine learning studies. The related outcomes can also lead to a further probe of the average performances of classification algorithms when applied to imbalanced data.

RQ2.2. This research question is formulated specifically to investigate whether the classification outcomes of learning algorithms are biased as a result of data imbalance. Since the outcomes and performance of a prediction model depend on the quality of the data applied in model training, there is a need to conduct an in-depth investigation of the learning outcomes and behaviors of classification models trained on imbalanced data.

RQ2.3. The aim of this research question is to encourage researchers performing machine learning studies to conduct a proper investigation of the self-reliance of learning algorithms when learning from imbalanced data. Although learning algorithms are typically expected to learn accurately and independently, there could be instances in which certain factors may hinder this capability; in particular, such a factor could result from inconsistencies in the dataset. Corresponding findings regarding the independence of learning algorithms can lead to rational conclusions concerning this issue.

RQ3. This research question guides an attempt to determine the average performance of classification algorithms when applied to imbalanced data. Although several studies have argued that the learning of classification algorithms may be biased as a result of data imbalance, others take a different view. Here, it is argued that class imbalance itself does not have a significant effect on the outcomes and average performance of learning algorithms; rather, what should be of major concern is the process used to clean the datasets (that is, data preprocessing). To strengthen this argument while providing answers to this specific research question, an empirical investigation is conducted to determine whether class imbalance has a major impact on the performance of learning algorithms.

RQ3.1. This research question is formulated to investigate whether certain classification algorithms outperform others in the same learning environment when trained using imbalanced data. The accuracy of a classification algorithm depends on the quality of the data used for model training. However, more broadly, there is a need to properly investigate whether certain classification algorithms outperform others when applied to imbalanced datasets. The findings from this investigation will assist in the reporting of the performances of certain classification algorithms compared to others in the same classification environment. The findings can also yield logical conclusions regarding the individual performances of different learning algorithms.

RQ3.2. Although some previous studies have reported that classification algorithms are biased due to the imbalanced nature of the datasets applied in machine learning studies, it is necessary to properly investigate these claims. RQ 3.2 is therefore formulated to guide this investigation and provide clear evidence regarding the bias, if any, of learning algorithms applied to imbalanced data.

RQ3.3. The ability to maintain the average information entropy among the learning algorithms also relies on the quality of the data applied in training these algorithms. However, if some learning algorithms can maintain their average performance when applied to imbalanced data, such algorithms can also be expected to maintain their average information entropies when applied to imbalanced data. It is necessary to accurately ascertain whether some classification algorithms can maintain their average information entropies when learning from imbalanced data. Therefore, RQ3.3 is formulated to support a proper investigation regarding the average information entropies of learning algorithms. The resulting findings can assist in inferring the information entropies of various learning algorithms.

RQ4. To achieve the objectives of the current study, it is necessary to investigate whether and how the derived variables obtained through the approach proposed to address this research problem influence the number of defects in a software project. Hence, RQ4 is formulated to aid in this assessment. The corresponding findings will help to establish facts regarding the influence of these derived variables and their relationship with the number of defects in a software project.

RQ4.1. This research question is formulated to investigate how the defect velocity can impact the number of software defects. The defect velocity, as one of the derived variables of interest, is investigated to determine its relationship with the number of defects in a software project.

RQ4.2. This research question is formulated to support the investigation of the impact of the defect density on the number of software defects. A clear relationship exists between the defect density and the number of defects in a software project, and this relationship must be properly investigated. Thus, we formulate RQ4.2 to aid in this investigation. The findings will assist in reporting the relationship between the defect density and the number of software defects.

RQ4.3. As a software project moves from one phase of the SDLC to another, opportunities may arise for a defect from one phase to be carried through to another along with the project transition. The times at which these defect transitions occur may impact the resulting number of defects. Therefore, it is also necessary to investigate whether the defect introduction time influences the number of software defects. RQ4.3 is formulated to guide this investigation.

To clearly show the relationship between the research objectives and the research questions, it is convenient to present a table that shows the mapping between the research objectives and the research questions as well as the phases of the research methodology in which these research objectives and questions are achieved and addressed, respectively. Table 1.1 shows this mapping relationship.

Table 1.1: Mapping of research objectives and questions with the corresponding phases of the research methodology

Research Objectives	Main Research Questions	Subquestions	Research Methodology Phases
RO1	RQ1	RQ1.1	
		RQ1.2	Literature Review
		RQ1.3	
RO2	RQ2	RQ2.1	
		RQ2.2	Proposed Approach
		RQ2.3	
RO3	RQ3	RQ3.1	
		RQ3.2	Evaluation and Analysis
		RQ3.3	
RO4	RQ4	RQ4.1	
		RQ4.2	Evaluation and Analysis
		RQ4.3	

1.7 Proposed solution

One of the expected outcomes of this research work is to propose a cost-effective and efficient data preprocessing framework to enhance the quality of the data applied in supervised machine learning studies. To achieve this aim, it is necessary to make an optimal decision that can lead to the actualization of the above objective. Such an optimal decision approach is also expected to be advantageous to the machine learning community. First, to address the key data-related issues in software defect prediction, a framework for data preprocessing is proposed in this study to address some of the challenges associated with the relevant datasets. An optimal decision procedure is used to design this data preprocessing framework, resulting in a prime advantage in ensuring effective data preprocessing. To evaluate the effectiveness of the proposed framework, this study investigates the average classification performances of several selected state-of-the-art classifiers across multiple projects with data imbalances between the defective and defect-free classes.

Second, to demonstrate a method of predicting the numbers of class- and method-level defects in an upcoming product release, the proposed framework is used as a basis to present such a method that relies on predictor variables derived from the defect acceleration, namely, the defect density, defect velocity and defect introduction time. These derived variables are then used to construct models that can predict the number of defects in a new product release.

1.8 Research scope

As defined by the standards set by the Software Engineering Body of Knowledge (SWEBOK) (Bourque et al., 2014) and the international standard ISO/IEC TR 19759:2005 (Seidman, 2008), the five main phases of the SDLC are requirements engineering, software design, software construction, software testing, and software maintenance. This research aims to support the software testing phase.

Software testing is an important phase of the SDLC that is carried out to find defects in a software program under development. For this reason, the ability to predict the presence of defects at both the class and method levels prior to testing will greatly assist the software team in optimizing their test efforts to improve software quality. To achieve reliable defect prediction outcomes, this research is driven by possibilities within the domain of supervised machine learning, which consists of regression and classification. Hence, the scope of this research revolves around these two phases of supervised machine learning. Figure 1.1 presents the components of the supervised machine learning process, which is the scope of this research. These components include regression, classification, the input datasets, the training set, the test set, the learning algorithms/predictive models and the output.

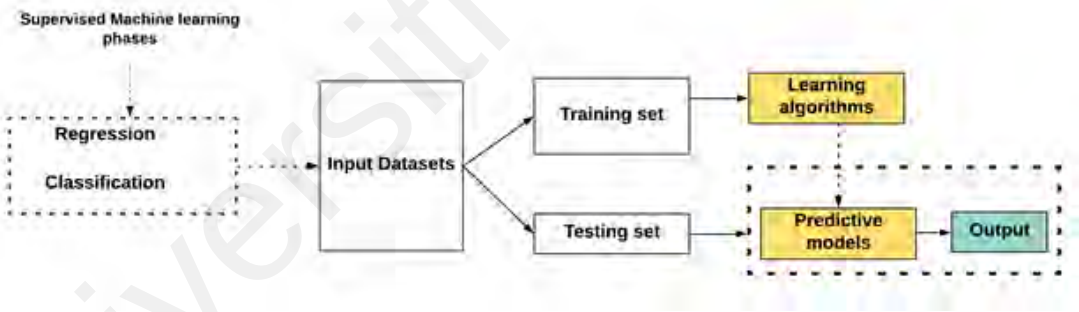


Figure 1.1: Scope of research

Regression: Regression analysis is an analytical approach for predicting the relationships between dependent and independent variables (Montgomery et al., 2012). Every supervised machine learning task includes some aspect of regression; in this research, linear regression models are applied to achieve certain prediction tasks, for instance, to predict the numbers of class- and method-level software defects. For the sake of achieving the objectives of this research, such regression models enable the researcher to draw

meaningful conclusions from the available evidence, allowing the possible number of defects in a new software version to be predicted. More details on regression models are provided in Chapters 3 and 4.

Classification: Supervised machine learning classification depends on the ability of models to accurately identify the target class to which a given element of a group belongs. In this context, an element refers to an observation based on the training data. For instance, a dataset may contain sets of observations with known classes, categories or labels, which may be defective or defect-free. However, if a new set of data that is independent of the initial training set is required to be classified into the defective and defect-free categories, a classification algorithm will be required to perform this task in the context of supervised machine learning. The behavior of such a learning algorithm for this task is determined by the initial data used to train it. Hence, one objective of this research is to implement such algorithms to accurately determine how they perform when trained on preprocessed training datasets, in comparison with their performance when trained on raw or unprocessed datasets. Further details regarding classification algorithms and their performances are provided in Chapters 3 and 4, respectively.

Input datasets: In this research, an essential concern is the quality of the input datasets. The quality of the datasets applied in machine learning studies cannot be ascertained because existing studies have not fully reported how such datasets are cleaned or preprocessed. Therefore, further studies are required not only to summarize the relevant data quality issues but also to clarify the detailed approaches applied to address the inconsistencies associated with existing datasets applied in machine learning studies. Furthermore, it is important to note that the outcomes of prediction models depend on the quality of the data used (Montgomery et al., 2012). Both classification and regression models for supervised machine learning depend solely on high-quality data to ensure accurate performance. As

such, the datasets applied in machine learning studies must be free from bias to avoid misleading research findings. One of the objectives of this research is to propose a practical and cost-effective framework to aid in data preprocessing for machine learning studies.

Test dataset: Although the test data are different from the training data, as much attention should be accorded to the quality of the test data as to the quality of the training data because the outcomes of learning algorithms are usually tested or verified using separate test datasets. Learning algorithms are trained using unbiased training datasets; then, after training, test datasets are applied to ascertain the algorithm performance. The datasets used for training and testing must be independent of each other to avoid model overfitting, which can result in misleading research findings. To avoid this, it is important to separate the training data and the test data. Such separation is also applied in this research to ensure accurate research findings.

Learning algorithms: In machine learning, learning algorithms are the predictive models that are responsible for performing prediction tasks based on the training received. As noted earlier, for these learning algorithms to produce reliable results, the quality of the datasets used is the fundamental influencing factor. The list of learning algorithms applied in this research is presented in Chapter 5.

Output: For a supervised machine learning process to be complete, an output is expected. Since supervised machine learning models rely on input data, the output is usually determined based on test data, as discussed in the previous subsection. Therefore, the output of a supervised machine learning process is the prediction result obtained based on the quality of the input data, which is validated using test data.

1.9 Research significance

There are several concerns regarding the quality of the datasets applied in machine learning studies, as well as the lack of a demonstrated approach for predicting the numbers

of defects in software at both the class and method levels. This research therefore offers a solution to address these data quality issues as well as a demonstration of how the possible number of defects in a new software version can be determined prior to testing. In this way, adequately preprocessed datasets suitable for predicting the number of defects in a new version of software at both class and method levels can be obtained, and the factors that influence the number of defects in a software product can be identified during data preprocessing.

In this study, an investigation of the existing literature confirms that the existing datasets are inconsistent and erroneous in nature. Given this scenario, an optimal decision framework is proposed for implementation in machine learning studies for data preprocessing. In addition, there is a need to properly evaluate the average performance of the learning algorithms applied in machine learning studies; hence, numerous evaluation metrics are applied in this study to evaluate the average classification performance across the different target classes in a dataset, i.e., the majority and minority classes. Such an evaluation will produce more reliable and accurate results regarding the performance of the learning algorithms applied in machine learning studies. Through this approach, software teams can properly allocate the available resources for software projects. By means of the proposed optimal decision framework, several predictor variables are identified that can be used to construct regression models that can then be used to predict the number of software defects. These new predictor variables will exert a meaningful influence on the number of defects present in a new software release.

The significance of this research can be summarized as follows:

1. Provide a cost-effective means of preprocessing the class- and method-level datasets applied in machine learning studies, which can be applied not only to software-related data but also to data from any sector of the economy.

2. Provide cost-effective prediction models.

3. Provide an approach that can assist in analyzing complex data and delivering faster and more accurate results.

4. Provide techniques that can assist software managers in making better decisions.

5. Provide a framework that can serve as a practical tool for software testing teams.

6. Enable better resource allocation by software companies.

7. Open up new directions of research.

If an acceptable technique for data preprocessing and defect prediction can be developed, it will provide a practical means of relieving the worries faced when building models for software defect prediction and also bring an end to the concerns regarding the quality of the data applied in defect prediction studies.

1.10 Thesis outline

The remainder of this thesis is organized as follows:

Chapter 2 presents a literature review. Chapter 3 describes the methodology. The proposed approach is presented in Chapter 4. Chapter 5 explains the evaluations of and experimentation with the proposed modeling approach. Chapter 6 reports the results and offers relevant discussions. Chapter 7 presents the conclusions and directions for future work. The organization of the thesis is summarized in Figure 1.2.

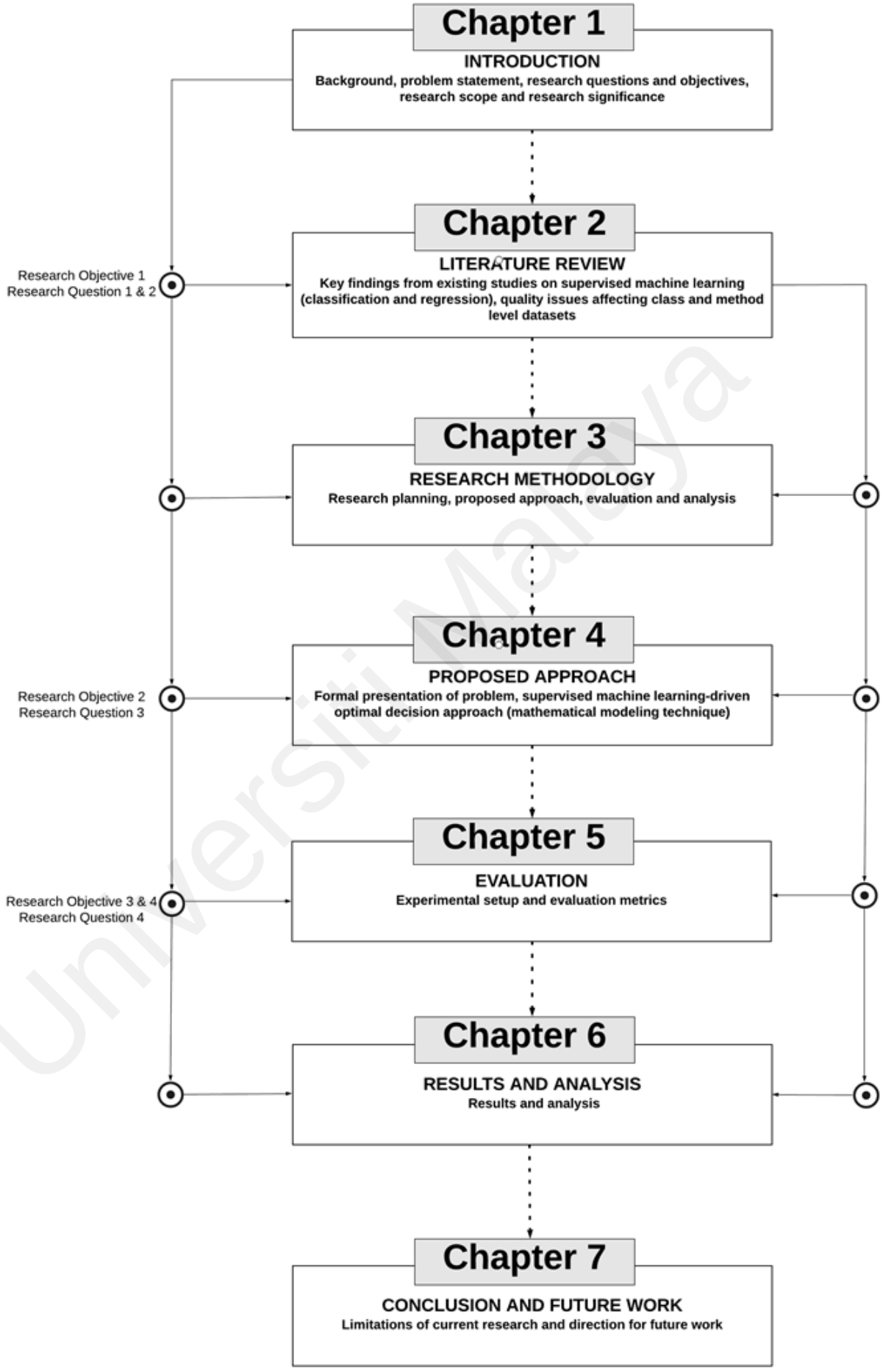


Figure 1.2: Thesis organization

CHAPTER 2: LITERATURE REVIEW

This chapter presents previous work related to data preprocessing, binary defect prediction and classification models as well as work done on predicting the numbers of defects in a new software version at both the class and method levels. A rigorous literature review is undertaken to provide a clear understanding of the approaches to data preprocessing and the performance of the existing learning algorithms as well as to investigate the approaches applied to improve algorithm performance. In addition, to aggregate clear evidence regarding the data preprocessing techniques available in the existing literature, studies related to data preprocessing are investigated to reveal several relevant data-related issues as well as the approaches applied to clean the datasets before using them in supervised machine learning studies. Proper reporting of data cleaning procedures is needed to possibly reveal new knowledge that will enhance the learning outcomes in machine learning studies. Through the rigorous review performed in this research, lists of several relevant studies that have attempted to address some of the identified data-related issues are presented. Finally, the existing literature is investigated to determine whether there is any existing approach for determining the number of defects in a new software product release that emphasizes the optimal real-time variables that influence the number of defects found in software products.

The lack of a generally acceptable means of preprocessing data applied in machine learning studies in the existing literature remains a crucial concern. Data preprocessing is performed before the construction of learning models to prepare reliable input datasets (Haixiang et al., 2017). As an unavoidable phase of machine learning studies, data preprocessing requires the understanding, identification and specification of data-related issues as well as a knowledge-based approach that can be used to address these issues

and, thus, make data more reliable for use in machine learning studies (S. Zhang et al., 2003). Notably, there is clear evidence that data preprocessing can impact the predictive performance of learning algorithms (Crone et al., 2006). If data are properly preprocessed, researchers can accurately identify and report the number of defects found in their data, thereby making their datasets more suitable for enabling learning models to learn independently and accurately from unbiased data to produce reliable results. The credibility of defect prediction is one factor influencing data quality (Hall et al., 2012; Liebchen & Shepperd, 2016; Hosseini et al., 2016; Hosseini, Turhan, & Mäntylä, 2017). In addition, the performances of the prediction models themselves consistently depend on the quality of the data used to train them. Such models are trained using historical data to identify defect-prone software modules (Tantithamthavorn et al., 2018).

Fan & Bifet (2013) noted that the quality of the data that can aid in decision-making remains a challenge and that such data quality issues have led to controversial conclusions in software engineering. Therefore, to address these data quality issues, the researcher has identified a list of challenges associated with existing datasets that require urgent attention in the machine learning community. The identified data-related issues are discussed in the following section.

2.1 Data quality issues in supervised machine learning

There are several concerns regarding the quality of the data applied in defect prediction studies. Obviously, the use of erroneous training datasets can lead to poor models and biased prediction results, and any research findings based on such erroneous data will continue to receive criticisms. Thus, the use of erroneous datasets in defect prediction studies has become a cause of great concern. The identified issues related to the existing datasets include class imbalance, data heterogeneity, high skewness, privacy concerns, irrelevant and redundant features, continuous data, collinearity among metrics and noise in

the data. Hosseini, Turhan, & Gunarathna (2017) also confirmed that these issues remain major challenges in software defect prediction studies and must be addressed without delay.

2.1.1 Class imbalance

One of the major characteristics of the existing datasets is class imbalance, which can be defined as a difference in data volume between the categories into which data are to be classified (Chawla, 2009). Class imbalance can also be regarded as a situation in which the number of defective (minority) instances and the number of defect-free (majority) instances in a project are not equal. Class imbalance in a dataset can affect the performance of classification algorithms, potentially leading to unfair and inaccurate evaluation outcomes when traditional assessment techniques are applied (Galar et al., 2012; H. He & Ma, 2013; Stefanowski, 2016; Van Hulse et al., 2007). Although approaches for overcoming the challenges posed by data imbalance have been proposed in many previous studies, such as those of G. E. Batista et al. (2004), Lusa et al. (2010), Błaszczyński & Stefanowski (2015), Branco et al. (2015), Chawla (2009), Galar et al. (2012), H. He & Garcia (2009), H. He & Ma (2013), Japkowicz & Stephen (2002), Stefanowski (2016), Y. Sun et al. (2009), Van Hulse et al. (2007), Schapire (1990), Wolpert (1992), Breiman (1996), Breiman (2001), Herbold (2013), Y. Ma et al. (2012), V. García et al. (2012), P. Yang et al. (2009), H. Cao et al. (2013), Q. Li et al. (2013), Anand et al. (2010), and Galar et al. (2013), the issue of imbalanced data in machine learning studies still remains unresolved. In some of the primary studies selected in this literature review, such as those of L. Chen et al. (2015), Ryu et al. (2016), Kamei et al. (2016), Ryu et al. (2017), Nekooeimehr & Lai-Yuen (2016), H. Cao et al. (2014), J. Li et al. (2017), Yun et al. (2016), D. Zhang et al. (2015), Zhai et al. (2017), Chetchotsak et al. (2015), Menardi & Torelli (2014), Das et al. (2015), Kumar et al. (2014), D'Addabbo & Maglietta (2015), Z. Sun et al. (2015), Ha & Lee (2016), Krawczyk et al. (2014), Cateni et al. (2014), Dubey et al. (2014), Díez-Pastor et al. (2015), P. Cao

et al. (2014), Sáez et al. (2015), J. Song et al. (2016), and Jian et al. (2016), resampling techniques, which involve repeatedly drawing samples from a given dataset, have been applied to address this problem. In addition, reweighting has been applied in previous studies, such as those of Ryu et al. (2016), X. Jing et al. (2015), and Ryu et al. (2017), to address the imbalance problem. Ryu et al. (2015) applied a selective learning technique to address the issue of imbalanced data.

Data imbalance appears to be a natural phenomenon. Therefore, software defect prediction studies should focus more on data preprocessing than on balancing existing datasets. This is because the use of well-cleaned and preprocessed data will lead to reliable prediction outcomes, whereas using noisy and erroneous data will result in biased outcomes (Tantithamthavorn et al., 2015; Saleem et al., 2014; Petrić et al., 2016; Iliou et al., 2015; Gupta & Gupta, 2017; P. He et al., 2015). Several studies have proposed methods of overcoming the class imbalance problem (G. E. Batista et al., 2004; Lusa et al., 2010; Błaszczyński & Stefanowski, 2015; Branco et al., 2015; Chawla, 2009; Galar et al., 2012; H. He & Garcia, 2009; H. He & Ma, 2013; Japkowicz & Stephen, 2002; Stefanowski, 2016; Y. Sun et al., 2009; Van Hulse et al., 2007). However, binary defect classification studies continue to grapple with issues related to imbalanced data. In these studies, the classifiers are typically biased toward the majority class and fail to accurately classify the minority class (H. He & Ma, 2013; Stefanowski, 2016; Krawczyk et al., 2014; Weiss & Provost, 2003).

To achieve better classification on imbalanced data, Ohsaki et al. (2017) proposed a confusion-matrix-based kernel logistic regression (LR) method with the objective of increasing the harmonic means of certain evaluation criteria, namely, the sensitivity, specificity, positive predictive value and negative predictive value, in a well-balanced manner.

Several studies have shown that certain ensemble techniques, such as bagging and boosting, can also be used to solve class imbalance problems (Schapire, 1990; Wolpert, 1992; Breiman, 1996, 2001). In addition, cost-sensitive methods can be useful when attempting to reduce the classification bias of a single classifier (Tang et al., 2009; S. Wang et al., 2012; X.-Y. Liu & Zhou, 2006; Castro & Braga, 2013; C. Zhang et al., 2016; Krawczyk, 2016). Several studies have also proposed ensemble methods for improving how imbalanced data are addressed (Krawczyk, 2016; Seiffert et al., 2010; Wallace et al., 2011; P. Yang et al., 2014; Nikulin et al., 2009; S. Wang & Yao, 2013; Z. Sun et al., 2015).

While investigating methods for improving the classification accuracy of learning algorithms, Galar et al. (2012) and H. He & Ma (2013) reported that the performances of these algorithms can be improved by considering combinations of individual metrics when confronting challenges related to imbalanced data. Meanwhile, G. E. Batista et al. (2004) performed an extensive experimental assessment of the performance of learning algorithms when applied to imbalanced data and reported that class imbalance does not completely prevent the successful application of learning algorithms. This study confirms the findings of G. E. Batista et al. (2004) in terms of the impact of data imbalance on the performance of learning algorithms. In binary classification studies, more than a few evaluation metrics are required to fully characterize the performance of each tested classifier. Therefore, a method of adequately preprocessing imbalanced datasets as well as evaluating the average performances of classifiers on imbalanced data is needed. Such a method will assist in characterizing the overall behavior of each classifier and in guiding the selection of appropriate classifiers for particular applications. As mentioned by Sommerville (2004), one of the philosophies of software engineering is to do it right the first time to support cost-effective decision-making.

Software project datasets are prone to class imbalance as a result of the emergence of

defects; however, several researchers have proposed techniques to solve such imbalance problems. For instance, class imbalance affects the performance of classification models (G. Batista et al., 2012; Estabrooks et al., 2004; P. He et al., 2015; Japkowicz & Stephen, 2002; Mao et al., 2017; Qiao et al., 2017; Rodriguez et al., 2014; Soda, 2011; Tang et al., 2009). To address this issue, Z.-W. Zhang et al. (2017) applied Laplacian score sampling for the selection of training data. Their results confirmed that Laplacian score sampling can improve data quality by addressing the imbalance issue in datasets. Another approach for improving the quality of the data used in prediction studies is through a relational association rule that can provide detailed information regarding the attributes of datasets (Czibula et al., 2014).

The early detection of errors in datasets can enable the improvement of data quality. Accordingly, Gray et al. (2011) and Gray et al. (2012) proposed several steps through which data preprocessing can be improved by identifying errors in datasets early. Menzies et al. (2007) noted that it is essential to consider the attributes of the datasets of interest when building a prediction model. However, these attributes can contain errors that contribute to data imbalance and can impact the performance of prediction models. One approach for overcoming the imbalanced nature of datasets was reported by X.-L. Yang et al. (2017). That approach can identify high-impact errors and is distinct from the traditional approach, in which only binary classification is considered and the effect of data errors is disregarded. Shepperd et al. (2013) outlined important rules for removing erroneous data from the National Aeronautics and Space Administration (NASA) datasets. Their effort was remarkable, and they also encouraged researchers to make available the sources of the data applied in their research studies, spend sufficient time in preprocessing their datasets and report the detailed steps followed during preprocessing.

In addition to these outlined rules for data preprocessing, Petrić et al. (2016) identified

additional rules that can be applied when removing erroneous data. However, despite the application of these rules during data preprocessing, datasets are still found to contain errors following data cleaning. In particular, datasets obtained from repositories are prone to errors and noise because of impurities in these datasets. These impurities can lead to inaccurate prediction outcomes and, consequently, misleading results (Ramya et al., 2012). Rodriguez et al. (2014) compared different data mining approaches, including sampling and cost-sensitive, ensemble and hybrid approaches. That study relied on the datasets cleaned and preprocessed by Shepperd et al. and concluded that the compared approaches could enhance the classification accuracy for the minority class.

Critical comment: *In the attempt to balance the classes and methods in a dataset, new defects may be introduced, and the original data may be altered. To avoid this undesirable situation, the researcher proposes and recommends a novel approach that can be applied in preprocessing the existing highly imbalanced class- and method-level datasets. This proposed approach accounts for the highly imbalanced nature of the datasets, which ultimately does not have an adverse effect on the average performance of learning algorithms.*

2.1.2 Data heterogeneity

Variance in the attributes of existing datasets can lead to differences in the outcomes of different machine learning studies. This is because data samples from different sources that are applied in such studies exhibit different characteristics. The extent of the variations among data sources and characteristics can lead to greater or lesser heterogeneity in datasets. This is because data from different sources may exhibit different characteristics, may be expressed using different representations, may be stored in incompatible formats, and may otherwise be both inconsistent and interrelated (Che et al., 2013).

In addition, most software projects are characterized by heterogeneity, with dissimilar

metrics spread throughout the datasets that constitute these projects (Canfora et al., 2015). Furthermore, as reported by Yu & Mishra (2012), data heterogeneity can be affected by various contextual factors, such as the domain, project size and programming language used. Notably, due to the more homogeneous nature of the data involved, some machine learning algorithms are likely to perform well when applied for within-project defect prediction (Hosseini, Turhan, & Gunarathna, 2017). However, Nam et al. (2013) confirmed that despite their success in within-project defect prediction due to the nature of the data, such algorithms may not be applicable for cross-project defect prediction. To address the issue of data heterogeneity, data transformation techniques have been applied in previous studies, such as those of Watanabe et al. (2008), Nam et al. (2013), Camargo Cruz & Ochimizu (2009), F. Zhang et al. (2016), and X. Jing et al. (2015). Filtering has also been applied to address data heterogeneity, as reported by Turhan et al. (2013), Turhan et al. (2009), Z. He et al. (2012), Jureczko & Madeyski (2010), Y. Ma et al. (2012), L. Chen et al. (2015), Peters et al. (2015), X. Jing et al. (2015), Ryu et al. (2015), and Ryu et al. (2017). Data normalization is another approach applied for this purpose, as reported by Uchigaki et al. (2012), Y. Liu et al. (2010), Herbold (2013), Nam et al. (2013), Panichella et al. (2014), Canfora et al. (2015), Ryu et al. (2016), X. Jing et al. (2015), and Ryu et al. (2017). Nam et al. (2017) reported the use of metric matching to address data heterogeneity, whereas Canfora et al. (2015) and Herbold (2013) addressed this issue by using clustering techniques.

Critical comment: *New defects may be introduced, and the original data may be altered, in the application of certain remedies such as data normalization L. Chen et al. (2015). In addition, both relevant and irrelevant metrics in the data may be matched when attempting to match metrics in datasets. These relevant and irrelevant metrics may be clustered. To address these concerns, the researcher proposes a practical approach*

for avoiding the introduction of further defects into the datasets as well as a means of preventing the matching of both relevant and irrelevant features in the datasets.

2.1.3 High skewness

Most of the time, the minority and majority classes in a dataset are not normally distributed, thus leading to data skewness. Data can be skewed either positively or negatively. Positively skewed data indicate a disadvantage to the right of the normal distribution line, whereas a negatively skewed data distribution shows a disadvantage to the left of the normal distribution line.

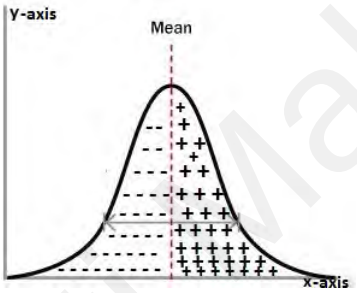


Figure 2.1: Normal skewness

Figures 2.1, 2.2 and 2.3 present normal, positively skewed and negatively skewed data distributions, respectively. The x-axis represents the frequency of the data, while the y-axis represents the data points for both the positive and negative classes.

As shown in Figure 2.1, a normal curve exhibits an almost perfectly symmetrical distribution. Reducing the data skewness, i.e., making the distribution more normal, can improve the performance of learning algorithms (Hosseini et al., 2016).

Figure 2.1 presents a normal distribution of data, in which the positive and negative classes are evenly distributed. In such a scenario, classification algorithms will generally show no bias when classifying both the positive and negative classes. In contrast, Figures 2.2 and 2.3 present data with positive and negative skewness, respectively, which may cause classification algorithms to be biased towards the majority class.

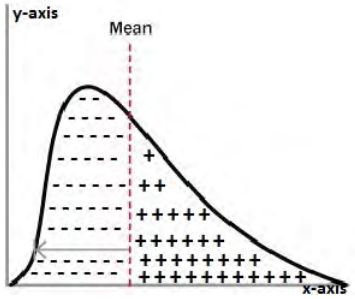


Figure 2.2: Positive skewness

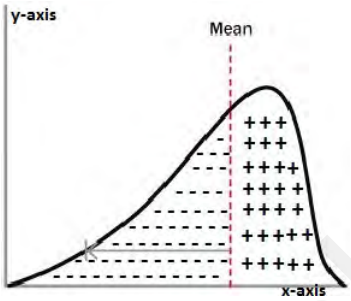


Figure 2.3: Negative skewness

Although previous authors, such as Uchigaki et al. (2012), Turhan et al. (2013), Turhan et al. (2009), and Ryu et al. (2015), have proposed means of addressing data skewness, this issue remains unresolved in general in machine learning studies. Data skewness can cause learning algorithms to produce poor and unreliable classification results (ÇATAL, 2016). In a normally distributed dataset, the mode is typically equal to the median and mean. For a positively skewed dataset, the mode is less than the median, which, in turn, is less than the mean. In contrast, in a dataset with a negative skew, the mode is greater than the median, and the median is greater than the mean. Logarithmic and automatic transformation schemes have been applied to address data skewness in previous studies, such as those of Uchigaki et al. (2012), Turhan et al. (2013), Turhan et al. (2009), Herbold (2013), Y. Ma et al. (2012), L. Chen et al. (2015), Ryu et al. (2015), Ryu et al. (2017), and Feng et al. (2016).

Critical comment: *Depending on the data distribution, it may be difficult to determine the accuracy performance of a model as a result of highly skewed datasets. By considering*

the skewed nature of the existing datasets, the researcher proposes a means of accurately determining the performance of learning algorithms despite the use of highly skewed data.

2.1.4 Irrelevant and redundant features in datasets

Most datasets contain irrelevant features, which can affect the performance of learning algorithms (Kwak & Choi, 2002; Gu et al., 2016). Training such algorithms only on relevant features can yield optimal results in machine learning studies. By the same token, irrelevant features can affect the accuracy of learning algorithms. Therefore, to encourage unbiased prediction outcomes, classification algorithms should be trained on an ideal set of features characterized by relevant information, simplicity and independence.

Techniques for selecting appropriate features from existing datasets can be applied to identify the most suitable subset of features related to a particular problem (Hossain et al., 2016). In addition, feature selection offers the ability to minimize the dimensionality of the data (Wahono, 2015). Selecting the most relevant attributes simultaneously removes irrelevant features from a dataset and enables better data management. For this reason, feature selection is essential when addressing imbalanced data (Kwak & Choi, 2002; Gu et al., 2016). To address the issue of irrelevant and redundant features in datasets, feature selection approaches have been applied in many previous studies, such as those of Yu & Mishra (2012), Lusa et al. (2010), W. Wei et al. (2013), Lane et al. (2012), Q. Li et al. (2013), Alibeigi et al. (2012), Gong & Huang (2012), X.-w. Chen & Wasikowski (2008), M.-H. Wei et al. (2013), P. He et al. (2015), Nam et al. (2017), Y. Zhang et al. (2015), Lima & Pereira (2015), Bae & Yoon (2015), J. Yang et al. (2016), Yijing et al. (2016), Beyan & Fisher (2015), Maldonado et al. (2014), Trafalis et al. (2014), Casañola-Martin et al. (2016), Al-Ghraibah et al. (2015), Haixiang et al. (2016), Dubey et al. (2014), L. Song et al. (2014), Moepya et al. (2014), Vong et al. (2015), N. Zhang (2016), Braytee et al. (2016), and Ng et al. (2016).

Attribute selection is one of the approaches that can be used to identify the most suitable subset of features related to a particular problem (Hossain et al., 2016). Such an approach enables the removal of irrelevant features in a dataset to enhance data management. For this reason, feature selection is essential when addressing imbalanced data (Kwak & Choi, 2002; Gu et al., 2016). Furthermore, a few authors, such as Laradji et al. (2015) and Shroff & Maheta (2015), have reported that feature selection can also be used to improve the quality of a dataset; consequently, applying dimensional reduction to datasets can improve the performance of prediction models. According to Tantithamthavorn (2016), the attributes of a dataset influence data preprocessing. That study explicitly described how data metrics can affect prediction outcomes and further reported that poor selection of data metrics reduces the accuracy of prediction models. In fact, the impact of poor metric selection can override the influence of the research team conducting the study and consequently lead to misleading results.

Critical comment: *It is usually difficult to select relevant features from datasets; consequently, irrelevant features are most often included among the selected features. The proposed data preprocessing framework offers a means of identifying relevant features in datasets.*

2.1.5 Continuous data

Continuous data are data whose values can be measured and divided into infinitely small increments, unlike discrete data, which take specific values. The majority of the existing datasets used in machine learning studies are characterized by continuous features, which can make it difficult to properly report these data in a discrete fashion (Fayyad & Irani, 1993).

Furthermore, the continuous nature of the existing data can lead to uncertainties in

the results produced by learning algorithms. Thus, for most learning algorithms to produce reliable classification outputs, a discrete feature space is required (Dougherty et al., 1995). To address the issue of continuous data, Ching et al. (1995) proposed a new information-theoretic data discretization technique. B. Ma et al. (2014) and L. Chen et al. (2015) also applied discretization techniques to continuous data. Discretization transforms quantitative data into qualitative data by partitioning the features into a small number of nonoverlapping intervals generated by implementing certain boundaries. Each feature value is assigned to its corresponding interval, thus making the data discrete (S. García et al., 2016).

Critical comment: *During preprocessing, new defects may be introduced, and the original data may be altered. To avoid this undesirable situation, a data preprocessing approach is proposed that can avoid any form of data alteration while also preventing the introduction of defects into the existing datasets.*

2.1.6 Data privacy

For obvious reasons, most project owners find it difficult to disclose their datasets because of the confidentiality of their projects (Hosseini, Turhan, & Gunarathna, 2017; Z. Li et al., 2017), although such projects may contain information that could be used to improve knowledge discovery in machine learning studies. Consequently, due to the privacy issues associated with these projects, such datasets are rarely made available to the machine learning community. Privacy concerns give rise to several controversial issues since the release of such information depends solely on the project owners or other responsible parties, who may not always be able to guarantee that their datasets will be used for the intended purposes (Smith et al., 2012).

However, as privacy issues continue to hinder the release of relevant datasets, these

concerns continue to pose a serious challenge in the big data mining community, which may sometimes require access to sensitive information in order to produce reliable results (Che et al., 2013). Consequently, encryption has been adopted by most researchers to ensure data privacy (Lin & Tzeng, 2012; N. Cao et al., 2014). In addition, multiparty data sharing has been applied to address privacy issues, as reported by Peters et al. (2015).

Critical comment: *Unfortunately, reliable datasets are not readily available for use. Therefore, the proposed data preprocessing approach aims to restore researchers' confidence in the ability to suitably preprocess existing datasets to avoid bias in the results obtained in defect prediction studies.*

2.1.7 Collinearity among metrics

In many existing datasets, some of the attributes are collinear, as reported by Gil & Lalouche (2017), Herraiz et al. (2011), Landman et al. (2016), Landman et al. (2014), Tantithamthavorn et al. (2016a), and F. Zhang et al. (2017). In other words, these metrics are correlated with each other. Such metrics are commonly used in regression models for defect prediction (Felix & Lee, 2017b). In multiple regression, however, the variables are usually independent; for instance, in the expression

$$Y = B_0 + B_1X_1 + B_2X_2 + \varepsilon$$

the variables X_1 and X_2 are treated as being independent of each other. However, multicollinearity will arise if X_1 and X_2 are correlated. In the above expression, Y represents the dependent variable and, as such, depends on X_1 and X_2; this indicates that X_1 and X_2 contain relevant information about Y. If X_1 and X_2 are collinear, this situation can lead to redundancy and collinearity, which will hinder the ability to generate accurate

prediction outcomes. When similar variables that are collinear are simultaneously applied in a prediction model, the correlation that exists among the supposedly independent variables tends to reduce the accuracy of the model (Hosseini, Turhan, & Gunarathna, 2017). Kamei et al. (2016) addressed the issue of collinearity among metrics by removing highly correlated metrics, whereas Jiarpakdee et al. (2018) combined clustered variables and applied the variance inflation factor technique to address the collinearity issue. In addition, a principal-component-based approach for addressing the collinearity among metrics has been proposed, as reported by Nagappan, Ball, & Murphy (2006), Briand et al. (2002), Nagappan, Ball, & Zeller (2006), Nelson et al. (2011), and Castaño & Gallón (2017).

Critical comment: *When attempting to address collinearity among metrics, relevant features may be eliminated. Consequently, the resulting model performance results may be misleading. The proposed framework for data preprocessing offers a suitable means of identifying collinearity among metrics.*

2.1.8 Noise in data

Noisy data will not produce meaningful outputs when they are used in prediction studies. Noise in data is usually associated with outliers. Outliers are data points that lie far away from the main cluster(s) of data. Outliers may occur in datasets as a result of measurement variations or may be a manifestation of experimental error. The removal of outliers can lead to a reduction in the noise found in a dataset (Ryu et al., 2015; W. Li et al., 2015; W. Liu et al., 2014). Noise filtering has also been reported to be an effective means of eliminating noise in a dataset, as reported by Hosseini, Turhan, & Gunarathna (2017) and Kang et al. (2017).

Critical comment: *Outliers in datasets cause those datasets to be noisy. During noise*

filtering, relevant features may be filtered out. Therefore, a filter-based feature selection approach is proposed to address such noise in datasets.

2.1.9 Missing values in datasets

Most datasets used in machine learning studies tend to contain missing values. Missing values are data that were not stored or recorded as a result of faulty sampling, which often occurs due to human or computer error. Missing values are often difficult to avoid because they are identified only during data analysis, and as such, most researchers face difficulties when dealing with missing values (S. García et al., 2016). If missing values are not adequately handled, they may lead to poor knowledge discovery or incorrect research findings. Notably, important information can be lost as a result of missing values (H. Wang & Wang, 2010). Several approaches, such as the imputation technique, have been proposed to address the issue of missing values, as reported by Luengo et al. (2012). There is also the possibility that the data may suffer from class overlap due to missing values.

Gupta & Gupta (2017) identified class overlap, which renders samples in a dataset invalid and noisy, as a significant factor affecting datasets. The authors concluded that training a classification model with data that are free of class overlap improves the model performance. Conversely, training such a model with data containing class overlap can decrease the model performance. To address the class overlap issue, G. Batista et al. (2012), Chawla et al. (2002), and Gupta & Gupta (2017) proposed a synthetic minority oversampling technique (SMOTE) that can identify the overlapping instances in a dataset and improve the prediction outcome. Chawla et al. (2002) applied a combination of an oversampling technique for the minority class and an undersampling technique for the majority class to improve the classification accuracy in terms of the area under the receiver operating characteristic (ROC) curve (AUC). This approach performs better than undersampling alone, as confirmed by Ha & Lee (2016), who identified issues with the

existing undersampling techniques in addressing the data imbalance problem. The authors reported that the existing undersampling techniques cannot improve the understandability of classification models in terms of the AUC. In an attempt to solve this problem with the existing undersampling techniques, a method called Genetic Algorithm-based Under-Sampling (GAUS) was proposed to improve the classification accuracy.

To address the persistent issue of class imbalance in datasets, Soda (2011) proposed a reliability-based balancing algorithm to effectively balance the defective and defect-free classes in a dataset. López et al. (2013) and Z. Sun et al. (2015) categorized some of the proposed approaches for this purpose into data sampling methods, cost-sensitive learning methods, methods based on algorithm modification and bagging, and boosting-based methods. These approaches result in the loss of essential portions of the data, which may lead to model overfitting (Z. Sun et al., 2015). Having investigated the existing literature, the researcher now summarizes the identified data-related issues and the techniques applied to address them in Table 2.1.

Table 2.1: Summary of class- and method-level data-related issues and corresponding preprocessing techniques and weaknesses

Data-related issue	Preprocessing technique	Study references	Weaknesses
Class imbalance	Rebalancing	H. Cao et al. (2014), P. Cao et al. (2014), Cateni et al. (2014), L. Chen et al. (2015), Chetchotsak et al. (2015), Das et al. (2015), Díez-Pastor et al. (2015), Dubey et al. (2014), D'Addabbo & Maglietta (2015), Ha & Lee (2016), Jian et al. (2016), Kamei et al. (2016), Krawczyk et al. (2014), Kumar et al. (2014), J. Li et al. (2017), Menardi & Torelli (2014), Nekooeimehr & Lai-Yuen (2016), Ryu et al. (2016), Ryu et al. (2017), Sáez et al. (2015), J. Song et al. (2016), Z. Sun et al. (2015), Yun et al. (2016), Zhai et al. (2017), D. Zhang et al. (2015)	New defects may be introduced, and original data may be altered.
	Reweighting	X. Jing et al. (2015), Ryu et al. (2016), Ryu et al. (2017)	Original data may be altered with data approximation.
	Selective learning	Ryu et al. (2015)	Relevant features in the data may be omitted.
	Data transformation	X. Jing et al. (2015), F. Zhang et al. (2016)	New defects may be introduced, and original data may be altered.
Data heterogeneity	Filtering	L. Chen et al. (2015), X. Jing et al. (2015), Peters et al. (2015), Ryu et al. (2015), Ryu et al. (2017)	Focuses only on noise in the dataset; relevant features may be filtered out.
	Data normalization	Canfora et al. (2015), X. Jing et al. (2015), Panichella et al. (2014), Ryu et al. (2016), Ryu et al. (2017)	New defects may be introduced.
	Metric matching	Nam et al. (2017)	Both relevant and irrelevant metrics in the data may be matched.
	Clustering	Canfora et al. (2015)	Both relevant and irrelevant metrics in the data may be clustered.
High skewness	Automatic/logarithmic transformation	L. Chen et al. (2015), Feng et al. (2016), Ryu et al. (2015), Ryu et al. (2017), Changyong et al. (2014)	Depends on the data distribution; difficult to determine the accuracy performance of a model.
Irrelevant and redundant features	Feature selection	Al-Ghraibah et al. (2015), Bae & Yoon (2015), Beyan & Fisher (2015), Braytee et al. (2016), Casañola-Martin et al. (2016), Dubey et al. (2014), Haixiang et al. (2016), P. He et al. (2015), Lima & Pereira (2015), Maldonado et al. (2014), Moepya et al. (2014), Nam et al. (2017), Ng et al. (2016), L. Song et al. (2014), Trafalis et al. (2014), Vong et al. (2015), Wahono (2015), J. Yang et al. (2016), Yijing et al. (2016), N. Zhang (2016), Y. Zhang et al. (2015)	Difficult to select relevant features; irrelevant features may be selected.
Continuous data	Discretization	L. Chen et al. (2015), B. Ma et al. (2014)	New defects may be introduced, and original data may be altered.
Data privacy	Multiparty data sharing	Peters et al. (2015)	Reliable datasets are not readily available for use.
Collinearity among metrics	Removal of highly correlated metrics	Kamei et al. (2016)	Relevant features may be removed.
	Combination of variables and variance inflation factor	Jiarpakdee et al. (2018), O'brien (2007)	Primarily focuses on multicollinearity; consequently, model performance results may be misleading.
	Principal-component-based technique	Castaño & Gallón (2017)	New defects may be introduced, and original data may be altered.
Noise in data	Outlier removal	W. Li et al. (2015), W. Liu et al. (2014), Ryu et al. (2015)	Focuses only on outliers
	Noise filtering	Hosseini, Turhan, & Gunarathna (2017), Kang et al. (2017)	Focuses only on noise.

2.1.10 Quality of cross-project datasets

Herbold (2013) proposed a distance-based approach for selecting the data to be used in defect prediction. The author found that a distance-based technique enhances the success rate of cross-project defect prediction. To address the problem of erroneous datasets affecting the performance of classification models, Beckmann et al. (2011) and Iliou et al. (2015) proposed a genetic algorithm for the oversampling of data. Based on this genetic algorithm, an evolutionary approach was used to create a synthetic minority-instance-oriented class with which to adjust the data. It was shown that the classification accuracy could be successfully improved via such data adjustment. Tantithamthavorn et al. (2015) argued that the outcome and accuracy of any prediction model depend on the data used for training. Therefore, prediction models may be overfitted and produce untrustworthy results if the datasets are not reliable.

To overcome this problem, Siebra & Mello (2015) encouraged researchers to apply adequate preprocessing techniques to achieve reliable results. M. H. Rahman et al. (2016) applied a feature-space transformation process in combination with data preprocessing and normalization to improve prediction accuracy, whereas Mausa et al. (2014) applied a standard procedure for collecting data to be used in a defect prediction study. The latter approach limits the influence of knowledge related to a biased dataset by focusing on the detailed capabilities of a bug-code analyzer. This procedure provides all of the functionalities necessary to create a software defect prediction dataset using a defect monitoring device capable of analyzing defects from the contents of source code management repositories.

Y. Zhang et al. (2015) presented a means of combining classification models across different projects by integrating multiple machine learning techniques to transfer a prediction model from one project to another. The models were trained using data from another project, and their performances were evaluated using two standard evaluation

metrics: the F-score and cost-effectiveness. One of the findings of that study was that for a new software project, insufficient training data are available for cross-project defect prediction. Mahmood et al. (2015) analyzed the predictive performance achieved when using imbalanced data for the prediction of software defects. When the data used for classification are of unequal proportions among different classes, the predictive accuracy of defect prediction studies appears to be low, whereas balanced data result in increased predictive performance. One measure that can be used to address such imbalance problems was highlighted by X.-Y. Jing et al. (2014).

To address the issues related to training data for different projects, Herbold (2013) reported how a training data selection strategy can be applied to improve performance in cross-project defect prediction. The proposed approach was evaluated on numerous case studies based on 44 datasets, which were collected from 14 projects available online, using 7 different prediction models, namely, a naïve Bayes model, a support vector machine (SVM) model, a Bayesian network model, a J48 decision tree model, a random forest (RF) model, a multilayer perceptron model and an LR model. The results of this study indicated a high success rate of cross-project defect prediction based on an evaluation in terms of the recall, precision and success rate. In a comparison of the results for the same project with those for different projects in terms of the recall and success rate, cross-project defect prediction yielded a higher recall (by 0.19) than that achieved in within-project prediction, whereas the cross-project success rate achieved with the training data selection strategy was surpassed by the within-project success rate of 0.19. Despite these noteworthy achievements in addressing data quality issues, the quality of the existing class- and method-level datasets is yet to be fully addressed, and consequently, data quality issues still lead to controversial research findings.

2.2 Classification model performance in software defect prediction

Previous studies have proposed models for both within-project and cross-project defect prediction at the source code level based on the software development process. These models have achieved noteworthy performance in terms of accuracy, reliability, fault detection and performance improvement in binary defect classification.

Several studies have focused on improving the outcomes and performances of the prediction models used in binary defect classification. In either within-project or cross-project prediction, the objective is to enhance the performance of the applied classification algorithms. For instance, X. Jing et al. (2015) successfully applied a defect prediction approach based on canonical correlation analysis to various heterogeneous projects. The outcomes in that study were evaluated using the false positive rate, recall, F-score and Matthews Correlation Coefficient (MCC).

To improve the detection of defect-prone modules, Shepperd et al. (2014) presented a meta-analysis of how relevant prediction factors influence predictive performance. The study further reported that classification outcomes primarily depend not on the model chosen but on factors related to how the data used to train the models are cleaned by the research group performing the study. According to the findings, the classifier can have an influence of as little as 1.3% on performance, compared with the significant explanatory factor of 31% related to the research group. In a study conducted to analyze the accuracy of defect prediction models for new software products, Cavezza et al. (2015) proposed a method of continuously improving a prediction model. This method was evaluated using the precision, recall and F-measure. The experiment performed in that study produced promising results and demonstrated how such an active method can be applied in defect prediction.

M. H. Rahman et al. (2016) reported how the accuracy of a prediction technique can be

improved via the application of a feature space transformation process. The outcomes of this prediction technique were evaluated using the confusion matrix and balance as the evaluation metrics. Maneerat & Muenchaisri (2011) reported the performances of various machine learning algorithms, namely, RF, naïve Bayes, LR, an Instance-Based Learner (IBL), a simple IBL (IBk), Voting Feature Intervals (VFI) and J48. The study analyzed how bad-smell prediction can be performed early in the Software Development Life Cycle (SDLC), and statistical significance tests were conducted to evaluate the prediction performance. Performance metrics related to accuracy, specificity and sensitivity as well as a confusion matrix table were used for evaluation.

Lessmann et al. (2008) evaluated the performance superiority of one classifier over another. In this study, the AUC and the truth table also known as confusion matrix were applied to evaluate the outcomes. The truth table shows the sensitivity (true positive rate) and false alarm rate (false positive rate) of a classifier, whereas the AUC is the most revealing and objective measure of the predictive accuracy of a model in the context of defect prediction. Halim (2013) analyzed how to measure the complexity of object-oriented software early in the SDLC to predict fault-prone classes. In this study, naïve Bayes and K-Nearest-Neighbors (KNN) models were applied to identify the relationship between module design complexity and fault-proneness. The performances of the models were evaluated based on the accuracy, precision and AUC.

Tantithamthavorn et al. (2015) investigated how the effectiveness of a prediction model can depend on the quality of the data used to train that model. The study concluded that if a prediction model is trained with noisy data, then that model may produce inaccurate results. The outcomes in that study were evaluated using the precision, recall and F-measure. Monden et al. (2013) described how the effort required for defect prediction can be reduced by adopting suitable evaluation measures that are likely to result in a reliable prediction

accuracy. The outcomes in that study were evaluated based only on the prediction accuracy and effort allocation.

Nugroho et al. (2010) applied several performance metrics, including accuracy, precision, sensitivity, specificity and a confusion matrix table, in the evaluation of early prediction outcomes for estimating avoidable faults early in the SDLC. The study further reported that a prediction model constructed using the Unified Modeling Language (UML) design standard is likely to achieve higher accuracy than a prediction model constructed using codes. However, a prediction model designed via a combination of both UML and codes can achieve significantly improved performance. Another study (Taba et al., 2013) investigated how to increase the accuracy of a bug prediction model by means of metric-based antipatterns. Such antipatterns indicate weaknesses in the design that may increase the risk of bugs in a software product. The F-score was applied to evaluate the outcomes in that study.

Investigating whether metrics that are available early in the SDLC can be applied to identify defective modules, Jiang et al. (2007) revealed that these metrics can influence project management. Such metrics can be used to drive efforts to improve the quality of software or to construct models to support verification and validation activities. In that study, the performances of two models built using the aforementioned metrics, namely, a requirement-based model and a code-based model, were compared using the following machine learning algorithms: OneR, naïve Bayes with a kernel, voted perceptron, LR, J48, VFI, IBk and RF. One drawback of the requirement-based model is that it cannot independently predict defects. However, a significant prediction result was achieved when the two models were combined. Nevertheless, only a few evaluation metrics were applied for accuracy assessment in that study, including the probability of detection (PD), also called the recall, as well as the probability of false alarms (PF) and the AUC.

In an attempt to reduce the likelihood of errors in a software system, Taba et al. (2013) enhanced the accuracy of defect prediction by applying metrics based on antipatterns. They used refactoring to correct poor designs and used antipatterns to identify weaknesses in a design that might increase the risk of future defects. If defects can be predicted using antipattern information, then the development team can use refactoring to reduce the risk of defects in the system.

Furthermore, Mori (2015) developed a prediction model with high accuracy and explanatory power by superposing a naïve Bayes model on an ensemble model, which resulted in an improvement in prediction performance. The naïve Bayes technique was found to be suitable for predicting the defect-proneness of a class using object-oriented metrics. This result is consistent with the finding of Menzies et al. that the naïve Bayes technique appears to be the best approach for constructing defect prediction models.

X.-Y. Jing et al. (2014) also achieved improved accuracy in software defect prediction using a technique based on collaborative representation classification. Their proposed metric-based software defect prediction method resulted in a considerably larger number of defect-free modules compared with the number of faulty modules. Although class imbalance was encountered in that study, the outcome of the study was not affected because the class imbalance was properly addressed through Laplacian score sampling for sample training, which resulted in an improved prediction accuracy.

Islam & Sakib (2014) used package-based clustering to enhance the accuracy of software defect prediction. These authors grouped software packages into multiple clusters according to their relationships and similarities and proposed a prediction model using this package-based clustering approach. The proposed approach achieved prediction rates of 54%, 71%, and 90%, all higher than those obtained using a prediction model based on BorderFlow and k-means clustering. However, the results thus obtained may be

inconsistent and, as such, lead to incorrect interpretations. Therefore, Malhotra & Raje (2014) addressed the problem of incorrect interpretations when verifying the performance of a defect prediction model. These authors used an object-oriented metric design suite and compared various machine learning techniques to investigate the impact of object-oriented metrics on erroneous classification.

Cavezza et al. (2015) examined defect prediction performance during the software development process. The findings of the study suggest that when a standard approach is applied for defect prediction, promising results can be obtained through continuous refinement of the prediction model using new commit data and by predicting whether any action introduced into a program introduces a bug. Lu et al. (2012) developed a semisupervised learning technique for program defect prediction using a variant of a self-study algorithm. The results confirmed that confidence fitting can be used as a substitute for established supervised algorithms. In combination with dimensional reduction, the developed semisupervised algorithm performed considerably better than an RF model when modules with typical faults were used for training. Xuan et al. (2015) comprehensively studied within-project defect prediction performance in a practical and sophisticated manner. These authors used a massive set of evaluation metrics and reported that a Bayesian network achieved good performance. However, other classification models may perform better in different scenarios because no single model dominates in binary defect classification performance.

Fukushima et al. (2014) evaluated a cross-project model using Just-In-Time (JIT) prediction via a case study of open-source projects. That study reported that within-project defect prediction models that are able to achieve high accuracy are uncommon compared with high-accuracy cross-project prediction models. However, cross-project prediction models trained on projects with identical correlations between the predictor and dependent

variables often exhibit good performance. Nam & Kim (2015) applied a collection of metric equalities to construct a prediction model for projects with diverse metric sets. They combined metric selection and metric combination to achieve a forecasting rate of 68%, which was higher than or comparable to the rates achieved for within-project defect prediction. Their proposed method also showed statistical significance.

X. Jing et al. (2015) presented a successful solution for mixed cross-company defect prediction by means of combined metric representations for data origin and destination. The performance of this approach depends on the correlation analysis that is established to achieve effective transfer learning for cross-company defect prediction. In this way, similar initial and resulting data distributions can be obtained. Panichella et al. (2014) improved the detection of defect-prone entities among software projects by means of a unified defect predictor that considers the groupings produced by various machine learning techniques.

Lessmann et al. (2008) proposed techniques for predicting fault-prone modules by prioritizing quality assurance efforts and used these techniques for selecting modules in accordance with their fault probabilities. To date, none of these techniques has demonstrated the ability to predict the number of defects that may exist in an upcoming product release. The approach proposed in the current study attempts to fill this gap to expedite actions taken for quality assurance by allowing the number of software defects to be predicted using the average defect density, average defect velocity, average defect introduction time and module design complexity. Notably, software complexity significantly affects the cost and time of software development and maintenance (Yousefi & Modiri, 2011).

Shepperd et al. (2014) conducted a meta-analysis of all relevant factors that influence predictive performance. These authors verified the performance of their defect prediction model by determining the factors that significantly influence the predictive outcomes of software defect classifiers, as determined based on the MCC. They found that the

choice of classifier only slightly influences the performance, whereas the model building factors (that is, factors related to the research group) exert a significant effect. This is because the research group is responsible for data preprocessing. If the data applied in a study are not properly cleaned, that study may produce a biased outcome. Consequently, the performance achieved in a prediction study depends predominantly on the research group and not on the choice of classifier. F. Zhang et al. (2014) attempted to construct a universal defect prediction prototype. However, developing a universal model of the primary connections between software metrics and defects is challenging because of the variations among predictors. Caglayan et al. (2010) therefore constructed a sensitive defect prediction model based on fault categories. By separating defect statistics into different classes for consideration in a defect prediction algorithm, practitioners are able to take proper actions to improve their prediction accuracy. Similarly, grouping and segmenting the errors in a program before applying a prediction model can also improve the prediction accuracy. Maneerat & Muenchaisri (2011) analyzed bad-smell prediction in an early phase of the SDLC by comparing the performances of different machine learning algorithms through statistical significance testing.

Halim (2013) proposed a model that can compute the complexity of object-oriented software in the design phase for the prediction of error-prone classes. He applied naïve Bayes and KNN models to identify the link between complexity and bug-proneness in a design. Parthipan et al. (2014) similarly proposed an evaluation model that captures the symptoms of design complexity using an aspect-oriented complexity evaluation model. Note that in the design phase and at the code level, defect prediction models are primarily designed either to discriminate between defective and defect-free modules (binary classification) or to forecast the number of defects (regression analysis) (Caglayan et al., 2010). The findings obtained from this extensive literature review confirm that for defect prediction

models to produce unbiased results, the quality of the data applied in training these models must be uncompromised. The findings further confirm that a well-preprocessed class- or method-level dataset is the key to ensure that the results obtained from any supervised machine learning study will be optimal and obtained based on unbiased data. Notably, the outcome of this literature review calls for an immediate solution to address the lingering data quality issues in software defect prediction.

2.3 List of existing performance evaluation metrics

This section presents the metrics used to evaluate the performance of learning algorithms applied in binary defect classifications. These metrics include the confusion matrix, Classification Accuracy (CA), precision, recall, specificity, Matthews Correlation Coefficient (MCC), J-coefficient, F-score, area under the receiver operating characteristic (ROC) curve (AUC), Brier score, information score and geometric mean.

2.3.1 Confusion matrix

A confusion matrix, also called a truth table, shows the numbers of accurately and inaccurately predicted outcomes compared with the real outcomes. The matrix has dimensions of $M \times M$, where M is the number of target classes (for example, defective and defect-free). In this study, the value of M is 2. Table 5.1 presents the 2×2 confusion matrix for the two classes, namely, the positive and negative classes, or the defective and defect-free classes, respectively.

Table 2.2: Truth table

		Target Values	
		Actually Positive	Actually Negative
Model Predictions	Positive	a = (TP)	b = (FP)
	Negative	c = (FN)	d = (TN)

In this table, a = the number of instances that are actually positive and are also predicted to be positive, also called True Positives (TP); b = the number of instances that are negative

but are predicted to be positive, also called False Positives (FP); c = the number of positive instances that are incorrectly predicted to be negative, also called False Negatives (FN); and d = the number of negative instances that are correctly predicted to be negative, also called True Negatives (TN).

2.3.2 Classification accuracy (CA)

The ratio of the number of correct predictions to the total number of predictions. The CA is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

2.3.3 Precision

The proportion of all predicted positive instances that are correctly predicted.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

2.3.4 Recall (sensitivity)

The proportion of all actually positive instances that are correctly predicted.

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

2.3.5 Specificity

The proportion of all actually negative instances that are correctly predicted.

$$Specificity = \frac{TN}{TN + FP} \tag{2.4}$$

2.3.6 Matthews correlation coefficient (MCC)

Applied in binary classification as a measure of the achieved performance, the MCC is an important correlation coefficient that represents an overall assessment of the observed and predicted classes in binary classification. It considers all four aspects of the confusion matrix, including the true negatives (Mahmood et al., 2015). It is calculated directly from the confusion matrix as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)\,(TP + FN)\,(TN + FP)\,(TN + FN)}} \tag{2.5}$$

2.3.7 J-coefficient

The sum of the recall and specificity minus one. It is equivalent to the recall minus the false positive rate (FPR).

$$J\text{-}coef = Recall - FPR = Recall + Specificity - 1 \tag{2.6}$$

2.3.8 F-score

Also called the **F-measure** or **F1**, the F-score is the harmonic average of the precision and recall. It is calculated using the following formula:

$$F\text{-}score = 2 \times (Precision \times Recall)\,(Precision + Recall) \tag{2.7}$$

2.3.9 Area under the ROC curve (AUC)

A metric based on a graphical representation of the performance of a binary classifier. It can also be used to compare different classifiers. The sensitivity, which represents the proportion of actually positive instances that are correctly classified, is plotted on the y-axis against 1 minus the specificity, which represents the proportion of actually

negative instances that are correctly classified, on the x-axis. A larger AUC value indicates better classifier performance. As reported by Tantithamthavorn et al. (2016b), the possible values of the AUC lie between 0 and 1. An AUC value of 0 indicates the worst possible performance, whereas a value of 1 indicates the best possible performance.

2.3.10 Geometric mean (G-mean)

The square root of the product of the recall and precision, as proposed by Kubat et al. (1997, 1998). Thus, the geometric mean is calculated as follows:

$$G\text{-}mean = \sqrt{Recall \times Precision} \tag{2.8}$$

2.3.11 Brier score (BS)

A score that measures the accuracy of probabilistic predictions. Tantithamthavorn et al. (2016b) applied the Brier score in their study to measure the gap between the estimated probability and the result achieved. A Brier score of 0 is the best achievable score, while 1 is the worst achievable score. The Brier score is calculated using the following formula:

$$BS = \frac{1}{N} \sum_{t=1}^{N} (f_t - o_t)^2 \tag{2.9}$$

where

f_t = the forecasted likelihood of occurrence, o_t = the true event result at time t, and

N = the number of predicted instances.

2.3.12 Information Score (IS)

The average amount of information per classified instance, as defined by Kononenko & Bratko (1991).

2.4 Regression models in software defect prediction

Researchers have been making great efforts to develop various techniques capable of predicting the number of software defects at the code level (Petrić, 2016). As noted earlier, several studies have proposed approaches for improving the outcomes of software defect prediction with a focus on binary classification, for example, those of Cavezza et al. (2015), Xuan et al. (2015), Lu et al. (2012), Fukushima et al. (2014), Nam & Kim (2015), X. Jing et al. (2015), Panichella et al. (2014), Lessmann et al. (2008), Shepperd et al. (2014), and F. Zhang et al. (2014). These studies have focused on improving the performance of learning algorithms at the class level. For instance, while investigating methods of improving the classification accuracy of learning algorithms, Galar et al. (2012) and H. He & Ma (2013) reported that the performances of these algorithms can be improved by considering combinations of individual metrics when confronting challenges related to imbalanced data. Lessmann et al. (2008) evaluated the performance superiority of one classifier over another. Using an alternative improvement approach, Taba et al. (2013) investigated how to increase the accuracy of a bug prediction model by means of metric-based antipatterns. G. E. Batista et al. (2004) performed an extensive experimental assessment of the performances of learning algorithms when applied to imbalanced data and reported that class imbalance does not completely prevent the successful application of learning algorithms. Notably, such learning algorithms are capable of classifying a module as either defective or defect-free. Petrić (2016) reported that the current prediction models handle defect prediction in a black-box manner. This is a weakness of the existing prediction models since they do not enable the prediction of the number of defects but rather focus on classification, which attempts only to forecast whether a software program will be defective (Zimmermann et al., 2007).

On the other hand, the application of regression models can allow a software team to

determine which models can best reveal the relationship between certain independent and dependent variables (Petrić, 2016). While attempting to address the lack of approaches for predicting the number of software defects, Bernstein et al. (2007) noted that if the number of software defects in a new software product can be predicted, both software managers and other stakeholders will benefit.

Nagappan & Ball (2005) presented a technique using a set of relative code churn measures for the early prediction of the software defect density. Their results showed that absolute measures of code churn are poor predictors of defect density. Meanwhile, Bernstein et al. (2007) extracted a certain number of temporal features, such as the number of revisions and the number of reported issues, from the CVS and Bugzilla repositories to predict the number of defects for the Eclipse project. Their results indicate that by using certain temporal features, a prediction model can be developed to predict whether a source file will contain a defect. That study achieved noteworthy results in predicting the number of software defects, although it was limited to only six Eclipse plugins, representing a single project.

To estimate the number of software defects, Felix & Lee (2017b) proposed certain prediction models that were first individually built using the average design complexity, average defect density, average defect introduction time and average defect velocity. The results obtained indicated that for determining the possible number of defects in a new software release, multiple regression models can be built based on combinations of the average defect density and average defect velocity and of the average design complexity and average defect velocity, and a prediction model can be built based on the average defect introduction time and average defect velocity. However, none of the existing studies has considered the prediction of the numbers of software defects at both the class and method levels simultaneously.

Hence, this research presents such an approach using derived variables first applied by Felix & Lee (2017b). Notably, in addition to not applying the proposed model for both class- and method-level prediction, Felix & Lee (2017b) also did not provide detailed regression model information, such as percentage errors on the results, and the quality of the data applied was low. Therefore, the current study addresses the corresponding gap in the literature by applying the proposed approach at both the class and method levels for various software projects to enable a detailed analysis of the approach, along with clear evidence of the reliability of the datasets. It is also important to note that the overall accuracy of the proposed approach is greater than that of other methods in the literature with respect to data quality and model performance. If the size of a software program can be predicted as reported by Laranjeira (1990) and Dolado (2000), then a corresponding method of predicting the possible numbers of defects likely to be present in a future release of a software product at both the method and class levels is needed. Hence, the current study attempts to provide such an approach.

2.5 Summary

While searching for answers to the current research questions to achieve the objectives of this research, the researcher found that researchers are still attempting to address the challenges related to the quality of the datasets applied in supervised machine learning studies. Although existing studies have offered noteworthy contributions in attempts to address issues related to the quality of the data used in software defect prediction, these issues remain unresolved. Consequently, such issues have led to numerous research controversies. Notably, some researchers have proposed various frameworks to address these challenges, such as the frameworks of Menzies et al., Lessmann et al. and Song et al. However, Wahono (2015) challenged the efficacy of these frameworks, arguing that they produce misleading findings while addressing issues associated with data inconsistencies.

Other researchers have also proposed techniques for addressing various data-related issues. However, these techniques also do not fully address the existing data quality issues; hence, our proposed framework attempts to fill this gap.

Importantly, the performances of existing prediction models have also been challenged as a result of data inconsistencies in machine learning studies. Taipale et al. (2013) maintained that these controversies involving defect prediction studies make it difficult to successfully implement their research findings in a real-world scenario. Panichella et al. (2014) further noted that no clear winner among these prediction models can be identified due to the controversies associated with the research findings. However, while investigating whether the existing literature has reported means of predicting the numbers of class- and method-level defects in a new software version, the researcher conducting this study found that Bernstein et al. (2007) considered the number of revisions and the number of reported issues to predict the number of defects for the Eclipse project, although without providing any general guidance on how such an approach could be applied to future versions of software products. Although their results indicate that certain temporal features can be used to develop a model for predicting whether a source file will contain a defect, that finding is limited to only six Eclipse plugins, representing a single project, which seems to be a drawback of their results.

Table 2.3: Summary of some related studies on data preprocessing

Author(s)	Topic area	Institution	Country	Year	Publication
Al-Ghraibah et al. (2015)	Feature selection	New Mexico State University	United States	2015	Conference
Lima & Pereira (2015)	Feature selection	Federal University of Minas Gerais	Brazil	2015	Conference
Maldonado et al. (2014)	Feature selection	Universidad de los Andes	Chile	2014	Journal
Bolon-Canedo et al. (2011)	Feature selection	University of A Coruña	Spain	2011	Journal
Braytee et al. (2016)	Feature extraction	University of Technology Sydney	Australia	2016	Conference
ÇATAL (2016)	Faulty data	Istanbul Kültür University	Turkey	2016	Journal
B. Ma et al. (2014)	Fault classification	Beijing University of Posts and Telecommunications	China	2014	Journal
Bae & Yoon (2015)	Imbalanced learning	Gwangju Institute of Science and Technology	South Korea	2015	Journal
Vong et al. (2015)	Imbalanced learning	University of Macau	Macau	2015	Journal
P. Cao et al. (2014)	Imbalanced learning	Northeastern University	China	2014	Journal
D. Zhang et al. (2015)	Imbalanced classification	Shandong Jianzhu University	China	2015	Conference
Moepya et al. (2014)	Class imbalance	University of Johannesburg	South Africa	2014	Conference
Wald et al. (2013)	Class imbalance	Florida Atlantic University	United States	2013	Conference
Longadge & Dongre (2013)	Class imbalance	G.H. Raisoni College of Engineering	India	2013	Journal
Kontos & Maragoudakis (2013)	Class imbalance	University of the Aegean	Greece	2013	Conference
Beyan & Fisher (2015)	Imbalanced data	University of Edinburgh	Scotland	2015	Journal
Yin et al. (2014)	Imbalanced data	Xi'an Jiaotong University	China	2014	Journal
Abolkarlou et al. (2014)	Imbalanced data	Graduate University of Advanced Technology	Iran	2014	Conference
Zhai et al. (2017)	Imbalanced data	Hebei University	China	2017	Journal
Díez-Pastor et al. (2015)	Imbalanced data	Bangor University	Wales	2015	Journal
Dubey et al. (2014)	Imbalanced data	Arizona State University	United States	2014	Journal
D'Addabbo & Maglietta (2015)	Imbalanced data	National Research Council	Italy	2015	Journal
Manikandan et al. (2016)	Imbalanced data	Bharathiar University	India	2016	Journal
López et al. (2013)	Imbalanced data	University of Granada	Spain	2013	Journal
Z. Sun et al. (2015)	Imbalanced data	Xi'an Jiaotong University	China	2015	Journal
Trafalis et al. (2014)	Imbalanced data	University of Oklahoma	United States	2014	Journal
Kumar et al. (2014)	Imbalanced data	JNTU	India	2014	Journal
J. Li et al. (2017)	Imbalanced data	University of Macau	Macau	2017	Journal
J. Yang et al. (2016)	Imbalanced data	Shenzhen University	China	2016	Journal
Yijing et al. (2016)	Imbalanced data	China University of Geosciences	China	2016	Journal
Casañola-Martin et al. (2016)	Imbalanced data	Universitat de València	Spain	2016	Journal
Haixiang et al. (2017)	Imbalanced data	University of Geosciences	China	2017	Journal
Haixiang et al. (2016)	Imbalanced data	University of Geosciences	China	2016	Journal
Jian et al. (2016)	Imbalanced data	Guangdong University of Technology	China	2016	Journal
Menardi & Torelli (2014)	Imbalanced data	Università degli Studi di Padova	Italy	2014	Journal
Nekooeimehr & Lai-Yuen (2016)	Imbalanced data	University of South Florida	United States	2016	Journal
Borrajo et al. (2011)	Imbalanced data	University of Vigo	Spain	2011	Journal
Y. Liu et al. (2011)	Imbalanced data	Shandong University	China	2011	Journal
Palacios et al. (2010)	Imbalanced data	Universidad de Oviedo	Spain	2010	Conference
Cateni et al. (2014)	Imbalanced datasets	TeCIP Institute	Italy	2014	Journal
Ng et al. (2016)	Imbalanced datasets in classification	South China University of Technology	China	2016	Journal
Krawczyk et al. (2014)	Imbalanced datasets in classification	Wrocław University of Technology	Poland	2014	Journal
J. Song et al. (2016)	Imbalanced datasets in classification	Communication University of China	China	2016	Conference
L. Song et al. (2014)	Imbalanced datasets in classification	Nanjing Medical University	China	2014	Journal
L. Chen et al. (2015)	Sample reduction	Chongqing University	China	2015	Journal
P. He et al. (2015)	Metric set	Wuhan University	China	2015	Journal
X. Jing et al. (2015)	Metric set	Wuhan University	China	2015	Conference
Castaño & Gallón (2017)	Multicollinearity	Universidad de Antioquia	Colombia	2017	Journal
W. Li et al. (2015)	Outliers in datasets	Baylor Research Institute	United States	2015	Journal
W. Liu et al. (2014)	Outliers in datasets	Watson Research Center	United States	2014	Conference
Z. Li et al. (2017)	Privacy	Wuhan University	China	2017	Journal
Ha & Lee (2016)	Undersampling	Sungkyunkwan University	South Korea	2016	Conference
Kang et al. (2017)	Undersampling	Tongji University	China	2017	Journal
Yun et al. (2016)	Oversampling	Sungkyunkwan University	South Korea	2016	Conference
H. Cao et al. (2014)	Oversampling	Research institute	Singapore	2014	Journal
Das et al. (2015)	Oversampling	Washington State University	United States	2015	Journal

Table 2.4: Table 2.3 continued

Author(s)	Topic area	Institution	Country	Year	Publication
Kamarulzalis et al. (2018)	Oversampling	Universiti Teknologi MARA	Malaysia	2018	Conference
Beckmann et al. (2011)	Oversampling	Federal University of Rio de Janeiro	Brazil	2011	Conference
Jiarpakdee et al. (2018)	Correlated metrics	University of Adelaide	Australia	2018	Journal
N. Zhang (2016)	Clustering	University of the District of Columbia	United States	2016	Conference
Y. Zhang et al. (2015)	Cross-project prediction	Zhejiang University	China	2015	Conference
Nam et al. (2017)	Cross-project prediction	University of Waterloo	Canada	2017	Journal
Hosseini, Turhan, & Gunarathna (2017)	Cross-project prediction	University of Oulu	Finland	2017	Journal
Kamei et al. (2016)	Cross-project prediction	Kyushu University	Japan	2016	Journal
Panichella et al. (2014)	Cross-project prediction	University of Salerno	Italy	2014	Conference
Peters et al. (2015)	Cross-project prediction	University of Limerick	Ireland	2015	Conference
Ryu et al. (2016)	Cross-project prediction	Korea Advanced Institute of Science and Technology	South Korea	2016	Journal
Ryu et al. (2015)	Cross-project prediction	Korea Advanced Institute of Science and Technology	South Korea	2015	Journal
Ryu et al. (2017)	Cross-project prediction	Korea Advanced Institute of Science and Technology	South Korea	2017	Journal
Bosu & MacDonell (2013)	Data quality	Auckland University of Technology	New Zealand	2013	Conference
Chetchotsak et al. (2015)	Data balancing	Khon Kaen University	Thailand	2015	Journal
Feng et al. (2016)	Data transformation	University of North Carolina	United States	2016	Journal
Canfora et al. (2015)	Defect prediction	University of Sannio	Italy	2015	Journal
Wahono (2015)	Defect prediction	Dian Nuswantoro University	Indonesia	2015	Journal
F. Zhang et al. (2016)	Defect prediction	Queen's University	Canada	2016	Journal
V. García et al. (2010)	Resampling	Universitat Jaume I	Spain	2010	Conference
Sáez et al. (2015)	Resampling	University of Granada	Spain	2015	Journal
Jojan & Srivihok (2013)	Preprocessing	Kasetsart University	Thailand	2013	Conference
Muresan et al. (2015)	Preprocessing	Technical University of Cluj-Napoca	Romania	2015	Conference
Tavares et al. (2013)	Preprocessing	Federal University of Pernambuco	Brazil	2013	Conference
Wong et al. (2013)	Preprocessing	Hong Kong Polytechnic University	Hong Kong	2013	Conference
Manek et al. (2013)	Preprocessing	Jawaharlal Nehru Technological University	India	2013	Conference
Lai & Leu (2017)	Preprocessing	Shih Chien University	Taiwan	2017	Conference
Idri et al. (2018)	Preprocessing	Université Mohammed V	Morocco	2018	Journal
Roy et al. (2018)	Preprocessing	University of Québec	Canada	2018	Journal
Xiaoli et al. (2015)	Preprocessing	Northeast Dianli University	China	2015	Journal
Benhar et al. (2018)	Preprocessing	Université Mohammed V	Morocco	2018	Conference
Pérez et al. (2015)	Preprocessing	Tecnológico Nacional de México	Mexico	2015	Journal
Fallahi & Jafari (2011)	Preprocessing	Shiraz University	Iran	2011	Journal
Coone (n.d.)	Preprocessing	Universiteit Gent	Belgium	2010	Journal
Bilgin & Camurcu (2010)	Preprocessing	Maltepe University	Turkey	2010	Journal
Rhoads (2011)	Preprocessing	United States patent	United States	2011	Other
Damaceno Borges et al. (2013)	Preprocessing	Federal University of Viçosa	Brazil	2013	Journal
López et al. (2012)	Preprocessing	University of Granada	Spain	2012	Conference
Meadem et al. (2013)	Preprocessing	University of Washington	United States	2013	Conference
Kamiran & Calders (2012)	Preprocessing	Eindhoven University of Technology	Netherlands	2012	Journal
Farquhar & Hill (2013)	Preprocessing	Radboud University	Netherlands	2013	Journal
Fournet et al. (2013)	Preprocessing	Microsoft	United States	2013	Conference
Shanab et al. (2012)	Preprocessing	Florida Atlantic University	United States	2012	Journal
K. Li et al. (2010)	Preprocessing	Hebei University	China	2010	Journal
Singh et al. (2013)	Preprocessing	National Institute of Technology Agartala	India	2013	Journal
Dórea et al. (2013)	Preprocessing	University of Prince Edward Island	Canada	2013	Journal
Kuhn & Johnson (2013)	Preprocessing	Pfizer Global Research and Development	United States	2013	Other
Błaszczyński et al. (2010)	Preprocessing	Poznań University of Technology	Poland	2010	Conference
Mohd et al. (2013)	Preprocessing	Universiti Malaysia Terengganu	Malaysia	2013	Conference
Marston et al. (2011)	Preprocessing	Washington State University	United States	2011	Conference
Hassan et al. (2011)	Preprocessing	University of Nottingham	Malaysia	2011	Conference
Balamurugan & Christopher (2012)	Preprocessing	Thiagarajar College of Engineering	India	2012	Conference
Nithya & Sumathi (2012)	Preprocessing	Manonmaniam Sundaranar University	India	2012	Conference
Verma & Gupta (2012)	Preprocessing	Pandit Dwarka Prasad Mishra Indian Institute of Information Technology, Design and Manufacturing	India	2012	Conference
Baskaran et al. (2010)	Preprocessing	National Cancer Centre Singapore	Singapore	2010	Conference
Florido et al. (2010)	Preprocessing	University of Granada	Spain	2010	Conference
Jayalskshmi & Santhakumaran (2010)	Preprocessing	CMS College of Science and Commerce	India	2010	Conference
Karunaratne et al. (2010)	Preprocessing	Stockholm University	Sweden	2010	Conference
Borghys & Perneel (2010)	Preprocessing	Royal Military Academy	Belgium	2010	Conference
Tsai & Chou (2011)	Preprocessing	National Central University	Taiwan	2011	Conference
Gray et al. (2011)	Preprocessing	University of Hertfordshire	England	2011	Conference
Armah et al. (2013)	Preprocessing	University for Development Studies	Ghana	2013	Conference

CHAPTER 3: RESEARCH METHODOLOGY

This chapter presents the methodology applied while conducting this research. It is important to provide a clear overview of the applied research methodology, not only to specify the steps taken but also to facilitate the understanding of the whole research process. To meet the research objectives, a standard methodological approach was applied in this research. Note that the key objectives of this research are to propose a suitable approach for addressing data-related issues at both the class and method levels of software programs and to propose a technique that can be applied to predict the numbers of defects in a new version of software at both the class and method levels. To achieve the formulated research objectives, an optimal decision approach was applied in each of the three main phases of the standard research methodology, namely, (i) the **planning phase**, (ii) the **proposed approach phase**, and (iii) the **evaluation and analysis phase**, as presented in Figure 3.1.

The flowchart presented in Figure 3.1 illustrates how the various stages of the research methodology are related to each other in terms of the activities performed in the individual phases. The phases of the research methodology are presented at the abstract level in Figure 3.3.

3.1 Planning phase

The planning phase was the initial stage of the research. During this starting phase, the research activities were broken down into individual logical segments to specify how the defined research objectives were to be achieved. Based on the optimal decision made to segment the research methodology phases, a clearer understanding of each phase and their relationships was achieved. This segmentation of the research phases enabled the researcher to effectively plan what was to be done and how throughout the entire research process. Having first identified the research problem, the researcher identified the needs of

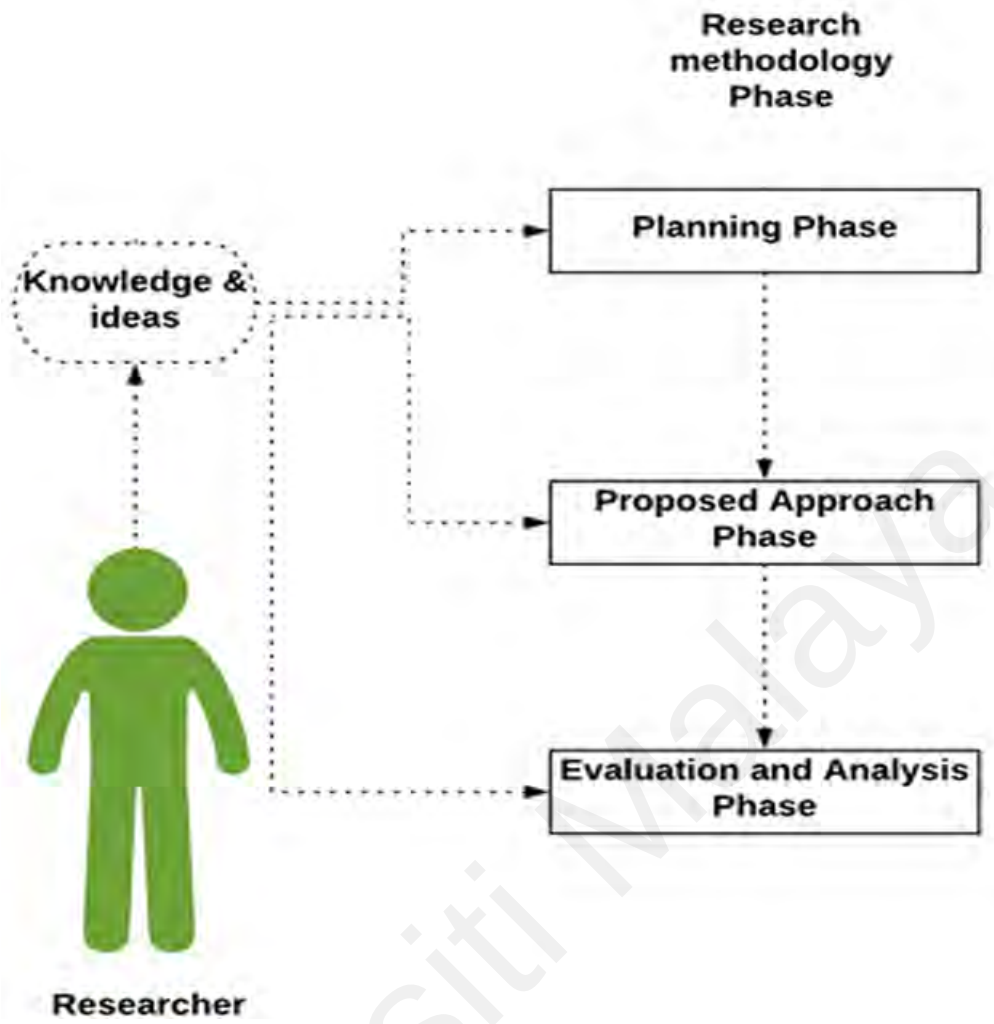


Figure 3.1: Phases of the research methodology

the research, formulated the research questions, formulated the research objectives, selected the electronic databases to be included in the research and clarified the reasons for their selection. Furthermore, the researcher planned the search process and structured a quality assessment of the materials to be included in the research. The researcher also planned the types of instruments to be applied in the data collection and preprocessing procedures, thus providing a basis for structuring the data analysis techniques and evaluating the proposed approach. This planning phase formed the foundation of the research methodology, as presented in Figure 3.1.

3.1.1 Needs of the research

During the planning phase, the needs of this research were identified. These needs are related to how the study is expected to contribute to the body of knowledge in the field of machine learning for software defect prediction. The importance of this research therefore forms the basis for the research significance, as presented in the previous chapter. In addition, the defined key objectives of this study, namely, to propose a framework to address the inconsistencies in the datasets applied in machine learning studies and to propose a technique that can provide actionable outputs to software managers, are in line with the research significance.

3.1.2 Planning the research questions

The research questions formulated in this study are answerable questions related to the scope of this research, with a focus on supervised machine learning. During the planning phase, a set of research questions were formulated to guide the investigation of the preprocessing of class- and method-level datasets as well as defect prediction techniques in the supervised machine learning domain. All research questions were required to possess the following characteristics: clarity, focus, concision, understandability and arguability.

Clarity: The researcher formulated research questions concerning the preprocessing of class- and method-level datasets as well as defect prediction to ensure that there would be no need to seek additional information on the interpretation of any particular research question. The researcher ensured that each research question provided sufficient information to be investigated.

Focus: The researcher avoided irrelevant contents unrelated to supervised machine learning when formulating the research questions; rather, only related research terms were included in the research questions. This was done to allow specific answers to be provided to each research question during the investigation of the processing of class- and method-level

datasets and defect prediction.

Concision: The researcher ensured that the formulated research questions were as simple as possible for ease of understanding.

Understandability: The researcher ensured that each research question would require not only a "yes" or "no" answer but rather an articulation of ideas in response to questions on the preprocessing of class- and method-level datasets and defect prediction.

Arguability: The research questions in this study were formulated to be capable of opening up new areas of research rather than merely accepting previous findings on supervised machine learning. The formulated research questions have been presented in Section 1.4.

3.1.3 Planning the search process, electronic databases and literature review

One of the crucial aspects of the planning phase of the research methodology applied in this study was the selection of reliable electronic databases and search processes to ensure the retrieval of information from reliable sources. Articles from journals and conference proceedings related to data preprocessing as well as articles related to class- and method-level defect prediction were extracted from electronic databases using both automated and conventional manual search processes. A list of the electronic databases searched is presented in Table 3.1. A flowchart of the search process is also presented in Figure 3.2. The search strings were formulated to retrieve as many articles as possible

Table 3.1: List of electronic databases searched

Database	URL
IEEE Explore	www.ieeexplore.ieee.org
SpringerLink	www.link.springer.com
ScienceDirect	www.sciencedirect.com
Web of Science	www.webofknowledge.com
Google Scholar	www.scholar.google.com
Wiley Online Library	www.onlinelibrary.wiley.com
ACM Digital Library	www.dl.acm.org
IET Software Digital Library	www.digital-library.theiet.org
Scopus	www.scopus.com

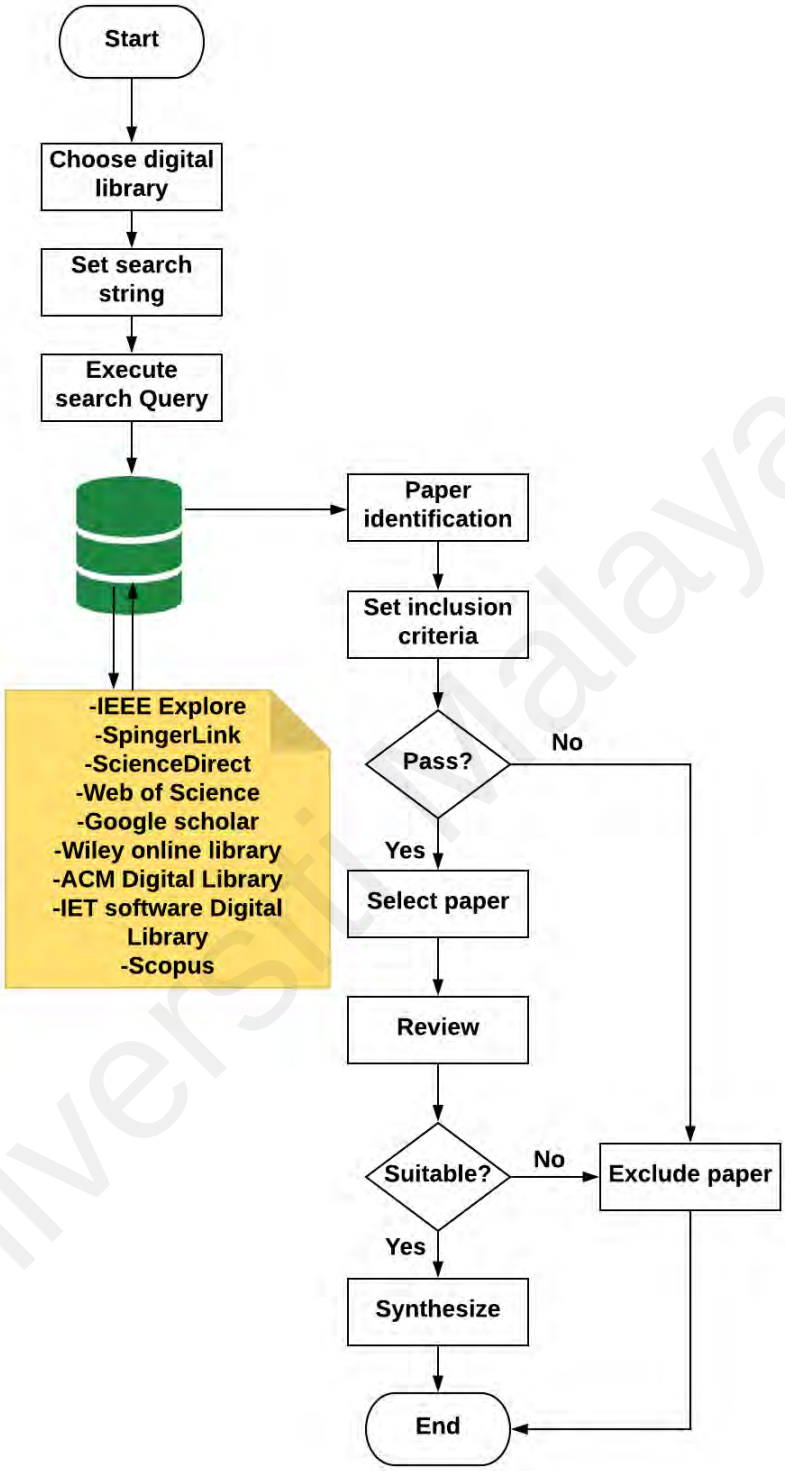


Figure 3.2: Search process flowchart

related to data preprocessing, the topic of interest in this literature review. The following search strings were formulated and applied: *(data OR data set OR data sets OR imbalanced*

data OR imbalanced data set OR imbalanced data sets) AND (preprocessing OR cleaning OR cleansing OR preparation OR sanitization OR purification at class and method levels of software OR program OR application) AND (defect prediction OR estimation OR forecast) AND (Machine learning)

(number OR amount OR quantity OR digits OR figure) AND (new version OR product OR program OR application) AND (defects OR errors OR faults OR failures at class and method levels of software OR program OR application) AND (defect prediction OR estimation OR forecast) AND (Machine learning)

During the literature review stage, relevant academic materials were investigated, such as books, scholarly articles (including academic journal articles and conference proceedings), reports, e-books and other documents relevant to the research problem, available mostly in online databases and from the university library. The relevant documents found in these repositories provided detailed information and evaluations of the work done in the supervised machine learning domain. In addition, for the purpose of establishing the research problem, the researcher performed an extensive literature review. Based on the relevant documents found in the literature, the research problem under investigation was formulated. The relevant documents accessed during the review period were organized so as to facilitate understanding of the sources of the materials reviewed while investigating the research problem as well as to demonstrate to the reader how this research can be applied within the software engineering community. Importantly, the literature review process enabled the researcher to gain an understanding of the academic progress achieved within the research domain and, thus, to uncover gaps in the existing work, which the present research attempts to fill. The researcher also considered the existing approaches proposed in the literature for addressing research problems related to the research problem investigated in this study. Thus, the literature review enabled the researcher to organize,

understand and acknowledge the existing contributions made by other researchers to address similar research problems. From the reviewed documents, the researcher could easily determine the correlations among the different studies and the possibilities for future work suggested in the existing studies. The literature review lasted approximately 6 months. Sufficient time was allocated to review all relevant materials, and at the end of the literature review, the researcher was able to define the research problem. As mentioned previously, this stage of the research methodology enabled the researcher to identify and review previous studies related to the research problem investigated in this study by engaging in a rigorous review process. Ultimately, the researcher acquired an overall understanding of the materials available in various databases to support the formulation of the research problem. The details of the literature review have been presented in Chapter 2.

3.1.4 Clarification of the research problem

The research problem investigated in this study was further clarified after a thorough review of all of the conceptual and empirical literature available within the databases searched. This problem is the main issue to which this research attempts to provide a solution. This problem clarification helped the researcher to focus on the specific research area related to the problem and on possibly suitable solutions. The formulation of this research problem clarifies the key issues associated with class- and method-level software defect prediction to provide guidance in developing and proposing a solution to address this problem. The details of the research problem have been presented in Section 1.2.

3.2 Proposed approach phase

After clarifying the research problem, the researcher then formulated the detailed structure of the proposed approach for addressing the fundamental research problem. The reason for formulating the proposed approach in this way was to guide the research

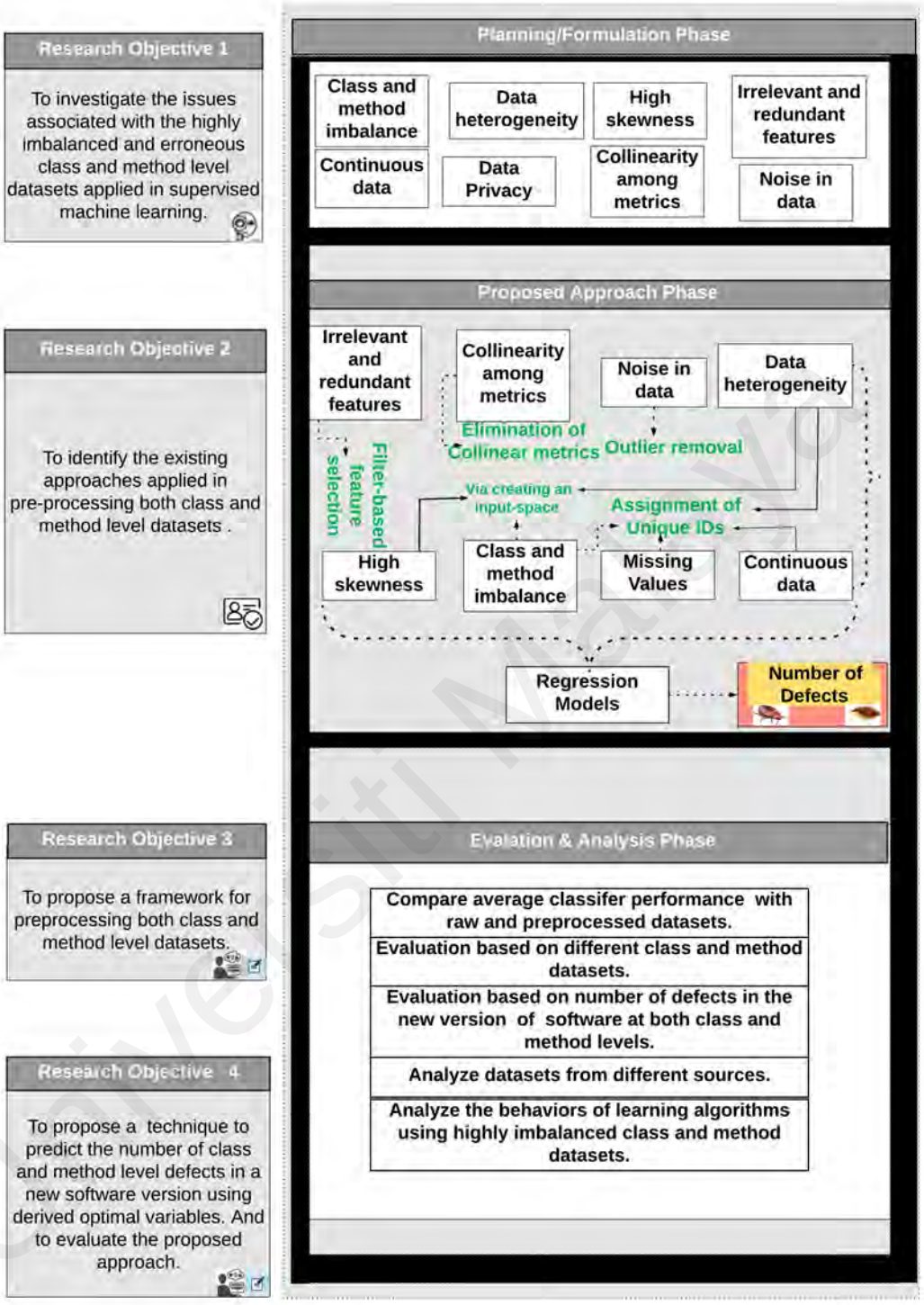


Figure 3.3: Phases of the research methodology at the abstract level

activities to yield an optimal solution to the problem under investigation as efficiently as possible, as presented in Figure 3.3. In essence, the formulation of the proposed approach assisted in the research effort by specifying a means of collecting relevant evidence to

support the research with minimum stress. In other words, the proposed approach provided a framework to ensure the smooth flow of the research process and to determine how the chosen approach could be applied to address the research problem. The approach formulated during the planning phase of this study provided a blueprint for addressing the research problem by accounting for the following:

1. The research problem.

2. A means to address the data quality issues encountered in defect prediction studies, which have been a major concern leading to numerous contradictory research findings in machine learning studies, and a demonstrated approach for predicting the numbers of defects in a new software version at both the class and method levels.

3.2.1 Proposed supervised optimal decision machine learning approach

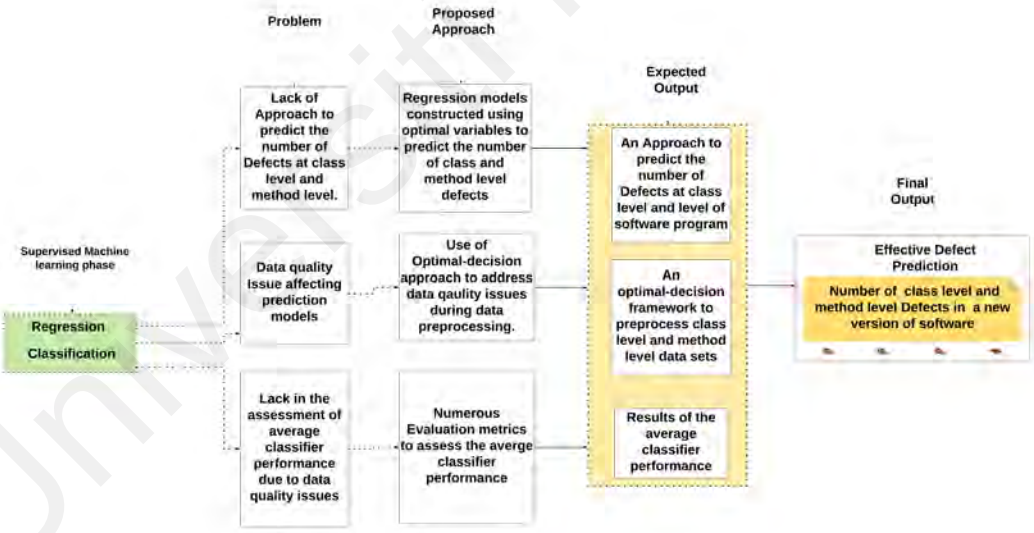


Figure 3.4: Flow diagram of the proposed approach

With the components presented in Figure 3.4, the proposed approach provided the researcher with a comprehensive and well-structured plan to reduce the risk of bias and enhance the reliability of the research findings. Through this well-structured approach, the researcher could achieve optimal results and ultimately provide a means for future

consideration of the research problem being addressed. Ultimately, all phases of the proposed approach were necessary to enable the researcher to meet the objectives of this study while addressing the research problem.

To address the issues related to the quality of the data applied for software defect prediction studies at both the class and method levels, which have been a major concern leading to numerous contradictory research findings in supervised machine learning studies, the researcher applied an optimal decision technique as part of the proposed approach to address these data quality issues through data preprocessing. First, the researcher ensured that every dataset would undergo the same preprocessing stages before it could be considered unbiased. The proposed approach includes a filter-based feature selection phase, in which only relevant features are selected from among the features available in the datasets. This phase ensures that redundant features are eliminated and that only features with high predictive power are selected. The predictive power of each selected feature is determined by applying a feature scoring method.

Second, each dataset is visualized to determine the number of outliers it contains. Outliers are automatically removed from each dataset using widgets provided in Orange 2.7. This ensures that only clusterable data are used throughout the experiment. Thereafter, a unique identifier is assigned to each inlier in each dataset for further preprocessing; the datasets are further thoroughly preprocessed with an input space created to recapture unreported faults. Via this proposed approach, researchers can properly ensure that the datasets applied in a machine learning study are free from bias. In addition, a proper evaluation of the average performances of various classifiers on imbalanced data can be conducted.

The proposed decision-based data preprocessing approach is expected to enable some learning algorithms to independently and accurately learn from properly preprocessed, highly imbalanced data while still generating suitable classification results, thus allowing

them to maintain their average performances. In addition, these classifiers should be able to make optimal classification decisions based on the training conducted with the preprocessed data and thus produce optimal outputs. The proposed approach ensures that incorrect predictions in supervised machine learning are avoided. If not addressed, biased predictions often lead to consequences with high costs (Chawla, 2009; Van Hulse et al., 2007). To reduce costs while achieving unbiased results in a defect prediction study, it is advisable to apply clearly specified procedures such as those of the proposed approach in order to achieve unbiased results at low cost, as reported by Doppa et al. (2014). Second, to address the lack of a demonstrated approach for predicting the numbers of defects in a new software version at both the class and method levels, based on an optimal decision approach, certain optimal variables are first derived as features that are relevant to and correlated with the number of defects. These variables are the defect density, defect introduction time and defect velocity, which have a certain correlation with the number of software defects and thus can be applied in constructing regression models to predict the number of defects in a new version of a software program. However, for the purpose of determining the average classifier performance, a different feature selection technique is applied. For each dataset, the individual rank score of each feature, denoted by $rank_{id}$, is determined to enable the identification of the relevant features and to ascertain their correlations with the target class or target modules (i.e., defective classes or methods) in the dataset. To achieve the above objective, an optimal threshold is set that is equal to the average rank score of all features, denoted by $Rank_{ave}$. Features whose rank scores are equal to or greater than this threshold are selected, and those with lower rank scores are discarded.

For this feature selection procedure, a filter-based selection module has been selected based on an optimal decision. This module provides multiple feature scoring algorithms

to ascertain the relationships between the independent variables (i.e., the features in the dataset) and the dependent variable (the target defective class or method). Among the feature scoring methods offered by the module, the Pearson correlation method is selected to accurately determine the linear relationship between each independent variable and the dependent variable, which is relevant for identifying meaningful features. The Pearson correlation is also used as an indicator to quantify the strength of meaningful features in terms of their correlation.

3.2.1.1 Feature selection

As previously stated, the first preprocessing activity is feature selection. As soon as the datasets are acquired from the repository, feature selection is performed to aid in the accurate selection of relevant features in each dataset. The filter-based feature selection approach applied in this research involves selecting a subgroup of relevant features from among the available features in both class- and method-level datasets. The reasons for performing filter-based feature selection include the following:

1. To ensure that the datasets applied in this research contain only relevant features.
2. To eliminate redundant features from the datasets.
3. To ensure that prediction models are constructed with relevant and simple feature sets.
4. To reduce the time needed to train the learning algorithms.
5. To avoid model overfitting, which may result in biased research findings.
6. To ensure that sparse data in the datasets are avoided.

3.2.1.2 Data visualization and outlier removal

After the selection of meaningful features from the datasets, the data are visualized to clearly identify and remove the outliers in the datasets to address the issue of noise in the data. Another reason for data visualization is to determine the patterns and concentrations

of the defective and defect-free classes and methods in the datasets. Through this data visualization, the impact of the defective classes and methods on software products is revealed. Often, such defective classes and methods may result in catastrophic damage, as reported by Felix & Lee (2017a); this situation confirms the need to predict the numbers of defective classes and methods present in a new software version.

3.2.1.3 Assigning unique identifiers to classes and methods in datasets

As noted earlier, the existing class- and method-level datasets are inherently imbalanced, and the proportions of defective and defect-free classes and methods vary from dataset to dataset. Typically, it is difficult to accurately determine the exact numbers of defective and defect-free classes and methods. Therefore, assigning unique identifiers to all classes and methods in a dataset makes it much easier to identify and track the proportions of defective and defect-free classes and methods. In the proposed approach, based on an optimal decision for addressing data quality issues, unique identifiers are assigned during data preprocessing for the following reasons:

1. For easy identification of defective and defect-free classes and methods in each dataset.
2. To identify and track records of missing values in each dataset.
3. To track outliers in each dataset.

3.2.1.4 Modeling technique

In this research, a modeling technique selected based on an optimal decision is applied to clearly address the research problem. Following the work done by Felix & Lee (2017b), the modeling technique applied in this research is based on Rayleigh's prediction model, which indicates that the numbers of class- and method-level defects in a software product increase over time throughout the Software Development Life Cycle (SDLC). The reasons for basing the modeling technique on Rayleigh's prediction model include the following:

1. To model an approach that can be used to address this research problem.
2. To ensure reliable defect prediction outcomes.
3. Rayleigh's model allows the modeling of the defect density and defect acceleration.
4. Rayleigh's model is based on the statistical defect distribution over time throughout the SDLC and, thus, is far superior to static models.
5. To allow modeling of the entire SDLC.

It is likely that the number of defects in a software product will increase as the software project transitions through the SDLC. Rayleigh's model, which describes the number of defects over time as a software project transitions, implies that the rate at which defects occur over time will lead to an increase in the number of defects, consequently resulting in an increase in the defect density of the software product. The defect density is calculated as the ratio of the number of defects to the project size. Based on this evolution of the defect density, it is confirmed that an increase in the number of defects is caused by the defect rate. The rate in this context represents the acceleration of defect occurrence, which leads to an increase in the defect density. This implies that the rate of defect occurrence is proportional to the defect density (that is, the ratio of the number of defects to the project size is proportional to the change in the defect velocity over the defect introduction time). Therefore, on the basis of the rate of defect occurrence, it is further confirmed that the defect acceleration drives the number of defects in a software product as well as its defect density. Thus, an increase in the defect acceleration will result in an increase in both the number of defects and the defect density of a software product. Acceleration is calculated as the rate of change in velocity over time. Hence, the defect acceleration is calculated as the change in the defect velocity over the defect introduction time. Since the ratio of the number of defects to the project size is proportional to the change in the defect velocity over the defect introduction time, the following expression holds:

$$\frac{no.\ of\ defects}{project\ size} \sim \frac{defect\ velocity\ v}{defect\ introduction\ time\ t} \tag{3.1}$$

Based on the above optimal relationship, the defect density, defect velocity and defect introduction time can be identified as optimal variables since they offer an optimal solution to the problem investigated in this research. Thus, these optimal variables are applied to further preprocess both class- and method-level datasets by determining the defect density, defect velocity and defect introduction time in each dataset. This information obtained from the current version of a software product during the preprocessing phase is helpful in constructing a model for predicting the numbers of defects in a new version at both the class and method levels.

3.2.1.5 Tracking unreported faults in the datasets

The optimal approach applied in this research reveals that not all faults can be captured while performing data analysis during the data preprocessing phase. Therefore, there is a need to ensure that these unreported faults, which were either omitted, hidden or unnoticed during data analysis, are fully tracked and reported. To track these unreported faults, an input space that uses a kernel-based algorithm is created. This kernel-based algorithm ensures that the outcomes in the input space are properly reported. First, the datasets are transformed into a feature representation by means of a similarity function, which is a real-valued function that quantifies the similarity between two objects. The kernel applied in the input space to perform this fault recapturing is the radial basis function (RBF), which will be discussed in the next chapter. Once the unreported faults have been recaptured, the datasets are assumed to be free from bias because an accurate number of faults has been reported.

3.3 Evaluation and analysis phase

This section explains the means of ascertaining the performance of the proposed optimal decision framework. The need to evaluate the proposed approach, to ensure that it is the best approach for overcoming the challenges associated with the existing class- and method-level datasets and developing a suitable means of predicting the number of software defects in the new version of a software product, cannot be overemphasized. To accurately evaluate the proposed optimal decision framework, both null and alternative hypotheses are formulated.

3.3.1 Formulation of the research hypotheses

The researcher has formulated a set of clearly defined research hypotheses to confirm the evaluation outcomes. It should be noted that at this stage, these hypotheses are unconfirmed assumptions. In other words, they may be either true or not until the relevant logical and empirical evidence has been tested and obtained through suitable evaluations. Because of the importance of these hypotheses, their formulation was given careful attention, especially with regard to the parameters used to construct them. These parameters are the variables assumed to be correlated with the research problem investigated in this study. Another reason for giving careful attention to the formulation of the research hypotheses is that they may influence the manner in which computations and evaluations are carried out. The research hypotheses provide clear guidance on the influence of the chosen variables applied in this research and further reveal vital information about the research problem, thus enabling the researcher to focus on the most suitable approach to the research problem. These hypotheses were tested using the evaluation measures specified for various phases in the evaluation diagram presented in Figure 3.5.

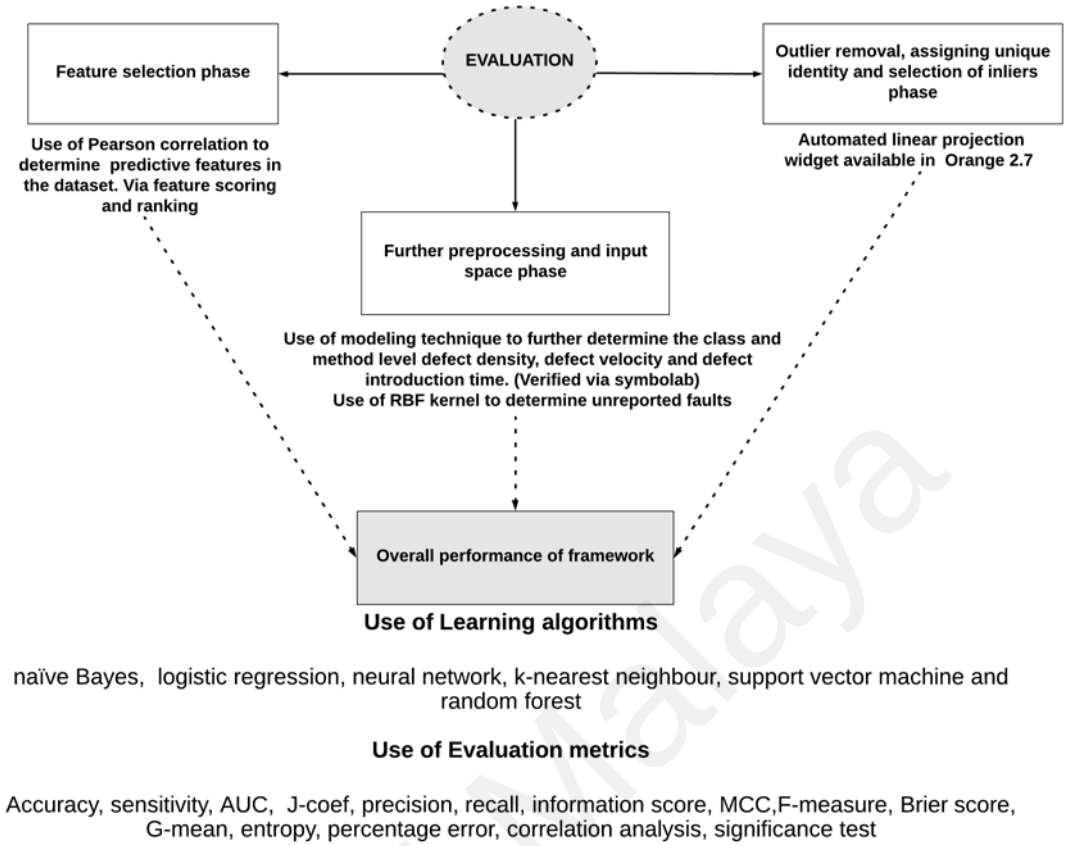


Figure 3.5: Evaluation diagram for the proposed approach

3.3.2 Hypothesis testing

This section introduces the testing of the formulated hypotheses in order to answer the research questions. These assumptions are the initial claims, which then require evaluation to determine their acceptance or rejection. The researcher evaluates each hypothesis by determining the likelihood that the stated relationship between the dependent and independent variables is true or not. The steps involved in hypothesis testing are presented in Figure 3.6.

Statement of hypotheses: The researcher must ensure that both the null and alternative hypotheses are clearly stated. The null hypothesis represents the first assumption in terms of the likelihood of acceptance, whereas the alternative hypothesis represents the way in which it is believed that the null hypothesis could be wrong.

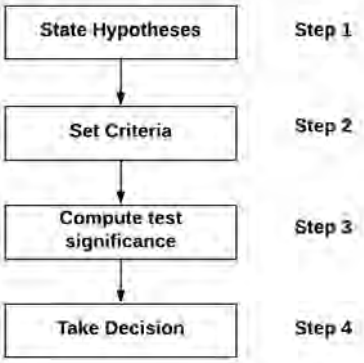


Figure 3.6: Steps of hypothesis testing

Setting of criteria: In this step, the researcher sets criteria to serve as a benchmark for decision-making. These criteria are defined to help determine whether the null hypothesis should be accepted or rejected. The set criteria are then compared against the results obtained in the computation step.

Computation of test significance: Significance tests are performed to determine the values to be compared with the values of the criteria set in Step 2.

Decision-making: A decision is made based on the values obtained in the computation phase. These values are compared against the benchmark defined to determine whether the null hypothesis should be accepted or rejected.

Feature selection phase: The feature selection phase is the first phase in the proposed optimal decision framework. Relevant features are selected, and irrelevant features are discarded. To evaluate the outcome of the filter-based feature selection applied in this phase, the Pearson correlation is applied to identify and select features with high predictive power.

Outlier removal, unique identifier assignment and inlier selection phase: This phase is executed using an automated approach available in Orange 2.7 to accurately determine the outliers in the class- and method-level datasets. The automated widgets provided in Orange 2.7 make it much easier to eliminate outliers and assign unique

identifiers before further preprocessing is performed on the inliers.

Further preprocessing and input space phase: The inliers are further preprocessed to determine the defect density, defect velocity and defect introduction time. To ensure thorough preprocessing, this phase is executed by means of a modeling technique to determine the relationships between these optimal variables (defect density, defect velocity and defect introduction time) and the numbers of class- and method-level defects.

Overall performance of the framework: To assess the overall performance of the proposed optimal decision framework, several learning algorithms, namely, naïve Bayes, logistic regression, neural network, K-nearest neighbor, support vector machine and random forest algorithms, were trained on the preprocessed training datasets. The performances of these learning algorithms were evaluated in terms of the accuracy, sensitivity, area under the receiver operating characteristic (ROC) curve (AUC), precision, recall, information score, percentage prediction error, entropy and geometric mean (G-mean). Furthermore, the performance of the framework was assessed by comparing the results achieved using the preprocessed datasets with those achieved using the raw datasets based on the learning algorithms and evaluation metrics listed above.

3.3.3 Instrumentation

To successfully test the hypotheses stated above, it was essential to select appropriate instruments to conduct the experiments in order to achieve the research objectives and avoid misinterpretation of the results. These instruments were applied in various phases of the proposed framework to evaluate the performance in every phase. For instance, to accurately select suitable features from the existing datasets, the researcher applied Microsoft's Azure Machine Learning Studio, which enabled filter-based feature scoring to guide the selection of relevant and predictive features. Thus, the researcher evaluated the performance of this feature scoring and ranking approach as a means of evaluating

the initial phase of the proposed data preprocessing framework. For outlier visualization and removal, the researcher applied automated widgets provided in Orange 2.7, which enabled the automatic elimination of outliers and the assignment of unique identifiers to the classes and methods in the datasets. Thereafter, the actual number of outliers in each dataset could be evaluated via linear projection. To further preprocess the class- and method-level datasets, the researcher applied a mathematical modeling technique to derive the defect density, defect velocity and defect introduction time from the current class- and method-level datasets. An open-source software tool called Symbolab was used to verify the accuracy of the mathematical modeling technique applied to derive the aforementioned optimal variables. Other statistical tools applied in this research include Microsoft SPSS and Microsoft Excel to perform analysis of variance (ANOVA) and to determine the correlations between the derived variables and the numbers of class- and method-level defects.

3.3.4 Experimental design

To achieve the objectives of this research, the experimental planning itself was of paramount importance. For the experimental design, the researcher carefully planned the types of tools to be used and the systematic collection of the data. An appropriate experimental design will enable accurate evaluation that will result in either acceptance or rejection of the research hypotheses.

3.3.5 Data collection

This research relied on secondary data collection. The data used in this study were collected from widely known repositories for supervised machine learning research. Notably, most of the reliable previous research in this field has also relied extensively on such secondary data collected by previous researchers, for instance, Elish & Elish

(2008), Q. Song et al. (2011), Z.-W. Zhang et al. (2017) and Qiao et al. (2017). However, this study did not rely on already preprocessed data; instead, the researcher applied the proposed optimal-decision-based data preprocessing technique to clean and preprocess the data to ensure the accuracy and reliability of the datasets applied in this study. The data collected for this research constitute the major factor affecting the ability to achieve the research objectives and answer the research questions. The means of data collection should enable the acquisition of concrete evidence that can yield reasonable answers to the research questions and can also be analyzed to provide a means of addressing future research problems. This research relied on the widely used National Aeronautics and Space Administration (NASA) datasets and the ELFF datasets. The NASA datasets are available from the PROMISE repository, while the ELFF datasets can be found at www.elff.org.uk/ESEM2016.

These datasets were used to ensure that the research findings would meaningfully contribute to the body of knowledge within the machine learning community and that the results obtained in this research could be readily compared with those in the existing literature. These datasets enabled the researcher to conduct experiments, validate the stated hypotheses, provide answers to the research questions and compute the experimental outcomes. The following factors were considered during data collection: reliability, suitability and adequacy.

Data reliability: To assess the data reliability, the researcher considered the following factors: the sources of the data, the person or group of persons who collected the data, the method of data collection, and the period of time during which the data were collected. The researcher also considered whether there was a possibility of bias in the measurement tools applied during data collection. With regard to the data sources, the researcher investigated the projects from which the data were collected. There was strong evidence in the literature

to support the sources of the datasets applied in the present research. Furthermore, the reliability of the individuals or groups responsible for data collection was investigated. This investigation proved that the people responsible for the collection of the data from their sources were reliable and that their goal was to make these repositories available to the software engineering community (Sayyad Shirabad & Menzies, 2005). Strong evidence for this reliability was confirmed by the quality of the journals in which similar research studies using these datasets were reported, such as the studies of Elish & Elish (2008), Q. Song et al. (2011), Z.-W. Zhang et al. (2017) and Qiao et al. (2017). Regarding the data collection procedures, reliable automated approaches with noteworthy accuracy were reported as the data collection methods. Regarding the periods of time during which the data were collected, for the NASA datasets, this period ranges from 2007 to 2018, whereas the time period for the ELFF data ranges from 2016 to 2018. Both sets of datasets were applied in validating the reliability of this research. Regarding the issue of bias in the measuring tools, the researcher was able to identify some erroneous data among the datasets. These errors in the datasets, which were identified during data cleaning, may have been the result of bias in the measuring tools applied. The errors could have arisen due to either human or computer faults. Despite these errors, the data show a considerable level of accuracy. For instance, the number of projects recorded was confirmed to be accurate for both sets of datasets during preprocessing.

Data suitability: Before these secondary data were collected, the researcher first ensured their suitability for addressing the research problem considered in this study. Datasets that were found to be unsuitable were not considered in this study; only suitable data were used. The suitability of these datasets makes it possible to carry out experiments with these data, analyze the data and compute the results. As noted in the previous section, several past studies on defect prediction have relied on the datasets considered in this

research, and as such, the suitability of these datasets for our purpose is confirmed by the literature.

Data adequacy: Since the level of accuracy found in these datasets was remarkable, the researcher concluded based on the evidence gathered from the literature that these datasets are suitable for use in the present research. The researcher ensured that these secondary data are precise and are strongly correlated with the research problem addressed in this study. The researcher thus concluded that both the NASA and ELFF datasets are reliable, suitable and adequate for this research. More information on the data collection process can be found in Chapter 4.

3.3.6 Data analysis

Data analysis is a process that involves the systematic application of statistical or logical techniques to evaluate data. Analyzing data can enable researchers to draw meaningful conclusions and to accurately report the behavior of the data attributes. To enforce the integrity of the data used in this research, the researcher applied suitable data analysis to accurately report the research findings to the best of his knowledge. To support the integrity of the findings, the researcher reports how the information obtained from the data was analyzed. The information thus obtained can not only assist in providing answers to the research questions but also assist the software engineering community in decision-making during software testing, with an emphasis on software defect prediction.

Certain statistical techniques and tools are used to ensure the accurate presentation of information relevant to this research. First, tables are used to clearly summarize related studies addressing data quality issues, including their countries of origin, the associated institutions and the years when these studies were published. In addition, the various techniques applied for data preprocessing are summarized in a table. Graphs are used to present the behavior of and relationships between the derived optimal variables and

the number of defects as well as to present the behavior of the learning algorithms on imbalanced class- and method-level datasets.

3.3.7 Threats to validity

In the course of searching for answers to the research questions and achieving the research objectives, unintentional errors might have occurred. Therefore, some threats to the validity of this study exist that may have affected the results. The threats to the validity of this research include construct, internal and external threats. Details of these threats to validity will be presented in Chapter 6 (Results and Discussion).

3.4 Summary

This chapter has presented the step-by-step research methodology applied in this research. The first of these steps was the planning phase, in which the researcher specified the research needs and formulated the research significance to ensure that the objectives of this research would be met. The research questions investigated in this research were also formulated in the planning phase, on the basis of a rigorous literature review. The second phase of the research methodology was the proposed approach phase. Accordingly, an optimal decision framework for preprocessing both class- and method-level datasets applied in machine learning studies has been developed and presented. This research also demonstrates how this framework can be applied in predicting the numbers of class- and method-level defects in new versions of software.

Finally, the proposed framework was evaluated based on its four key phases: the feature selection phase, which was evaluated on the basis of feature scoring and ranking; the outlier removal phase, which was evaluated via the automated linear projection tool available in Orange 2.7 (Demšar et al., 2013); additional data preprocessing, which was evaluated by means of a mathematical modeling technique; and the overall framework performance,

which was evaluated through a comparison of the results obtained via training on raw and preprocessed datasets. This comparison was achieved by evaluating the performances of the various learning algorithms applied in this research. The performance comparison of these algorithms was conducted by training these algorithms on raw and preprocessed datasets, individually, and evaluating the results on the basis of various evaluation metrics, namely, the accuracy, sensitivity, AUC, J-coefficient, precision, recall, information score, Matthews Correlation Coefficient (MCC), F-score, Brier score, G-mean and entropy.

CHAPTER 4: PROPOSED APPROACH

This chapter presents the proposed optimal decision approach applied to address the problems identified in the previous chapters. Before presenting the proposed approach, the research problems are first analyzed to ensure that the proposed approach is suitable to address them. The problems addressed in this study include the lack of an appropriate preprocessing technique for class- and method-level data and the lack of an appropriate technique for predicting the numbers of class- and method-level defects in a new version of software in advance, as presented in Section 1.2. Based on the optimal decision-making that led to the proposed approach, several variables (referred to as optimal variables due to their influence on the numbers of class- and method-level defects) have been derived, namely, the defect density, defect velocity and defect introduction time. The essential purpose of deriving these optimal variables is to ensure that their underlying influences on the numbers of class- and method-level defects are investigated and revealed. In addition, these optimal variables can be applied not only in preprocessing both class- and method-level datasets but also in constructing regression models to predict the numbers of class- and method-level defects in a new version of a software program.

4.1 Problem analysis

After a rigorous investigation of the literature, the researcher can confirm that most defect prediction models applied in machine learning studies appear to be ineffective due to the challenges these models face. These challenges can arise before, during or after model construction as a result of data quality issues.

However, many attempts to improve the performance and accuracy of these learning algorithms have been made; for instance, studies performed by X. Jing et al. (2015), Shepperd et al. (2014), M. H. Rahman et al. (2016), Taba et al. (2013), Taba et al. (2013),

and Herbold (2013) have reported improved performance of learning algorithms. However, despite their high performance, the existing prediction models still produce biased results when applied in machine learning studies. As a result of these biased findings, some controversies exist with regard to the accuracy of the defect prediction models used in existing machine learning studies. These bias-related controversies contribute to the current lack of a reliable prediction model that can be used to predict the number of defects in a new software version. This lack is reflected by the conflicting results reported in the existing literature, which can be traced to the following root causes: the poor quality of the class- and method-level data applied in defect prediction studies and poor model construction due to the lack of appropriate variables that are highly correlated with the number of software defects. Moreover, data quality issues remain an important factor in defect prediction studies because the outcomes of such studies depend on the quality of the data applied.

4.1.1 Emphasis on data quality

The backbone of every prediction model is the quality of the data applied. In practice, every organization or industry relies on information acquired from datasets available within that organization or industry to aid in better decision-making. As existing studies have proven, however, the quality of much of this data is poor, which is an issue that has yet to be fully addressed. Therefore, it is important to accurately analyze how the existing poor-quality datasets affect industries and organizations. These negative impacts include but are not limited to the following: (1) distrust of the findings obtained based on these poor-quality datasets and (2) potential high costs incurred within an organization.

4.1.1.1 Distrust of findings based on poor-quality datasets

Most decision-makers base their decision-making on information obtained from available datasets. However, if one cannot trust these datasets due to their inconsistent quality, there is a high risk of inaccurate and biased decision-making (Rob Manser, 2016). When a biased or inaccurate decision is made, additional effort will be required to investigate its cause, which may ultimately lead to re-evaluation of the datasets applied during the decision-making process. As discussed in Section 1.1, organizations may suffer enormous losses as a result of software defects; hence, the data extracted in such scenarios must be reliable to enable the construction of reliable prediction models to prevent the recurrence of such events. If the datasets obtained from such catastrophic losses are not reliable, then there will be even a higher risk that the models constructed with such data will produce misleading results. Moreover, if decisions are made based on biased and unreliable datasets, especially when organizations have placed their confidence in such decisions, distrust is likely to set in, which can result in a loss of customer confidence and loyalty (Akbar & Parvez, 2009; Nguyen et al., 2013). Distrust among customers as a result of poor data quality can also diminish the availability of resources within an organization if not addressed. One of the ways to address such a situation while ensuring that available datasets are free from bias is through the optimal decision framework proposed in this research, which can address most relevant data quality issues.

4.1.1.2 High cost implications

There are always high costs associated with correcting the damage that occurs as a result of applying poor-quality data in decision-making (Rob Manser, 2016). For instance, with a cost-effective data preprocessing approach, it may cost an organization 1 dollar to prevent poor data quality. On the other hand, it may cost the same organization 100 dollars to correct the damages caused as a result of poor data quality. Moreover, if this

remediation is not achieved in a timely fashion, the consequences may incur thousands of dollars of additional costs or may even result in catastrophic damage from which the organization may be unable to recover. Figure 4.1 illustrates the relative cost of correcting defects throughout the Software Development Life Cycle (SDLC) and shows that the cost of preventing defects is always less than the cost of correction over time.

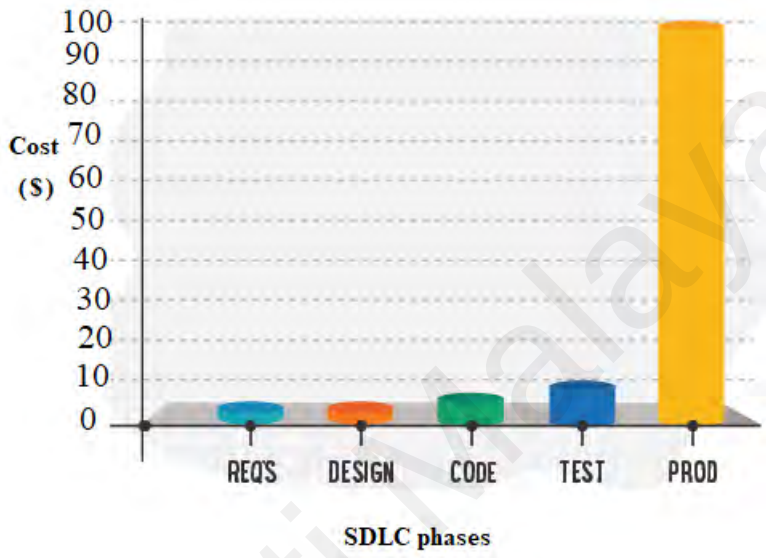


Figure 4.1: Relative cost of correcting defects (LaTonya Pearson, 2014)

If poor-quality data are applied in constructing prediction models to aid in decision-making, the outcomes of such models will lead to the following results:

a. Reduced customer satisfaction and loyalty.

b. High loss in the organization's income/profit.

c. Reduced productivity of the organization.

d. High consumption of revenue and a high cost of maintenance.

e. Distrust in the organization, leading to a loss of customers.

It is therefore important to note that class- and method-level datasets applied in defect prediction studies are, by default, inconsistent due to their imbalanced nature. As such, classification algorithms may produce inconsistent results due to this method and class imbalance. As a result of these inconsistencies, many conflicting and questionable findings

arise in the field of machine learning, and this situation requires urgent attention. If these problems are not addressed, the outcomes of research on defect prediction will continue to face criticism. It is also important to note that these inherent issues result in erroneous data that, when applied in model construction, can lead to model overfitting and concept drift, causing the constructed prediction models to perform poorly (Hosseini, Turhan, & Gunarathna, 2017). In addition, if a prediction model is trained using erroneous data, the model will tend to produce biased results since the outcome of a prediction model depends on the quality of the data on which it is trained. Consequently, it is essential to adequately preprocess these data before using them to train any prediction model to avoid misleading results. To address this need, a preprocessing framework is proposed in this research to supplement the existing preprocessing techniques available in the literature. The proposed framework is needed because data preprocessing, which is performed before the construction of learning models to prepare reliable input datasets (Haixiang et al., 2017), is a crucial concern in machine learning research. As a fundamental phase of machine learning studies, data preprocessing requires the understanding, identification and specification of data-related issues as well as a knowledge-based approach that can be used to address these issues and thus make data more reliable for use in machine learning (S. Zhang et al., 2003). Notably, there is clear evidence that data preprocessing can impact the predictive performance of learning algorithms (Crone et al., 2006). If data are properly preprocessed, researchers can accurately identify and report the number of defects found in their data, thereby making their datasets more suitable for enabling learning models to learn independently and accurately from unbiased data to produce reliable prediction results to aid in decision-making. The credibility of defect prediction is one factor influencing data quality (Hall et al., 2012; Liebchen & Shepperd, 2016; Hosseini et al., 2016; Hosseini, Turhan, & Mäntylä, 2017). Thus, it is necessary to focus on eliminating errors from a

dataset by means of preprocessing rather than on altering the original dataset, which may lead to the introduction of additional errors. Such alteration of the original datasets has been identified as a weakness of the existing solutions offered in the literature to address data quality issues. The removal of errors from an existing dataset can be achieved through data preprocessing, which can be defined as the process of making the original dataset less erroneous and more suitable for use in training learning algorithms to make unbiased predictions.

4.1.1.3 Emphasis on learning algorithms

Notably, when learning algorithms are trained with unbiased data, such algorithms tend to produce reliable results. It is also important to note that with the existing learning algorithms, no approach has been demonstrated that can be applied to predict the number of defects in a new software product prior to testing. Consequently, these prediction models have not offered actionable outputs for use in the software industry. One reason for this lack is that the existing prediction models have not been built using metrics that are significantly correlated with the number of defects found in software systems. Therefore, it is crucial to bridge this existing gap by means of the optimal decision approach proposed in this research. In addition to the quality of the data, the metrics applied when constructing prediction models can also play an important role in determining the ability of the resulting regression models to predict the number of software defects. For instance, the defect density is an important metric in a software system but has not been fully utilized in model construction for defect prediction studies. The number of defects in a software project can be characterized by its defect density, which is defined as the ratio of the number of defects to the size of the project. If prediction models are not constructed with metrics that exhibit a sufficient correlation with the number of defects, then such models can produce misleading results. Therefore, it is important to consider meaningful metrics

when constructing defect prediction models since the existing defect prediction models are unable to estimate the number of future defects in a new software product. This lack could also be the result of the unavailability of optimal metrics suitable for constructing such models. If a prediction model is constructed with irrelevant metrics, this can similarly result in false prediction outcomes, which can lead to software failure. If such a situation occurs, the cost of correcting the software can be high. Consequently, biased prediction outcomes can also lead a software team to deliver a defective software product to the end user, which can reduce customer satisfaction (Koru & Liu, 2005). Notably, optimal metrics for the construction of defect prediction models are currently lacking, and as a result, the findings obtained in defect prediction studies continue to face challenges. To address this lack, the researcher has identified such optimal metrics, namely, the defect density, defect velocity and defect introduction time, which are real-time metrics with considerable influence on the number of defects found in a software product. These metrics enable the prediction of the number of defects in a new software release due to their influence on and correlation with the number of defects. However, these real-time metrics have not yet been utilized in the existing literature for the construction of prediction models, either because these optimal metrics have simply been neglected or because they are expensive (Ramler et al., 2014). The inability to utilize such metrics has impacted the outcomes of most prediction studies in the field of machine learning. Supervised machine learning requires techniques for selecting appropriate features from existing datasets to identify the most suitable subset of features for solving data-related issues (Hossain et al., 2016). It is known that the existing datasets contain irrelevant features, which can affect the performance of learning algorithms; consequently, a proper means of selecting the most relevant features is needed (Kwak & Choi, 2002; Gu et al., 2016). With the proper selection of relevant features, learning algorithms can be trained on these features to yield optimal

results in machine learning studies. By the same token, irrelevant features can affect the accuracy of learning algorithms. Therefore, to encourage unbiased prediction outcomes, classification algorithms should be trained on an ideal set of metrics characterized by relevant information about the expected outcome. Selecting the most relevant attributes thus simultaneously removes irrelevant features from a dataset and enables better data management and understanding; for this reason, the choice of metrics is essential when handling inconsistent datasets (Kwak & Choi, 2002; Gu et al., 2016).

The quality of data, in terms of accuracy, consistency and completeness, must be enhanced before attributes can be successfully extracted from such data; such enhancement will assist researchers in producing unbiased results (Ramler et al., 2014). Again, proper data enhancement can be achieved through data preprocessing to ensure that a dataset is properly cleaned and complete before learning algorithms are applied. Thus, proper preprocessing will make it possible to extract optimal metrics that can be applied in model construction, which can greatly support decision-making. Notably, as a result of the inconsistencies that exist in the data and metrics applied in machine learning studies, it is difficult for most researchers to accurately report the methods applied when performing data cleaning, and these inconsistencies have also led to conflicting practical results in machine learning studies (Shepperd et al., 2014).

Therefore, it is necessary for researchers to report in detail the measures applied during data preprocessing (Felix & Lee, 2017b). One suitable means of achieving better data preprocessing is through an optimal preprocessing approach characterized by the optimal decision procedure proposed in this research. Clearly, the existing literature lacks such an optimal data preprocessing approach. To bridge this gap, a data preprocessing framework is proposed in this study to address some of the challenges associated with typical class- and method-level software datasets. An optimal decision procedure was used to design

this data preprocessing framework, resulting in a prime advantage in ensuring effective data preprocessing. This approach can lead to knowledge discovery in defect prediction studies and possibly open up new areas of research.

Notably, the presence of defects in software products markedly reduces software performance. Consequently, such defects make it difficult for the user to understand how a software product works. Furthermore, the defects present in a software system can be expensive to eliminate and can result in a high cost of maintaining the software product (D'Ambros et al., 2010). These defects can lead to software malfunction and, ultimately, to failure. Such failures caused by defects can result in the unavailability of software systems (Sullivan & Chillarege, 1991). The defects in a software product can limit both its dependability and its security (Pereira et al., 2016).

Often, defects in a software system persist even after extensive quality assurance procedures. Due to the issues raised here, it is extremely difficult to develop a generally applicable defect prediction model that can provide actionable outputs to software teams by predicting the number of defects in a new software version. Through extensive research efforts, previous studies have demonstrated various achievements with regard to the prediction of software defects. Nevertheless, additional research on software defect prediction is still required to produce a pragmatic prediction model for managers and development teams. To the best of our knowledge, a model that can successfully predict the number of possible defects in an upcoming product release has not yet been proposed for either cross-project or within-project defect prediction at either the class or method level in software, with the exception of work done by Felix & Lee (2017b).

To date, the underlying factors responsible for an increase in the number of defects in a software product (for example, the defect acceleration, as characterized by the defect velocity and defect introduction time) have not been fully considered in defect prediction

studies. In view of the fact that the existing literature has not demonstrated a way to bridge this gap, this research therefore aims to address this need by presenting a cost-effective data preprocessing approach for enhancing the quality of the data applied in defect prediction studies while revealing the factors that contribute to an increase in the number of defects in a software product. Thereafter, it is demonstrated how the relationship between the defect acceleration and the number of defects can be useful in predicting the number of defects in a new software version, as presented in Section 4.2.2.

4.2 Proposed optimal decision approach

This section presents how an optimal decision-making approach can be applied in preprocessing datasets to be used in machine learning studies with the aim of addressing the research problem. This research suggests that such an approach can be advantageous for ensuring suitable preprocessing of class- and method-level datasets and reliable prediction outcomes. Additionally, the researcher recommends that such an optimal decision-making approach be applied in the selection of both the sets of evaluation metrics and the learning algorithms applied in machine learning studies to aid in the proper assessment of algorithm performance. It is often wise to choose the factors that can lead to optimal results when such a choice is available (Goldberg, 2016); hence, this study offers such a choice. The term **optimal decision**, as applied here, refers to a decision made in every stage of data preprocessing to determine a suitable setpoint that can lead to an optimal result. Here, the term **setpoint** refers to setting specifications that can result in a prime advantage. First, a decision is made to apply filter-based feature selection to select suitable features from the datasets before further preprocessing.

Feature selection is the process of selecting a subset of appropriate features that can help machine learning models produce reliable results. One advantage of feature selection is that it can reduce noise in the data, which can also lead to an improvement in model

performance. If proper feature selection is not performed, the following issues are likely to affect the performance of the learning algorithms:

1. Redundant or irrelevant features in the datasets will add no value for model performance.
2. No relevant information regarding model performance can be obtained if the datasets contain redundant features; rather, such feature redundancy can increase the time required to train the learning algorithms.
3. Multicollinearity can arise in the presence of redundant features.

Therefore, it is important to make an optimal decision to address the above concerns through reliable and effective filter-based feature selection. More concretely, based on the decision made, every attribute or feature d selected (note that the terms 'attribute' and 'feature' are used interchangeably throughout this thesis) from a dataset DS will result in an outcome a as follows:

$$d : DS \rightarrow a = fd \tag{4.1}$$

In other words, the outcome a is a function of the decision made during the selection process. Here, the term **outcome** refers to the result of a decision or choice. For every selection made, multiple sets of possible outcomes may exist, represented as a set of possibilities P that may be advantageous for decision-making. We can assign a prime advantage, denoted by $P'_a a$, to this set of possibilities P. The term **prime advantage** in this context means a highly relevant or significant outcome of a decision or choice. To ascertain the prime advantage of a set of possibilities, reliable information must be extracted about both the decision made and the relevance of the outcome based on that decision.

The relevance of the outcome should be positive, meaning that it contributes to addressing the research problem; if it is not, then a new decision will be made. Thus,

the prime advantage of the decision to select a particular attribute can be represented as follows:

$$P'_{DS}d = P'_{a}fd \tag{4.2}$$

The prime advantage P' is a function of the selected attribute and thus may vary from one selected attribute to another depending on the strength and correlation of each; consequently, it is possible to identify an optimal attribute based on its predictive power in terms of a rank score. Such a rank score is a measure of the relevance of the selected attribute. The optimal attribute is denoted by $d_{optimal}$. Here, the term **optimal** refers to the most advantageous feature, or the feature with the highest relevance. The optimal attribute selection as a function of the feature subset d_s is expressed as follows:

$$d_{optimal} = \arg\max_{d_s \in DS} P'_{DS}d_s \tag{4.3}$$

where $arg\,max$ denotes the argument of the maximum, representing the point at which the optimal value is achieved.

Notably, the advantage of each selected attribute is important, and this advantage is not compromised in this research. To avoid bias in feature selection, the average rank score on each dataset is determined and expressed in the form of an objective function, which is used as a selection criterion for every feature subset. The objective function is described as follows. Given a set of features d, first, it is important to determine the strength of each feature in terms of correlation as characterized by its rank score. The outcome a based on the rank score of each feature is then compared with the objective function, denoted by f_{ave}. Any feature whose rank score is equal to or greater than the average score is selected; otherwise, the feature is discarded. Considering the prime advantage $P'_{DS}d_s$ as a function of the selected feature subset d_s, the expected prime advantage P'_{exp} can be expressed as

follows:

$$P'_{expDS}d_s = P'f_{ave}d_s \tag{4.4}$$

The objective function f_{ave}, which represents the average rank score, is calculated as follows:

$$f_{ave} = \frac{1}{n}\sum_{i=1}^{n} d_n = \frac{1}{n}(d_1 + d_2 + \ldots d_n) \tag{4.5}$$

where

n = the total number of features in the dataset,

f_{ave} = the average rank score, and

d_i = the rank score of the i-th individual feature.

Hence, the optimal attribute $d_{optimal}$ depends on the objective function f_{ave}, which is reflected in the expected prime advantage P'_{exp} of the given feature subset, $P'_{expDS}d_s$. Therefore, the optimal attribute can be selected as follows:

$$d_{optimal} = \arg\max_{d_s \in DS} P'_{expDS}d_s \tag{4.6}$$

Similarly, an optimal decision-making approach can be applied in the selection of both evaluation metrics and classifiers. Every metric m selected from the set of evaluation metrics SM will result in an outcome a as follows:

$$m : SM \rightarrow a = fm \tag{4.7}$$

The purpose of applying optimal decision-making in metric and classifier selection is to properly determine the average performance of the learning algorithms. For optimal

selection among a set of evaluation metrics, we have the following expression:

$$m_{optimal} = \arg\max_{m \in SM} P'_{expSM} m \tag{4.8}$$

To ensure satisfactory assessment of the average classifier performance, this optimal decision-making approach was applied in this study to select from among the following evaluation metrics: Classification Accuracy (CA), precision, recall, Matthews Correlation Coefficient (MCC), area under the receiver operating characteristic (ROC) curve (AUC), Brier score, and entropy. The decision to select these metrics was made because the information obtained from these metrics can be used to assess classification algorithm performance, and this research also suggests that these metrics can provide an overall assessment of the performance of each classifier (X. Jing et al., 2015).

In addition, in a manner similar to that expressed in Equations 4.1-4.4, every classifier c selected from the set of classification algorithms SC will result in an outcome a as follows:

$$c : SC \rightarrow a = fc \tag{4.9}$$

The corresponding expression for optimal classifier selection is

$$c_{optimal} = \arg\max_{c \in SC} P'_{expSC} c \tag{4.10}$$

As defined earlier, the prime advantage is the relevance or significance of an outcome based on a decision or choice. By means of the proposed framework, the following prime advantages can be achieved when addressing various data-related issues:

1. Using the filter-based feature selection approach in the proposed framework, the relative proportions of relevant and irrelevant features in a dataset can be revealed. Then, the

relevant features can be selected to eliminate irrelevant and redundant features.

2. The data visualization phase of the proposed framework enables the identification and elimination of outliers and noise present in a dataset. Outliers in a dataset can negatively impact the outcomes of prediction models (Hosseini, Turhan, & Gunarathna, 2017).

3. To facilitate the optimal decision approach described above, in the proposed framework, a unique identifier is assigned to each module in each dataset. These unique identifiers enable the determination of the level of skewness within a dataset and the accurate identification of the defective classes in each dataset.

4. Data resampling, corresponding to the creation of an input space for capturing unreported faults, is considered in the proposed framework to identify instances that might have been omitted in the initial data sampling phase to ensure that the data are accurately preprocessed to achieve unbiased outcomes.

5. The optimal decision approach enables an appropriate feature scoring method for determining the strength of each feature in terms of correlation.

6. Through the proposed framework, the average performances of classification algorithms can be determined.

7. The proposed framework enables the mutual information due to entropy within a dataset to be assessed.

4.2.1 Proposed optimal decision framework

The proposed data preprocessing framework is based on optimal decision-making to ensure the proper preprocessing of datasets, the accurate identification of the levels of imbalance in the datasets and, ultimately, the generation of unbiased datasets that are suitable for training learning algorithms, as presented in Figure 4.2. For learning algorithms to learn accurately, the datasets on which they are trained must be accurately preprocessed. The selected data preprocessing technique p should be chosen to improve

the quality of the data while addressing the research problem. Then, the performance of the classification algorithms will depend on the data quality, which, in turn, will ultimately depend on the preprocessing technique applied. Thus, the outcome a of preprocessing is a function of the decision regarding the selection of the preprocessing technique. Every technique p chosen from the set of preprocessing techniques SP will result in an outcome a as follows:

$$p : SP \rightarrow a = fp \tag{4.11}$$

In every phase of the proposed framework, an optimal decision is made, which will lead to other sets of possible outcomes p, which may similarly be more or less advantageous for data preprocessing. We can assign a prime advantage to each of these possibilities p, denoted by $P'_a a$, similar to Equation 4.2. It could be advantageous to extract only meaningful attributes from the datasets while disregarding irrelevant attributes (Kwak & Choi, 2002; Gu et al., 2016). It might also be advantageous to visualize the imbalanced nature of the data to distinguish inliers from outliers, as only inliers are useful in obtaining reliable classification results. The optimal preprocessing technique can be expressed as a function of the selected attributes as follows:

$$p_{optimal} = \arg\max_{p \in SP} P'_{SP} p \tag{4.12}$$

where p is the preprocessing technique selected through the optimal decision procedure proposed in this study, SP is the set of preprocessing techniques, and a is the outcome and is a function of the choice p. Such optimal decision-making was applied in the preprocessing of the datasets used in this study to enable the learning algorithms to learn accurately from unbiased datasets.

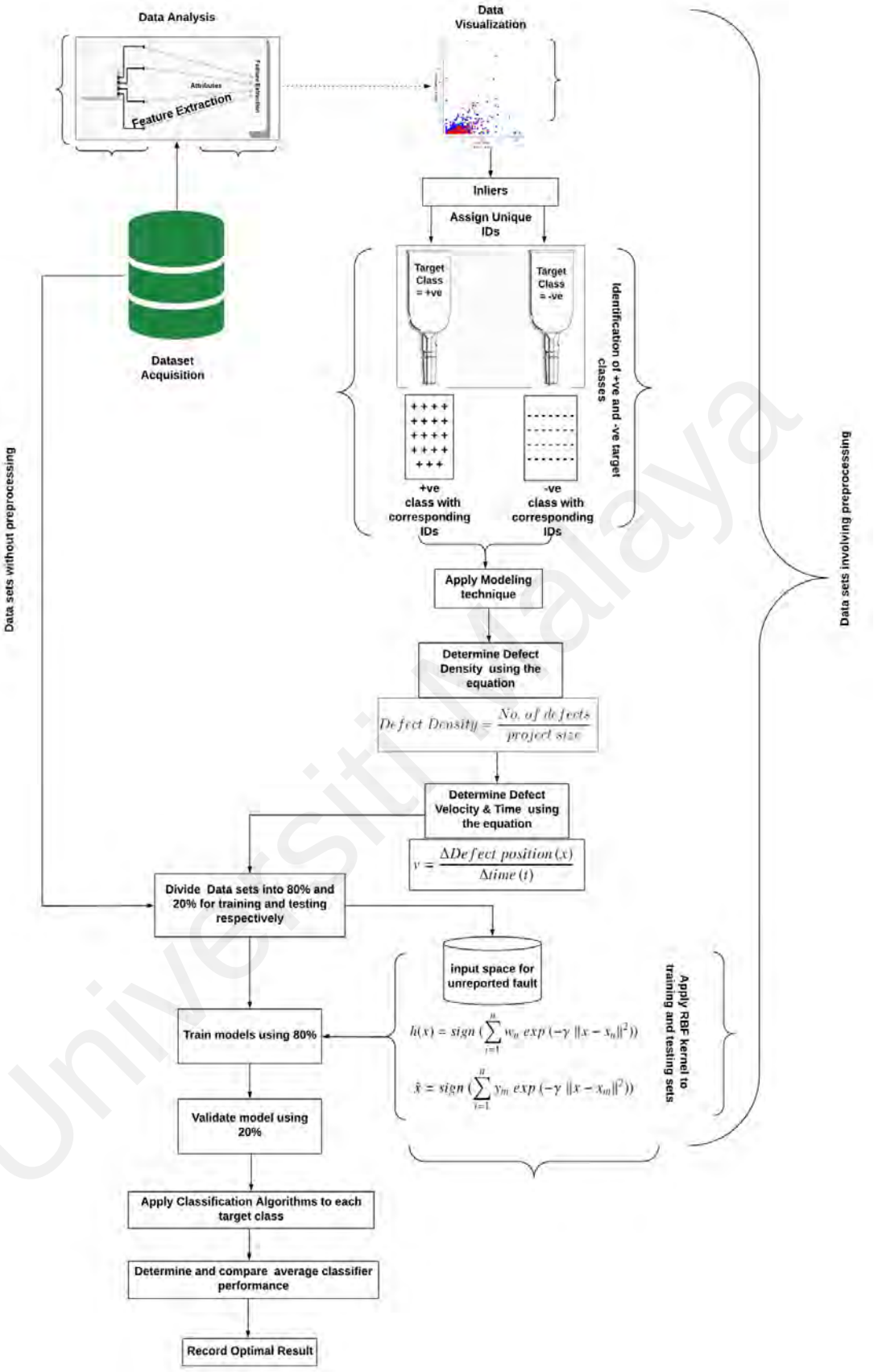


Figure 4.2: Proposed optimal decision framework for data preprocessing

4.2.1.1 Outlier removal from imbalanced class- and method-level datasets

A dataset is imbalanced if the categories into which the data are to be classified are not almost equally represented (Chawla, 2009). Thus, data imbalance (or class or method imbalance) refers to the case in which the proportions of defective (minority) and defect-free (majority) modules in a project are not equal. In addition to feature selection, another important phase of data preprocessing is outlier removal. Therefore, the data were first checked to determine whether they contained outliers before any further preprocessing tasks were performed.

Outliers are data points that lie far away from the main cluster(s) of data. Outliers may occur in datasets as a result of measurement variations or may be a manifestation of experimental error. In this study, outliers were identified and eliminated during the preprocessing phase to prevent them from impacting the results obtained. Figure 4.3 illustrates how the outliers were identified and removed from the datasets based on the optimal decision to eliminate outliers. To identify outliers, the data file was first loaded into the Orange workspace to visualize the inliers and outliers in the data. An outlier widget was then connected to the data file to enable the separation of the outliers. Thereafter, the outlier widget signal was reset to capture only the inliers in the data. To achieve a reliable, bias-free dataset that would be suitable for use in prediction, it was then necessary to subject the inliers to further preprocessing. The reason for removing outliers is to enable prediction models to learn only from the main clusters of data to ensure an accurate assessment of the models' performance. If a prediction model is trained on data containing outliers, the resulting noise in the data can cause the model to produce misleading results.

The outlier widget was configured with a distance metric to enable accurate outlier identification; the Euclidean distance was applied, with K=3. The Euclidean distance metric was chosen because it not only can be used to calculate the distance between two

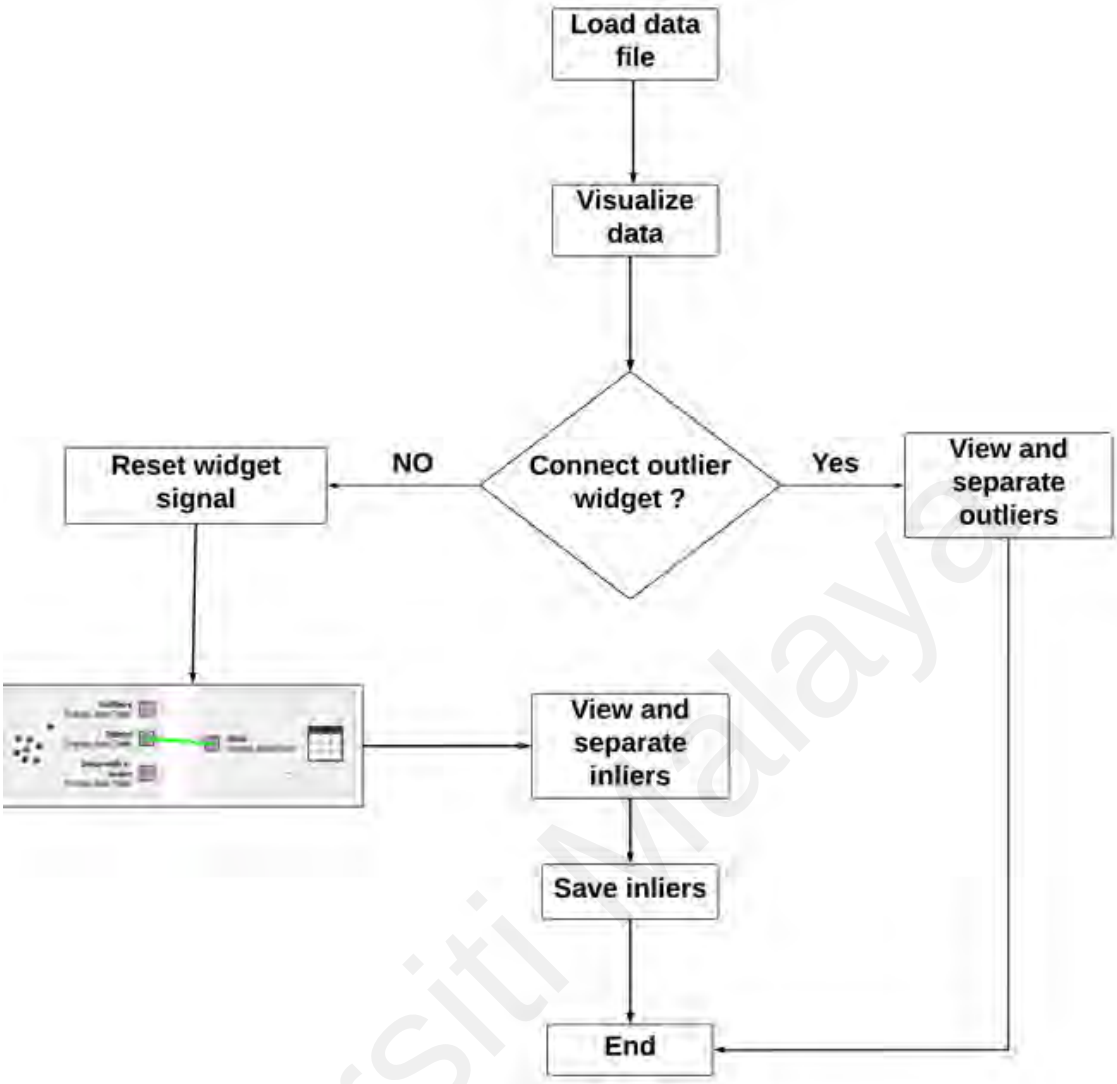


Figure 4.3: Outlier removal procedure

points on a plane but also is a consistent measure that treats all dimensions equally.

Consider two points (A and B) on a plane, as presented in Figure 4.4. The distance between A and B can be calculated from their x and y coordinates using the Pythagorean theorem.

$$dA, B = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.13}$$

Similarly, the Euclidean distance between two points x and y in n dimensions can be

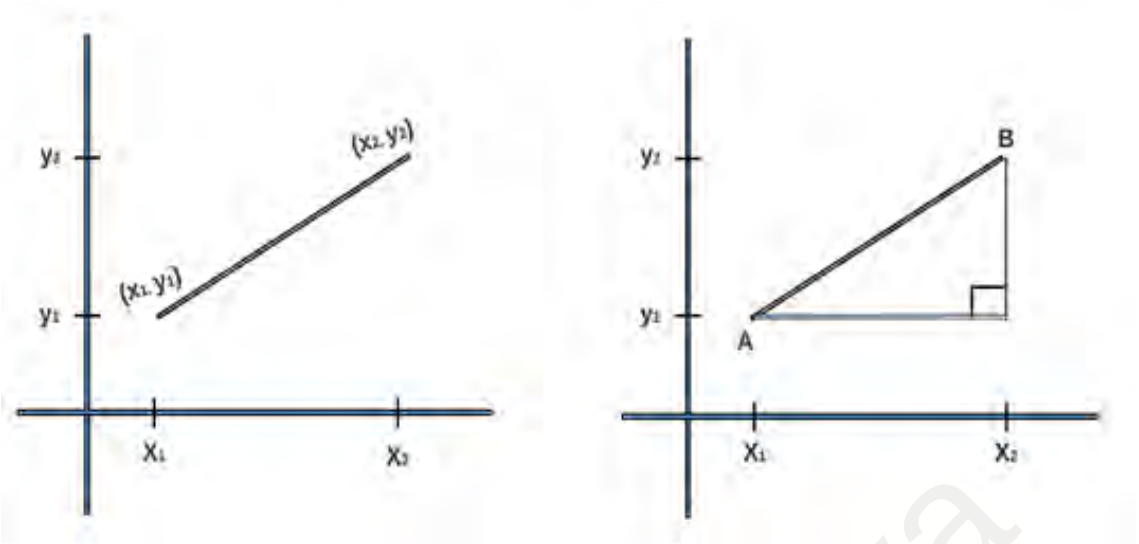


Figure 4.4: Distance function

calculated as

$$dx, y = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.14}$$

4.2.1.2 Further preprocessing of inliers

Inliers are the clustered data used for classification. Some inliers may be defective but may be grouped together with defect-free instances in the data distribution, which could pose a challenge in distinguishing defective from defect-free data. Therefore, it is necessary to identify defective inliers during data preprocessing. The defective inliers were captured with the help of an input space that allowed unreported faults (i.e., faults omitted in the early phase of data preprocessing) to be recaptured. To visualize the data and ensure that only inliers would be used in the experiments, we reset the widget signal to capture only the inliers. Based on the optimal decision outcome, it was then necessary for the inliers to undergo further preprocessing to make the data suitable for use in training and validating the learning algorithms.

4.2.1.3 Input space

Some unreported faults in class- and method-level datasets might be omitted, without proper identification of their category as defective or defect-free. Such omission can occur due to either human or computer error during the initial stage of data preprocessing. Therefore, an input space was created to capture such omitted errors before the application of the learning algorithms. For this purpose, a kernel-based approach was chosen to determine the classification outcomes for unlabeled data not present in the training set. In this approach, a similarity function is applied to both the training set and the unlabeled set as represented in the feature space in which unreported faults are to be captured. The kernel classifier computes a weighted sum of the similarities between the unlabeled and training sets. The outcome, based on the sign function, will determine whether the unlabeled set is negative or positive.

Therefore, to determine the classification outcomes for each inlier in the dataset, a kernel-based algorithm was applied. The Radial Basis Function (RBF) kernel, which is applied in various kernelized learning algorithms (Chang et al., 2010), was used for this purpose. The RBF kernel takes as input samples x_n , $y_n \in D$, where x_n is the data point and y_n is the weighted influence that determines the outcome for this data point.

However, each data point x_n in the dataset DS is influenced by the distance $\|x - x_n\|$, which is the Euclidean distance between the feature vectors in DS. To determine the outcome hx for sample x, which is hypothesized to be either +ve or -ve based on the input signal s, the RBF kernel defined by Schölkopf et al. (2004) is adopted, as follows:

$$hx = \sum_{i=1}^{n} w_n \, exp \, -\gamma \, \|x - x_n\|^2 \tag{4.15}$$

where x is the sample for which the outcome is to be determined and x_n is a data point

contained in the training sample. Here, w_n is used to denote the weighted influence in a more general form; in this case, w_n is equivalent to y_n. The notation indicates that all related influences (the weighted influences of the input signal) are summed to determine the outcome for x. The value of γ determines the distance range considered, where $\gamma > 0$, meaning that larger values of γ correspond to more negative values of $-\gamma \ \|x - x_n\|^2$ (Schölkopf et al., 2004). The expected outcome hx can be represented as the input signal s; thus,

$$s = \sum_{i=1}^{n} w_n \ exp \ -\gamma \ \|x - x_n\|^2 \tag{4.16}$$

The value of the RBF kernel decreases with increasing distance, ranging from 0 at an infinite distance limit to 1 when $x = x_n$ (Shashua, 2009). Here, the goal is to minimize the mean square error between the input signal and the weight; thus, $mins - y^2 \ \in \ D$ is taken to ensure that the range of values will be between +1 and -1. The value +1 represents a defective outcome, whereas any value below zero (between -1 and 0) will be regarded as 0. Based on the similarity between the input signal and the hypothesis, as shown in Equations 4.15 and 4.16, it is found that $hx = signs$. Therefore, Equation 4.15 can be rewritten as

$$hx = sign \sum_{i=1}^{n} w_n \ exp \ -\gamma \ \|x - x_n\|^2 \tag{4.17}$$

Here, x_n, y_n is used to denote the i-th training sample and weighted influence drawn from the 80% of the data designated as the training data; this sample has a corresponding weight, denoted by w_n in general form, which corresponds to y_n. The 20% of the data to be used for evaluation are treated by applying a similar RBF kernel between x_m and y_m, where $\hat{x}$

denotes an unlabeled input in the validation set. Hence, the outcome of the kernelized binary classifier can be expressed as

$$\hat{x} = sign \sum_{i=1}^{n} y_m \, exp \, -\gamma \, \|x - x_m\|^2 \tag{4.18}$$

where $\hat{x} \in \{-1, +1\}$ is the label predicted by the kernelized classifier for the evaluation sample denoted by $\hat{x}$, whose outcome is the target of interest; x_m represents the validation sample; and y_m is the corresponding weight. Furthermore, the sign function, as applied in Equations 4.17 and 4.18, is used to identify whether the predicted outcome is positive or negative; when positive, the output value is set to 1, while when negative (-1 or less), it is set to 0. Thus,

$$f\hat{x} = \begin{cases} 1, & \text{if } error\_count \geq 1 \\ 0, & \text{-1 or less} \end{cases}$$

4.2.2 Modeling technique

The modeling technique applied in this research is based on the relationship between the number of defects and the defect density as a function of the defect acceleration. This relationship is based on the fact that an increase in the number of defects results in an increase in the defect density of a project. Based on the application of the optimal decision technique to derive variables that are correlated with the number of defects, this study presents a simplified step-by-step modeling approach for further preprocessing of class- and method-level datasets as well as for predicting the class- and method-level defects in a new version of a software product. Figure 4.5 illustrates the basis of the proposed modeling technique.

The modeling technique applied to derive the defect density, defect velocity and defect

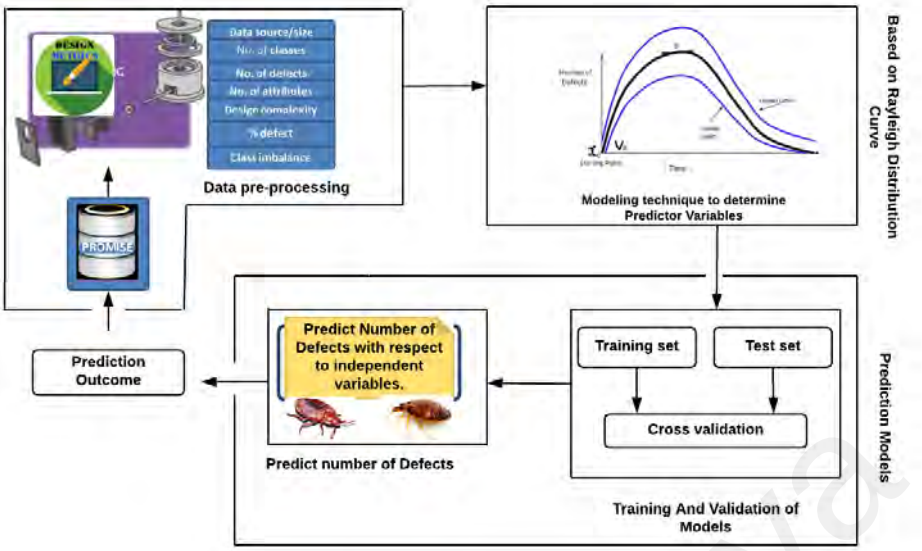


Figure 4.5: Proposed modeling technique for data preprocessing

introduction time is based on a Rayleigh distribution curve, which represents the number of defects over time throughout the life cycle of a project (Aydin & Tarhan, 2014), as shown in Figure 4.5. This curve reveals how software defects evolve with time throughout the development process. As the phases of software development proceed, the number of errors increases if these errors are not caught and eliminated. Furthermore, the Rayleigh model also shows the relationships between other variables, such as the defect acceleration, and the number of defects over time throughout the SDLC. These variables were used as predictor variables to construct the regression models used in predicting the numbers of class- and method-level defects in a new software version.

4.2.2.1 Definitions of metrics

The defect density, as an essential attribute for determining software reliability, is one of the optimal variables applied in this research. This attribute has an exponential relationship with the number of class- or method-level defects. A software product can be released on the market only once its defect density is considered low enough to avoid criticism (Malaiya & Denton, 2000). The quality of a software product can be evaluated based on

its defect density (Shah et al., 2012; Mohagheghi et al., 2004). A software product with a high defect density tends to perform less well, while a product with a low defect density has the ability to run without failure (Westfall, 2013; Kelly et al., 1992). Specifically, a high defect density indicates that a software program contains a large number of defects; as such, the ability to estimate the defect density makes it easier for a testing team to focus on identifying and correcting as many defects as possible with limited resources (Westfall, 2013). Software companies tend to benefit from the early detection of defects that are likely to be present in software, which is facilitated by knowledge of the defect density in a software product of interest (Nagappan & Ball, 2005; Compton & Withrow, 1990; Planning, 2002). In addition, the defect density provides information that can help development teams keep proper records while attempting to reduce the number of defects in upcoming versions of a current product (Westfall, 2013).

Moreover, the defect density provides a development team with helpful information for keeping track of the progress made in reducing the number of defects during software project transitions (Westfall, 2013). It is also important to note that while the overall software quality is a matter of concern, the aspects that require the greatest attention are mainly the components with the highest defect densities; thus, the ability to determine the defect density is key (Knab et al., 2006). Furthermore, based on the information provided by the defect density of a software product, the number of defects in an upcoming release of that product can be estimated (Knab et al., 2006).

Based on the relevant mathematical relationship, the optimal variables chosen as the basis of this study were derived to provide a means of estimating the numbers of defects likely to be present in various software products, as reported by Felix & Lee (2017b). If the number of defects can be determined prior to software testing, it will be easier for the testing team to focus on addressing as many defects as possible with the limited resources

available (Westfall, 2013). Typically, following the SDLC, software developers aim to meet the project deadline and thus carry out software development as quickly as possible, without proper elaboration of the impact of such speedy transitions. Such high-velocity transitions through the SDLC can expose a software product to defects. Therefore, to improve software quality, it is wise to determine the impact of the average defect velocity on the number of software defects as a software product transitions from one phase of the SDLC to another. Any significant improvement in the SDLC will lead to a reduction in the rate at which defects occur, reduce the need for software rework and ultimately improve software quality and productivity (Diaz & King, 2002). Hence, it would be in the best interests of the machine learning community if the numbers of defects in a new version of software at both the class and method levels could be successfully estimated. In addition, it would be beneficial to have an idea of the rate at which these defects occur and the impact of this rate on software products. On the basis of the in-depth relationship between the defect density as an optimal variable and the numbers of class- and method-level defects, all of the optimal variables considered in this study are now explicitly defined, as follows.

Defect density: The defect density g represents the ratio of the number of defects to the size of a software project. An increase in the number of defects in a software project will result in an increase in the defect density (Felix & Lee, 2017b). The defect density, as applied in this study, is expressed as a number per unit project size and is calculated using the following equation:

$$defect\ density = \frac{no.\ of\ defects}{project\ size} \tag{4.19}$$

Defect velocity: The defect velocity v is the change in the defect position with respect to

time t and is measured in units of defects per day. The defect position here ranges from the requirement phase of the SDLC to the implementation phase and represents when defects might have occurred during software development.

Defect introduction time: The defect introduction time t is the time at which defects occur in a software product and is measured in days. The Rayleigh distribution curve depicts how the number of defects increases over time in a software project. Based on the Rayleigh model, the integral of the relationship between the defect acceleration and the number of defects can be taken to derive the defect density, defect velocity and defect introduction time.

4.2.2.2 Derivations and formulations

This section presents the step-by-step derivation of the aforementioned optimal variables applied in this research. At the starting point of a project, the number of defects is zero, as illustrated in Figure 4.6. The chance of defect introduction increases over time as the project proceeds from one phase to the next. With further phase transitions during software development, the defect acceleration increases, which can lead to an increase in the number of defects. The defect acceleration is the change in the defect velocity at a given instant of time. The increase in the number of defects and the defect density are therefore functions of the defect acceleration.

In Figure 4.6, x_0 = initial defect status, v_0 = initial defect velocity, t = time, and g = defect density.

The transitioning of a software project from one phase of development to the next can be accompanied by an increase in the number of defects. This is simply because of the possibility of more defects being added to the project over time as the project transitions. Furthermore, the defects present in the requirement phase can potentially transition to the design phase, and so on, as the project continues. It is therefore necessary to determine

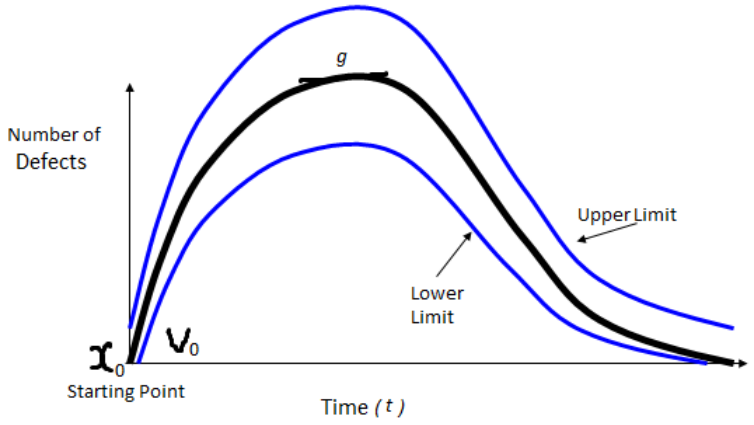


Figure 4.6: Rayleigh distribution curve (Linda M. Laird, 2005)

the defect position as the project progresses. To this end, a mathematical relationship is applied in the model to determine the defect position over time throughout the SDLC, which, in turn, allows the defect introduction time to be determined. Accordingly, the time at which a defect appears is also determined since the position of the defect can be ascertained. On the Raleigh distribution curve, the variable x_0 represents the initial defect position, and v_0 and t_0 are the initial defect velocity and time, respectively. Under the assumption that the project size is constant, i.e., the sizes of the previous and current versions of the same project are the same, the numbers of defects in both versions are influenced by the rate at which defects occur in those versions; this also implies that the number of defects in a software project depends on the acceleration characterizing defect occurrence while the project is ongoing.

Therefore, the defect density g of a software product depends on the number of defects, as presented in the expression below.

$$fno.\ of\ defects \quad \Rightarrow \quad g$$

This expression indicates that the defect density g is affected by the number of defects in a project, which, in turn, depends on the defect acceleration.

The defect acceleration is the change in the defect velocity over a period of time. As presented below, the number of defects within a software project is a function of the defect acceleration (Felix & Lee, 2017b). Thus,

$$f defect\ acceleration \quad \Rightarrow \quad no.\ of\ defects$$

If the defect acceleration is low, then the number of defects will also be low, and because the defect acceleration is defined as the change in the defect velocity over time, the defect density, which is affected by the defect velocity, will be low as well.

If the defect acceleration increases, this increased acceleration will lead to an increase in the number of defects and, hence, an increase in the defect density g.

Therefore,

$$f defect\ acceleration \quad \Rightarrow \quad g$$

The defect density g is a function of the defect acceleration, which characterizes defect occurrence given a constant project size. The defect acceleration can be calculated as the change in the defect velocity over the change in time, as follows:

$$defect\ acceleration = \frac{\Delta\ velocity\ v}{\Delta\ time\ t} \tag{4.20}$$

At a constant project size, the number of defects and, consequently, the defect density g of a project vary only as functions of the defect acceleration because the defect density depends on the rate at which defects occur. Thus,

$$g = \frac{no.\ of\ defects}{const.\ project\ size} \propto \frac{fdefect\ acceleration}{const.\ project\ size} \tag{4.21}$$

For a constant project size, the defect density g, which depends on the defect acceleration, can be expressed as follows:

$$g = \frac{fdefect\ acceleration}{const.\ project\ size} \tag{4.22}$$

Hence, the defect density g is equivalent to the defect acceleration at a constant project size (Felix & Lee, 2017b). Thus,

$$g = \frac{\Delta velocity\ (v)}{\Delta time\ (t)} \tag{4.23}$$

$$v = gt \tag{4.24}$$

By integrating the defect velocity v over time t as described by Felix & Lee (2017b), the following equation is obtained:

$$vdt = gdt \tag{4.25}$$

$$g\ dt = gt + c \tag{4.26}$$

Replacing the constant c with the initial velocity v_0 yields

$$g\ dt = gt + v_0 \tag{4.27}$$

By eliminating v_0 in Equation 4.27, such that the velocity at the starting point is 0, the following equation is obtained:

$$g\ dt = gt \tag{4.28}$$

The initial velocity is zero; in this state, the project has not yet commenced. To predict the defect position x as a function of time t, the defect velocity is used, as described by Felix & Lee (2017b). The defect velocity is expressed as follows:

$$v = \frac{\Delta Defect\ position\ (x)}{\Delta time\ (t)} \tag{4.29}$$

The change in the defect position, Δx, represents the transitioning of the project through the SDLC. Therefore, the target defect position x as a function of time t can be determined by integrating v with respect to time t on the basis of Equation 4.28. Accordingly, the following equation is obtained:

$$x\ (t) = \ vdt \tag{4.30}$$

Similar to Equation 4.25, i.e., $vdt = \ gdt$, integrating the defect velocity with respect to time yields

$$vdt = \ gt + v_0 dt \tag{4.31}$$

Equation 4.31 can be further decomposed into a sum of two integrals to determine the average defect introduction time t:

$$gt + v_0 dt = gtdt + v_0 dt \tag{4.32}$$

From this sum of two integrals, the constants g and v_0 are first factored out, as follows:

$$gtdt + v_0 dt = g\ tdt + v_0\ dt \tag{4.33}$$

To solve for the resultant polynomial, the power rule of integration, of the form $t^n dt = \frac{1}{n+1}t^{n+1} + c$, is applied to the first integral to obtain

$$g\ tdt = g\frac{1}{2}t^2 + c \tag{4.34}$$

For the second part of the resultant polynomial in Equation 4.32, the following equation is obtained:

$$v_0 dt = v_0 t + c \tag{4.35}$$

Hence, combining Equations 4.34 and 4.35 yields

$$g\ tdt + v_0 dt = g\left(\frac{1}{2}t^2\right) + v_0 t + c \tag{4.36}$$

The constant of integration c becomes the initial defect position x_0. At this initial point, the number of defects and the defect velocity are both equal to zero. Thus, the equation

$$x\,(t) = g\left(\frac{1}{2}t^2\right) + v_0 t + x_0 \tag{4.37}$$

can be rewritten as

$$x\left(t\right) = g\left(\frac{1}{2}t^2\right) \tag{4.38}$$

Both v_0t and x_0 are equal to 0 at the initial point.

The defect introduction time t can then be calculated by solving Equation 4.38 for t, that is, rewriting the formula such that the left-hand side is t. Consequently, the following equation is obtained:

$$t = \sqrt{\frac{2x}{g}} \tag{4.39}$$

where x is the defect position at either the class or method level. In this study, x represents the number of classes or methods, and g is the average defect density, measured as the number of defects per unit project size.

Equations 4.19-4.39 show the relationships between the number of defects and the optimal variables. The defect density is determined using Equation 4.19, the defect velocity is determined using Equation 4.24, and the defect introduction time is determined using Equation 4.39. Among these derived optimal variables, the defect velocity is hypothesized to have a strong, positive correlation with the numbers of class- and method-level defects. Based on such a correlation, the defect velocity can be applied in constructing regression models to predict the numbers of class- and method-level defects. This implies that the defect velocity accounts for the increase in the numbers of class- and method-level defects in a software program.

4.3 Summary

The proposed optimal decision approach applied to achieve the research objectives has been presented in this chapter. In addition, further analysis of the problem investigated in this research is also presented to ensure that the proposed solution is suitable to address the research problem. Notably, there are clear indications that the proposed solution

should be suitable for addressing the research problem and that it can also serve as a blueprint for the proper preprocessing of datasets applied in machine learning studies. The proposed approach can render both class- and method-level datasets free from bias. For instance, to address irrelevant and redundant features in a dataset, a filter-based technique is applied in the proposed approach to form a feature subset based on the rank scores of the selected features. To address the presence of outliers, which can hinder the accuracy of learning algorithms, an automated linear projection widget is used in the proposed solution to separate outliers from the rest of the data. To ensure a favorable outcome, optimal decision-making is applied in every stage of the proposed framework to ensure proper preprocessing of class- and method-level datasets to aid in effective defect prediction. The literature suggests that an effective data preprocessing method is urgently needed as a tool to support the research findings in the supervised machine learning domain; therefore, the aim of this research is to develop a cost-effective approach to ensure that the datasets applied in machine learning studies are free from bias.

CHAPTER 5: EVALUATIONS AND EXPERIMENTATION

This chapter presents the metrics applied in evaluating the proposed approach presented in the previous chapter and the sets of experiments carried out to validate the stated hypotheses.

This section further explains the means of ascertaining the performance of the proposed optimal decision framework. The evaluation of the proposed framework is important to ensure that the best approach is applied for overcoming the challenges associated with the existing class- and method-level datasets and developing a suitable means of predicting the number of software defects in the new version of a software product. The methods used to evaluate the phases of the optimal decision framework proposed in Chapter 3, including the feature selection phase as well as the phase of outlier removal, unique identifier assignment and inlier selection and the phase of further preprocessing and construction of the input space, are presented here along with the means of evaluating the overall performance of the proposed framework.

To evaluate the outcome of the filter-based method applied in the feature selection phase, the Pearson correlation was applied to identify and select features with high predictive power. The predictive power of the selected features was determined in the form of an average rank score, as previously presented in Equation 4.5, Section 4.2.

To evaluate the outlier removal phase, the Euclidean distances between data points were considered while applying Orange 2.7 to accurately determine the outliers in the class- and method-level datasets. The automated widgets provided in Orange 2.7 made it much easier to eliminate outliers and assign unique identifiers to the inliers of the datasets before performing further preprocessing on the inliers.

To evaluate the outcome of the further preprocessing phase of the proposed approach, the

defect density, defect velocity and defect introduction time of the inliers in each dataset were determined by applying a modeling technique to extract the relationships between these variables (defect density, defect velocity and defect introduction time) and the numbers of class- and method-level defects. The defect density was evaluated using Equation 4.21. The defect velocity was determined using Equation 4.24, whereas the defect introduction time was determined using Equation 4.39.

5.1 Performance evaluation metrics

To assess the overall performance of the proposed optimal decision framework, several learning algorithms, namely, naïve Bayes, logistic regression, neural network, K-nearest neighbor, support vector machine and random forest algorithms, were trained on the preprocessed training datasets. The performances of these learning algorithms were evaluated in terms of the Classification Accuracy (CA), precision, recall, specificity, Matthews Correlation Coefficient (MCC), J-coefficient, F-score, area under the receiver operating characteristic (ROC) curve (AUC), Brier score, information score, information entropy and geometric mean. Furthermore, the performance of the framework was assessed by comparing the results achieved using the preprocessed datasets with those achieved using the raw datasets based on the learning algorithms and evaluation metrics listed above. The details of these metrics have been presented in Section 2.3.

Other metrics applied in evaluating the proposed framework include the average classifier performance, the average performance loss/gain, the average information entropy and the percentage prediction error.

5.1.1 Average classifier performance

To properly assess the overall performance of the learning algorithms applied in this study, the average performance of each classifier was evaluated. The average classifier performance ($\bar{x}$), which represents the mean performance of a given classifier for the target class of True or False, is calculated as follows:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_n = \frac{1}{n} (x_1 + x_2 + \ldots x_n) \tag{5.1}$$

where n = the number of experiments performed,

$\bar{x}$ = the average classifier performance, and

x_i = the individual classifier performance in the i-th experiment (here, each experiment corresponds to a different project for both the True and False classes).

5.1.2 Average performance loss/gain

The average performance loss/gain ($P_{lossgain}$) of a classifier is used to determine the extent to which that classifier's average performance differs between the majority and minority classes. The average performance loss/gain is characterized in terms of the information entropy and is calculated as follows:

$$P_{lossgain} = \frac{VI_T - VI_F}{VI_k} \tag{5.2}$$

where $VI_T - VI_F$ = the difference in average performance between the majority (False) and minority (True) classes and VI_k = the average performance for the target class.

5.1.3 Average information entropy

In a situation where the average uncertainties for an information source regarding the different possible outcomes of an event are unequal, the concept of entropy can be applied

to the information concerning the outcome. Since each learning algorithm performs differently when applied to imbalanced datasets, the amount of information available regarding an algorithm's performance may be uncertain and consequently exhibit variations. Therefore, the average information entropy was applied to assess the level of uncertainty of the information obtained among these learning algorithms.

Entropy refers to the average amount of information obtained regarding an information source. The entropy of an information source X is denoted by HX. Here, the information sources are the metrics in which the classifiers exhibit losses or gains as a result of the information entropy. These metrics include the specificity, F-score, recall, precision and G-mean. To determine HX, the following equation is used:

$$HX = \sum_{i=1}^{n} p_i \, log_2 \frac{1}{p_i} \tag{5.3}$$

This equation can be rewritten as

$$HX = p_1 \, log_2 \frac{1}{p_1} + \ldots + p_n \, log_2 \frac{1}{p_n} \tag{5.4}$$

where p_i is the probability of a single outcome with respect to the information source and n is the number of outcomes.

5.2 Classifiers

Classifiers are machine learning models for data analysis tasks. The purpose of the classifiers considered in this research is to properly assess the overall performance of the proposed solution. The classifiers investigated in this research are naïve Bayes, Logistic Regression (LR), neural network, K-Nearest-Neighbors (KNN), Support Vector Machine (SVM) and Random Forest (RF) classifiers.

5.2.1 Naïve Bayes

A naïve Bayes classifier is an effective classifier that is capable of generating accurate prediction results and is a suitable choice for classification problems involving variates of independent variables (Larsen, 2005). It is capable of handling large datasets and often outperforms various sophisticated classification models. The Bayes classification model is derived from Bayes' theorem under the hypothesis of independence among the variables. Bayes' theorem provides an avenue for estimating the conditional probability $prba \mid b$ from $prba$, $prbb$ and $prbb \mid a$. The probability of class a given attribute b is calculated as

$$prba \mid b = \frac{prbb \mid a}{prbb} \tag{5.5}$$

For a large dataset, the posterior probability of class a is calculated as

$$prba \mid b = prbb_1 \mid aprbb_2 \mid a...prbb_n \mid aprba \tag{5.6}$$

where $prba \mid b$ = the posterior probability of class a given attribute b,

$prba$ = the prior probability of class a,

$prbb$ = the prior probability of attribute b,

n = the number of instances, and

$prbb \mid a$ = the likelihood, that is, the probability, of attribute b given class a.

The class with the highest posterior probability is the prediction outcome.

5.2.2 Logistic regression (LR)

LR estimates the likelihood of an event with two possible values (binary classification). This is a predictive analysis conducted when the possible values of the independent variables are 0 and 1. The LR model is employed to analyze data to interpret the relationship

between one dependent binary variable and one or more independent variables.

5.2.3 Neural network

A neural network is a classification model inspired by the neural network of the human brain. The units in a neural network perform tasks similar to those of neurons in the human brain. This network has an input layer, an output layer, and one or more hidden layers in between. During classification, data pass through the units of the neural network, which perform various mathematical computations. These units interact with each other via connections linking the various layers. Each connection has a number, called a weight, associated with it. When the neural network takes an input, it processes that input by performing calculations based on the weights to produce an output. A neural network is trained by adjusting the values of the connections; this can be achieved by means of a backpropagation algorithm, which trains from the input layer to the output layer and vice versa. Alternatively, a neural network can be trained by trying to reduce a loss function over a training set using a gradient-based method (Goldberg, 2016).

5.2.4 K-nearest neighbors (KNN)

A KNN classifier is based on a straightforward classification model that considers all possible outcomes and classifies each instance in advance based on a resemblance factor. An instance is classified based on the voting strength of its neighbors. In the classification outcome, each instance is assigned to the class with the highest number of corresponding votes, as determined by a distance factor. In this study, the Euclidean distance between two neighbors is used as the distance factor.

For instance, consider two points (X and Y) on a plane. The distance between X and Y, which also represents the length of $\overline{XY}$, can be calculated from the a and b coordinates of these points using the Pythagorean theorem.

$$dx, y = \sqrt{(a_2 - b_1)^2 + (b_2 - a_1)^2} \tag{5.7}$$

Similarly, the Euclidean distance between two points a and b in n dimensions can be calculated as

$$da, b = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{5.8}$$

5.2.5 Support vector machine (SVM)

An SVM classifier finds the line of best fit that maximizes the margin between two classes that are linearly separable. First, the SVM treats every instance of a class as a vector of the input variables. Thus,

$$p = x_1, x_2, ..., x_n \tag{5.9}$$

Consequently, a dataset D takes the form of pairs of vectors and classes:

$$D = \{x_1, y_1, x_2, y_2, ..., x_n, y_n\} \tag{5.10}$$

where x = the vectors, y = the classes associated with the vectors (either +ve or -ve), and n = the number of instances.

Then, the SVM locates the two closest points between the two classes. The SVM connects these two points with a line and then draws a perpendicular line bisecting this connection line. Thus, the two closest points define the line of best fit. This line of best fit has an intercept b and a normal vector w, which represents a weight vector, such that all x on the line satisfy the linear equation

$$w^T x = -b \tag{5.11}$$

Here, T denotes the transpose operation; in particular, w^T is the transpose of the weight vector w. Equation 5.11 can be rewritten as

$$w^T x + b = 0 \tag{5.12}$$

such that the scalar product of the weight vector and the point vector is equal to the intercept of the line of best fit. From Equation 5.10, the training dataset can be written as

$$D = \{x_i, y_i\} \tag{5.13}$$

in the form of point vectors x_i paired with their corresponding classes y_i. Thus, the linear classification process can be expressed as a function of the point vector x as follows:

$$fx = signw^T x + b \tag{5.14}$$

5.2.6 Random forest (RF)

An RF classifier is a large collection of decision trees. These random trees are said to make up a forest. A reasonable number of decision trees are generated based on a random selection of data and variables. The RF predicts the class of the dependent variable based on the class predicted by the largest number of trees. Thus, the decision trees are used to predict the classification outcome. RF classifiers can outperform certain other sophisticated classifiers (Masetic & Subasi, 2016; Subasi et al., 2017).

The classifiers introduced above were used to accurately assess and report the impact of imbalance in class- and method-level datasets on the performance of learning algorithms.

5.2.7 Prediction models

A regression model was constructed for predicting the number of defects. The prediction outcomes were then compared with the actual numbers of defects present in the datasets. In the regression models presented in Equations 5.15-5.22, Y represents the dependent variable (that is, the number of defects); B_0 represents the intercept, while B_1 is the coefficient of the independent variable X_1; and ε represents the standard error of the prediction.

Regression models were constructed with only one independent variable X_1, i.e., the defect velocity, because this variable exhibits a strong positive correlation with the number of defects. This method of model construction also ensured that there was no collinearity among independent variables in the constructed regression models. This is important because when collinear variables are applied to construct a prediction model, the correlation among these supposedly independent variables tends to reduce the accuracy of the model (Hosseini, Turhan, & Gunarathna, 2017). Hence, to avoid the potential influence of collinearity on the prediction accuracy, only simple regression models that could explicitly and accurately reflect the performance of a single independent variable were considered. Based on an analysis of variance (ANOVA) of the defect velocity at the method level, the intercept and coefficient for the defect velocity were found to be approximately -96 and 17.7, respectively, as presented in Figure 5.1 along with the corresponding values of the standard error, t-statistic and p-value. These coefficient and intercept values were applied as the standard parameters of the regression equation. The above values were used in the regression equation shown in Equation 5.15 below to estimate the number of defects:

$$Y = B_0 + B_1 X_1 + \varepsilon \tag{5.15}$$

NASA MODULE LEVEL DATASET				
	Coefficients	Standard Error	t Stat	P-value
Intercept	-105.2798	15.07864421	-6.98205	0.000115
Defect Velocity	15.721201	1.040042862	15.11592	3.63E-07

ELFF METHOD-LEVEL DATASET				
	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	-96.4122051	12.49590222	-7.71551	7.1994E-11
Defect Velocity	17.74843768	0.876329326	20.25316	1.6415E-30

ELFF CLASS-LEVEL DATASET				
	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	-158.0494	20.42883679	-7.7365834	7.183E-11
Defect Velocity	21.635627	0.997571265	21.688302	5.3E-32

Figure 5.1: Evidence for the intercept and coefficient values applied in the regression models

Hence, the resulting regression equation for estimating the number of defects at the method level was found to be

$$Y = -96 + 17.7X_1 + \varepsilon \tag{5.16}$$

Example 1. Suppose that a new version of the $Unicore$ project, similar to an existing version, is to be developed. It is possible to estimate the number of defects in the new project at the method level using this regression equation. For instance, when developing a $Unicore1.n$ version that is similar to the existing $Unicore1.3$ project, information on the existing project, such as the rate at which defects occurred in the current project (i.e., the average defect velocity), can be helpful for predicting the defect characteristics of the future project. The $Unicore1.3$ project was characterized by an average defect velocity of 6.50 defects per day. Therefore, the regression equation for estimating the possible number of defects in $Unicore1.n$ at the method level is

$$Y = -96 + 17.7 * 6.50 + \varepsilon = 19 + \varepsilon \tag{5.17}$$

This means that the predicted number of method-level defects in the $Unicore1.n$ version is 19 plus the standard error ε. For comparison, the actual number of defects in the $Unicore1.3$ project is 21.

From an ANOVA of the defect velocity at the class level, the intercept and coefficient for the defect velocity were found to be approximately -158 and 22.64, respectively. Therefore, the regression equation for class-level defect prediction is

$$Y = -158 + 21.64X_1 + \varepsilon \tag{5.18}$$

Example 2. Suppose that a new version of the $OmegaT$ project, similar to an existing version, is to be developed. It is possible to estimate the number of defects in the new project at the class level using this regression equation. For instance, when developing an $OmegaT3.n$ version that is similar to the $OmegaT3.1$ project, information such as the average defect velocity of the existing project can again be helpful for predicting the defect characteristics of the future project. The $OmegaT3.1$ project was characterized by an average defect velocity of 9.49 defects per day. Therefore, the resulting regression equation for estimating the possible number of defects in $OmegaT3.n$ at the class level is

$$Y = -158 + 21.64 * 9.49 + \varepsilon = 47 + \varepsilon \tag{5.19}$$

This means that the predicted number of class-level defects in the $OmegaT3.1$ version is 44 plus the standard error ε. The numbers of defects predicted as described above are compared with the actual numbers of defects in the results section.

To further confirm the results obtained from the regression models, an additional experiment was performed using the NASA module-level datasets. From the ANOVA results for the defect velocity at the class level, the intercept and coefficient were found to

be approximately -105 and 15.72, respectively. These values were used to construct the regression equation. As before, the regression equation below was applied to estimate the number of class-level defects based on the above values:

$$Y = B_0 + B_1 X_1 + \varepsilon \tag{5.20}$$

Thus, the resulting regression equation for estimating the number of defects using the NASA module-level data was found to be

$$Y = -105 + 15.72 X_1 + \varepsilon \tag{5.21}$$

Example 3. Suppose that a new version of the KC project, similar to an existing version, is to be developed. It is possible to estimate the number of defects in the new project at the module level using this regression equation. For instance, when developing a KC_n version that is similar to the $KC1$ project, information such as the average defect velocity of the existing project can be helpful for predicting the defect characteristics of the future project. The $KC1$ project was characterized by an average defect velocity of 25.49 defects per day. Therefore, the resulting regression equation for estimating the possible number of defects in KC_n at the module level is

$$Y = -105 + 15.72 * 25.49 + \varepsilon = 296 + \varepsilon \tag{5.22}$$

This means that the predicted number of module-level defects in the KC_n version is approximately 296 plus the standard error ε.

5.3 Experiments

The computing environment used in the experiments was a computer with a 1.70 GHz Intel(R) Core(TM) i5-3317U CPU and 4 GB of RAM running 64-bit Windows 10. Orange 2.7 was used to evaluate the performances of 6 state-of-the-art classifiers, namely, a naïve Bayes classifier, an LR classifier, a neural network classifier, a KNN classifier, an SVM classifier and an RF classifier. C++ was used as the programming language alongside the regression models for predicting the numbers of defects in a new version of software at both the class and method levels.

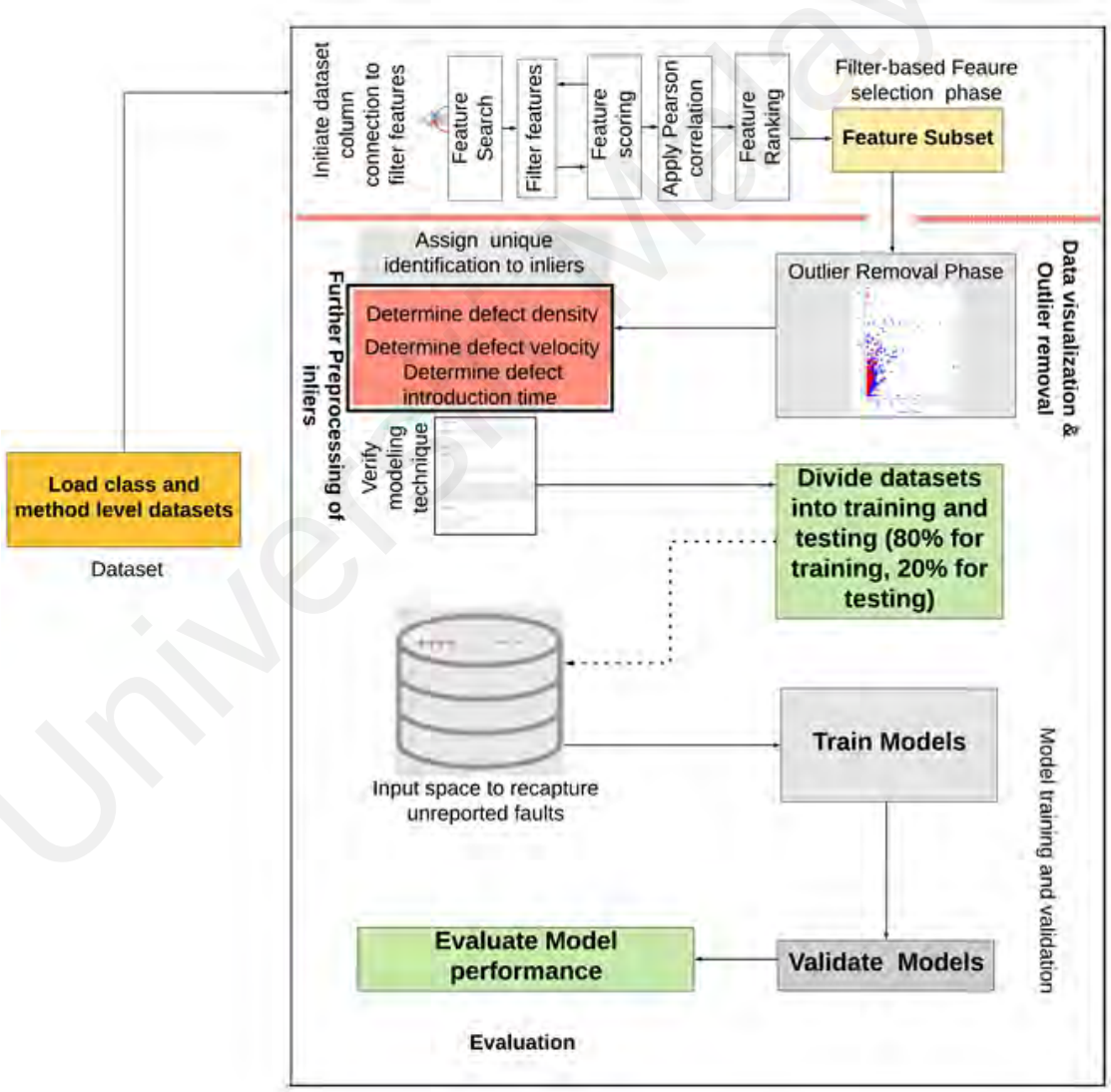


Figure 5.2: Experimental setup

The experimental setup, as presented in Figure 5.2, was based on the well-planned experimental design presented in Chapter 3. This design enabled the proper setup of experiments based on proper instruments to aid in the generation of reliable research findings. As stated previously in Chapter 3, appropriate and reliable tools were used throughout the experiments to ensure consistent and reliable outcomes. The tools applied in the experiments included Microsoft's Azure Machine Learning Studio, which was applied in the early stage of the experiments to perform filter-based feature scoring to enable the selection of relevant and predictive features, and an automated outlier widget provided in Orange 2.7, which enabled the automatic elimination of outliers and the subsequent assignment of unique identifiers to the inlier classes and methods in the datasets for further preprocessing. To further preprocess the class- and method-level datasets, a mathematical modeling technique was applied to derive the defect density, defect velocity and defect introduction time from the available class- and method-level data. An open-source software tool called Symbolab was used to verify the accuracy of the technique used to derive these optimal variables. Other statistical tools applied in this research included Microsoft SPSS and Microsoft Excel, which were used to perform ANOVA and to determine the correlations between the derived variables and the numbers of class- and method-level defects.

5.3.1 Hypotheses

As discussed previously in Chapter 3, the researcher formulated lists of hypotheses to enable the evaluation of the proposed optimal decision framework. Both null and alternative hypotheses were considered to ensure a proper evaluation.

5.3.1.1 List of null hypotheses (H_0)

The null hypotheses (H_0) served as the starting point and as the basis for the alternative hypotheses. The null hypotheses were evaluated to determine the likelihood that the corresponding assumptions could be accepted or rejected. To consider the possibility that the null hypotheses could be wrong, an alternative hypothesis was also formulated as the contrary counterpart to each null hypothesis.

In this research, several hypotheses regarding the relationships between the number of software defects (dependent variable) and the derived optimal variables (independent variables), namely, the defect density, defect velocity and defect introduction time, were formulated and tested. Both null hypotheses (H_0) and alternative hypotheses were considered to avoid bias and enable us to test our assumptions through evaluation. The following null hypotheses were formulated in the present research.

a. Irrelevant and redundant features in class- and method-level datasets may not affect the average performance of learning algorithms.

b. Outliers present in highly imbalanced class- and method-level datasets may not affect the average performance of learning algorithms.

c. Learning algorithms may exhibit degradation in their average performance when applied to imbalanced class- and method-level datasets.

d. The performance of learning algorithms may not be affected as a result of highly imbalanced class- and method-level datasets.

e. Learning algorithms may not maintain their average information entropy when applied to highly imbalanced class- and method-level datasets.

f. The outcome of a defect prediction study may not depend on the quality of the datasets applied in that study.

g. The defect velocity v is not expected to influence the number of defects in a software

project.

h. The defect density g is expected to influence the number of defects in a software project.

i. The defect introduction time t is expected to influence the number of defects in a software project.

j. The defect density g of a software project is not expected to increase with an increase in the number of defects.

5.3.1.2 List of alternative hypotheses (H_n)

Alternative hypotheses (H_n) are formulated statements that contradict the null hypotheses. They are used to state certain assumptions about the ways in which the null hypotheses may be wrong. The following alternative hypotheses were formulated in this study:

a. Irrelevant and redundant features in class- and method-level datasets may affect or hinder the average performance of learning algorithms.

b. Outliers present in highly imbalanced class- and method-level datasets may affect the average performance of learning algorithms.

c. Not all classifiers may exhibit degradation in their average performance when applied to imbalanced data.

d. The performance of some learning algorithms may be affected as a result of highly imbalanced class- and method-level datasets.

e. Some learning algorithms may maintain their average information entropy when applied to highly imbalanced class- and method-level datasets.

f. The outcome of a defect prediction study may depend on the quality of the datasets applied in that study.

g. The defect velocity v is expected to influence the number of defects in a software project.

h. The defect density g is not expected to influence the number of defects in a software

project.

i. The defect introduction time t is not expected to influence the number of defects in a software project.

j. The defect density g of a software project is expected to increase with an increase in the number of defects.

These hypotheses were tested by means of the evaluation metrics presented in Sections 2.3, 4.2.2, 5.1.1, 5.1.2, 5.1.3 and 5.2.7.

5.3.2 Correlation analysis

A correlation analysis was performed to determine the relationships between pairs of continuous variables, for instance, between an independent and a dependent variable or between two independent variables. This analysis also allowed the numerical strengths of the relationships between the dependent and independent variables to be determined. The performance of each model was evaluated using the p-value, the adjusted R-square, the F-statistic and the standard error.

5.3.2.1 P-value

The p-value is used to indicate how well a model performs; it represents the statistical significance of the model (Ioannidis, 2005; Carterette, 2015).

5.3.2.2 F-statistic

The F-statistic enables a comparison of the average significance of the variables used in model construction.

5.3.2.3 Adjusted R-square

The adjusted R-square measures the goodness of fit of a model and also indicates the influence of a significant variable in a model. The adjusted R-square was considered in this

study because its value increases only when a significant variable is included in a model.

5.3.2.4 Standard error

In a regression analysis, there is the possibility of variations in measurements. These variations are often high but sometimes low relative to the actual measurements. The standard error can be used as an indicator of the reliability of the sample estimates relative to the actual measurements.

5.3.2.5 Mean magnitude of relative error (MMRE)

The mean magnitude of relative error (MMRE) is the most widely used evaluation metric for comparing the performance of competing software prediction models, and one of its purposes is to help a software team to select the best-performing model. However, the MMRE was not applied as an evaluation criterion in this study because the findings of several previous studies, such as those of Foss et al. (2003) and Miyazaki et al. (1994), suggest that the MMRE may be unreliable under certain conditions, leading to the selection of the worse candidate between two competing models; in particular, the MMRE tends to prefer a model that underestimates to a model that accurately estimates the expected value. These studies cast doubt on the results of previous studies that have relied on the MMRE to compare the accuracy of different prediction models.

5.3.3 Data collection

First, 10 different datasets representing different projects were obtained from the PROMISE repository (Sayyad Shirabad & Menzies, 2005). These NASA Metric Data Program datasets have been widely used in software defect prediction studies. Second, 69 open-source projects called the ELFF datasets, which contain 131,034 classes and 289,132 methods and were first used by Shippey et al. (2016), were obtained from the repository available at www.elff.org.uk/ESEM2016. All of these datasets require a significant amount

of cleaning to be suitable for defect prediction studies and to produce reliable results. Table 5.1 presents the statistics of the 10 different NASA datasets collected from the PROMISE repository. The columns in the table, from first to last, contain the dataset name, the number of instances, the number of defects, the number of attributes and the percentage of defects. Tables 5.2 and 5.3 present the statistics of the ELFF datasets at the class and method levels, respectively. The columns in each table, from first to last, contain the project name, the number of classes or methods, the number of defects, the number of attributes (for instance, lines of code and branch count) and the percentage of defects.

Table 5.1: NASA dataset statistics

Dataset	No. of Modules	No. of Defects	No. of Attributes	% Defects
KC1	2109	326	21	15.46
KC2	522	107	21	20.50
KC3	458	43	39	9.39
MC1	9466	68	38	0.72
MC2	161	48	39	29.81
MW1	403	31	37	7.69
PC1	1109	77	21	6.94
PC2	5589	21	36	0.38
PC3	1563	159	37	10.17
PC4	1458	174	37	11.93
Total	22838	1054	326	4.62

Table 5.2: ELFF class-level dataset statistics

Project Name	No. of Defective Classes	No. of Defects	No. of Attributes	% Defective Classes
AutoPlot 2012	2800	192	42	6.86
Cdk1	1678	0	42	0
Cdk1.1	1670	261	42	15.63
Cdk1.2	1717	0	42	0
Cmusphinx3.6	665	15	42	2.26
Cmusphinx3.7	665	10	42	1.5
Controltier3	1659	0	42	0
Controltier3.1	1658	117	42	7.06
Controltier3.2	1683	0	42	0
Drjava2008	2697	1013	42	37.56
Drjava2009	3196	774	42	24.22
Drjava2010	3549	403	42	11.36
Eclemma2	196	9	42	4.59
Eclemma2.1	233	37	42	15.88
Genoviz5.4	1111	827	42	74.44
Genoviz6	1077	840	42	77.99
Genoviz6.1	1059	817	42	77.15
Genoviz6.2	1156	306	42	26.47
Genoviz6.3	1242	226	42	18.2
HTMLUnit2008	497	427	42	85.92
HTMLUnit2009	1059	121	42	11.43
HTMLUnit2010	1296	364	42	28.09
JEdit5.2	1265	13	42	1.03
Jikesrvm2	1332	107	42	8.03
Jikesrvm3	2098	440	42	20.97
Jikesrvm3.1	2290	71	42	3.1
Jitterbit1.1	6141	533	42	8.68
Jitterbit1.2	12247	145	42	1.18
Jmol2	268	38	42	14.18
Jmol3	275	35	42	12.73
Jmol4	297	81	42	27.27
Jmol5	324	92	42	28.4
Jmol6	378	294	42	77.78
Jmol7	405	248	42	61.23
Jmol8	453	145	42	32.01
Jmol9	459	176	42	38.34
Jmol10	556	107	42	19.24
Jmri2	2730	124	42	4.54
Jmri2.2	3337	175	42	5.24
Jmri2.4	3727	1388	42	37.24
Jmri2.6	4059	252	42	6.21
Jppf4	1933	258	42	13.35
Jppf4.1	1934	133	42	6.88
Jppf4.2	1956	135	42	6.9
Jppf5	1879	274	42	14.58
Jppf5.1	1912	55	42	2.88
Jtds23072009	156	35	42	22.44
Jump1.5	2791	131	42	4.69
Jump1.6	2909	104	42	3.58
Jump1.7	3016	96	42	3.18
Jump1.8	3064	107	42	3.49
Jump1.9	3231	281	42	8.7
OmegaT3.1	1204	45	42	3.74
OmegaT3.5	1391	96	42	6.9
OmegaT3.6	1476	97	42	6.57
Runawfe3.5	5029	0	42	0
Runawfe3.6	5237	5	42	0.1
Runawfe4.1	2882	369	42	12.8
Runawfe4.2	3408	0	42	0
Saros1.0.6	329	69	42	20.97
Tango2008	4299	151	42	3.51
Unicore1.2	412	90	42	21.84
Unicore1.3	466	54	42	11.59
Unicore1.4	477	202	42	42.35
Unicore1.5	728	69	42	9.48
Unicore1.6	834	234	42	28.06
Xaware5	882	120	42	13.61
Xaware5.1	994	51	42	5.13
Xaware6	1001	0	42	0

Table 5.3: ELFF method-level dataset statistics

Project Name	No. of Methods	No. of Defective Methods	No. of Attributes	% Defective Methods
AutoPlot 2012	15781	191	33	1.2
Cdk1	9576	0	33	0
Cdk1.1	4276	73	33	1.7
Cdk1.2	4366	0	33	0
Cmusphinx3.6	4819	15	33	0.30
Cmusphinx3.7	4826	10	33	0.20
Controltier3	6078	0	33	0
Controltier3.1	5799	52	33	0.9
Controltier3.2	4946	0	33	0
Drjava2008	15748	1012	33	6.40
Drjava2009	3333	130	33	3.90
Drjava2010	4946	100	33	0.2
Eclemma2	896	9	33	1.00
Eclemma2.1	1081	37	33	3.40
Ejit3	3357	48	33	1.43
Genoviz5.4	1451	141	33	9.7
Genoviz6	1269	117	33	9.20
Genoviz6.1	4701	504	33	10.70
Genoviz6.2	5704	210	33	3.70
Genoviz6.3	8509	221	33	2.6
HTMLUnit2008	4715	427	33	9.00
HTMLUnit2009	1096	16	33	1.50
HTMLUnit2010	7747	259	33	3.30
JEdit5.2	5400	9	33	0.17
Jikesrvm2	4489	43	33	0.96
Jikesrvm3	5113	149	33	2.90
Jikesrvm3.1	3890	20	33	0.50
Jitterbit1.1	1155	22	33	1.90
Jitterbit1.2	11246	26	33	0.23
Jmol2	1347	38	33	2.80
Jmol3	1402	35	33	2.50
Jmol4	1419	81	33	5.70
Jmol5	89	4	33	4.50
Jmol6	2170	280	33	12.90
Jmol7	2484	248	33	9.98
Jmol8	1910	85	33	4.50
Jmol9	3433	176	33	5.12
Jmol10	3957	81	33	2.05
Jmri2	4910	42	33	0.85
Jmri2.2	17011	175	33	1.03
Jmri2.4	11564	802	33	6.90
Jmri2.6	2637	30	33	1.13
Jppf4	2054	57	33	2.78
Jppf4.1	2058	28	33	1.36
Jppf4.2	298	5	33	1.67
Jppf5	448	16	33	3.57
Jppf5.1	3618	19	33	0.53
Jtds23072009	2005	27	33	1.34
Jump1.5	5194	51	33	0.98
Jump1.6	3692	30	33	0.81
Jump1.7	4064	26	33	0.64
Jump1.8	3090	13	33	0.42
Jump1.9	11661	201	33	1.72
OmegaT3.1	4347	35	33	0.81
OmegaT3.5	1812	30	33	1.66
OmegaT3.6	2331	34	33	1.46
Runawfe3.5	3282	0	33	0
Runawfe3.6	470	0	33	0
Runawfe4.1	1402	46	33	3.28
Runawfe4.2	2136	0	33	0
Saros1.0.6	749	31	33	4.13
Tango2008	3246	18	33	0.55
Unicore1.2	1756	67	33	3.80
Unicore1.3	952	21	33	2.20
Unicore1.4	2575	202	33	7.84
Unicore1.5	4007	69	33	1.72
Unicore1.6	2171	113	33	5.20
Xaware5	792	18	33	2.27
Xaware5.1	6033	43	33	0.71
Xaware6	2843	0	33	0

5.3.4 Data preprocessing

To ensure the reliability of the collected datasets, first, the researcher gained a basic understanding of the datasets used in this research by determining the origin, size, and format of each dataset. The proposed optimal decision framework was applied to properly preprocess the datasets. This research was conducted using the well-known NASA datasets from the PROMISE repository (Sayyad Shirabad & Menzies, 2005) and the ELFF datasets (Shippey et al., 2016).

Each dataset was preprocessed using the proposed optimal decision technique to select feature subsets with high predictive strength from among the available features, followed by the elimination of outliers from the datasets. A unique identifier was then assigned to each module in each dataset to further preprocess both the class- and method-level data, followed by the determination of the defect density, defect velocity and defect introduction time in each dataset. This process was repeated for all datasets. Unique identifiers were assigned to both defective and defect-free modules to allow the number of defective modules in each dataset to be easily recorded and to avoid redundancy among the data points. From the number of defective modules, the percentage of defective modules in each dataset could be correctly determined based on the assigned unique identifiers. A class or method was considered defective if the error count among the attributes was greater than or equal to 1. Files with incomplete attributes were not counted. This process was repeated for each file in every dataset. This was a time-consuming process; however, all 22,838 modules in the NASA datasets as well as all 131,034 classes and 289,132 methods in the ELFF projects used in this research were carefully cleaned and preprocessed. This cleaning process was necessary to determine whether the datasets were suitable for training and validation and to ensure that the prediction models (regression models and classifiers) applied in this study produced unbiased outcomes.

An input space was created to recapture unreported faults. Thereafter, the samples were divided into training and validation sets with a 10-fold cross-validation structure to ensure unbiased results. The training sets consisted of 80% of the data and were used to train the prediction models. If models are properly trained, they will produce reliable results during validation; this is why cross-validation sampling was applied when training the models to ensure highly accurate results with regard to model performance. The steps of the applied data preprocessing procedure are further elucidated in Algorithms 1, 2 and 3.

5.3.5 Steps of filter-based feature selection

Algorithm 1 presents the steps applied to perform filter-based feature selection. For each dataset, the individual rank score of each feature, denoted by $rank_{id}$, was determined to identify the relevant features and ascertain their correlations with the target class of modules (classes or methods) in the dataset, i.e., the defective class. To achieve the above objective, an optimal threshold was set that was equal to the average rank score of all features, denoted by $Rank_{ave}$. Features whose rank scores were equal to or greater than this threshold were selected, and those with lower rank scores were discarded, as presented in Algorithm 1.

5.3.6 Steps of determining the impacts of the defect density, defect introduction time and defect velocity on the numbers of defects at the class and method levels

Algorithm 2 presents the steps of determining the impacts of the defect density, defect introduction time and defect velocity on the number of defects, and Algorithm 3 presents the steps of the data cleaning approach applied to determine the average classifier performance. In all three algorithms (Algorithms 1, 2, and 3), the module-, class- and method-level datasets serve as the input for (1) filtering the features in the class- and method-level datasets; (2) determining the impact of the defect density, defect velocity and defect

Algorithm 1 Steps of filter-based feature selection

procedure STEPS OF FEATURE SELECTION FROM EACH DATASET
Input: Datasets DS with the universal feature set
Output: Datasets with the selected feature subset
    Analyze the features in each dataset
    Record the total number of features in each dataset
    Apply the feature selection module to each dataset
    Apply the feature scoring method to rank each feature
    Record the rank score of each feature in each dataset
    Determine the $average$ rank score of all features in each dataset
    for i = 1 to n (each feature) **do**
        Set optimal value = average rank score ($Rank_{ave}$)
        if $individual\ feature\ rank\ score\ rank_{id} \geq Rank_{ave}$ **then**
            Include feature in subset
        else
            Discard feature
            Record the total number of selected features for which $rank_{id} \geq Rank_{ave}$
        end if
    end for
end procedure

introduction time; and (3) determining the average classifier performance. In these algorithms, DS represents the datasets. In Algorithm 2, CD represents both classes and methods in the datasets. m represents the number of datasets, and n represents the number of classes and methods in the datasets. To determine the impact of the defect density, defect velocity and defect introduction time, the following steps were applied. First, all classes and methods in each dataset were analyzed to assess the defect density in each class and method. Subsequently, the defect velocity and defect introduction time in each class and method were evaluated, as presented in Algorithm 2. The defect density of each class and method was determined by dividing the number of defects in the class or method by the size of the class or method, and subsequently, the average defect introduction time and average defect velocity were determined. The defect velocity v is expressed as

$$v = \frac{\Delta\ defect\ position\ (x)}{\Delta\ time\ (t)} \tag{5.23}$$

Algorithm 2 Steps of determining the impacts of the defect density, defect introduction time and defect velocity in terms of correlation with the number of defects

procedure STEPS OF DETERMINING THE IMPACTS OF THE DEFECT DENSITY, DEFECT INTRODUCTION TIME, AND DEFECT VELOCITY IN TERMS OF CORRELATION WITH THE NUMBER OF DEFECTS AT BOTH THE CLASS AND METHOD LEVELS
Input: Datasets DS
Output: Impacts of the defect density, defect introduction time and defect velocity on the number of defects
    Analyze datasets DS_{1-m}
    Analyze classes and methods CD_{1-n} of each DS_{1-m}
    for i = 1 to n **do**
        Determine the average defect density, $g = \frac{no.\ of\ defects}{class\ or\ method\ size}$
        Determine the average defect velocity and defect introduction time, $v = \frac{\Delta Defect\ position(x)}{\Delta time(t)}$
        Determine the correlations between the average defect density g, average defect velocity v and average defect introduction time t and the number of defects in datasets DS_{1-m}
        Record the impacts of g, v and t on the number of defects
    end for
end procedure

5.3.7 Steps of determining the average classifier performance

This section elaborates on the process used to accurately determine the average classifier performances of the six state-of-the-art classifiers investigated in this study. The datasets served as the input, as presented in Algorithms 1, 2 and 3. In the cross-validation approach, the data were split into 10 folds, with a relative training set size of 80% and a validation set size of 20%. The training data and validation data were independent of each other to ensure unbiased prediction outcomes. The reason for using 80% of the data for training was to effectively train the classification algorithms to avoid biased results. Notably, if models are trained using only a small amount of data, they may learn inaccurately and produce unreliable results. Thus, all the classification algorithms applied in this study were trained using 80% of the available data. For this purpose, cross-validation sampling was chosen as the approach used in this study. Throughout the experiment, $A \times B$ cross-validation was used, representing a 10-fold cross-validation strategy for both the training and validation sets, with a value of K=10, as presented in Algorithm 3. n represents the number of classes

Algorithm 3 Data cleaning to determine average classifier performance

procedure STEPS OF DETERMINING AVERAGE CLASSIFIER PERFORMANCE
Input: Datasets DS
Output: Average classifier performance $\bar{x}$ based on $A \times B$ cross-validation
    A = 10 /* number of folds for cross-validation, K=10
    B = 10 /* number of repetitions of training/validation
    Preprocess datasets (DS_{1-n})
    for i = 1 to n **do**
        Assign an ID to each instance of (DS_{1-n})
        Check attributes of each instance
        Check for missing values in each instance
        Check for outliers and inliers
        for j = 1 to n **do**
            Identify and record faulty classes and methods
            Divide DS_{1-n} into two parts: 80% and 20%, for **training** and **validation**, respectively
            Create an input space S for unreported faulty instances $\hat{x}$ and x
            Apply RBF kernel to training and validation data
            Compute a weighted sum (w_n and y_m) for x and $\hat{x}$

$$hx = sign \sum_{i=1}^{n} w_n \, exp \, -\gamma \, \|x - x_n\|^2 \, , \hat{x} = sign \sum_{i=1}^{n} y_m \, exp \, -\gamma \, \|x - x_m\|^2$$

$$\hat{x} \; and \; x \in \{0, 1\} = outcome \; that \; is \; of \; interest$$

$$= \begin{cases} 1, & \text{if } error\_count \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

            if `error_count` ≥ 1 **then**
                Target class = defective
            else
                Target class = defect-free
                Record the number of defects
                Calculate the percentage of defects in each class
            end if
            Evaluate the individual classifier performance x
            Calculate the average classifier performance $\overline{x} = \frac{1}{n} \sum_{i=1}^{N} x_n = \frac{1}{n} (x_1 + x_2 + \ldots x_n)$
            Determine the variation of information related to the classifier performance metrics
            Compute the average entropy of the performance metrics
        end for
    end for
end procedure

or methods in the datasets.

Each dataset was preprocessed as follows. First, a unique identifier was assigned to each module in every dataset to enable the researcher to pinpoint the defective modules. Thereafter, the attributes in each dataset were checked to easily identify missing values. The outliers were then separated from the inliers because only the clustered inliers are suitable for use in prediction. The number of errors in each dataset was recorded as well. If the error count among the attributes was greater than or equal to 1, the corresponding module instance was considered defective, whereas an error count of 0 indicated a defect-free module. Thereafter, an input space was created to recapture unreported faults in both the training and validation datasets, as presented in Algorithm 3. The six state-of-the-art classifiers were implemented on each dataset to determine their individual classification performances. Thereafter, the average classification performance of each classifier on all datasets was determined. Finally, these average performances were compared.

5.3.8 Sampling strategy

This section elaborates on the selection of the sampling strategy applied in this study. The sampling strategy was selected using an optimal decision-making approach similar to that described previously. The assessment of any classifier's performance depends on the data sampling method applied. For the experiments reported here, three potential sampling strategies were considered, namely, cross-validation, leave-one-out and random sampling.

5.3.8.1 Cross-validation

The researcher chose to apply a cross-validation sampling strategy to achieve accurate results and also because, for a large dataset, the cross-validation sampling of data split into 10 folds with relative training and validation set sizes of 80% and 20%, respectively, performs better than leave-one-out or random sampling. Figure 5.3 illustrates how the

cross-validation sampling strategy was applied in this study. Each dataset was split into 10 different folds, with each fold containing both training and and testing data. Notably, cross-validation sampling ensures that the training and validation data are independent of one another; consequently, with this strategy, learning algorithms tend to produce more reliable results (Tantithamthavorn et al., 2017). For instance, Q. Li et al. (2014) applied cross-validation sampling and achieved promising results.

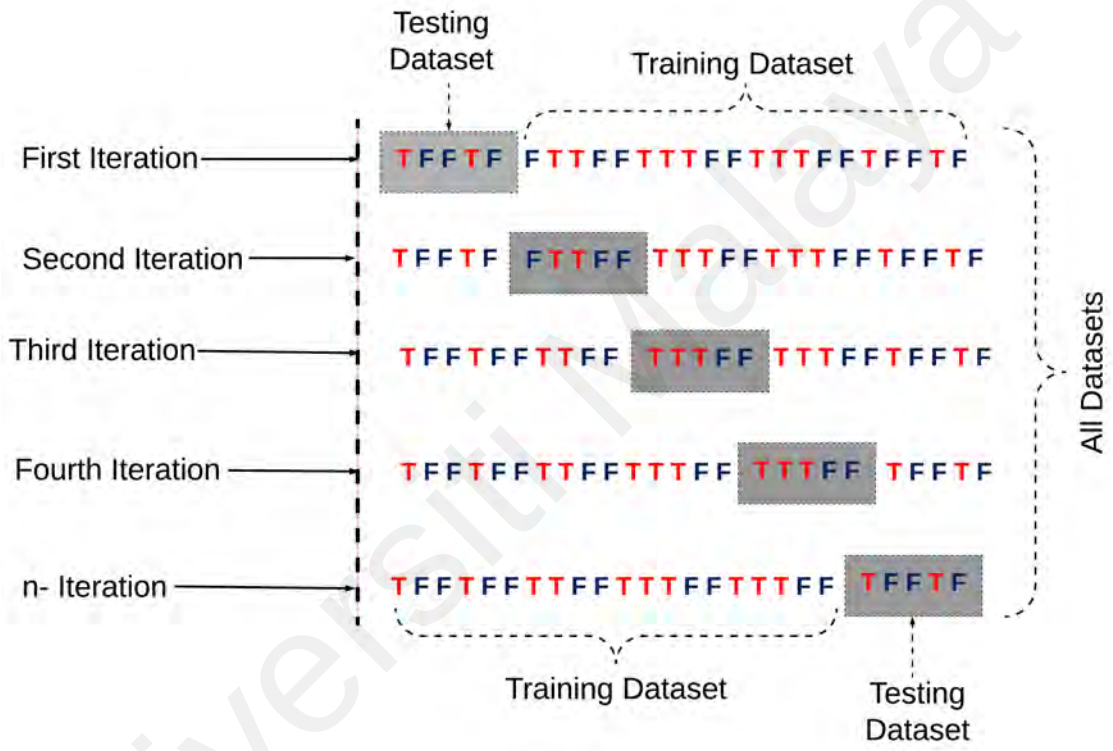


Figure 5.3: Cross-validation sampling strategy

5.3.8.2 Leave-one-out sampling strategy

The leave-one-out sampling strategy requires a longer processing time because only one sample is left out at a time. This sampling method produces more accurate and reliable output but is very slow compared with cross-validation. Tantithamthavorn et al. (2016b) reported that for Brier score evaluations, the leave-one-out sampling strategy is the most stable choice.

5.3.8.3 Random sampling

Random sampling follows an unsystematic approach in which the data are split into training and validation sets in specific proportions such that every sample in the dataset has the same chance of being chosen. The entire procedure is repeated a specified number of times. Because of the random nature of the selection process, the results of this sampling strategy cannot be controlled and, consequently, may be biased.

5.3.9 Classification procedure

This subsection explains in detail the experimental procedure used to evaluate the average performances of the classifiers. The experiments were performed using Orange 2.7. Figure 5.4 illustrates the classification procedure used to evaluate the average performances of the selected classifiers.

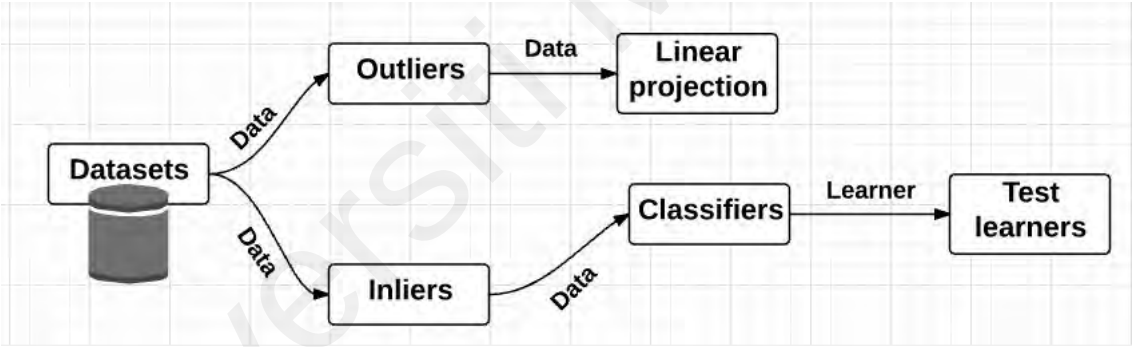


Figure 5.4: Classification procedure

5.3.9.1 Outliers

The classification procedure is illustrated in Figure 2. First, the data were checked to determine whether they contained outliers before the classification algorithms were applied. Outliers are data points that lie far away from the main cluster(s) of data. Outliers may occur in datasets as a result of measurement variations or may be an indication of

experimental error. In this study, outliers were identified during the training phase because they might impact model performance.

5.3.9.2 Inliers

Inliers are the clustered data used for classification. Some inliers may be defective but nevertheless grouped together with defect-free instances in the data distribution, which could pose a challenge in distinguishing defective from defect-free data.

5.3.9.3 Linear projection

To visualize the outliers, each data file was input into an outliers widget followed by a linear projection widget. By feeding the data file into the linear projection widget, the outliers could be identified as data points that were seen to be far from the clustered data (inliers). These clustered data were then fed into the classifier for classification.

5.3.9.4 Test learners

Test learners were implemented to visualize the performance of each classifier. Learners were connected in between the classifiers and the test learners to evaluate the performance of each classifier.

5.4 Summary

This chapter has presented the detailed experimental setup that formed the basis for the evaluation of the proposed approach. The experimental outcomes enabled meaningful conclusions to be drawn when evaluating the clearly defined research hypotheses. The details of the activities performed in the various phases of the proposed optimal decision framework to achieve unbiased experimental outcomes have been presented in the current chapter. For instance, the activities involved in preprocessing both class- and method-level datasets include filter-based feature selection; outlier removal; the derivation of the defect

density, defect velocity and defect introduction time for further preprocessing of the inliers; and the creation of an input space for recapturing unreported faults. These preprocessing activities were performed on the class- and method-level datasets before the application of the learning algorithms to avoid misleading results. In addition, the evaluation metrics applied to determine the performance of the proposed framework have been presented. These evaluation metrics were used to accurately assess the performance of the proposed approach. As described, the proposed optimal decision framework presented in the current research attempts to address data-related issues at both the class and method levels, which, in turn, will help to ensure that the data applied in supervised machine learning studies are free from bias.

CHAPTER 6: RESULTS AND DISCUSSION

This chapter focuses on the experimental results and the interpretation and discussion thereof. The experimental design and modeling have been discussed in the previous chapter, along with the metrics applied for evaluation. The results obtained when applying these metrics during the evaluations are presented in this chapter and are used to draw conclusions. It is therefore important to note that the results presented in this chapter conform to the various phases of the experimental setup as discussed in the previous chapter. With respect to the preprocessing of the class- and method-level datasets, the results obtained during the feature selection phase are presented first, followed by those obtained in the outlier removal and further preprocessing phases. Numerous evaluation metrics are used to assess the consistency of the classifiers' performance. These evaluation metrics include the Classification Accuracy (CA), area under the receiver operating characteristic (ROC) curve (AUC), Brier score, Matthews Correlation Coefficient (MCC), recall, specificity, J-coefficient, F-score, precision, geometric mean (G-mean), information score, and entropy. A large number of metrics are used because the use of many evaluation metrics increases the likelihood of obtaining useful information on the model accuracy, such as the mean performance, and thus results in more reliable predictions (X. Jing et al., 2015). Relevant discussions on the research questions are also provided in this chapter.

6.1 Filter-based feature selection phase

The technique applied to select highly predictive features (that is, features that are highly correlated with the number of defects present in the dataset) is a filter-based feature selection technique, which makes it possible for the researcher to separate irrelevant and redundant features from the relevant features in a dataset. This selection is based on the average rank score of the features in the dataset, which is compared against the individual

rank score of each feature to identify features with high predictive power. Features whose individual rank scores are equal to or greater than the average rank score of all features in the dataset are selected, whereas features whose individual rank scores are less than the average rank score are discarded due to their low predictive strength. Tables 6.1-6.10 list the features present in the National Aeronautics and Space Administration (NASA) class-level datasets, the corresponding rank score of each feature, the selection or nonselection of each feature for use in the analysis, and the reasons for selection or nonselection. The average rank score of the features in each dataset was adopted as a benchmark against which each feature in the dataset was compared. When the individual rank score of a given feature was greater than or equal to the average rank score, that feature was selected as a relevant feature; otherwise, it was not selected. Specifically, the average rank score for the KC1 dataset was 0.303, and a total of 12 of the 21 features were selected, amounting to 57% of the features present in the dataset. The average rank score for the KC2 dataset was 0.371, and 11 of the 21 features (52%) were selected. The average rank score for the KC3 dataset was 0.268, and 27 of the 39 features (69%) were selected. The average rank score for the MC1 dataset was 0.108, and 20 of the 38 features (53%) were selected.

The average rank score for the MC2 dataset was 0.248, and 27 of the 39 features (69%) were selected, whereas the average rank score for the MW1 dataset was 0.217, and 22 of the 37 features (59%) were selected. The PC1 dataset yielded an average rank score of 0.192, leading to the selection of 13 of the 21 features (62%). The average rank score for the PC2 dataset was 0.122, and 18 of the 36 features (50%) were selected; the average rank score for the PC3 dataset was 0.108, and 13 of the 37 features (35%) were selected; and the PC4 dataset yielded an average rank score of 0.143, leading to the selection of 20 of the 37 features (54%). From the feature selection results, we can conclude that most datasets contain numerous irrelevant features, which, if not eliminated, can lead to

unreliable prediction outcomes. It is also important to note that some datasets can contain more irrelevant than relevant features; for example, in the PC3 dataset, the percentage ratio of relevant to irrelevant features was 35%:65%. Note that the rank scores for the ELFF datasets are not listed because the ELFF datasets contain enormous numbers of both classes and methods; consequently, only the results for the NASA datasets are listed for clarity. However, the same feature selection technique was applied to all class- and method-level datasets used in this research. For the same reason, the NASA datasets are used throughout the rest of this chapter to provide a detailed illustration of the results where necessary.

Table 6.1: Features and their corresponding rank scores for the KC1 dataset (f_{ave} = 0.303)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	McCabe's line count of code (loc)	0.3484	✓	×	rank score $\geq f_{ave}$
2	McCabe's "cyclomatic complexity" (v(g))	0.2956	×	✓	rank score $< f_{ave}$
3	McCabe's "essential complexity" (ev(g))	0.2052	×	✓	rank score $< f_{ave}$
4	McCabe's "design complexity" (iv(g))	0.2957	×	✓	rank score $< f_{ave}$
5	Halstead's total operators + operands (n)	0.3551	✓	×	rank score $\geq f_{ave}$
6	Halstead's "volume" (v)	0.3395	✓	×	rank score $\geq f_{ave}$
7	Halstead's "program length" (l)	0.2328	×	✓	rank score $< f_{ave}$
8	Halstead's "difficulty" (d)	0.3875	✓	×	rank score $\geq f_{ave}$
9	Halstead's "intelligence" (i)	0.3429	✓	×	rank score $\geq f_{ave}$
10	Halstead's "effort" (e)	0.2701	×	✓	rank score $< f_{ave}$
11	Halstead's error estimator (b)	0.3390	✓	×	rank score $\geq f_{ave}$
12	Halstead's time estimator (t)	0.2701	×	✓	rank score $< f_{ave}$
13	Halstead's line count (lOCode)	0.3415	✓	×	rank score $\geq f_{ave}$
14	Halstead's count of lines of comments (lOComment)	0.2329	×	✓	rank score $< f_{ave}$
15	Halstead's count of blank lines (lOBlank)	0.3236	✓	×	rank score $\geq f_{ave}$
16	lOCodeAndComment	0.0051	×	✓	rank score $< f_{ave}$
17	unique operators (uniq_Op)	0.3856	✓	×	rank score $\geq f_{ave}$
18	unique operands (uniq_Opnd)	0.3867	✓	×	rank score $\geq f_{ave}$
19	total operators (total_Op)	0.3488	✓	×	rank score $\geq f_{ave}$
20	total operands (total_Opnd)	0.3625	✓	×	rank score $\geq f_{ave}$
21	branchCount	0.2978	×	✓	rank score $< f_{ave}$

Table 6.2: Features and their corresponding rank scores for the KC2 dataset (f_{ave} = 0.371)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	McCabe's line count of code (loc)	0.4069	✓	×	rank score $\geq f_{ave}$
2	McCabe's "cyclomatic complexity" (v(g))	0.3732	✓	×	rank score $\geq f_{ave}$
3	McCabe's "essential complexity" (ev(g))	0.3079	×	✓	rank score $< f_{ave}$
4	McCabe's "design complexity" (iv(g))	0.3517	×	✓	rank score $< f_{ave}$
5	Halstead's total operators + operands (n)	0.3811	✓	×	rank score $\geq f_{ave}$
6	Halstead's "volume" (v)	0.3295	×	✓	rank score $< f_{ave}$
7	Halstead's "program length" (l)	0.3153	×	✓	rank score $< f_{ave}$
8	Halstead's "difficulty" (d)	0.4897	✓	×	rank score $\geq f_{ave}$
9	Halstead's "intelligence" (i)	0.4709	✓	×	rank score $\geq f_{ave}$
10	Halstead's "effort" (e)	0.2388	×	✓	rank score $< f_{ave}$
11	Halstead's error estimator (b)	0.3322	×	✓	rank score $< f_{ave}$
12	Halstead's time estimator (t)	0.2157	×	✓	rank score $< f_{ave}$
13	Halstead's line count (lOCode)	0.3844	✓	×	rank score $\geq f_{ave}$
14	Halstead's count of lines of comments (lOComment)	0.3514	×	✓	rank score $< f_{ave}$
15	Halstead's count of blank lines (lOBlank)	0.4061	✓	×	rank score $\geq f_{ave}$
16	lOCodeAndComment	0.3059	×	✓	rank score $< f_{ave}$
17	unique operators (uniq_Op)	0.4958	✓	×	rank score $\geq f_{ave}$
18	unique operands (uniq_Opnd)	0.4777	✓	×	rank score $\geq f_{ave}$
19	total operators (total_Op)	0.3844	✓	×	rank score $\geq f_{ave}$
20	total operands (total_Opnd)	0.3989	✓	×	rank score $\geq f_{ave}$
21	branchCount	0.3637	×	✓	rank score $< f_{ave}$

Table 6.3: Features and their corresponding rank scores for the KC3 dataset (f_{ave} = 0.268)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.3496	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.3094	✓	×	rank score $\geq f_{ave}$
3	call_pairs	0.3503	✓	×	rank score $\geq f_{ave}$
4	loc_code_and_comment	0.3436	✓	×	rank score $\geq f_{ave}$
5	loc_comments	0.1485	×	✓	rank score $< f_{ave}$
6	condition_count	0.2787	✓	×	rank score $\geq f_{ave}$
7	cyclomatic_complexity	0.3165	✓	×	rank score $\geq f_{ave}$
8	cyclomatic_density	0.1543	×	✓	rank score $< f_{ave}$
9	decision_count	0.2834	✓	×	rank score $\geq f_{ave}$
10	decision_density	0.2534	×	✓	rank score $< f_{ave}$
11	design_complexity	0.3095	✓	×	rank score $\geq f_{ave}$
12	design_density	0.1199	×	✓	rank score $< f_{ave}$
13	edge_count	0.3334	✓	×	rank score $\geq f_{ave}$
14	essential_complexity	0.2160	×	✓	rank score $< f_{ave}$
15	essential_density	0.1441	×	✓	rank score $< f_{ave}$
16	loc_executable	0.3324	✓	×	rank score $\geq f_{ave}$
17	parameter_count	0.0498	×	✓	rank score $< f_{ave}$
18	global_data_complexity	0.3118	✓	×	rank score $\geq f_{ave}$
19	global_data_density	0.0318	×	✓	rank score $< f_{ave}$
20	halstead_content	0.3439	✓	×	rank score $\geq f_{ave}$
21	halstead_difficulty	0.3192	✓	×	rank score $\geq f_{ave}$
22	halstead_effort	0.2703	✓	×	rank score $\geq f_{ave}$
23	halstead_error_est	0.3160	✓	×	rank score $\geq f_{ave}$
24	halstead_length	0.3294	✓	×	rank score $\geq f_{ave}$
25	halstead_level	0.2208	×	✓	rank score $< f_{ave}$
26	halstead_prog_time	0.2703	✓	×	rank score $\geq f_{ave}$
27	halstead_volume	0.3161	✓	×	rank score $\geq f_{ave}$
28	maintenance_severity	0.2642	×	✓	rank score $< f_{ave}$
29	modified_condition_count	0.2728	✓	×	rank score $\geq f_{ave}$
30	multiple_condition_count	0.2784	✓	×	rank score $\geq f_{ave}$
31	node_count	0.3345	✓	×	rank score $\geq f_{ave}$
32	normalized_cyclomatic_complexity	0.1403	×	✓	rank score $< f_{ave}$
33	num_operands	0.3214	✓	×	rank score $\geq f_{ave}$
34	num_operators	0.3336	✓	×	rank score $\geq f_{ave}$
35	num_unique_operands	0.3447	✓	×	rank score $\geq f_{ave}$
36	num_unique_operators	0.3273	✓	×	rank score $\geq f_{ave}$
37	number_of_lines	0.3353	✓	×	rank score $\geq f_{ave}$
38	percent_comments	0.1344	×	✓	rank score $< f_{ave}$
39	loc_total	0.3364	✓	×	rank score $\geq f_{ave}$

Table 6.4: Features and their corresponding rank scores for the MC1 dataset (f_{ave} = 0.108)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.1779	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.1054	×	✓	rank score $<f_{ave}$
3	call_pairs	0.1775	✓	×	rank score $\geq f_{ave}$
4	loc_code_and_comment	0.1524	✓	×	rank score $\geq f_{ave}$
5	loc_comments	0.1306	✓	×	rank score $\geq f_{ave}$
6	condition_count	0.0882	×	✓	rank score $<f_{ave}$
7	cyclomatic_complexity	0.0573	×	✓	rank score $<f_{ave}$
8	cyclomatic_density	0.1191	✓	×	rank score $\geq f_{ave}$
9	decision_count	0.0821	×	✓	rank score $<f_{ave}$
10	design_complexity	0.0265	×	✓	rank score $<f_{ave}$
11	design_density	0.0981	×	✓	rank score $<f_{ave}$
12	edge_count	0.1317	✓	×	rank score $\geq f_{ave}$
13	essential_complexity	0.0810	×	✓	rank score $<f_{ave}$
14	essential_density	0.1830	✓	×	rank score $\geq f_{ave}$
15	loc_executable	0.1813	✓	×	rank score $\geq f_{ave}$
16	parameter_count	0.0082	×	✓	rank score $<f_{ave}$
17	global_data_complexity	0.0117	×	✓	rank score $<f_{ave}$
18	global_data_density	0.0515	×	✓	rank score $<f_{ave}$
19	halstead_content	0.1744	✓	×	rank score $\geq f_{ave}$
20	halstead_difficulty	0.0822	×	✓	rank score $<f_{ave}$
21	halstead_effort	0.0136	×	✓	rank score $<f_{ave}$
22	halstead_error_est	0.1213	✓	×	rank score $\geq f_{ave}$
23	halstead_length	0.1256	✓	×	rank score $\geq f_{ave}$
24	halstead_level	0.0749	×	✓	rank score $<f_{ave}$
25	halstead_prog_time	0.0136	×	✓	rank score $<f_{ave}$
26	halstead_volume	0.1217	✓	×	rank score $\geq f_{ave}$
27	maintenance_severity	0.0213	×	✓	rank score $<f_{ave}$
28	modified_condition_count	0.0935	×	✓	rank score $<f_{ave}$
29	multiple_condition_count	0.0887	×	✓	rank score $<f_{ave}$
30	node_count	0.1359	✓	×	rank score $\geq f_{ave}$
31	normalized_cyclomatic_complexity	0.0888	×	✓	rank score $<f_{ave}$
32	num_operands	0.1259	✓	×	rank score $\geq f_{ave}$
33	num_operators	0.1248	✓	×	rank score $\geq f_{ave}$
34	num_unique_operands	0.2010	✓	×	rank score $\geq f_{ave}$
35	num_unique_operators	0.1363	✓	×	rank score $\geq f_{ave}$
36	number_of_lines	0.1948	✓	×	rank score $\geq f_{ave}$
37	percent_comments	0.1267	✓	×	rank score $\geq f_{ave}$
38	loc_total	0.1927	✓	×	rank score $\geq f_{ave}$

Table 6.5: Features and their corresponding rank scores for the MC2 dataset (f_{ave} = 0.248)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.3237	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.3292	✓	×	rank score $\geq f_{ave}$
3	call_pairs	0.2373	×	✓	rank score $< f_{ave}$
4	loc_code_and_comment	0.1708	×	✓	rank score $< f_{ave}$
5	loc_comments	0.3456	✓	×	rank score $\geq f_{ave}$
6	condition_count	0.3365	✓	×	rank score $\geq f_{ave}$
7	cyclomatic_complexity	0.3209	✓	×	rank score $\geq f_{ave}$
8	cyclomatic_density	0.0956	×	✓	rank score $< f_{ave}$
9	decision_count	0.3412	✓	×	rank score $\geq f_{ave}$
10	decision_density	0.1210	×	✓	rank score $< f_{ave}$
11	design_complexity	0.3032	✓	×	rank score $\geq f_{ave}$
12	design_density	0.1357	×	✓	rank score $< f_{ave}$
13	edge_count	0.3474	✓	×	rank score $\geq f_{ave}$
14	essential_complexity	0.3216	✓	×	rank score $\geq f_{ave}$
15	essential_density	0.2946	✓	×	rank score $\geq f_{ave}$
16	loc_executable	0.2480	✓	×	rank score $\geq f_{ave}$
17	parameter_count	0.1659	×	✓	rank score $< f_{ave}$
18	global_data_complexity	0.2896	✓	×	rank score $\geq f_{ave}$
19	global_data_density	0.0173	×	✓	rank score $< f_{ave}$
20	halstead_content	0.0878	×	✓	rank score $< f_{ave}$
21	halstead_difficulty	0.3782	✓	×	rank score $\geq f_{ave}$
22	halstead_effort	0.2881	✓	×	rank score $< f_{ave}$
23	halstead_error_est	0.2515	✓	×	rank score $\geq f_{ave}$
24	halstead_length	0.2739	✓	×	rank score $\geq f_{ave}$
25	halstead_level	0.2561	✓	×	rank score $\geq f_{ave}$
26	halstead_prog_time	0.2881	✓	×	rank score $\geq f_{ave}$
27	halstead_volume	0.2509	✓	×	rank score $\geq f_{ave}$
28	maintenance_severity	0.0132	×	✓	rank score $< f_{ave}$
29	modified_condition_count	0.3307	✓	×	rank score $\geq f_{ave}$
30	multiple_condition_count	0.3369	✓	×	rank score $\geq f_{ave}$
31	node_count	0.3522	✓	×	rank score $\geq f_{ave}$
32	normalized_cyclomatic_complexity	0.1063	×	✓	rank score $< f_{ave}$
33	num_operands	0.2656	✓	×	rank score $\geq f_{ave}$
34	num_operators	0.2778	✓	×	rank score $\geq f_{ave}$
35	num_unique_operands	0.1456	×	✓	rank score $< f_{ave}$
36	num_unique_operators	0.3404	✓	×	rank score $\geq f_{ave}$
37	number_of_lines	0.3069	✓	×	rank score $\geq f_{ave}$
38	percent_comments	0.1244	×	✓	rank score $< f_{ave}$
39	loc_total	0.2511	✓	×	rank score $\geq f_{ave}$

Table 6.6: Features and their corresponding rank scores for the MW1 dataset (f_{ave} = 0.217)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.3405	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.2945	✓	×	rank score $\geq f_{ave}$
3	call_pairs	0.3711	✓	×	rank score $\geq f_{ave}$
4	loc_code_and_comment	0.0177	×	✓	rank score $< f_{ave}$
5	loc_comments	0.3415	✓	×	rank score $\geq f_{ave}$
6	condition_count	0.3177	✓	×	rank score $\geq f_{ave}$
7	cyclomatic_complexity	0.2619	✓	×	rank score $\geq f_{ave}$
8	cyclomatic_density	0.0607	×	✓	rank score $< f_{ave}$
9	decision_count	0.3146	✓	×	rank score $\geq f_{ave}$
10	decision_density	0.0964	×	✓	rank score $< f_{ave}$
11	design_complexity	0.2771	✓	×	rank score $\geq f_{ave}$
12	design_density	0.0032	×	✓	rank score $< f_{ave}$
13	edge_count	0.3539	✓	×	rank score $\geq f_{ave}$
14	essential_complexity	0.1584	×	✓	rank score $< f_{ave}$
15	essential_density	0.0258	×	✓	rank score $< f_{ave}$
16	loc_executable	0.3112	✓	×	rank score $\geq f_{ave}$
17	parameter_count	0.0171	×	✓	rank score $< f_{ave}$
18	halstead_content	0.2934	✓	×	rank score $\geq f_{ave}$
19	halstead_difficulty	0.1351	×	✓	rank score $< f_{ave}$
20	halstead_effort	0.1462	×	✓	rank score $< f_{ave}$
21	halstead_error_est	0.2451	✓	×	rank score $\geq f_{ave}$
22	halstead_length	0.2482	✓	×	rank score $\geq f_{ave}$
23	halstead_level	0.1443	×	✓	rank score $< f_{ave}$
24	halstead_prog_time	0.1462	×	✓	rank score $< f_{ave}$
25	halstead_volume	0.2450	✓	×	rank score $\geq f_{ave}$
26	maintenance_severity	0.1053	×	✓	rank score $< f_{ave}$
27	modified_condition_count	0.3076	✓	×	rank score $\geq f_{ave}$
28	multiple_condition_count	0.3007	✓	×	rank score $\geq f_{ave}$
29	node_count	0.3689	✓	×	rank score $\geq f_{ave}$
30	normalized_cyclomatic_complexity	0.0557	×	✓	rank score $< f_{ave}$
31	num_operands	0.2542	✓	×	rank score $\geq f_{ave}$
32	num_operators	0.2415	✓	×	rank score $\geq f_{ave}$
33	num_unique_operands	0.3306	✓	×	rank score $\geq f_{ave}$
34	num_unique_operators	0.1636	×	✓	rank score $< f_{ave}$
35	number_of_lines	0.3515	✓	×	rank score $\geq f_{ave}$
36	percent_comments	0.0751	×	✓	rank score $< f_{ave}$
37	loc_total	0.3136	✓	×	rank score $\geq f_{ave}$

Table 6.7: Features and their corresponding rank scores for the PC1 dataset (f_{ave} = 0.192)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	McCabe's line count of code (loc)	0.2676	✓	×	rank score $\geq f_{ave}$
2	McCabe's "cyclomatic complexity" (v(g))	0.1575	×	✓	rank score $< f_{ave}$
3	McCabe's "essential complexity" (ev(g))	0.1134	×	✓	rank score $< f_{ave}$
4	McCabe's "design complexity" (iv(g))	0.1548	×	✓	rank score $< f_{ave}$
5	Halstead's total operators + operands (n)	0.2215	✓	×	rank score $\geq f_{ave}$
6	Halstead's "volume" (v)	0.2289	✓	×	rank score $\geq f_{ave}$
7	Halstead's "program length" (l)	0.0047	×	✓	rank score $< f_{ave}$
8	Halstead's "difficulty" (d)	0.0946	×	✓	rank score $< f_{ave}$
9	Halstead's "intelligence" (i)	0.2136	✓	×	rank score $\geq f_{ave}$
10	Halstead's "effort" (e)	0.1177	×	✓	rank score $< f_{ave}$
11	Halstead's error estimator (b)	0.2338	✓	×	rank score $\geq f_{ave}$
12	Halstead's time estimator (t)	0.1177	×	✓	rank score $< f_{ave}$
13	Halstead's line count (lOCode)	0.2579	✓	×	rank score $\geq f_{ave}$
14	Halstead's count of lines of comments (lOComment)	0.2653	✓	×	rank score $\geq f_{ave}$
15	Halstead's count of blank lines (lOBlank)	0.2705	✓	×	rank score $\geq f_{ave}$
16	lOCodeAndComment	0.2347	✓	×	rank score $\geq f_{ave}$
17	unique operators (uniq_Op)	0.1971	✓	×	rank score $\geq f_{ave}$
18	unique operands (uniq_Opnd)	0.2829	✓	×	rank score $\geq f_{ave}$
19	total operators (total_Op)	0.2236	✓	×	rank score $\geq f_{ave}$
20	total operands (total_Opnd)	0.2171	✓	×	rank score $\geq f_{ave}$
21	branchCount	0.1505	×	✓	rank score $< f_{ave}$

Table 6.8: Features and their corresponding rank scores for the PC2 dataset (f_{ave} = 0.122)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	branch_count	0.1432	✓	×	rank score $\geq f_{ave}$
2	call_pairs	0.1113	×	✓	rank score $<f_{ave}$
3	loc_code_and_comment	0.1861	✓	×	rank score $\geq f_{ave}$
4	loc_comments	0.1149	×	✓	rank score $<f_{ave}$
5	condition_count	0.1001	×	✓	rank score $<f_{ave}$
6	cyclomatic_complexity	0.1690	✓	×	rank score $\geq f_{ave}$
7	cyclomatic_density	0.0673	×	✓	rank score $<f_{ave}$
8	decision_count	0.0884	×	✓	rank score $<f_{ave}$
9	decision_density	0.0651	×	✓	rank score $<f_{ave}$
10	design_complexity	0.1511	✓	×	rank score $\geq f_{ave}$
11	design_density	0.0796	×	✓	rank score $<f_{ave}$
12	edge_count	0.1400	✓	×	rank score $\geq f_{ave}$
13	essential_complexity	0.0632	×	✓	rank score $<f_{ave}$
14	essential_density	0.0227	×	✓	rank score $<f_{ave}$
15	loc_executable	0.1115	×	✓	rank score $<f_{ave}$
16	parameter_count	0.0332	×	✓	rank score $<f_{ave}$
17	halstead_content	0.0652	×	✓	rank score $<f_{ave}$
18	halstead_difficulty	0.1689	✓	×	rank score $\geq f_{ave}$
19	halstead_effort	0.2178	✓	×	rank score $\geq f_{ave}$
20	halstead_error_est	0.1636	✓	×	rank score $\geq f_{ave}$
21	halstead_length	0.1735	✓	×	rank score $\geq f_{ave}$
22	halstead_level	0.0455	×	✓	rank score $<f_{ave}$
23	halstead_prog_time	0.2178	✓	×	rank score $\geq f_{ave}$
24	halstead_volume	0.1634	✓	×	rank score $\geq f_{ave}$
25	maintenance_severity	0.0708	×	✓	rank score $<f_{ave}$
26	modified_condition_count	0.1102	×	✓	rank score $<f_{ave}$
27	multiple_condition_count	0.1001	×	✓	rank score $<f_{ave}$
28	node_count	0.1296	✓	×	rank score $\geq f_{ave}$
29	normalized_cyclomatic_complexity	0.0473	×	✓	rank score $<f_{ave}$
30	num_operands	0.1647	✓	×	rank score $\geq f_{ave}$
31	num_operators	0.1781	✓	×	rank score $\geq f_{ave}$
32	num_unique_operands	0.1463	✓	×	rank score $\geq f_{ave}$
33	num_unique_operators	0.1475	✓	×	rank score $\geq f_{ave}$
34	number_of_lines	0.1786	✓	×	rank score $\geq f_{ave}$
35	percent_comments	0.0746	×	✓	rank score $<f_{ave}$
36	loc_total	0.1800	✓	×	rank score $\geq f_{ave}$

Table 6.9: Features and their corresponding rank scores for the PC3 dataset (f_{ave} = 0.108)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.3347	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.0757	×	✓	rank score $<f_{ave}$
3	call_pairs	0.1726	✓	×	rank score $\geq f_{ave}$
4	loc_code_and_comment	0.2233	✓	×	rank score $\geq f_{ave}$
5	loc_comments	0.2724	✓	×	rank score $\geq f_{ave}$
6	condition_count	0.0662	×	✓	rank score $<f_{ave}$
7	cyclomatic_complexity	0.0784	×	✓	rank score $<f_{ave}$
8	cyclomatic_density	0.1024	×	✓	rank score $<f_{ave}$
9	decision_count	0.0593	×	✓	rank score $<f_{ave}$
10	decision_density	0.1417	✓	×	rank score $\geq f_{ave}$
11	design_complexity	0.0804	×	✓	rank score $<f_{ave}$
12	design_density	0.0782	×	✓	rank score $<f_{ave}$
13	edge_count	0.0819	×	✓	rank score $<f_{ave}$
14	essential_complexity	0.0302	×	✓	rank score $<f_{ave}$
15	essential_density	0.0232	×	✓	rank score $<f_{ave}$
16	loc_executable	0.1006	×	✓	rank score $<f_{ave}$
17	parameter_count	0.0465	×	✓	rank score $<f_{ave}$
18	halstead_content	0.1152	✓	×	rank score $\geq f_{ave}$
19	halstead_difficulty	0.0693	×	✓	rank score $<f_{ave}$
20	halstead_effort	0.0010	×	✓	rank score $<f_{ave}$
21	halstead_error_est	0.0515	×	✓	rank score $<f_{ave}$
22	halstead_length	0.0753	×	✓	rank score $<f_{ave}$
23	halstead_level	0.1019	×	✓	rank score $<f_{ave}$
24	halstead_prog_time	0.0010	×	✓	rank score $<f_{ave}$
25	halstead_volume	0.0517	×	✓	rank score $<f_{ave}$
26	maintenance_severity	0.1729	✓	×	rank score $\geq f_{ave}$
27	modified_condition_count	0.0722	×	✓	rank score $<f_{ave}$
28	multiple_condition_count	0.0734	×	✓	rank score $<f_{ave}$
29	node_count	0.0826	×	✓	rank score $<f_{ave}$
30	normalized_cyclomatic_complexity	0.1199	✓	×	rank score $\geq f_{ave}$
31	num_operands	0.0740	×	✓	rank score $<f_{ave}$
32	num_operators	0.0759	×	✓	rank score $<f_{ave}$
33	num_unique_operands	0.1492	✓	×	rank score $\geq f_{ave}$
34	num_unique_operators	0.1737	✓	×	rank score $\geq f_{ave}$
35	number_of_lines	0.1988	✓	×	rank score $\geq f_{ave}$
36	percent_comments	0.2401	✓	×	rank score $\geq f_{ave}$
37	loc_total	0.1166	✓	×	rank score $\geq f_{ave}$

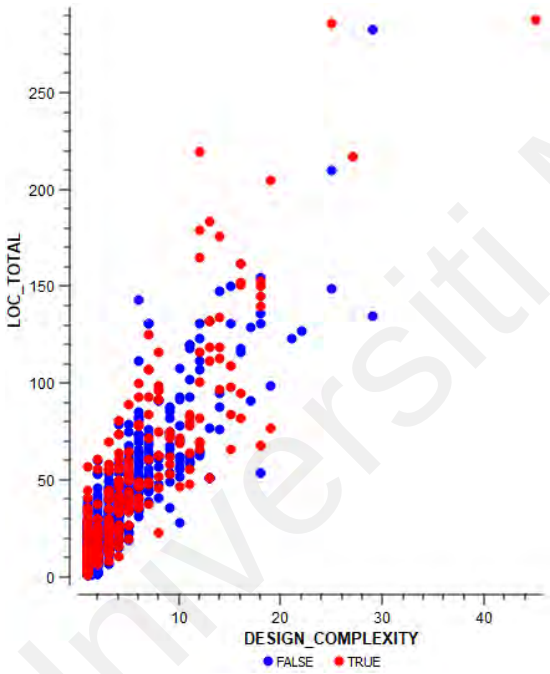
Table 6.10: Features and their corresponding rank scores for the PC4 dataset (f_{ave} = 0.143)

S/N	Feature	Rank score	Selected	Not selected	Reason for selection/nonselection
1	loc_blank	0.1782	✓	×	rank score $\geq f_{ave}$
2	branch_count	0.0051	×	✓	rank score $<f_{ave}$
3	call_pairs	0.0906	×	✓	rank score $<f_{ave}$
4	loc_code_and_comment	0.4243	✓	×	rank score $\geq f_{ave}$
5	loc_comments	0.0889	×	✓	rank score $<f_{ave}$
6	condition_count	0.1530	✓	×	rank score $\geq f_{ave}$
7	cyclomatic_complexity	0.0011	×	✓	rank score $<f_{ave}$
8	cyclomatic_density	0.1868	✓	×	rank score $\geq f_{ave}$
9	decision_count	0.1560	✓	×	rank score $\geq f_{ave}$
10	decision_density	0.3056	✓	×	rank score $\geq f_{ave}$
11	design_complexity	0.0188	×	✓	rank score $<f_{ave}$
12	design_density	0.0892	×	✓	rank score $<f_{ave}$
13	edge_count	0.0370	×	✓	rank score $<f_{ave}$
14	essential_complexity	0.0612	×	✓	rank score $<f_{ave}$
15	essential_density	0.0456	×	✓	rank score $<f_{ave}$
16	loc_executable	0.1698	✓	×	rank score $\geq f_{ave}$
17	parameter_count	0.0828	×	✓	rank score $<f_{ave}$
18	halstead_content	0.1167	×	✓	rank score $<f_{ave}$
19	halstead_difficulty	0.1378	×	✓	rank score $<f_{ave}$
20	halstead_effort	0.1339	×	✓	rank score $<f_{ave}$
21	halstead_error_est	0.1762	✓	×	rank score $\geq f_{ave}$
22	halstead_length	0.1902	✓	×	rank score $\geq f_{ave}$
23	halstead_level	0.0869	×	✓	rank score $<f_{ave}$
24	halstead_prog_time	0.1339	×	✓	rank score $<f_{ave}$
25	halstead_volume	0.1765	✓	×	rank score $\geq f_{ave}$
26	maintenance_severity	0.1845	✓	×	rank score $\geq f_{ave}$
27	modified_condition_count	0.1484	✓	×	rank score $\geq f_{ave}$
28	multiple_condition_count	0.1462	✓	×	rank score $\geq f_{ave}$
29	node_count	0.0485	×	✓	rank score $<f_{ave}$
30	normalized_cyclomatic_complexity	0.1668	✓	×	rank score $\geq f_{ave}$
31	num_operands	0.1912	✓	×	rank score $\geq f_{ave}$
32	num_operators	0.1819	✓	×	rank score $\geq f_{ave}$
33	num_unique_operands	0.1642	✓	×	rank score $\geq f_{ave}$
34	num_unique_operators	0.1365	×	✓	rank score $<f_{ave}$
35	number_of_lines	0.1491	✓	×	rank score $\geq f_{ave}$
36	percent_comments	0.2909	✓	×	rank score $\geq f_{ave}$
37	loc_total	0.2397	✓	×	rank score $\geq f_{ave}$

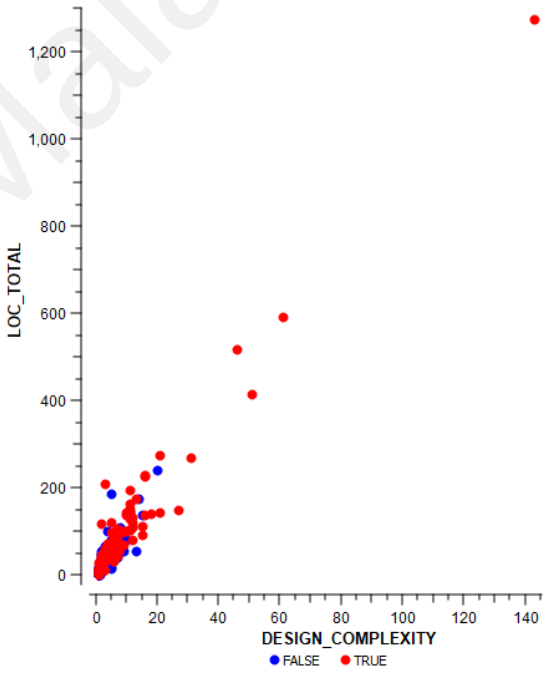
6.2 Outlier removal phase

The results obtained during the outlier removal phase using the NASA datasets are presented in Table 6.11.

The results of outlier removal from the ELFF datasets are presented in Appendix A. Again, these results are not tabulated here due to the large numbers of classes and methods contained in the ELFF datasets. During data preprocessing, it was necessary to visualize the imbalanced nature of the datasets to identify outliers. Figures 6.1-6.5 illustrate the imbalanced nature of the class- and method-level datasets, using the NASA datasets as examples.

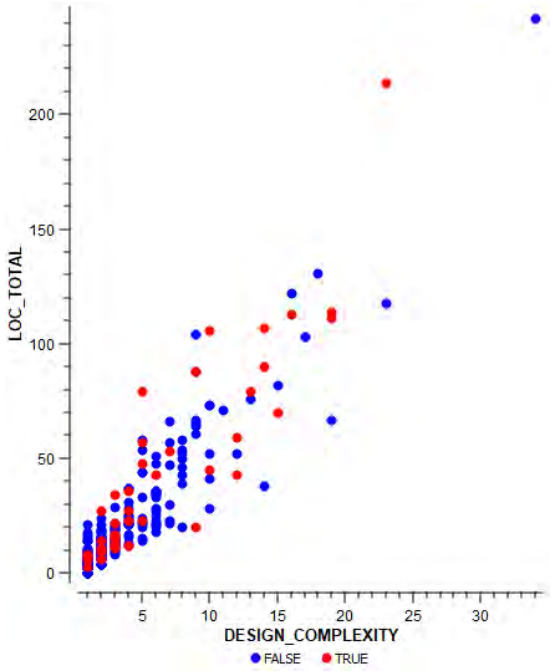


(a) Scatter plot showing data imbalance in KC1

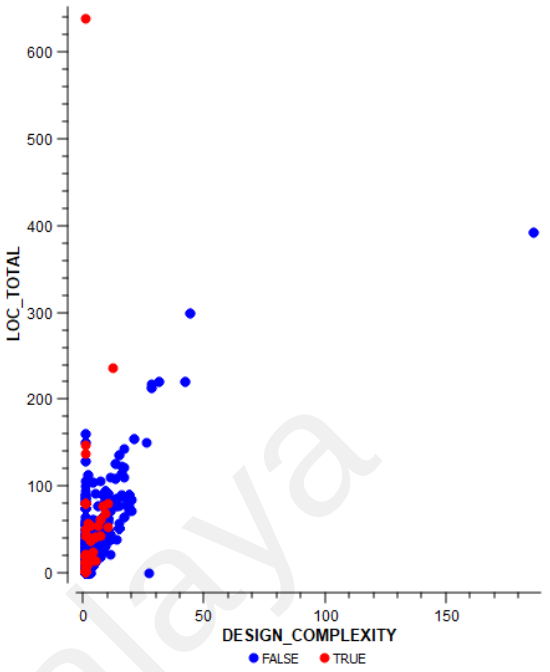


(b) Scatter plot showing data imbalance in KC2

Figure 6.1: Imbalance in KC1 and KC2

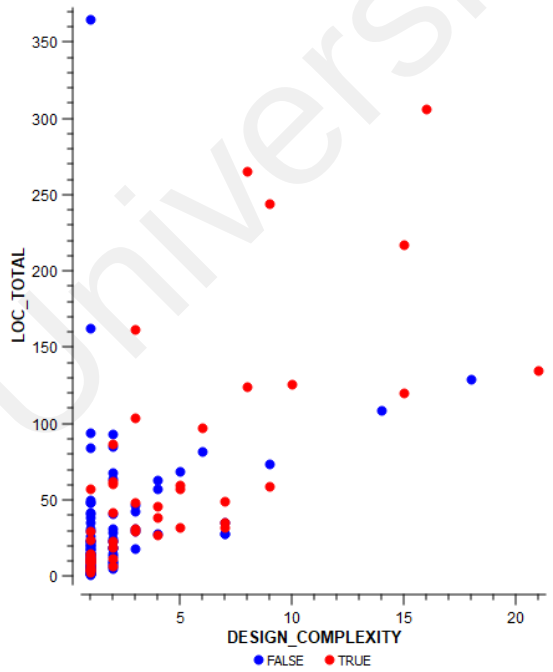


(a) Scatter plot showing data imbalance in KC3

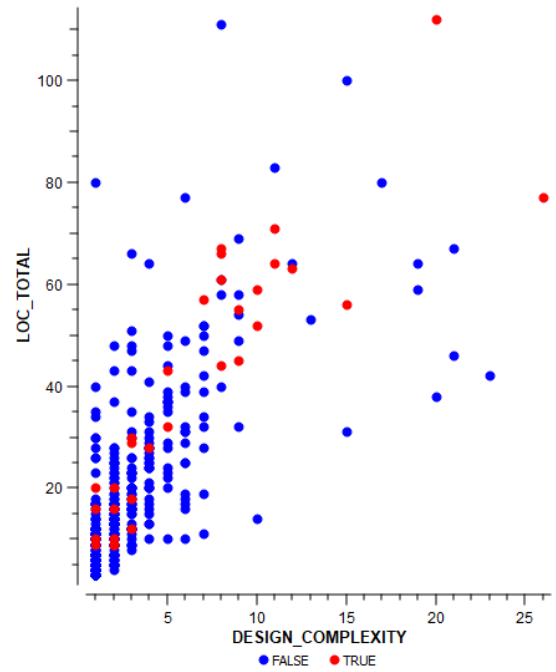


(b) Scatter plot showing data imbalance in MC1

Figure 6.2: Imbalance in KC3 and MC1

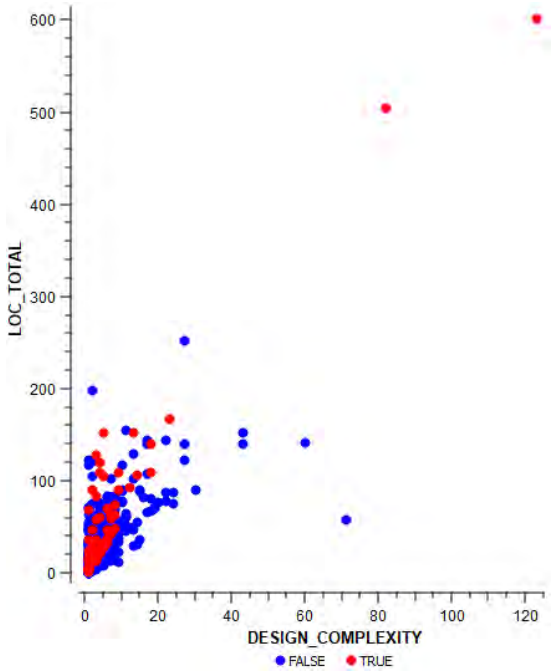


(a) Scatter plot showing data imbalance in MC2

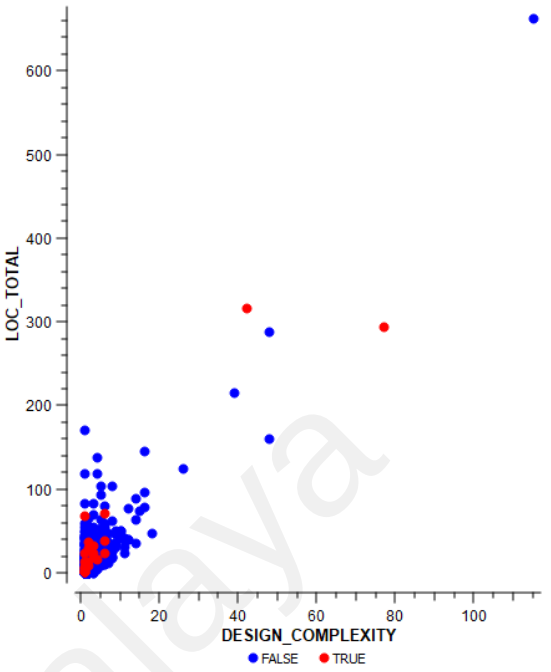


(b) Scatter plot showing data imbalance in MW1

Figure 6.3: Imbalance in MC2 and MW11

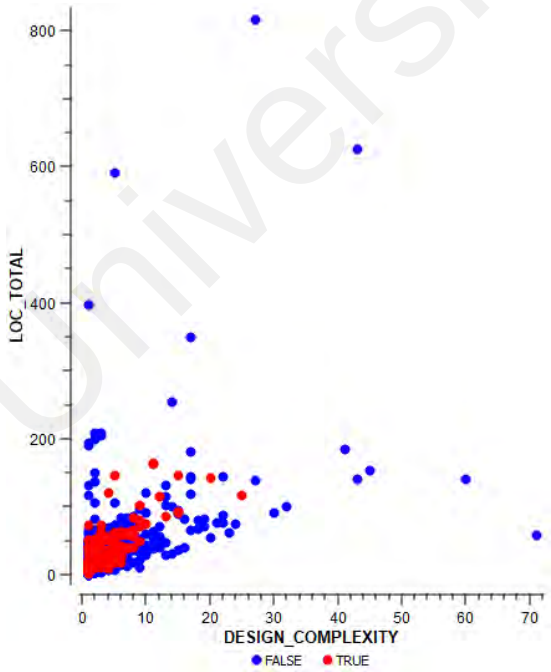


(a) Scatter plot showing data imbalance in PC1

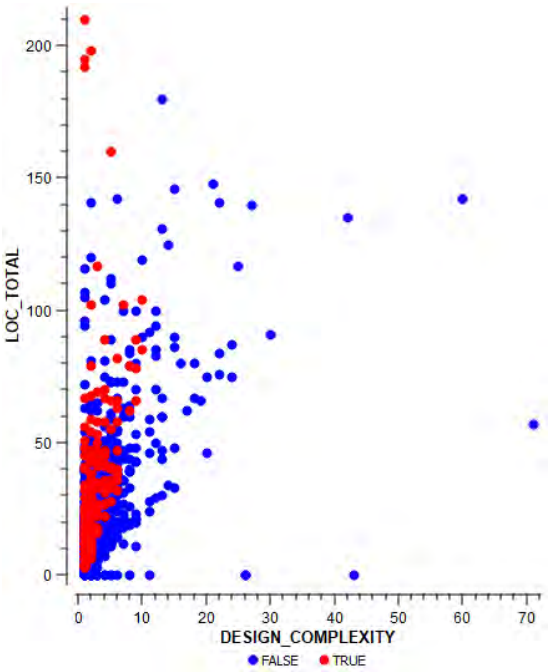


(b) Scatter plot showing data imbalance in PC2

Figure 6.4: Imbalance in PC1 and PC2



(a) Scatter plot showing data imbalance in PC3



(b) Scatter plot showing data imbalance in PC4

Figure 6.5: Imbalance in PC3 and PC4

Table 6.11: Outliers present in the NASA datasets

Dataset	Total number of outliers	Outliers = minority class	Outliers = majority class
KC1	20	12	8
KC2	5	5	0
KC3	2	1	1
MC1	53	6	47
MC2	1	0	1
MW1	4	1	3
PC1	6	2	4
PC2	42	3	39
PC3	7	0	7
PC4	10	4	6
Total	150	34	116

6.3 Assigning unique IDs to classes and methods for identifying defective modules

One of the necessary steps of data preprocessing is to assign unique identifiers to all classes and methods in each dataset based on the optimal decision made for accurate preprocessing. Based on these unique identifiers, the defective classes and methods can be easily identified and recorded. Detailed lists of the identifiers of all defective classes (minority class) in the NASA datasets are presented for illustration. If an identifier is listed, this indicates that the corresponding method or class is defective. Defective methods and classes were selected as follows: if the number of errors among the attributes of the dataset was found to be greater than or equal to 1 for a given method or class, that method or class was considered defective, whereas an error count of 0 indicated a defect-free method or class. Through this means, it was possible to accurately determine the number and percentage of defective classes or methods in each dataset. Considering the time and energy required for data preprocessing, the identifiers of the defective instances in the NASA datasets, which contain 22,838 instances in total, are explicitly presented. However, the individual identifiers of the defective instances for the PC1 and KC1 projects are not listed because they are consecutive. The identifiers of the defective instances for the KC1 project range from 2 to 327; thus, the defective modules constitute 326 of the 2109 total

instances, corresponding to an error percentage of 15.46%. The PC1 dataset was found to contain 77 defective instances out of 1109 total instances, with consecutive identifiers ranging from 2 to 78; thus, the error percentage of PC1 is 6.94%. Tables 6.12 to 5.19 present the identifiers of the defective instances constituting the minority class in the KC2, KC3, MC1, MC2, MW1, PC2, PC3 and PC4 datasets, respectively.

For the ELFF class- and method-level datasets, the identifiers of defective classes and methods are not explicitly presented due to the large numbers of classes and methods in these datasets. However, the percentages of defects found after accurately identifying the defective classes and method are reported for each of these datasets.

Table 6.12 presents the identifiers of the defective classes used to identify the minority class in the KC2 dataset during data preprocessing. The KC2 dataset was found to contain 107 defective instances out of 522 total instances with 21 attributes. This number of defective instances represents 20.50% of the dataset.

Table 6.12: Identifiers of defective modules in KC2

2	3	4	5	6	7	422	423	424	425	426
427	428	429	430	431	432	433	434	435	436	437
438	439	440	441	442	443	444	445	446	447	448
449	450	451	452	453	454	455	456	457	458	459
460	461	462	463	464	465	466	467	468	469	470
471	472	473	474	475	476	477	478	479	480	481
482	483	484	485	486	487	488	489	490	491	492
493	494	495	496	497	498	499	500	501	502	503
504	505	506	507	508	509	510	511	512	513	514
515	516	517	518	519	520	521	522	-	-	-

Table 6.13 presents the identifiers of the defective modules used to identify the minority class in the KC3 dataset during data preprocessing. The KC3 dataset contains 43 defective instances out of 458 total instances with 39 attributes. This number of defective instances represents 9.39% of the dataset.

Table 6.14 presents the identifiers of the defective modules used to identify the minority class in the MC1 dataset during data preprocessing. The MC1 dataset contains 68 defective

Table 6.13: Identifiers of defective modules in KC3

20	21	69	76	88	118	122	142	150	155	156
164	174	179	185	190	195	197	201	214	224	225
245	254	274	275	282	289	296	298	309	341	343
356	359	363	370	429	431	433	441	444	452	-

instances out of 9466 total instances with 38 attributes. This number of defective instances represents 0.72% of the dataset.

Table 6.14: Identifiers of defective modules in MC1

279	354	448	542	560	797	806	878	938	1272	1274
1471	1515	1567	1622	1647	1763	2206	2255	2362	2448	2844
2926	2948	2998	3022	3054	3082	3296	3390	3437	3571	3764
3827	3963	4101	4186	4298	4402	4411	4460	4462	4574	4781
4922	5031	5236	5291	5310	5537	5557	5743	5998	6320	6719
6840	6973	7017	7049	8052	8239	8348	8384	8506	8574	8704
8803	9025	-	-	-	-	-	-	-	-	-

Table 6.15 presents the identifiers used to identify the minority class in the MC2 dataset during data preprocessing. The MC2 dataset contains 48 defective instances out of 161 total instances with 39 attributes. This number of defective instances represents 29.81% of the dataset.

Table 6.15: Identifiers of defective modules in MC2

2	5	12	14	16	21	26	28	33	37	40
44	45	46	50	53	59	60	65	68	77	78
86	87	91	92	93	96	100	102	106	125	128
130	136	138	142	143	144	146	147	148	149	150
151	154	158	160	-	-	-	-	-	-	-

Table 6.16 presents the identifiers used to identify the minority class in the MW1 dataset during data preprocessing. The MW1 dataset contains 31 defective instances out of 403 total instances with 37 attributes. This number of defective instances represents 7.69% of the dataset.

Table 6.16: Identifiers of defective modules in MW1

32	34	48	58	65	75	89	129	136	149	186
192	195	196	213	214	218	257	271	277	281	296
328	330	337	354	370	372	375	389	394	-	-

Table 6.17 presents the identifiers used to identify the minority class in the PC2 dataset during data preprocessing. The PC2 dataset contains 21 defective instances out of 5589 total instances with 36 attributes. This number of defective instances represents 0.38% of the dataset.

Table 6.17: Identifiers of defective modules in PC2

327	330	770	860	1418	1682	2069	2071	2193	2233	2938
2963	3136	3272	3310	3489	3605	4016	4758	4819	5061	-

Table 6.18 presents the identifiers used to identify the minority class in the PC3 dataset during data preprocessing. The PC3 dataset contains 159 defective instances out of 1563 total instances with 37 attributes. This number of defective instances represents 10.17% of the dataset. Table 6.19 presents the identifiers used to identify the minority class in the PC4 dataset during data preprocessing. The PC4 dataset contains 174 defective instances out of 1458 total instances with 37 attributes. This number of defective instances represents 11.93% of the dataset.

Table 6.18: Identifiers of defective modules in PC3

8	17	47	59	75	81	100	109	110	117	127
131	134	141	161	175	183	185	186	194	203	214
227	235	236	237	244	263	300	315	317	327	348
368	390	394	407	415	436	449	458	464	467	471
482	489	492	493	500	506	516	517	533	558	582
588	590	593	610	616	621	654	660	676	727	735
744	746	761	763	770	799	800	815	834	841	845
850	853	866	879	882	888	900	907	914	917	926
939	948	955	964	965	966	970	975	977	984	1005
1024	1025	1030	1040	1045	1085	1107	1115	1118	1119	1124
1132	1138	1139	1148	1160	1204	1207	1212	1230	1241	1254
1262	1276	1279	1283	1284	1288	1294	1302	1303	1310	1312
1319	1322	1338	1340	1344	1348	1349	1361	1362	1384	1401
1415	1427	1429	1432	1434	1441	1443	1475	1489	1490	1495
1497	1501	1503	1509	1528	-	-	-	-	-	-

6.4 Results obtained from further preprocessing of the datasets

After the assignment of unique identifiers to each class and method in the datasets, the inliers in the datasets were further preprocessed to ensure the consistency and reliability

Table 6.19: Identifiers of defective modules in PC4

9	46	49	59	60	67	76	95	101	103	105
106	129	134	146	180	186	195	197	199	204	217
221	224	227	250	253	255	263	269	290	292	308
334	347	349	352	355	363	373	382	383	393	400
410	419	432	437	441	454	468	470	479	483	491
492	501	515	516	533	543	559	567	569	602	621
623	646	654	662	682	687	690	691	704	705	719
722	730	733	739	744	777	779	791	797	798	811
813	816	821	822	830	831	836	838	852	875	878
892	895	908	911	919	921	935	942	958	960	964
974	979	985	987	989	992	1005	1006	1011	1028	1037
1049	1051	1065	1070	1076	1080	1109	1119	1138	1158	1163
1166	1170	1175	1177	1199	1200	1222	1225	1233	1236	1250
1253	1258	1268	1275	1287	1288	1290	1295	1309	1312	1313
1315	1316	1323	1324	1338	1341	1344	1345	1353	1354	1356
1390	1393	1399	1402	1410	1417	1431	1439	1447	-	-

of the data applied in the experiments. In this stage of data preprocessing, the predictor variables (defect density, defect velocity and defect introduction time) in each class- and method-level dataset were derived to exploit their correlation with the number of defects. After verifying the mathematical accuracy of the derived variables, a correlation analysis was performed to determine the relationship between each derived variable and the number of defects.

6.4.1 Correlation coefficient results

The results for the correlation coefficients of the derived predictor variables are presented in Tables 6.20-6.22.

Table 6.20: Correlation coefficients between the predictor variables and the number of defects for the NASA datasets

Variable	Correlation Coefficient
Defect Introduction Time	-30%
Defect Density	22%
Defect Velocity	98%

On the NASA datasets, the average defect introduction time was found to have a correlation coefficient of -0.30, indicating a negative correlation with the number of defects; the average defect density had a correlation coefficient of 0.22, indicating a weak

Table 6.21: Correlation coefficients between the predictor variables and the number of defects for the ELFF datasets at the class level

Variable	Correlation Coefficient
Defect Introduction Time	-11%
Defect Density	61%
Defect Velocity	94%

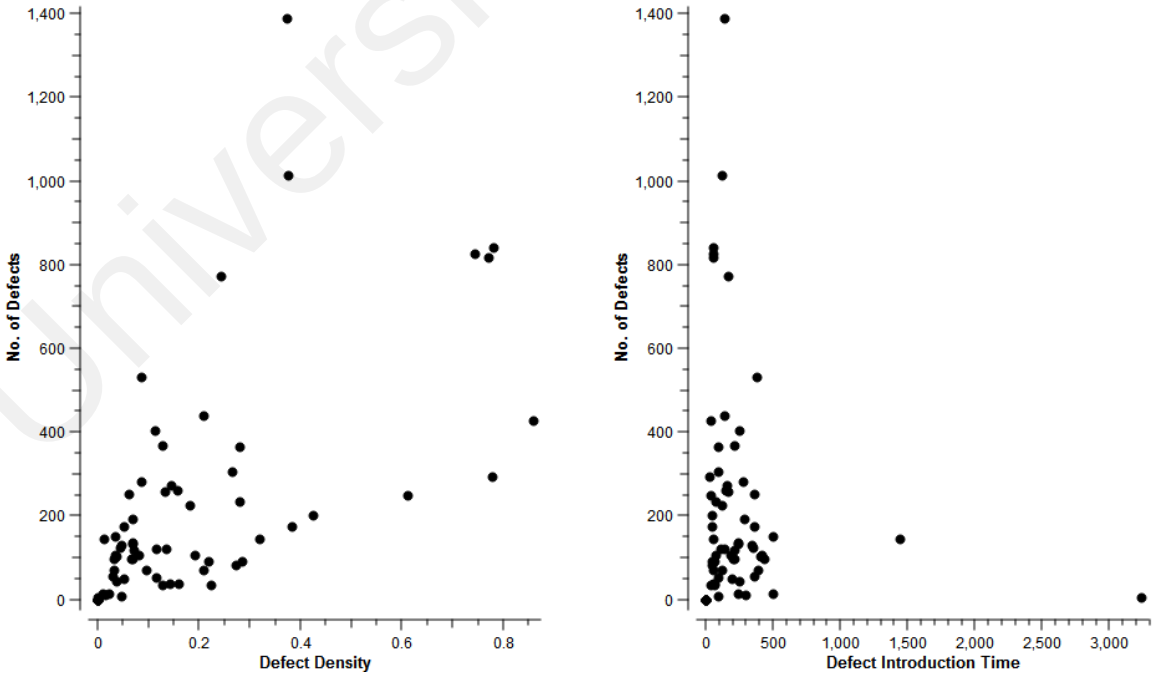
Table 6.22: Correlation coefficients between the predictor variables and the number of defects for the ELFF datasets at the method level

Variable	Correlation Coefficient
Defect Introduction Time	-4%
Defect Density	60%
Defect Velocity	93%

positive correlation with the number of defects; and the average defect velocity had a correlation coefficient of 0.98, showing a strong positive correlation with the number of defects, as presented in Table 6.20. A similar class-level correlation analysis was also performed using the ELFF datasets to determine the relationships between the predictor variables and the number of defects. The average defect introduction time was found to have a correlation coefficient of -0.11, similarly indicating a negative correlation with the number of defects at the class level; the average defect density had a correlation coefficient of 0.61, indicating a moderate positive correlation with the number of defects; and the average defect velocity had a correlation coefficient of 0.94, similarly showing a strong positive correlation with the number of defects, as presented in Table 6.21. To confirm these correlation results, a method-level analysis was further conducted using the ELFF datasets. The average defect introduction time was found to have a correlation coefficient of -0.04, again indicating a negative correlation with the number of defects; the average defect density had a correlation coefficient of 0.60, again indicating a moderately positive correlation with the number of defects; and the average defect velocity had a high correlation coefficient of 0.93, again showing a strong positive correlation with the number of defects, as presented in Table 6.22.

6.4.2 Statistics of the preprocessed class- and method-level datasets and graphical illustration of the impact of the derived optimal variables

The module-level statistics of the preprocessed NASA datasets are presented in Table 6.23. The statistics of the preprocessed class- and method-level ELFF datasets are presented in Tables 6.24 and 6.25, respectively. Figure 6.6 (a and b) presents graphical illustrations of the impacts of the defect density and defect introduction time on the number of defects as found at the class level in the ELFF datasets, whereas Figure 6.7 illustrates the impact of the defect velocity on the number of defects in the class-level ELFF datasets. Similarly, Figure 6.8 (a and b) presents graphical illustrations of the impacts of the defect density and defect introduction time on the number of defects as found at the method level in the ELFF datasets, whereas Figure 6.9 illustrates the impact of the defect velocity on the number of defects in the method-level ELFF datasets. In each plot in these figures, the number of defects is plotted on the y-axis, whereas the corresponding predictor variable is plotted on the x-axis.



(a) Effect of the defect density

(b) Effect of the defect introduction time

Figure 6.6: Effects of the defect density and defect introduction time on the number of defects at the class level in the ELFF datasets

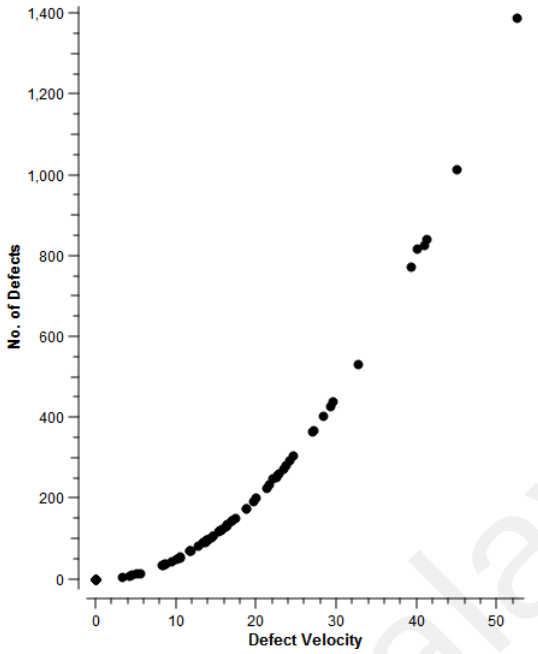
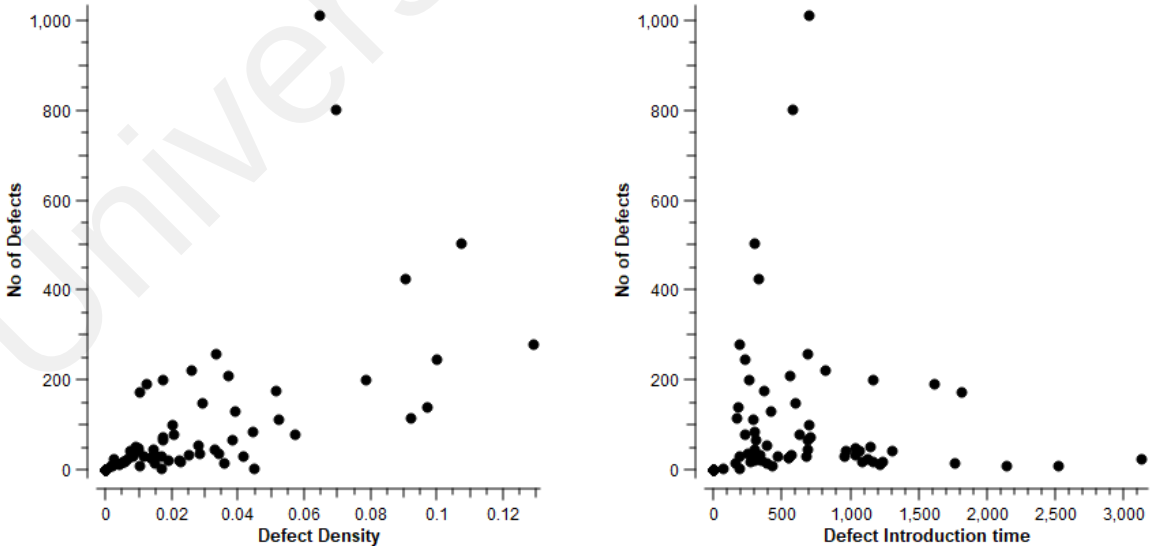


Figure 6.7: Effect of the defect velocity on the number of defects at the class level in the ELFF datasets



(a) Effect of the defect density

(b) Effect of the defect introduction time

Figure 6.8: Effects of the defect density and defect introduction time on the number of defects at the method level in the ELFF datasets

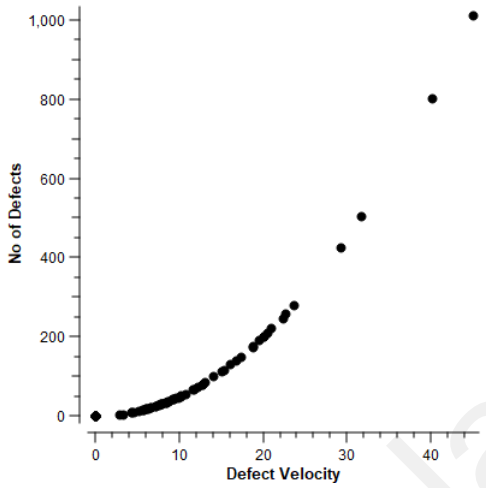


Figure 6.9: Effect of the defect velocity on the number of defects at the method level in the ELFF datasets

Table 6.23: Module-level statistics of the preprocessed NASA datasets

Dataset	No. of Modules	No. of Defective Modules	No. of Attributes	% Defective Modules	Defect Density (num/unit project size)	Defect Introduction Time (days)	Defect Velocity (num/day)
KC1	2109	326	21	15.46	0.1545	165	25.49
KC2	522	107	21	20.50	0.205	71	14.56
KC3	458	43	39	9.39	0.0934	99	9.25
MC1	9466	68	38	0.72	0.0072	1622	11.68
MC2	161	48	39	29.81	0.2981	33	9.84
MW1	403	31	37	7.69	0.0769	102	7.84
PC1	1109	77	21	6.94	0.0694	179	12.42
PC2	5589	21	36	0.38	0.0038	1715	6.52
PC3	1563	159	37	10.17	0.1017	175	17.80
PC4	1458	174	37	11.93	0.1193	165	18.61
Total	22,838	1054	326	4.62	1.1293	4317	134.01

Table 6.24: Class-level statistics of the preprocessed ELFF datasets

Project Name	No. of Classes	No. of Defects	No. of Attributes	% Defects	Defect Density (num/unit project size)	Defect Introduction Time (days)	Defect Velocity (num/day)
AutoPlot 2012	2800	192	42	6.86	0.0686	286	19.62
Cdk1	1678	0	42	0	0	0	0
Cdk1.1	1670	261	42	15.63	0.1563	146	22.82
Cdk1.2	1717	0	42	0	0	0	0
Cmusphinx3.6	665	15	42	2.26	0.0226	243	5.49
Cmusphinx3.7	665	10	42	1.5	0.015	298	4.47
Controltier3	1659	0	42	0	0	0	0
Controltier3.1	1658	117	42	7.06	0.0706	217	15.32
Controltier3.2	1683	0	42	0	0	0	0
Drjava2008	2697	1013	42	37.56	0.3756	120	45.07
Drjava2009	3196	774	42	24.22	0.2422	162	39.24
Drjava2010	3549	403	42	11.36	0.1136	250	28.4
Eclemma2	196	9	42	4.59	0.0459	92	4.22
Eclemma2.1	233	37	42	15.88	0.1588	54	8.58
Genoviz5.4	1111	827	42	74.44	0.7444	55	40.94
Genoviz6	1077	840	42	77.99	0.7799	53	41.33
Genoviz6.1	1059	817	42	77.15	0.7715	52	40.12
Genoviz6.2	1156	306	42	26.47	0.2647	93	24.62
Genoviz6.3	1242	226	42	18.2	0.182	117	21.29
HTMLUnit2008	497	427	42	85.92	0.8592	34	29.21
HTMLUnit2009	1059	121	42	11.43	0.1143	136	15.54
HTMLUnit2010	1296	364	42	28.09	0.2809	96	26.97
JEdit5.2	1265	13	42	1.03	0.0103	496	5.11
Jikesrvm2	1332	107	42	8.03	0.0803	182	14.61
Jikesrvm3	2098	440	42	20.97	0.2097	141	29.57
Jikesrvm3.1	2290	71	42	3.1	0.031	384	11.9
Jitterbit1.1	6141	533	42	8.68	0.0868	376	32.64
Jitterbit1.2	12247	145	42	1.18	0.0118	1441	17
Jmol2	268	38	42	14.18	0.1418	61	8.65
Jmol3	275	35	42	12.73	0.1273	66	8.4
Jmol4	297	81	42	27.27	0.2727	47	12.82
Jmol5	324	92	42	28.4	0.284	48	13.63
Jmol6	378	294	42	77.78	0.7778	31	24.11
Jmol7	405	248	42	61.23	0.6123	36	22.04
Jmol8	453	145	42	32.01	0.3201	53	16.97
Jmol9	459	176	42	38.34	0.3834	49	18.79
Jmol10	556	107	42	19.24	0.1924	76	14.62
Jmri2	2730	124	42	4.54	0.0454	347	15.75
Jmri2.2	3337	175	42	5.24	0.0524	357	18.71
Jmri2.4	3727	1388	42	37.24	0.3724	141	52.51
Jmri2.6	4059	252	42	6.21	0.0621	362	22.48
Jppf4	1933	258	42	13.35	0.1335	170	22.69
Jppf4.1	1934	133	42	6.88	0.0688	237	16.31
Jppf4.2	1956	135	42	6.9	0.069	238	16.42
Jppf5	1879	274	42	14.58	0.1458	161	23.47
Jppf5.1	1912	55	42	2.88	0.0288	364	10.48
Jtds23072009	156	35	42	22.44	0.2244	37	8.3
Jump1.5	2791	131	42	4.69	0.0469	345	16.18
Jump1.6	2909	104	42	3.58	0.0358	403	14.43
Jump1.7	3016	96	42	3.18	0.0318	436	13.86
Jump1.8	3064	107	42	3.49	0.0349	419	14.62
Jump1.9	3231	281	42	8.7	0.087	273	23.75
OmegaT3.1	1204	45	42	3.74	0.0374	254	9.49
OmegaT3.5	1391	96	42	6.9	0.069	201	13.87
OmegaT3.6	1476	97	42	6.57	0.0657	212	13.93
Runawfe3.5	5029	0	42	0	0	0	0
Runawfe3.6	5237	5	42	0.1	0.001	3236	3.24
Runawfe4.1	2882	369	42	12.8	0.128	212	27.14
Runawfe4.2	3408	0	42	0	0	0	0
Saros1.0.6	329	69	42	20.97	0.2097	56	11.74
Tango2008	4299	151	42	3.51	0.0351	495	17.37
Unicore1.2	412	90	42	21.84	0.2184	61	13.32
Unicore1.3	466	54	42	11.59	0.1159	90	10.43
Unicore1.4	477	202	42	42.35	0.4235	47	19.9
Unicore1.5	728	69	42	9.48	0.0948	124	11.76
Unicore1.6	834	234	42	28.06	0.2806	77	21.61
Xaware5	882	120	42	13.61	0.1361	114	15.52
Xaware5.1	994	51	42	5.13	0.0513	197	10.11
Xaware6	1001	0	42	0	0	0	0

Table 6.25: Method-level statistics of the preprocessed ELFF datasets

Project Name	No. of Methods	No. of Defects	No. of Attributes	% Defects	Defect Density (num/unit project size)	Defect Introduction Time (days)	Defect Velocity (num/day)
AutoPlot 2012	15781	191	33	1.2	0.0121	1615	19.50
Cdk1	9576	0	33	0	0	0	0
Cdk1.1	4276	73	33	1.7	0.0171	707	12.10
Cdk1.2	4366	0	33	0	0	0	0
Cmusphinx3.6	4819	15	33	0.30	0.0031	1763	5.50
Cmusphinx3.7	4826	10	33	0.20	0.0021	2144	4.50
Controltier3	6078	0	33	0	0	0	0
Controltier3.1	5799	52	33	0.9	0.0089	1142	10.20
Controltier3.2	4946	0	33	0	0	0	0
Drjava2008	15748	1012	33	6.40	0.0643	700	45.00
Drjava2009	3333	130	33	3.90	0.039	413	16.10
Drjava2010	4946	100	33	0.2	0.0020	700	14.1
Eclemma2	896	9	33	1.00	0.0100	423	4.20
Eclemma2.1	1081	37	33	3.40	0.0.0342	251	8.6
Ejit3	3357	48	33	1.43	0.0143	685	9.80
Genoviz5.4	1451	141	33	9.7	0.0972	173	16.80
Genoviz6	1269	117	33	9.20	0.0922	166	15.30
Genoviz6.1	4701	504	33	10.70	0.1072	292	31.70
Genoviz6.2	5704	210	33	3.70	0.0368	557	20.50
Genoviz6.3	8509	221	33	2.6	0.0259	811	20.90
HTMLUnit2008	4715	427	33	9.00	0.0906	323	29.20
HTMLUnit2009	1096	16	33	1.50	0.0146	387	5.70
HTMLUnit2010	7747	259	33	3.30	0.0334	681	22.70
JEdit5.2	5400	9	33	0.17	0.0017	2521	4.30
Jikesrvm2	4489	43	33	0.96	0.0096	967	9.30
Jikesrvm3	5113	149	33	2.90	0.0291	593	17.30
Jikesrvm3.1	3890	20	33	0.50	0.0051	1235	6.30
Jitterbit1.1	1155	22	33	1.90	0.019	349	6.60
Jitterbit1.2	11246	26	33	0.23	0.0023	3127	7.20
Jmol2	1347	38	33	2.80	0.0282	309	8.70
Jmol3	1402	35	33	2.50	0.024	336	8.40
Jmol4	1419	81	33	5.70	0.0571	223	12.70
Jmol5	89	4	33	4.50	0.0449	63	2.80
Jmol6	2170	280	33	12.90	0.129	183	23.70
Jmol7	2484	248	33	9.98	0.0998	223	22.30
Jmol8	1910	85	33	4.50	0.0445	293	13.00
Jmol9	3433	176	33	5.12	0.0513	366	18.80
Jmol10	3957	81	33	2.05	0.0205	621	12.70
Jmri2	4910	42	33	0.85	0.0086	1066	9.20
Jmri2.2	17011	175	33	1.03	0.0103	1817	18.70
Jmri2.4	11564	802	33	6.90	0.0694	577	40.10
Jmri2.6	2637	30	33	1.13	0.0114	680	7.80
Jppf4	2054	57	33	2.78	0.0278	384	10.70
Jppf4.1	2058	28	33	1.36	0.0136	550	7.50
Jppf4.2	298	5	33	1.67	0.0168	188	3.20
Jppf5	448	16	33	3.57	0.0357	158	5.70
Jppf5.1	3618	19	33	0.53	0.0053	1168	6.20
Jtds23072009	2005	27	33	1.34	0.0135	545	7.40
Jump1.5	5194	51	33	0.98	0.0098	1030	10.10
Jump1.6	3692	30	33	0.81	0.0081	955	7.70
Jump1.7	4064	26	33	0.64	0.0064	1127	7.20
Jump1.8	3090	13	33	0.42	0.0042	1213	5.10
Jump1.9	11661	201	33	1.72	0.0172	1164	20.00
OmegaT3.1	4347	35	33	0.81	0.0081	1036	8.40
OmegaT3.5	1812	30	33	1.66	0.0166	467	7.80
OmegaT3.6	2331	34	33	1.46	0.0146	565	8.30
Runawfe3.5	3282	0	33	0	0	0	0
Runawfe3.6	470	0	33	0	0	0	0
Runawfe4.1	1402	46	33	3.28	0.0328	292	9.60
Runawfe4.2	2136	0	33	0	0	0	0
Saros1.0.6	749	31	33	4.13	0.0414	190	7.90
Tango2008	3246	18	33	0.55	0.0055	1086	6.00
Unicore1.2	1756	67	33	3.80	0.0382	303	11.60
Unicore1.3	952	21	33	2.20	0.0221	294	6.50
Unicore1.4	2575	202	33	7.84	0.0784	256	20.10
Unicore1.5	4007	69	33	1.72	0.0172	683	11.70
Unicore1.6	2171	113	33	5.20	0.052	289	15.00
Xaware5	792	18	33	2.27	0.0227	264	6.00
Xaware5.1	6033	43	33	0.71	0.0071	1304	9.30
Xaware6	2843	0	33	0	0	0	0

6.5 Comparison of model performance before and after data preprocessing

For the purpose of comparing the performance of the proposed approach with models trained on raw and preprocessed datasets, the detailed results obtained using only the class-level datasets from both the NASA and ELFF projects will now be presented.

The average classification performances of the six selected classification models on the NASA datasets before and after preprocessing are presented in Tables 6.26 and 6.27. The results indicate that the NASA datasets contain a high number of inconsistencies and consequently require considerable time and energy for preprocessing. The results obtained after preprocessing show that the errors present in the datasets can hinder the proper assessment of the average performance of the learning algorithms. These errors contribute to the level of uncertainty in the datasets.

As seen from a comparison of the average classification performances of the six selected classification models on the ELFF datasets before and after preprocessing, as presented in Tables 6.28 and 6.29, the classifier performance results obtained after data preprocessing are noteworthy. In particular, the results suggest that the ELFF datasets contain fewer inconsistencies than the NASA datasets and consequently require less preprocessing time and energy.

In terms of the levels of uncertainty in the datasets, the results show promising average performances of the learning algorithms after preprocessing, indicating that the preprocessed datasets contain less uncertainty. After the preprocessing of the NASA datasets, the naïve Bayes model achieved the best performance in terms of the CA, recall, specificity, AUC, F-score, MCC, J-coefficient and G-mean. The naïve Bayes model achieved a CA of 80%, whereas the other learning algorithms each achieved a CA of 70%, except the K-Nearest-Neighbors (KNN) algorithm, which achieved a CA of 60%. The Brier score is an important performance metric for probabilistic prediction; a score of 100% is

the worst achievable score, whereas a score close to 0% indicates good performance. The naïve Bayes model, with a Brier score of 24.48%, achieved the best performance in terms of this metric, whereas the other models showed worse performance, with Brier scores higher than 24.48%.

Table 6.26: Average classifier performance before data preprocessing for the NASA datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	0.5929	0.6582	0.6582	0.7144	0.4892	0.4868	0.6582	0.7943	0.2314	0.3164	-1.6998	0.5328
LR	0.8978	0.5949	0.5949	0.8181	0.6194	0.7272	0.5949	0.1549	0.2865	0.1899	-0.1448	0.4030
Neural Network	0.8969	0.6183	0.6184	0.8399	0.6475	0.7351	0.6183	0.1524	0.3300	0.2368	0.0106	0.4489
KNN	0.8799	0.6427	0.6427	0.7459	0.6543	0.5213	0.6427	0.2720	0.3123	0.2854	0.0456	0.4768
SVM	0.8933	0.5581	0.5581	0.7348	0.5654	0.7292	0.5581	0.1656	0.2091	0.1163	0.0130	0.3145
RF	0.8930	0.5842	0.5842	0.8116	0.6076	0.7217	0.5842	0.1586	0.2679	0.1684	-0.0347	0.3794

Table 6.27: Average classifier performance after data preprocessing for the NASA datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	0.8000	0.7619	0.7619	0.9524	0.7619	0.7619	0.7619	0.2448	0.5238	0.5238	0.4969	0.6667
LR	0.7000	0.5952	0.5952	0.5238	0.6000	0.6250	0.5952	0.3997	0.2182	0.1904	0.1089	0.6050
Neural Network	0.7000	0.5000	0.5000	0.6667	0.4118	0.3500	0.5000	0.3711	0.0000	0.0000	0.0921	0.4184
KNN	0.6000	0.4286	0.4286	0.6190	0.6543	0.3700	0.3333	0.4286	0.4898	-0.2182	-0.1429	0.3780
SVM	0.7000	0.5000	0.5000	0.0000	0.4118	0.3500	0.5000	0.5073	0.0000	0.0000	-0.1134	0.4184
RF	0.7000	0.5000	0.5000	0.0000	0.4118	0.3500	0.5000	0.5447	0.0000	0.0000	-0.0140	0.4184

Table 6.28: Average classifier performance before data preprocessing for the ELFF datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	0.8551	0.8541	0.5267	0.5602	0.9148	0.9932	0.8541	0.2803	0.1714	0.3808	-0.5890	0.9210
LR	0.9755	0.9992	0.0203	0.5402	0.9876	0.9763	0.9992	0.0442	0.0583	0.0195	-0.0405	0.9877
Neural Network	0.9980	0.9994	0.6235	0.6663	0.9989	0.9986	0.9994	0.0029	0.6338	0.6229	0.1355	0.9989
KNN	0.9791	0.9965	0.1919	0.5443	0.9892	0.9822	0.9965	0.0346	0.2849	0.1915	0.0002	0.9893
SVM	0.9754	0.9996	0.0058	0.3769	0.9875	0.9758	0.9996	0.0469	0.0255	0.0054	-0.0146	0.9876
RF	0.9972	0.9992	0.6053	0.6665	0.9986	0.9979	0.9992	0.0050	0.6225	0.6045	0.0865	0.9985

Table 6.29: Average classifier performance after data preprocessing for the ELFF datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	0.9571	0.9754	0.9754	0.9939	0.9085	0.8637	0.9754	0.0856	0.8316	0.9508	0.3538	0.9139
LR	0.9857	0.9375	0.9375	0.9672	0.9626	0.9919	0.9375	0.0761	0.9278	0.8750	0.1174	0.9637
Neural Network	0.9429	0.7500	0.7500	0.8627	0.8175	0.9623	0.7500	0.1071	0.6850	0.5000	0.1871	0.8380
KNN	0.9857	0.9375	0.9375	0.9334	0.9626	0.9919	0.9375	0.0366	0.9278	0.8750	0.3970	0.9637
SVM	0.9405	0.9129	0.9129	0.9918	0.8723	0.8416	0.9129	0.0590	0.7511	0.8258	0.2389	0.8747
RF	0.9714	0.8750	0.8750	0.9805	0.9205	0.9842	0.8750	0.0505	0.8522	0.7500	0.3215	0.9250

On the ELFF datasets, both the LR and KNN classifiers achieved the same CA of 98.57%. The RF model achieved an average CA of 97.14%, while the naïve Bayes and neural network models achieved CA values of 95.71% and 94.29%, respectively. Notably,

the LR and KNN models appeared to tie in terms of the recall, specificity, F-score, MCC, J-coefficient and G-mean, with average results of 93.75%, 96.26%, 92.78%, 87.50% and 96.37%, respectively. The naïve Bayes classifier achieved the best performance in terms of the AUC, with an average value of 99.39%. The AUC captures all performance-related information and allows the essential links related to classifier performance to be clearly observed (Drummond & Holte, 2006). The KNN classifier achieved the best results in terms of the Brier score and information score, with values of 3.66% and 39.70%, respectively. Notably, the results of this study are significant compared with the results reported by Xuan et al. (2015).

6.5.1 Researcher's view of imbalanced datasets

Based on the results obtained, the researcher can report that the imbalanced nature of the data itself does not have a significant impact on the average classifier performance G. E. Batista et al. (2004) and Shepperd et al. (2014). Instead, care is required during dataset preprocessing to avoid misclassification of the data. If the data are significantly misclassified, this means that the applied preprocessing approach was poor. Therefore, when addressing imbalanced data, both the minority and majority classes must be correctly identified to avoid any misleading results during classification. Accordingly, in this research, the classes were carefully identified to avoid misclassification, and the average performance results for each target class were then compared to assess the performance of the proposed approach when applied to imbalanced datasets. This performance evaluation was conducted based on the six state-of-the-art classifiers considered in this research.

The findings show that incorrect preprocessing of a dataset can generate misleading results in a defect prediction study. Moreover, there is a possibility that classifiers may lose their performance capabilities when impurities, such as inconsistencies and uncertainties, are present in a dataset.

6.5.2 Average performance loss/gain of learning algorithms

The average performance loss/gain results, characterized by the information entropy among the classification algorithms, are presented in Table 6.30.

Table 6.30: Average performance loss/gain results characterized by information entropy

Classifier	Specificity	F-score	Recall	Precision	G-mean
Naïve Bayes	15.6% gain	52.4% loss	15.6% loss	68.2% loss	37.5% loss
LR	77.5% loss	68.1% loss	77.5% gain	40.2% loss	25.2% loss
Neural Network	72.2% loss	62.1% loss	72.2% gain	38.9% loss	24.1% loss
KNN	63.2% loss	59% loss	63.2% gain	53.6% loss	32.7% loss
SVM	86.8% loss	79.6% loss	86.8% gain	37.7% loss	24.6% loss
RF	80.0% loss	70.4% loss	80.0% gain	40.4% loss	25.6% loss

Data imbalance remains a known problem in machine learning and can affect the performance of learning algorithms (Chawla et al., 2004; G. Batista et al., 2012; Rodriguez et al., 2014). As a result of variations in the data, classification performance can be affected by class imbalance. The major differences in the average performances of the six state-of-the-art classifiers tested here were observed in terms of the specificity, F-score, recall, precision and G-mean, as presented in Table 6.30. Only the naïve Bayes classifier showed a gain in specificity (of 15.6%); all other classifiers showed losses when the target class was varied.

The naïve Bayes classifier suffered a loss in terms of recall, while the other classifiers conversely gained in recall what they lost in specificity. All six state-of-the-art classifiers showed losses in terms of the F-score, precision and G-mean. Based on these inconsistencies in performance between the majority and minority classes, it can be concluded that there is evidence of performance loss, characterized by the information entropy, that could be due to the imbalanced nature of the data. It can also be concluded that the imbalanced nature of the data does not totally hinder the performance of these learning algorithms. However, data imbalance can affect the performances of prediction models (Japkowicz &

Stephen, 2002; Van Hulse et al., 2007; Soda, 2011; Z. Sun et al., 2015; Mao et al., 2017; Qiao et al., 2017). The findings obtained in this study indicate that a research study can yield misleading results because of the use of an erroneous dataset and that imbalanced data can lead to classifier performance loss when applied in any classification study.

The average levels of uncertainty are presented in terms of the information entropy in *bits* in Tables 6.31 and 6.32.

Table 6.31: Average classifier information entropy in bits on the NASA datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	≈ 0.72	≈ 0.78	≈ 0.78	≈ 0.28	≈ 0.78	≈ 0.78	≈ 0.78	≈ 0.77	≈ 0.64	≈ 0.64	≈ 0.99	≈ 0.91
LR	≈ 0.87	≈ 0.96	≈ 0.96	≈ 0.63	≈ 0.96	≈ 0.95	≈ 0.96	≈ 0.97	≈ 0.75	≈ 0.70	≈ 0.49	≈ 0.98
Neural Network	≈ 0.87	≈ 1.00	≈ 1.00	≈ 0.91	≈ 0.97	≈ 0.94	≈ 1.00	≈ 0.95	≈ 0.00	≈ 0.00	≈ 0.44	≈ 0.97
KNN	≈ 0.96	≈ 0.96	≈ 0.96	≈ 0.94	≈ 0.91	≈ 0.95	≈ 0.88	≈ 0.96	≈ 0.99	≈ 0.77	≈ 0.51	≈ 0.94
SVM	≈ 0.87	≈ 1.00	≈ 1.00	≈ 0.00	≈ 0.97	≈ 0.94	≈ 1.00	≈ 0.98	≈ 0.00	≈ 0.00	≈ 0.49	≈ 0.97
RF	≈ 0.87	≈ 1.00	≈ 1.00	≈ 0.00	≈ 0.97	≈ 0.94	≈ 1.00	≈ 0.97	≈ 0.00	≈ 0.00	≈ 0.09	≈ 0.97

Table 6.32: Average classifier information entropy in bits on the ELFF datasets

Classifier	CA	Sens	Spec	AUC	F-score	Prec	Recall	Brier	MCC	J-coef	IS	G-mean
Naïve Bayes	≈ 0.24	≈ 0.16	≈ 0.16	≈ 0.05	≈ 0.14	≈ 0.56	≈ 0.16	≈ 0.41	≈ 0.64	≈ 0.13	≈ 0.90	≈ 0.41
LR	≈ 0.10	≈ 0.33	≈ 0.33	≈ 0.20	≈ 0.23	≈ 0.07	≈ 0.33	≈ 0.38	≈ 0.36	≈ 0.54	≈ 0.51	≈ 0.22
Neural Network	≈ 0.31	≈ 0.80	≈ 0.80	≈ 0.57	≈ 0.68	≈ 0.23	≈ 0.80	≈ 0.47	≈ 0.92	≈ 1.00	≈ 0.66	≈ 0.62
KNN	≈ 0.10	≈ 0.33	≈ 0.33	≈ 0.35	≈ 0.23	≈ 0.07	≈ 0.33	≈ 0.22	≈ 0.37	≈ 0.54	≈ 0.95	≈ 0.22
SVM	≈ 0.32	≈ 0.42	≈ 0.42	≈ 0.68	≈ 0.55	≈ 0.62	≈ 0.42	≈ 0.32	≈ 0.78	≈ 0.65	≈ 0.78	≈ 0.54
RF	≈ 0.18	≈ 0.54	≈ 0.54	≈ 0.13	≈ 0.39	≈ 0.12	≈ 0.54	≈ 0.08	≈ 0.60	≈ 0.80	≈ 0.90	≈ 0.38

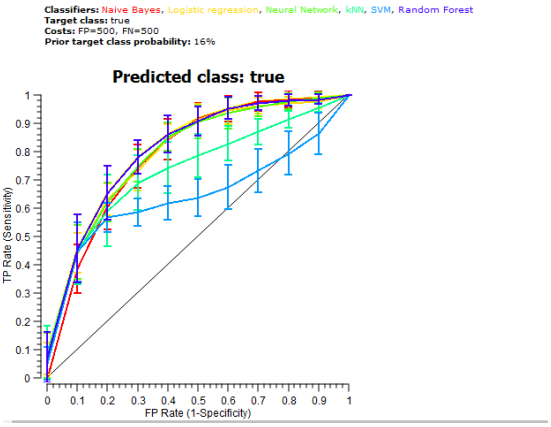
The results show that the NASA datasets contain a higher level of uncertainty due to the erroneous nature of these data, as presented in Table 6.31. From a comparison of the module-level information entropy results for the NASA and ELFF datasets, as presented in Tables 6.31 and 6.32, respectively, it is clear that the ELFF datasets contain fewer impurities. The higher information entropy values for the NASA datasets indicate higher uncertainties, whereas the ELFF data have lower information entropy, indicating lower uncertainties regarding the accuracy of the obtained results. The information entropy reveals further details regarding the uncertainty of an event. If the outcome of an event is already known, then this outcome has a low entropy. By contrast, high entropy or uncertainty could characterize an event whose outcome is yet unknown. By using the proposed optimal decision technique, reliable outcomes can be achieved with regard to

classifier performance on imbalanced data.

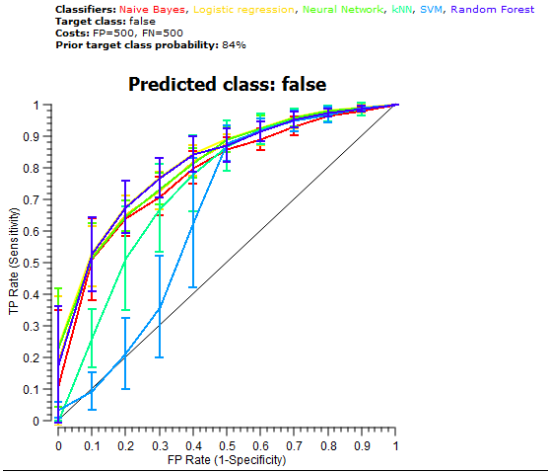
This study has shown that average classifier performance can be characterized in terms of the information entropy values of the learning algorithms in terms of the specificity, F-score, recall, precision and G-mean, as presented in Table 6.30. Because some classifiers maintained their average classification performance, it can be concluded that these classifiers exhibit consistency in their average information entropies. As demonstrated in this thesis, to accurately ascertain the performances of learning algorithms when applied to imbalanced data, consistent and reliable information is necessary, which the proposed optimal decision technique provides.

6.5.3 Analysis of the ROC curves

An analysis of the imbalanced nature of the NASA datasets is presented in Figures 6.5-6.9. The AUC performances of all six state-of-the-art classifiers were also compared with respect to both the majority and minority classes. This was done because the ROC curve is a graphical representation that simultaneously considers two aspects of the problem (plotted on the x- and y-axes) and is suitable for analyzing cases of class imbalance; hence, the NASA datasets are used as representative examples to illustrate the class imbalance. The x-axis represents the false positive rate, that is, the number of instances that are negative but are predicted to be positive, whereas the y-axis represents the true positive rate, i.e., the number of instances that are actually positive and are also predicted to be positive. Another reason for choosing to analyze the ROC curve is that it depicts all performance-related information and allows the essential links related to classifier performance to be clearly observed (Drummond & Holte, 2006). The prior target class probabilities for both the majority and minority classes were obtained from the ROC curves in Figures 6.10 to 6.19.

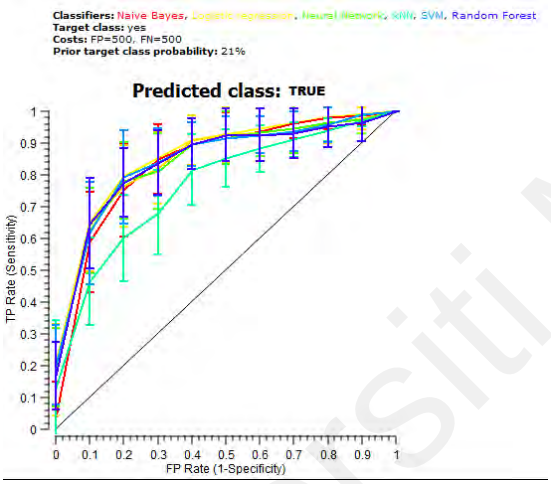


(a) ROC curve for KC1, target = True

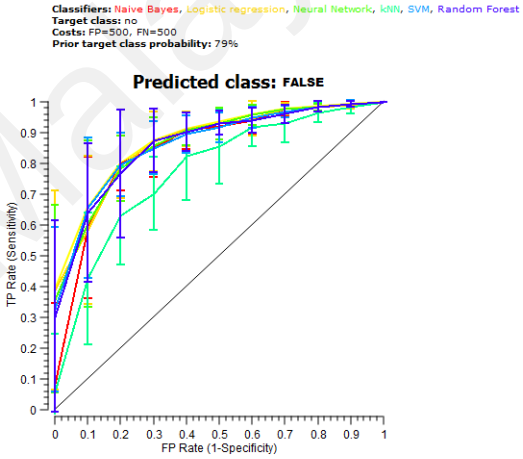


(b) ROC curve for KC1, target = False

Figure 6.10: ROC curves for KC1

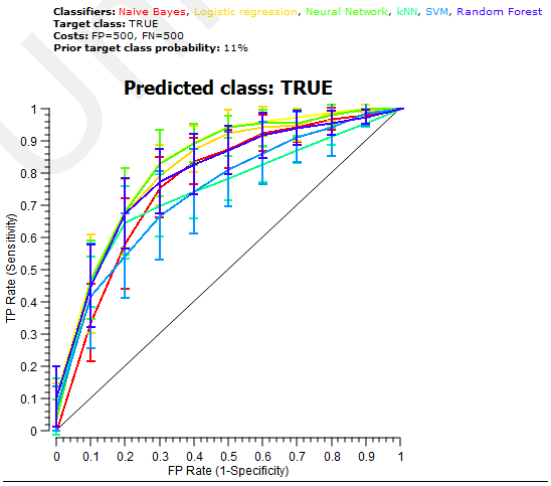


(a) ROC curve for KC2, target = True

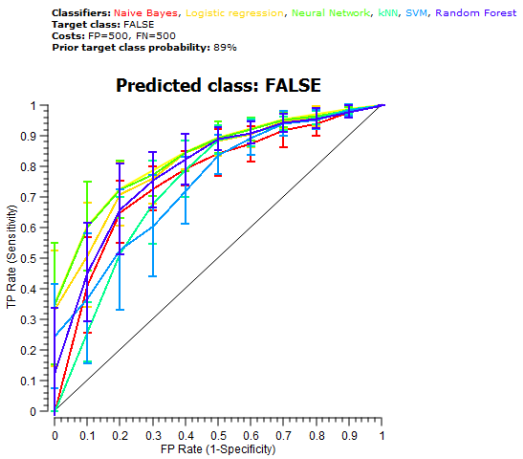


(b) ROC curve for KC2, target = False

Figure 6.11: ROC curves for KC2

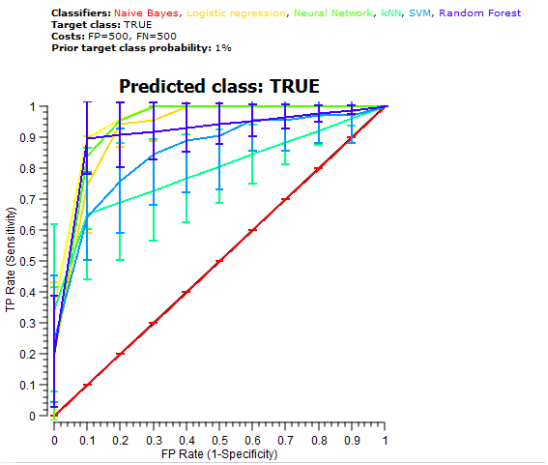


(a) ROC curve for KC3, target = True

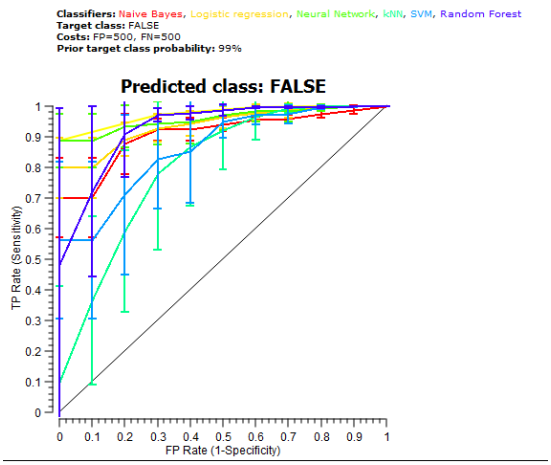


(b) ROC curve for KC3, target = False

Figure 6.12: ROC curves for KC3

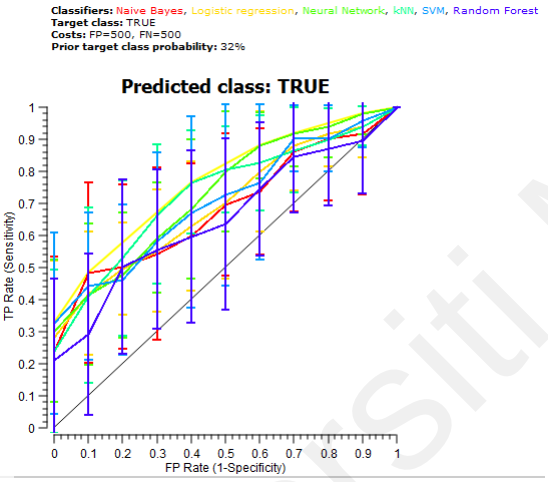


(a) ROC curve for MC1, target = True

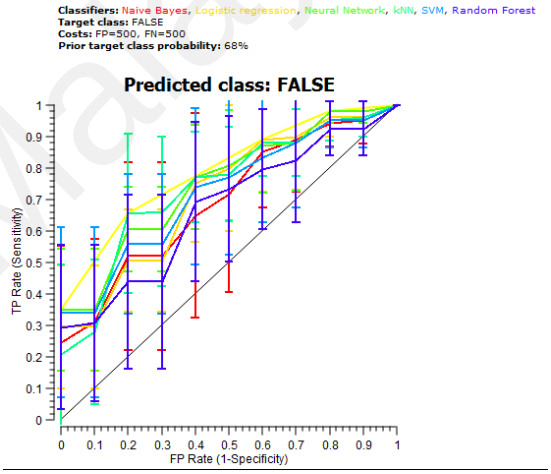


(b) ROC curve for MC1, target = False

Figure 6.13: ROC curves for MC1

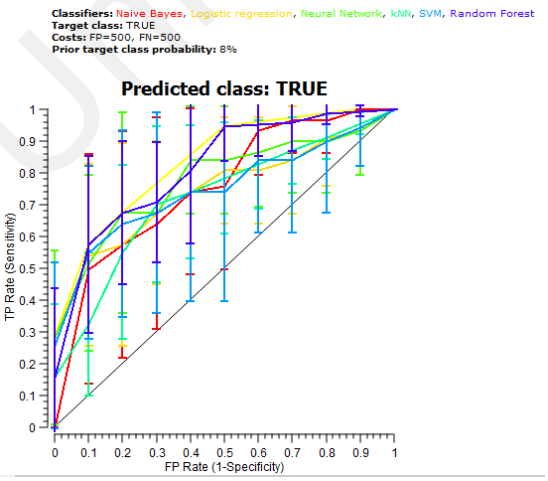


(a) ROC curve for MC2, target = True

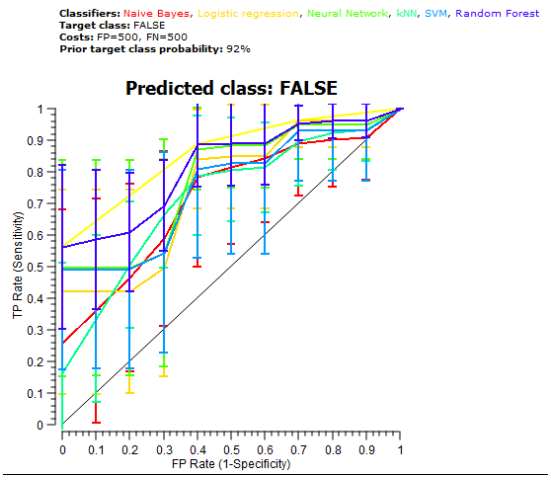


(b) ROC curve for MC2, target = False

Figure 6.14: ROC curves for MC2

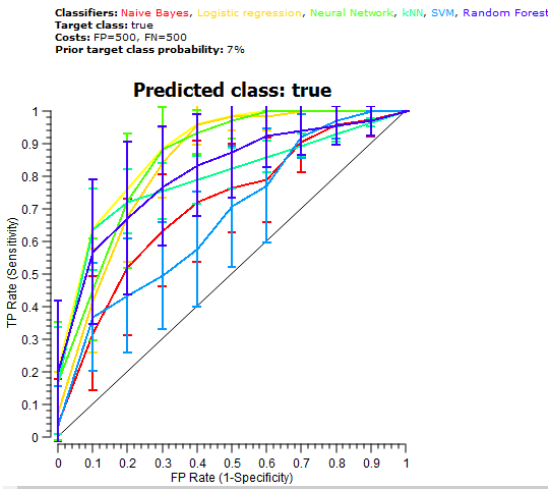


(a) ROC curve for MW1, target = True

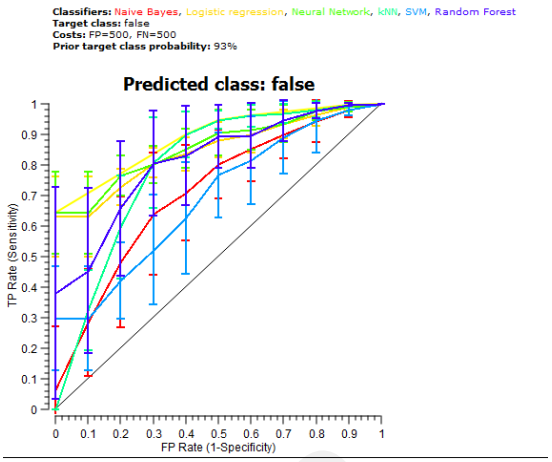


(b) ROC curve for MW1, target = False

Figure 6.15: ROC curves for MW1

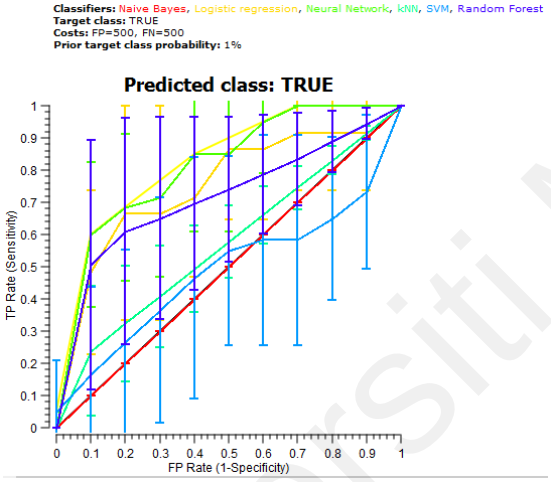


(a) ROC curve for PC1, target = True

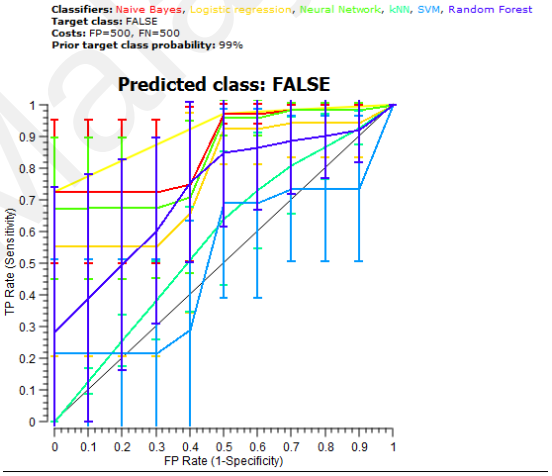


(b) ROC curve for PC1, target = False

Figure 6.16: ROC curves for PC1

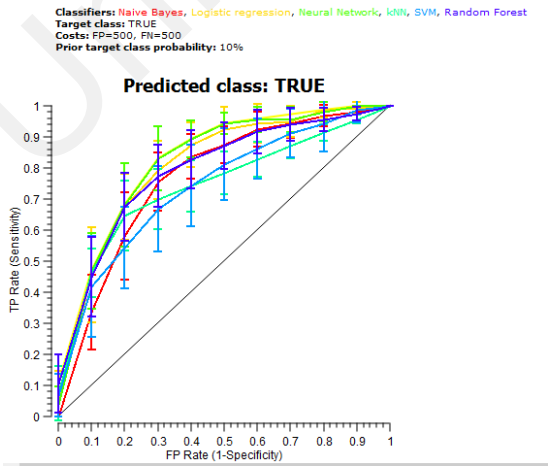


(a) ROC curve for PC2, target = True

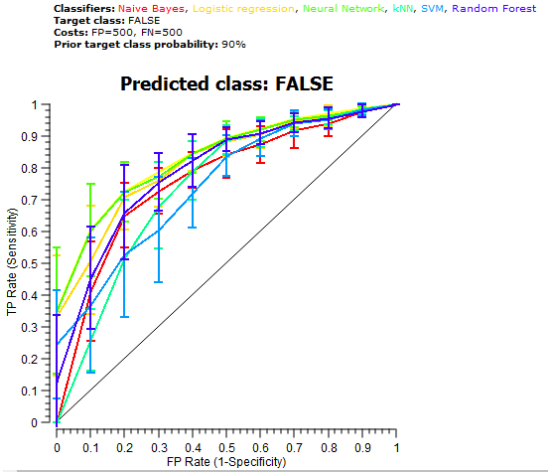


(b) ROC curve for PC2, target = False

Figure 6.17: ROC curves for PC2

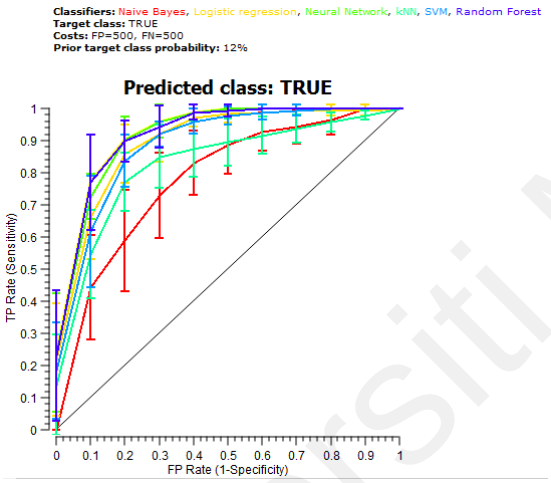


(a) ROC curve for PC3, target = True

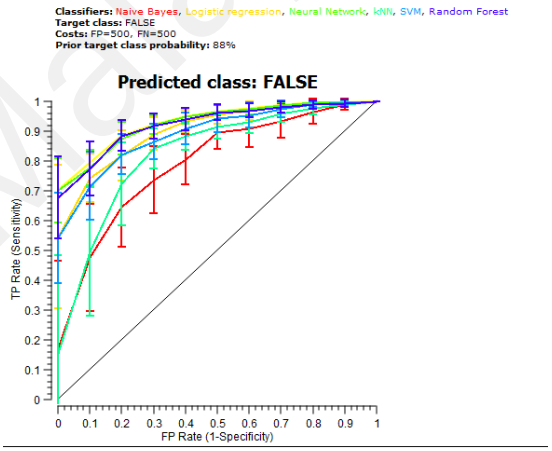


(b) ROC curve for PC3, target = False

Figure 6.18: ROC curves for PC3



(a) ROC curve for PC4, target = True



(b) ROC curve for PC4, target = False

Figure 6.19: ROC curves for PC4

The y-axes of these graphs represent the true positive rates of the corresponding classification algorithms, whereas the x-axes represent the true negative rates. The results are presented in a different form in Table 6.33. It can be concluded that the number of False instances (majority class) is greater than the number of True instances (minority class) based on the fact that the actual percentage of False instances (majority class) is far greater than the actual percentage of True instances (minority class). Such class imbalance scenarios also arise in real-life situations, in which the number of nonfaulty modules in a software system, which allow the software system to remain functional, is always greater than the number of faulty modules, which cause system downtime.

In this study, the researcher hypothesized that even in a scenario in which proper data preprocessing is applied, some classifiers will maintain their average performance when applied to imbalanced data, whereas the average performance of other classifiers may be affected. Based on this hypothesis, the researcher concluded that the overall performance of a classification algorithm should depend on both the target class (defective or defect-free) and the data imbalance. Notably, prediction model performance depends more on the nature of the data on which the model was trained than on the choice of classifier. As previously reported, the choice of classifier family affects the predictive accuracy (as quantified by the MCC) with an impact factor of only 1.3%, compared with the 31% impact factor related to the research group performing the study (Shepperd et al., 2014; Shepperd, 2015). The findings of this research further elaborate on this reported 31% impact of the research group on the quality of the data applied in supervised machine learning defect prediction studies. However, when prediction models are trained with noisy data, the model performance may contradict this finding (Ghotra et al., 2015).

Table 6.33: Actual class imbalance percentages

Dataset	% Target Class = False	% Target Class = True	Total %
KC1	84	16	100
KC2	79	21	100
KC3	89	11	100
MC1	99	1	100
MC2	68	32	100
MW1	92	8	100
PC1	93	7	100
PC2	99	1	100
PC3	90	10	100
PC4	88	12	100

6.6 Results obtained using the regression models

This section presents the results obtained for the predicted numbers of class- and method-level defects in new versions of software products and compare these results with the actual numbers of defects present in the current version at both the class and method levels. Table 6.34 presents the results of defect prediction at the class level for the ELFF datasets.

From left to right, the columns show the project name, the number of classes, the number of defects present at the class level in the current version, the predicted number of defects in the future version and the percentage error of our prediction. The results obtained are promising, with class-level prediction errors of less than 20%, implying that the accuracy of the results at the class level is greater than 80%. Note that in some cases, the predicted values are negative, implying that the number of predicted defects is less than 0. In such a scenario, the predicted number of defects is assumed to actually be 0. For instance, if the predicted number of defects is -166, which is below 0, the corresponding value is reported as 0 in Tables 6.34 and 6.35. In addition, not all prediction results are presented because a certain number of the datasets were used to validate the proposed approach.

Table 6.34: Comparison between the actual and predicted numbers of defects at the class level in the ELFF datasets and the corresponding percentage errors

Project Name	No. of Classes	No. of Defects in Current Version	Predicted No. of Defects in New Version	Percentage Error
Cdk1	1678	0	0	0%
Cdk1.2	1717	0	0	0%
Controltier3	1659	0	0	0%
Controltier3.2	1683	0	0	0%
Drjava2009	3196	774	691	11%
Genoviz5.4	1111	827	728	12%
Genoviz6	1077	840	736	12%
Genoviz6.1	1059	817	710	13%
Jikesrvm3	2098	440	482	10%
Jitterbit1.1	6141	533	548	3%
OmegaT3.1	1204	45	47	4%
Runawfe3.5	5029	0	0	0%
Xaware5.1	994	51	60	18%
Xaware6	1001	0	0	0%

Table 6.35: Comparison between the actual and predicted numbers of defects at the method level in the ELFF datasets and the corresponding percentage errors

Project Name	No. of Methods	No. of Defects in Current Version	Predicted No. of Defects in New Version	Percentage Error
Cdk1	9576	0	0	0%
Cdk1.2	4366	0	0	0%
Controltier3	6078	0	0	0%
Controltier3.2	4946	0	0	0%
Genoviz6.1	4701	504	466	8%
HTMLUnit2008	4715	427	422	1.2%
HTMLUnit2010	7747	259	307	19%
Jitterbit1.1	1155	22	21	5%
Jitterbit1.2	11246	26	31	19%
Jmol6	2170	280	325	16%
Jmol7	2484	248	300	21%
Jppf5.1	3618	19	14	26%
Jump1.7	4064	26	32	23%
Jikesrvm3.1	3890	20	16	20%
Runawfe3.5	3282	0	0	0%
Runawfe3.6	470	0	0	0%
Runawfe4.2	2136	0	0	0%
Unicore1.3	952	21	19	10%
Xaware6	2843	0	0	0%

Table 6.36: Comparison between the actual and predicted numbers of defects at the module level in the NASA datasets and the corresponding percentage errors

Project Name	No. of Modules	No. of Defects in Current Version	Predicted No. of Defects in New Version	Percentage Error
KC1	2109	326	296	9%
KC3	458	43	40	7%
MC2	161	48	49	2%
PC3	1563	159	174	9%
PC4	1458	174	187	8%

Table 6.35 presents the results of defect prediction at the method level for the ELFF datasets. From left to right, the columns similarly show the project name, the number of methods, the number of defects present at the method level in the current version of the software, the predicted number of defects in the future version and the standard error of the prediction in the form of a percentage. The method-level prediction errors are less than 30%, implying that the accuracy of the results at the method level is greater than 70%.

Table 6.36 presents the prediction results obtained for the NASA datasets. The class-level prediction errors are less than 10%, implying that the accuracy of the results at the class level is greater than 90%.

6.7 Regression statistics

To evaluate the significance of the impact of the derived optimal variables on the numbers of class- and method-level defects, the significance of the regression models was evaluated in terms of the p-value, the adjusted R-square and the F-statistic. The p-value indicates how well a model performs; it represents the statistical significance of a model (Carterette, 2015; Ioannidis, 2005). The F-statistic enables a comparison of the average significance of the variables used in constructing the models. The adjusted R-square measures the goodness of fit of a model and also indicates the influence of a significant variable in a model. The adjusted R-square was considered in this study because its value increases only when a significant variable is included in a model, as reported by Felix & Lee (2017b). Tables 6.37, 6.38 and 6.39 present the statistical significance of the defect density, defect introduction time and defect velocity, respectively.

Table 6.37: Defect density significance statistics

Dataset	NASA module level	ELFF class level	ELFF method level
P-value	0.5317	<0.001	<0.001
F-statistic	0.4271	40.5640	37.3613
Adjusted R-square	-0.0679	0.3678	0.3451

Table 6.38: Defect introduction time significance statistics

Dataset	NASA module level	ELFF class level	ELFF method level
P-value	0.3852	0.3504	0.7718
F-statistic	0.4271	0.8842	0.0846
Adjusted R-square	-0.0177	-0.0017	-0.0134

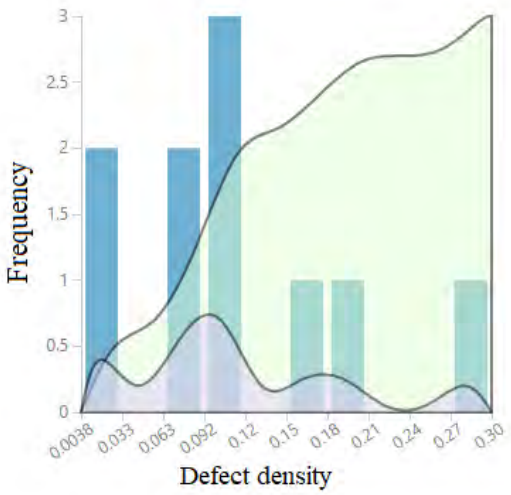
The results indicate that the defect density has inconsistent and weak significance based on the regression statistics obtained for both the NASA and ELFF datasets. The

Table 6.39: Defect velocity significance statistics

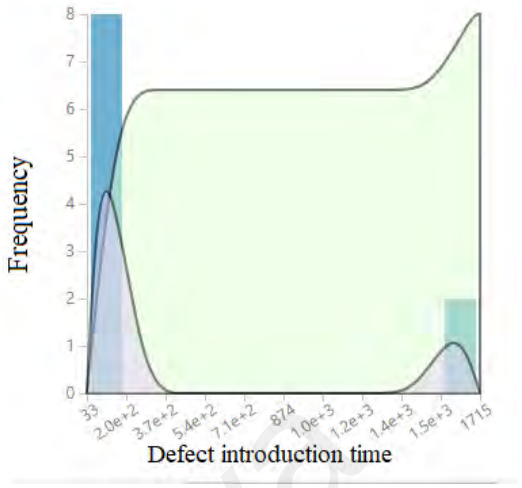
Dataset	NASA module level	ELFF class level	ELFF method level
P-value	<0.001	<0.001	<0.001
F-statistic	228.4909	470.3825	410.1905
Adjusted R-square	0.9619	0.8734	0.8557

inconsistency of this significance can be seen from the p-value: the defect density achieved a significant p-value of <0.001 for both the class- and method-level ELFF datasets but a nonsignificant p-value of 0.5317 for the NASA module-level datasets. Therefore, the inconsistent significance of the defect density makes it unsuitable for constructing regression models for defect prediction. In addition, the defect density achieved a relatively low F-statistic, supporting the hypothesis that the defect density does not influence the number of defects in a software product. In terms of the adjusted R-square, the defect density achieved negative and inconsistent values on the NASA and ELFF datasets, further indicating that the defect density has little impact on the increase in the number of defects in a software program. The defect introduction time achieved the worst significance values on both the NASA and ELFF datasets in terms of the p-value, F-statistic and adjusted R-square. By contrast, the defect velocity showed a strong positive significance, with p-values of <0.001 on all datasets at both the class and method levels and strong F-statistics of 288.4909, 470.3825 and 410.1905 on the NASA module-level datasets, ELFF class-level datasets and ELFF method-level datasets, respectively. The values of the adjusted R-square further indicate that the defect velocity exhibits a high predictive value for defect modeling. Based on the significance of the defect velocity, it was used to construct the regression model for predicting the number of defects in a new software product.

Figures 6.20-6.25 illustrate the cumulative distributions and probability densities representing the influence of the derived optimal variables on the number of defects. In each of these figures, the y-axis represents the frequency, and the x-axis represents the



(a)



(b)

Figure 6.20: Cumulative distributions and probability densities representing the influence of the defect density and defect introduction time on the number of defects at the class level in the NASA datasets

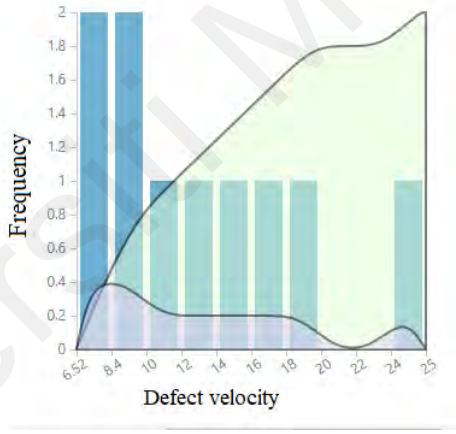
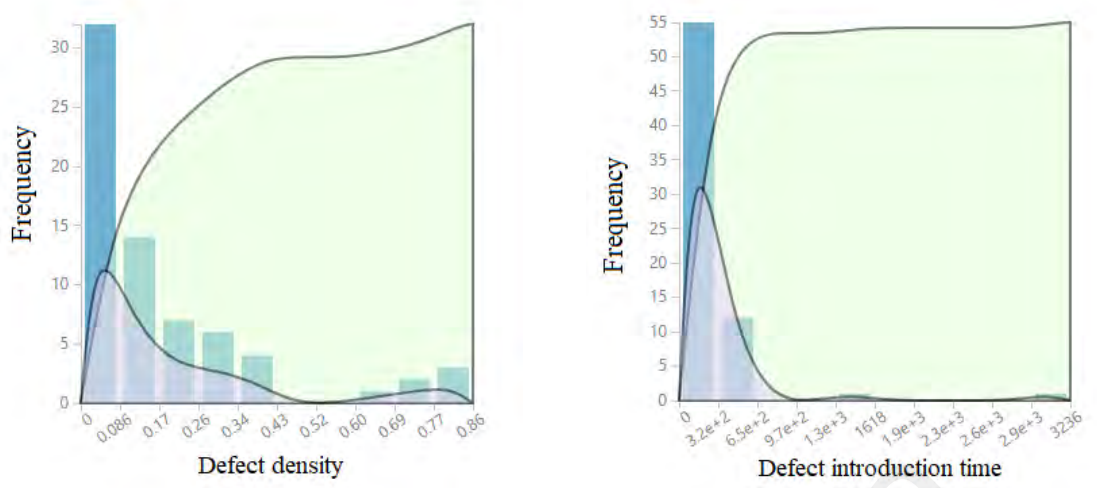


Figure 6.21: Cumulative distribution and probability density representing the influence of the defect velocity on the number of defects at the class level in the NASA datasets

variable distribution. Figure 6.20 (a and b) illustrates the cumulative distributions and probability densities representing the influence of the defect density and defect introduction time on the number of defects at the class level in the NASA datasets. It can be deduced from Figure 6.20(a) that the cumulative distribution and probability density for the defect density are inconsistent starting from the origin, indicating a weak probability that the defect density has an impact on the number of defects throughout the Software Development Life



(a) (b)

Figure 6.22: Cumulative distributions and probability densities representing the influence of the defect density and defect introduction time on the number of defects at the class level in the ELFF datasets

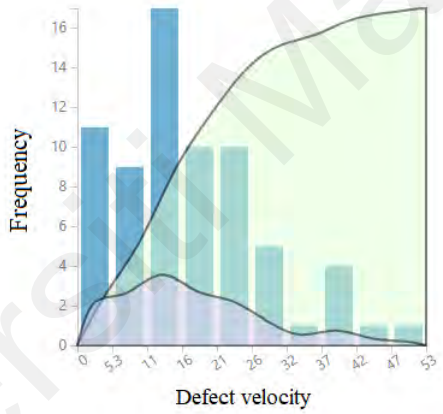
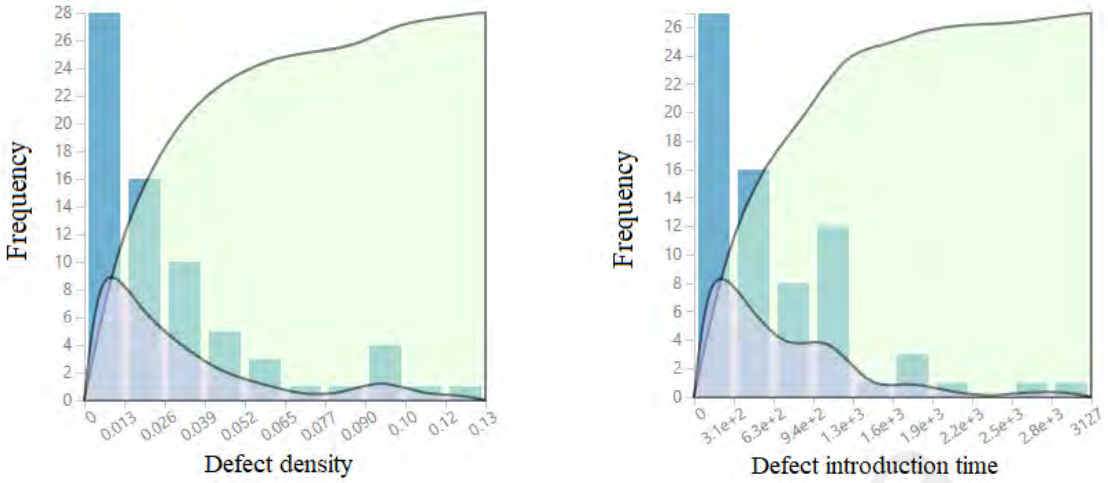


Figure 6.23: Cumulative distribution and probability density representing the influence of the defect velocity on the number of defects at the class level in the ELFF datasets

Cycle (SDLC), even though the defect density of each software program varies depending on the number of defects present in that program. Hence, by comparing the cumulative distributions and probability densities corresponding to the defect density in Figures 6.22 and 6.24 for the ELFF datasets, we can deduce that the defect density varies with a weak and inconsistent cumulative distribution. Figures 6.20(b), 6.22(b) and 6.24(b) illustrate the cumulative distributions and probability densities representing the influence of the defect introduction time on the numbers of defects at the class level in the NASA datasets and at



(a) (b)

Figure 6.24: Cumulative distributions and probability densities representing the influence of the defect density and defect introduction time on the number of defects at the method level in the ELFF datasets

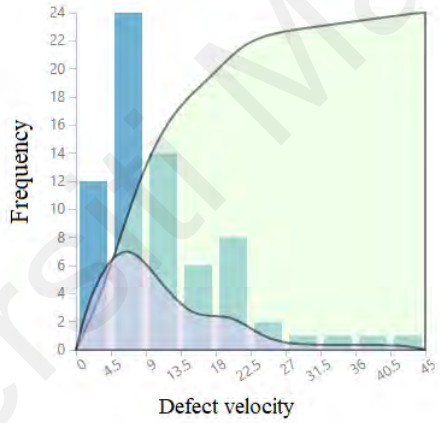


Figure 6.25: Cumulative distribution and probability density representing the influence of the defect velocity on the number of defects at the method level in the ELFF datasets

the class and method levels in the ELFF datasets, respectively. We can similarly deduce from these figures that the defect introduction time across the SDLC does not affect the number of defects in a software program. The probability of defect introduction remains constant throughout the SDLC, as illustrated in the figures. In contrast, Figures 6.21, 6.23 and 6.25 illustrate the consistent cumulative cumulative distributions and probability densities representing the influence of the defect velocity in the NASA and ELFF datasets.

These figures illustrate how the defect velocity influences the number of defects, beginning in the initial phase of the SDLC. The defect velocity exhibits an exponential relationship with the number of defects, as illustrated in these figures, which implies that the probability of an increase in the number of defects becomes higher with an increase in the defect velocity.

6.8 Threats to validity

This section presents the potential threats to the validity of this research. To this end, three types of threats are distinguished, namely, construct, internal and external threats.

6.8.1 Construct threats

Mathematical modeling was performed in this research to demonstrate the relationships between the derived variables and the numbers of class- and method-level defects. In addition, equations were constructed to calculate the average performances and performance losses/gains of the six state-of-the-art classifiers investigated in this study with the goal of determining the overall performance of each classifier on imbalanced data for both the majority and minority classes. These equations may pose some threat to the validity of this study.

6.8.2 Internal threats

Although previous studies, such as that of Xuan et al. (2015), have achieved noteworthy results in evaluating average performance for binary defect classification, this study also investigated the average performances of the six selected state-of-the-art classifiers on binary target classes (majority and minority classes) to obtain detailed and unbiased information regarding the average performance of each classifier when applied to imbalanced data. The use of the average information entropy as the performance indicator may have affected

the validity of this study. Thus, a more accurate approach for evaluating average classifier performance for binary defect classification is needed.

6.8.3 External threats

Experiments were conducted using datasets from the NASA and UCI repositories as well as datasets obtained from the ELFF projects. These data could also pose some threat to the validity of the results of this study. Notably, these datasets are widely used and have been applied in several previous prediction studies, such as those of Elish & Elish (2008), Q. Song et al. (2011), Z.-W. Zhang et al. (2017) and Qiao et al. (2017). The correctness of the results obtained in this research depends on the ability to correctly apply the proposed approach to identify the defective classes and methods in these datasets during data preprocessing. Therefore, other researchers are encouraged to replicate this study to improve the overall performance achieved in defect prediction studies.

6.9 Summary

This chapter presents the results obtained during the experimental analysis and evaluation of the proposed optimal decision technique applied in this research. The proposed approach was evaluated in various phases, including the filter-based feature selection phase, the outlier removal phase, and the further preprocessing of the datasets, which includes the assignment of unique identifiers to inliers. The results show that the datasets applied in supervised machine learning studies require proper preprocessing to avoid misleading results. A comparison of the results obtained on raw and preprocessed datasets has also been presented in this chapter to serve as the basis for assessing the performance of the proposed optimal decision framework for data preprocessing. The analysis presented in this chapter shows that the datasets used in this research are highly imbalanced and contain irrelevant and redundant features. The basis for and the results of selecting only highly

predictive and relevant features from the datasets have also been presented in the current chapter. Moreover, the results for the average information entropy, which characterizes the inconsistent and imbalanced nature of the datasets, have been presented. The reason for presenting these results is to enable an assessment of the uncertainty of the performance of the proposed approach on the basis of several learning algorithms. Furthermore, the results on the significance of the derived optimal variables have been presented, along with the results on the predicted numbers of defects in future versions of software products and their comparison with the numbers of defects in the current versions of the same software products. Finally, this chapter has presented the cumulative distribution curves and probability densities representing the influence of the derived optimal variables on the number of defects throughout the SDLC. The researcher presents in Appendix B the C++ code which can be used to automate the prediction of number of NASA module level defects. The evidence of C++ code which can be used to automate the prediction of number of ELFF class level defects is presented in Appendix C. The evidence of C++ code which can be used to automate the prediction of number of ELFF method level defects is presented in Appendix D whereas in Appendix E the researcher presents the evidence on validation of the modeling technique applied in deriving the predictor variables.

CHAPTER 7: CONCLUSION

This chapter concludes the work done in this research. The following sections present (i) a summary of the main findings regarding the formulated research questions, (ii) a summary of the research objectives, (iii) the limitations of statistical methods in supervised machine learning, (iv) the benefits of the proposed optimal decision approach, and (v) future work.

7.1 Summary of findings regarding the research questions

The research questions formulated for this research are reiterated and discussed below.

RQ1. How do researchers perform and report the techniques applied during data pre-processing?

It is clear that the existing datasets suffer from numerous quality issues and that these issues, if not addressed, will affect the performance of prediction models. A dataset will be free from bias if almost all quality issues associated with it are addressed. As part of this work, the researcher was able to locate studies that have reported means of partially addressing some of the challenges associated with datasets, such as the class imbalance issue, irrelevant and redundant features, noise in the data and collinearity among metrics. A summary of the collected evidence on the existing techniques for addressing data-related issues is presented in Chapter 2. The techniques applied to address these issues have achieved noteworthy results. However, none of the existing studies has demonstrated a holistic approach for thoroughly cleaning and preprocessing both class- and method-level datasets applied in supervised machine learning studies. In contrast, the proposed optimal decision approach applied in this research addresses all of the data quality issues associated with existing class- and method-level datasets. This proposed approach consists of a replicable step-by-step technique for addressing the inconsistencies associated with the

datasets applied in supervised machine learning studies.

RQ1.1 Do the reported data preprocessing techniques satisfactorily address data inconsistency issues?

Although the techniques reported in the literature have achieved noteworthy results, inconsistencies in the datasets still exist. The available evidence shows that the reported data preprocessing techniques only partially address the issues associated with the existing datasets. Consequently, if the remaining issues affecting a dataset are not addressed, the prediction outcomes will be incorrect and misleading. Therefore, there is a need to implement an approach that can address almost all, if not all, of the issues facing existing class- and method-level datasets.

RQ1.2 Does a generally accepted data preprocessing technique exist?

As of yet, a generally accepted data preprocessing technique is lacking in the machine learning community. As stated previously in Section 1.1 of Chapter 1, several studies have proposed frameworks for addressing some of the challenges associated with the existing datasets, including the frameworks of Menzies et al., Lessmann et al. and Song et al. However, the reliability of these frameworks has been challenged by Wahono (2015), who argued that these frameworks produce misleading findings while addressing the issues associated with data inconsistencies. Based on this evidence, the researcher concludes that a generally accepted data preprocessing technique is not yet available. Hence, an optimal decision framework is proposed to address this lack.

RQ1.3 To what extent do the existing data preprocessing techniques offer suitable solutions for data preprocessing?

The existing data preprocessing techniques offer suitable solutions to some of the data quality issues affecting the outcomes of defect prediction. However, the solutions offered by existing preprocessing techniques do not address the majority of the inconsistencies

affecting a given dataset. Therefore, further efforts are still required to overcome the remaining issues that the existing techniques do not address.

RQ2. How do existing prediction models perform?

The existing literature has confirmed that the performance of learning algorithms depends on the quality of the data on which they are trained. The performances of learning algorithms at both the class and method levels will ultimately depend on the quality of the class- and method-level training datasets, respectively. There is evidence that the existing models have achieved noteworthy performance, as presented in Chapter 2. However, criticisms have also been leveled with regard to the quality of the data applied in such studies, as previously reported. In addition, some of the existing studies have relied on datasets prepared by others, and the researchers did not themselves ascertain the reliability of these datasets. In such a situation, the results obtained can be misleading. Therefore, to avoid further criticism of the results obtained by learning algorithms, a reliable data preprocessing technique is required.

RQ2.1 Do classifiers exhibit degradation in their average performance as a result of imbalanced data?

Inconsistencies in datasets can affect the performance of learning algorithms. However, if datasets are properly preprocessed, the imbalanced nature of both class- and method-level data may not have an adverse impact on the performance of learning algorithms. Specifically, the results of this research show that if datasets are properly preprocessed, not all classifiers may show degradation in their average performance. Even in the case of highly imbalanced classes and methods, some classifiers have the ability to maintain their average performance when trained on unbiased data.

RQ2.2 Does data imbalance result in unfair and inaccurate evaluation outcomes?

Some studies have reported that classifiers may be biased as a result of imbalance in the

datasets on which they are trained. For instance, H. He & Ma (2013), Stefanowski (2016), Krawczyk et al. (2014) and Weiss & Provost (2003) have reported that classifiers may be biased towards the minority classes and methods. However, the outcome of a classification algorithm does not inherently depend on the class or method ratio within the datasets. Rather, the outcome depends on the quality of the data applied in such a study. Therefore, from the findings of this research, it can be concluded that data imbalance among the classes and methods in a dataset does not result in unfair and inaccurate evaluation outcomes.

RQ2.3 Do classification algorithms learn independently from imbalanced data? Classification algorithms can learn independently from imbalanced data provided that the data are unbiased. If the datasets applied in training the classification algorithms are biased, then the algorithms will be unable to achieve independent learning. Such an algorithm may suffer from overfitting and concept drift (a situation in which the prediction models are trained and validated with biased datasets) and consequently produce misleading results. Therefore, it is necessary to train classification algorithms with unbiased data to ensure accurate and independent performance of these learning algorithms when faced with highly imbalanced datasets.

RQ3. How do supervised classification algorithms perform on average when applied to raw and preprocessed imbalanced data?

There is a clear distinction in performance between models trained on raw and preprocessed datasets. One of the objectives of this research was to assess the average performance of classification models built using the proposed optimal decision framework. The results obtained show that classification algorithms trained on raw datasets tend to produce misleading results due to the inconsistencies in the data. However, as clearly seen by comparing the performance achieved by learning algorithms trained on raw datasets with the performance of algorithms trained on preprocessed datasets, the removal of impurities

from datasets allows a complete assessment of the performance of the learning algorithms by ensuring more reliable model performance results.

RQ3.1 Do certain supervised classification algorithms outperform others on average when applied to imbalanced data?

When applied in defect prediction studies, classification algorithms maintain certain parameter settings. These parameter settings may cause some classifiers to outperform others in terms of their average performance. Generally, some classification algorithms can maintain their average performance when trained on properly preprocessed datasets, even if these datasets are highly imbalanced in terms of classes and methods.

RQ3.2 How does class imbalance result in biased outcomes from supervised learning algorithms?

Learning algorithms can yield biased outcomes if they are trained using raw or unprocessed datasets. The impurities within the datasets may not allow the learning algorithms to produce accurate and reliable results.

RQ3.3 To what extent do supervised classification algorithms maintain their average information entropy when applied to imbalanced data?

The training of a classifier depends on the nature of the training dataset, that is, the numbers of classes and methods in the dataset. The higher the numbers of classes and methods are, the more training the classifier will receive. For instance, between the National Aeronautics and Space Administration (NASA) and ELFF datasets, the ELFF projects contain higher numbers of classes and methods. Hence, learning algorithms will tend to learn more and produce more reliable results when trained on the ELFF datasets. Consequently, the learning algorithms considered in this research could achieve classification accuracies of up to 80% to 99% in the cases of the NASA and ELFF datasets, respectively.

RQ4. How do the derived optimal predictor variables aid in data preprocessing and

influence the number of defects in a software project?

To enable the accurate determination of the values of the defect density, defect velocity and defect introduction time at both the class and method levels, these derived optimal variables are calculated during the further preprocessing stage of the proposed framework. These optimal variables reveal the rates at which software defects occur and the exact proportions of defects with respect to the project size at both the class and method levels in a software project.

Furthermore, investigation of the derived optimal variables reveals strong positive correlations between the defect velocity and the numbers of defects at both the class and method levels. Based on these strong correlations, the defect velocity was applied in this research to construct regression models to predict the numbers of defects at both the class and method levels in a software program.

RQ4.1 How does the defect velocity impact the number of software defects?

As stated previously, the results of this research confirm that the defect velocity has a strong correlation with the number of defects. The increase in the defect velocity throughout the Software Development Life Cycle (SDLC) results in an increase in the number of defects as a project transitions from one phase of the SDLC to the next. Based on this correlation, the defect velocity is found to be a significant variable that can influence the number of defects in a software program. In addition, the associated cumulative distributions and probability densities show that the defect velocity exhibits a high probability of influencing the numbers of defects in software at both the class and method levels.

RQ4.2 Does the defect density influence the number of software defects?

The defect density of a software program does not influence the numbers of class- and method-level defects. However, the results of this research confirm that the defect density varies from one software product to another depending on the number of defects present in

each software program.

RQ4.3 Does the defect introduction time impact the number of software defects?

The defect introduction time was found to have a weak and negative correlation with the numbers of class- and method-level defects in a software program. Hence, it does not influence the number of software defects.

7.2 Summary of the research objectives

As presented in Section 1.5 of Chapter 1, there is a need for a cost-effective and efficient framework for preprocessing both class- and method-level datasets. If properly implemented, such a framework is expected to improve the quality of the datasets applied in defect prediction studies. In addition, there is also a need for a demonstrated means of predicting the numbers of class- and method-level defects in a new version of software. These objectives were met in this research.

First, the issues associated with the existing imbalanced class- and method-level datasets were investigated. The findings of this investigation are presented in Chapter 2, Section 2.1. Through these findings, research objective 1 was achieved.

Second, the performance and accuracy of the existing learning algorithms applied in supervised machine learning studies were investigated. A summary of the corresponding findings is presented in Chapter 2, Section 2.2. Through these findings, objective 2 of this research was achieved. By means of the data preprocessing framework presented here, the researcher was able to determine the average performance and accuracy of the learning algorithms applied in this research.

Furthermore, the proposed data preprocessing framework revealed certain optimal variables, namely, the defect density, defect velocity and defect introduction time, which can be applied when constructing regression models to predict the numbers of defects in a new software version at both the class and method levels using information available from

the current version of the software, thus achieving objective 3 of this research, as presented in Chapter 4. Finally, to achieve research objective 4, the proposed data preprocessing framework and the approach for predicting the numbers of class- and method-level defects in a new software version were evaluated. The details of the evaluation metrics applied in this research are presented in Chapter 5.

7.3 Limitations of statistical methods in supervised machine learning

Most existing software defect prediction models show weak performance as a result of their inability to capture the as-yet-unknown correlation between defects and failures (Fenton & Neil, 1999). In the researcher's opinion, this is because the predictor variables that actually influence the number of defects in a software product have not been fully exploited in the construction of a suitable prediction model. In this study, such predictor variables were incorporated into the proposed models; according to the findings, the defect velocity, which characterizes the defect introduction rate, exerts the greatest influence on the numbers of class- and method-level defects in a software product. Therefore, prediction models built using the defect velocity tend to produce reliable prediction outcomes regarding the number of defects in a future version of software product. However, statistical approaches such as those applied in this study are subject to certain limitations, including limitations related to multicollinearity, model fitting and the quality of the data points.

7.3.1 Multicollinearity

One of the most common problems encountered with the application of statistical methods in the existing literature, multicollinearity occurs when two or more predictor variables are highly positively or negatively correlated with each other (Fenton & Neil, 1999). When multicollinearity arises in a statistical analysis, it may lead to inconsistent signs

of the correlation coefficients and misleading conclusions. In this study, the correlation coefficient between the number of defects and the defect introduction time had a negative value. This negative correlation was consistent when tested on both the NASA and ELFF datasets. Notably, linear regression models rely on the assumption that the correlations between the predictor variables are always zero, meaning that they are independent of each other (Manly & Alberto, 2016). In this study, it was ensured that the applied prediction models did not suffer from multicollinearity issues since this issue is addressed in the early phase of the proposed data preprocessing framework.

7.3.2 Model fitting

Almost all evaluations of prediction models are primarily concerned with model fitting, with little focus on the quality of the data used in training the models. Instead, attempts are typically made to demonstrate how well these models explain the historical data through least-squares fitting and the goodness of fit. By contrast, this study focused on accurately preprocessing the datasets, predicting the number of defects, and determining the effect of each predictor variable on the number of defects. A reliable model is one that is capable of predicting the number of defects in a future version of a software module (Fenton & Neil, 1999). Because of a lack of reliable data, the authors of some existing studies, such as Compton & Withrow (1990) and Hatton (1970), have used only their own data for model fitting, without performing a proper evaluation of these models on newer datasets, which may lead to misleading research findings. Thus, to avoid misleading results, one of the objectives of this research was to develop an optimal decision framework to ensure that the datasets applied in model training are properly preprocessed. Accurately preprocessed datasets will produce reliable results and prevent model overfitting and concept drift.

7.3.3 Quality of the data points

It is somewhat challenging to determine which studies in the existing literature have included procedures for controlling the quality of the data points during preprocessing, although if such procedures were applied, some justification for doing so should be provided (Fenton & Neil, 1999). In this study, the quality of the data points was addressed when applying the proposed optimal decision framework. Data points were removed during the data preprocessing phase based on the identification of incomplete attributes because such incomplete data may impact model performance. These incomplete data points were removed to enable us to gain full knowledge of the data while ensuring that further data preprocessing and analysis would be performed using datasets with complete data points.

7.3.4 Limitations of the proposed solution

To achieve the objectives of this study, an optimal decision framework is proposed to address some of the challenges associated with datasets applied in supervised machine learning studies. However, there are some limitations of the proposed framework which includes: (a) is time consuming, (b) is semi-automated and (c) margin of errors. **Time consuming**: The proposed framework takes more time to be implemented during data preprocessing and as such may not serve the intended purpose for use in a software industry. **Semi-automated**: There are a lot of manually controlled aspects of the proposed framework, for instance, the assignment of unique identifiers to classes and methods of each project. This manual aspect of the proposed framework needs to be fully automated to reduce data preprocessing time.

Margin of errors: The proposed framework contains some mathematical components which may pose some threat to the validity of the results obtained. As such, the results obtained through these mathematical components may subject the proposed framework to certain degree of errors. Therefore, these limitations call for future work to improve the

quality of the proposed solution.

7.4 Benefits of the proposed optimal decision approach

In terms of cost and quality, the proposed approach represents an easy-to-use and cost-effective data preprocessing technique that can assist software companies in gaining a comprehensive and accurate understanding of their existing datasets and in performing appropriate preprocessing thereof. At the same time, the proposed approach requires fewer lines of code and less time and energy than existing methods. Generally, good prediction models are capable of learning independently and accurately from reliable data such that they are able to produce reliable prediction results. Furthermore, the proposed approach can assist managers in decision-making regarding the allocation of available resources for software projects. The experimental results obtained in this study suggest that the defect velocity is the primary factor affecting the number of defects in a software project; consequently, software companies of all sizes must pay adequate attention to the rate at which software projects transition from one phase to another. This does not imply that a slow development approach should be adopted for software projects; rather, more attention should be paid in every stage of the development process to reduce errors. However, the initial phase of a software project may require more than the available number of experts to accurately determine whether the project contains nearly zero defects and thus is ready to transition to the next development phase. If an enormous software project is handled by fewer than the required number of experts, then that project may be more prone to errors, whereas a project that is attended to by the required number of experts is capable of approaching nearly zero defects and can be delivered to the end user on time.

The challenges posed by software defects remain an important issue to be addressed in software engineering. Software companies continue to seek possible ways of reducing or eliminating errors in software products. The regression models applied in this study

can assist managers in forecasting the future status of a software product by predicting the number of defects that are likely to be present in a new product release. In addition, these prediction models can support decision-making regarding the proper allocation of available resources for a software project, which will lead to a more profitable future for software companies.

7.5 Future work

The results of this work must be confirmed to verify the suitability of the proposed optimal decision approach for both class- and method-level data preprocessing and defect prediction. In future work, the researcher hopes to apply the most recent datasets from one or more software companies to validate this method of predicting the number of defects in an upcoming software release while also considering additional predictor variables. In addition, the researcher hopes to fully automate the proposed data preprocessing framework to reduce time required to clean the datasets as well to improve the effectiveness of the proposed solution.

REFERENCES

Abolkarlou, N. A., Niknafs, A. A., & Ebrahimpour, M. K. (2014). Ensemble imbalance classification: Using data preprocessing, clustering algorithm and genetic algorithm. In *4th international econference on computer and knowledge engineering (iccke)* (pp. 171–176).

Agrawal, A., & Menzies, T. (2018). Is better data better than better data miners?: on the benefits of tuning smote for defect prediction. In *Proceedings of the 40th international conference on software engineering* (pp. 1050–1061).

Akbar, M. M., & Parvez, N. (2009). Impact of service quality, trust, and customer satisfaction on customers loyalty. *ABAC journal*, *29*(1).

Al-Ghraibah, A., Boucheron, L. E., & McAteer, R. J. (2015). A study of feature selection of magnetogram complexity features in an imbalanced solar flare prediction data-set. In *IEEE international conference on data mining workshop (icdmw)* (pp. 557–564).

Alibeigi, M., Hashemi, S., & Hamzeh, A. (2012). Dbfs: An effective density based feature selection scheme for small sample size and high dimensional imbalanced data sets. *Data & Knowledge Engineering*, *81*, 67–103.

Anand, A., Pugalenthi, G., Fogel, G. B., & Suganthan, P. (2010). An approach for classification of highly imbalanced data using weighting and undersampling. *Amino acids*, *39*(5), 1385–1391.

Armah, G. K., Luo, G., & Qin, K. (2013). Multi_level data pre_processing for software defect prediction. In *Information management, innovation management and industrial engineering (iciii), 2013 6th international conference on* (Vol. 2, pp. 170–174).

Aydin, A., & Tarhan, A. (2014). Investigating defect prediction models for iterative software development when phase data is not recorded lessons learned. In *International conference on evaluation of novel approaches to software engineering (enase)* (pp. 1–11).

Bae, S.-H., & Yoon, K.-J. (2015). Polyp detection via imbalanced learning and discriminative feature learning. *IEEE transactions on medical imaging*, *34*(11), 2379–2393.

Balamurugan, S. A. A., & Christopher, A. A. (2012). Data tuner for effective data pre-processing. In *Advances in engineering, science and management (icaesm), 2012 international conference on* (pp. 804–810).

Baskaran, N., Kwoh, C. K., & Kam, M. H. (2010). Outcomes of gene association analysis of cancer microarray data are impacted by pre-processing algorithms. In *IEEE international conference on bioinformatics and biomedicine (bibm)* (pp. 228–233).

Batista, G., Silva, D., & Prati, R. (2012). An experimental design to evaluate class imbalance treatment methods. In *2012 11th international conference on machine learning and applications (icmla)* (Vol. 2, pp. 95–101).

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, *6*(1), 20–29.

Beckmann, M., de Lima, B. S. L., & Ebecken, N. F. (2011). Genetic algorithms as a pre processing strategy for imbalanced datasets. In *Proceedings of the 13th annual conference companion on genetic and evolutionary computation* (pp. 131–132).

Benhar, H., Idri, A., & Fernández-Alemán, J. (2018). Data preprocessing for decision making in medical informatics: Potential and analysis. In *World conference on information systems and technologies* (pp. 1208–1218).

Bernstein, A., Ekanayake, J., & Pinzger, M. (2007). Improving defect prediction using temporal features and non linear models. In *Ninth international workshop on principles of software evolution: in conjunction with the 6th esec/fse joint meeting* (pp. 11–18).

Beyan, C., & Fisher, R. (2015). Classifying imbalanced data sets using similarity based hierarchical decomposition. *Pattern Recognition*, *48*(5), 1653–1672.

Bilgin, T. T., & Camurcu, A. Y. (2010). An efficient preprocessing stage for the relationship-based clustering framework. *Intelligent Data Analysis*, *14*(6), 731–748.

Błaszczyński, J., Deckert, M., Stefanowski, J., & Wilk, S. (2010). Integrating selective pre-processing of imbalanced data with ivotes ensemble. In *International conference*

on rough sets and current trends in computing (pp. 148–157).

Błaszczyński, J., & Stefanowski, J. (2015). Neighbourhood sampling in bagging for imbalanced data. *Neurocomputing*, *150*, 529–542.

Bolon-Canedo, V., Sanchez-Marono, N., & Alonso-Betanzos, A. (2011). Feature selection and classification in multiple class datasets: An application to kdd cup 99 dataset. *Expert Systems with Applications*, *38*(5), 5947–5957.

Borghys, D., & Perneel, C. (2010). Study of the influence of pre-processing on local statistics-based anomaly detector results. In *Hyperspectral image and signal processing: Evolution in remote sensing (whispers), 2010 2nd workshop on* (pp. 1–4).

Borrajo, L., Romero, R., Iglesias, E. L., & Marey, C. R. (2011). Improving imbalanced scientific text classification using sampling strategies and dictionaries. *Journal of integrative bioinformatics*, *8*(3), 90–104.

Bosu, M. F., & MacDonell, S. G. (2013). Data quality in empirical software engineering: a targeted review. In *Proceedings of the 17th international conference on evaluation and assessment in software engineering* (pp. 171–176).

Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (swebok (r)): Version 3.0*. IEEE Computer Society Press.

Branco, P., Torgo, L., & Ribeiro, R. (2015). A survey of predictive modelling under imbalanced distributions. *arXiv preprint arXiv:1505.01658*.

Braytee, A., Liu, W., & Kennedy, P. (2016). A cost-sensitive learning strategy for feature extraction from imbalanced data. In *International conference on neural information processing* (pp. 78–86).

Brecher, V. H., Hall, R. W., Parisi, D. M., Rao, R., Riley, S. L., Sturzenbecker, M. C., et al. (1996, August 6). *Automated defect classification system.* Google Patents. (US Patent 5,544,256)

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Briand, L. C., Melo, W. L., & Wust, J. (2002). Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE transactions on Software Engineering*, *28*(7), 706–720.

Caglayan, B., Misirli, A. T., Bener, A. B., & Miranskyy, A. (2015). Predicting defective modules in different test phases. *Software Quality Journal*, *23*(2), 205–227.

Caglayan, B., Tosun, A., Miranskyy, A., Bener, A., & Ruffolo, N. (2010). Usage of multiple prediction models based on defect categories. In *Proceedings of the 6th international conference on predictive models in software engineering* (p. 8).

Camargo Cruz, A. E., & Ochimizu, K. (2009). Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement* (pp. 460–463).

Canfora, G., Lucia, A. D., Penta, M. D., Oliveto, R., Panichella, A., & Panichella, S. (2015). Defect prediction as a multiobjective optimization problem. *Software Testing, Verification and Reliability*, *25*(4), 426–459.

Cao, H., Li, X.-L., Woon, D. Y.-K., & Ng, S.-K. (2013). Integrated oversampling for imbalanced time series classification. *IEEE Transactions on Knowledge and Data Engineering*, *25*(12), 2809–2822.

Cao, H., Tan, V. Y., & Pang, J. Z. (2014). A parsimonious mixture of gaussian trees model for oversampling in imbalanced and multimodal time-series classification. *IEEE transactions on neural networks and learning systems*, *25*(12), 2226–2239.

Cao, N., Wang, C., Li, M., Ren, K., & Lou, W. (2014). Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on parallel and distributed systems*, *25*(1), 222–233.

Cao, P., Yang, J., Li, W., Zhao, D., & Zaiane, O. (2014). Ensemble-based hybrid probabilistic sampling for imbalanced data learning in lung nodule cad. *Computerized Medical Imaging and Graphics*, *38*(3), 137–150.

Carterette, B. (2015). Statistical significance testing in information retrieval: Theory and practice. In *Proceedings of the 2015 international conference on the theory of information retrieval* (pp. 7–9).

Casañola-Martin, G., Garrigues, T., Bermejo, M., González-Álvarez, I., Nguyen-Hai, N., Cabrera-Pérez, M. Á., ... others (2016). Exploring different strategies for imbalanced adme data problem: case study on caco-2 permeability modeling. *Molecular diversity*, *20*(1), 93–109.

Castaño, E., & Gallón, S. (2017). A solution for multicollinearity in stochastic frontier production function models. *Lecturas de Economía*(86), 9–24.

Castro, C. L., & Braga, A. P. (2013). Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data. *IEEE transactions on neural networks and learning systems*, *24*(6), 888–899.

ÇATAL, Ç. (2016). The use of cross-company fault data for the software fault prediction problem. *Turkish Journal of Electrical Engineering & Computer Sciences*, *24*(5), 3714–3723.

Cateni, S., Colla, V., & Vannucci, M. (2014). A method for resampling imbalanced datasets in binary classification tasks for real-world problems. *Neurocomputing*, *135*, 32–41.

Cavezza, D. G., Pietrantuono, R., & Russo, S. (2015). Performance of defect prediction in

rapidly evolving software. In *IEEE/ACM 3rd international workshop on release engineering (releng)* (pp. 8–11).

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., & Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, *11*(Apr), 1471–1490.

Changyong, F., Hongyue, W., Naiji, L., Tian, C., Hua, H., Ying, L., et al. (2014). Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, *26*(2), 105.

Charette, R. N. (2005). Why software fails [software failure]. *IEEE Spectrum*, *42*(9), 42–49.

Chawla, N. V. (2009). Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook* (pp. 875–886). Springer.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Editorial: special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, *6*(1), 1–6.

Che, D., Safran, M., & Peng, Z. (2013). From big data to big data mining: challenges,

issues, and opportunities. In *International conference on database systems for advanced applications* (pp. 1–15).

Chen, L., Fang, B., Shang, Z., & Tang, Y. (2015). Negative samples reduction in cross-company software defects prediction. *Information and Software Technology*, *62*, 67–77.

Chen, X.-w., & Wasikowski, M. (2008). Fast: a roc-based feature selection metric for small samples and imbalanced data classification problems. In *Proceedings of the 14th acm sigkdd international conference on knowledge discovery and data mining* (pp. 124–132).

Chetchotsak, D., Pattanapairoj, S., & Arnonkijpanich, B. (2015). Integrating new data balancing technique with committee networks for imbalanced data: Grsom approach. *Cognitive neurodynamics*, *9*(6), 627–638.

Ching, J. Y., Wong, A. K. C., & Chan, K. C. C. (1995). Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(7), 641–651.

Compton, B. T., & Withrow, C. (1990). Prediction and control of ada software defects. *Journal of Systems and Software*, *12*(3), 199–207.

Coone, A. (n.d.). A study on different preprocessing and machine.

Crone, S. F., Lessmann, S., & Stahlbock, R. (2006). The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, *173*(3), 781–800.

Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*, *264*, 260–278.

Damaceno Borges, K. C. A., de Barcelos Tronto, I. F., de Aquino Lopes, R., da Silva, S., & Demisio, J. (2013). A data pre-processing method for software effort estimation using case-based reasoning. *Clei electronic journal*, *16*(3), 7–7.

D'Ambros, M., Bacchelli, A., & Lanza, M. (2010). On the impact of design flaws on software defects. In *10th international conference on quality software (qsic)* (pp. 23–31).

Das, B., Krishnan, N. C., & Cook, D. J. (2015). Racog and wracog: Two probabilistic oversampling techniques. *IEEE transactions on knowledge and data engineering*, *27*(1), 222–234.

Demšar, J., Curk, T., Erjavec, A., Črt Gorup, Hočevar, T., Milutinovič, M., ... Zupan, B. (2013). Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, *14*, 2349-2353. Retrieved from `http://jmlr.org/papers/v14/demsar13a.html`

Dheeru, D., & Karra Taniskidou, E. (2017). *UCI machine learning repository.* Retrieved

from http://archive.ics.uci.edu/ml

Diaz, M., & King, J. (2002). How cmm impacts quality, productivity, rework, and the bottom line. *CrossTalk*, *15*(3), 9–14.

Díez-Pastor, J. F., Rodríguez, J. J., García-Osorio, C., & Kuncheva, L. I. (2015). Random balance: ensembles of variable priors classifiers for imbalanced data. *Knowledge-Based Systems*, *85*, 96–111.

Dolado, J. J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, *26*(10), 1006–1021.

Doppa, J. R., Fern, A., & Tadepalli, P. (2014). Hc-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research*, *50*, 369–407.

Dórea, F. C., McEwen, B. J., McNab, W. B., Revie, C. W., & Sanchez, J. (2013). Syndromic surveillance using veterinary laboratory data: data pre-processing and algorithm performance evaluation. *Journal of the Royal Society Interface*, *10*(83), 20130114.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995* (pp. 194–202). Elsevier.

Drummond, C., & Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance. *Machine learning*, *65*(1), 95–130.

Dubey, R., Zhou, J., Wang, Y., Thompson, P. M., Ye, J., Initiative, A. D. N., et al. (2014). Analysis of sampling techniques for imbalanced data: An n= 648 adni study. *NeuroImage*, *87*, 220–241.

D'Addabbo, A., & Maglietta, R. (2015). Parallel selective sampling method for imbalanced and large data classification. *Pattern Recognition Letters*, *62*, 61–67.

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, *81*(5), 649–660.

Estabrooks, A., Jo, T., & Japkowicz, N. (2004). A multiple resampling method for learning from imbalanced data sets. *Computational intelligence*, *20*(1), 18–36.

Fallahi, A., & Jafari, S. (2011). An expert system for detection of breast cancer using data preprocessing and bayesian network. *International Journal of Advanced Science and Technology*, *34*, 65–70.

Fan, W., & Bifet, A. (2013). Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, *14*(2), 1–5.

Farquhar, J., & Hill, N. J. (2013). Interactions between pre-processing and classification methods for event-related-potential classification. *Neuroinformatics*, *11*(2), 175–

192.

Fayyad, U., & Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning.

Felix, E. A., & Lee, S. P. (2017a). Impact of defect velocity at class level. In *2017 international conference on robotics and automation sciences (icras)* (pp. 182–188).

Felix, E. A., & Lee, S. P. (2017b). Integrated approach to software defect prediction. *IEEE Access*, *5*, 21524–21547.

Feng, Q., Hannig, J., & Marron, J. (2016). A note on automatic data transformation. *Stat*, *5*(1), 82–87.

Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on software engineering*, *25*(5), 675–689.

Florido, J. P., Pomares, H., Rojas, I., Urquiza, J., Herrera, L. J., & Claros, M. G. (2010). Effect of pre-processing methods on microarray-based svm classifiers in affymetrix genechips. In *Neural networks (ijcnn), the 2010 international joint conference on* (pp. 1–6).

Foss, T., Stensrud, E., Kitchenham, B., & Myrtveit, I. (2003). A simulation study of the model evaluation criterion mmre. *IEEE Transactions on Software Engineering*,

29(11), 985–995.

Fournet, C., Kohlweiss, M., Danezis, G., Luo, Z., et al. (2013). Zql: A compiler for privacy-preserving data processing. In *Usenix security symposium* (pp. 163–178).

Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., & Ubayashi, N. (2014). An empirical study of just-in-time defect prediction using cross-project models. In *Proceedings of the 11th working conference on mining software repositories* (pp. 172–181).

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). 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)*, *42*(4), 463–484.

Galar, M., Fernández, A., Barrenechea, E., & Herrera, F. (2013). Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*, *46*(12), 3460–3471.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., & Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, *1*(1), 9.

García, V., Sánchez, J. S., Martín-Félez, R., & Mollineda, R. A. (2012). Surrounding neighborhood-based smote for learning from imbalanced data sets. *Progress in Artificial Intelligence*, *1*(4), 347–362.

García, V., Sánchez, J. S., & Mollineda, R. A. (2010). Exploring the performance of resampling strategies for the class imbalance problem. In *International conference on industrial, engineering and other applications of applied intelligent systems* (pp. 541–549).

Ghotra, B., McIntosh, S., & Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th international conference on software engineering-volume 1* (pp. 789–800).

Ghunaim, H., & Dichter, J. (2019). Applying the fahp to improve the performance evaluation reliability of software defect classifiers. *IEEE Access*, *7*, 62794–62804.

Gil, Y., & Lalouche, G. (2017). On the correlation between size and metric validity. *Empirical Software Engineering*, *22*(5), 2585–2611.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.(JAIR)*, *57*, 345–420.

Gong, R., & Huang, S. H. (2012). A kolmogorov–smirnov statistic based segmentation approach to learning from imbalanced datasets: With application in property refinance prediction. *Expert Systems with Applications*, *39*(6), 6192–6200.

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2011). The misuse of the nasa metrics data program data sets for automated software defect prediction. In *15th annual conference on evaluation & assessment in software engineering (ease*

2011) (pp. 96–103).

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the nasa mdp data sets. *IET software*, *6*(6), 549–558.

Gu, S., Cheng, R., & Jin, Y. (2016). Feature selection for high-dimensional classification using a competitive swarm optimizer. *Soft Computing*, 1–12.

Gupta, S., & Gupta, A. (2017). A set of measures designed to identify overlapped instances in software defect prediction. *Computing*, *99*(9), 889–914.

Ha, J., & Lee, J.-S. (2016). A new under-sampling method using genetic algorithm for imbalanced data classification. In *Proceedings of the 10th international conference on ubiquitous information management and communication* (p. 95).

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, *73*, 220–239.

Haixiang, G., Yijing, L., Yanan, L., Xiao, L., & Jinling, L. (2016). Bpso-adaboost-knn ensemble learning algorithm for multi-class imbalanced data classification. *Engineering Applications of Artificial Intelligence*, *49*, 176–193.

Halim, A. (2013). Predict fault-prone classes using the complexity of UML class diagram. In *International conference on computer, control, informatics and its applications*

(ic3ina) (pp. 289–294).

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304.

Hassan, M., Rajkumar, R., Isa, D., & Arelhi, R. (2011). Kalman filter as a pre-processing technique to improve the support vector machine. In *IEEE conference on sustainable utilization and development in engineering and technology (student)* (pp. 107–112).

Hatton, L. (1970). The automation of software process and product quality. *WIT Transactions on Information and Communication Technologies*, *4*.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, *21*(9), 1263–1284.

He, H., & Ma, Y. (2013). *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons.

He, P., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, *59*, 170–190.

He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, *19*(2), 167–199.

Herbold, S. (2013). Training data selection for cross-project defect prediction. In *Proceedings of the 9th international conference on predictive models in software engineering* (p. 6).

Herraiz, I., German, D. M., & Hassan, A. E. (2011). On the distribution of source code file sizes. In *Icsoft (2)* (pp. 5–14).

Hossain, M. A., Jia, X., & Benediktsson, J. A. (2016). One-class oriented feature selection and classification of heterogeneous remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *9*(4), 1606–1612.

Hosseini, S., Turhan, B., & Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*.

Hosseini, S., Turhan, B., & Mäntylä, M. (2016). Search based training data selection for cross project defect prediction. In *Proceedings of the the 12th international conference on predictive models and data analytics in software engineering* (p. 3).

Hosseini, S., Turhan, B., & Mäntylä, M. (2017). A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Information and Software Technology*.

Idri, A., Benhar, H., Fernández-Alemán, J., & Kadi, I. (2018). A systematic map of medical

data preprocessing in knowledge discovery. *Computer methods and programs in biomedicine*.

Iliou, T., Anagnostopoulos, C.-N., Nerantzaki, M., & Anastassopoulos, G. (2015). A novel machine learning data preprocessing method for enhancing classification algorithms performance. In *Proceedings of the 16th international conference on engineering applications of neural networks (inns)* (p. 11).

Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS medicine*, *2*(8), e124.

Islam, R., & Sakib, K. (2014). A package based clustering for enhancing software defect prediction accuracy. In *17th international conference on computer and information technology (iccit)* (pp. 81–86).

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, *6*(5), 429–449.

Jayalskshmi, T., & Santhakumaran, A. (2010). Impact of preprocessing for diagnosis of diabetes mellitus using artificial neural networks. In *Machine learning and computing (icmlc), 2010 second international conference on* (pp. 109–112).

Jian, C., Gao, J., & Ao, Y. (2016). A new sampling method for classifying imbalanced data based on support vector machine ensemble. *Neurocomputing*, *193*, 115–122.

Jiang, Y., Cukic, B., & Menzies, T. (2007). Fault prediction using early lifecycle data. In *The 18th ieee international symposium on software reliability, issre'07* (pp. 237–246).

Jiarpakdee, J., Tantithamthavorn, C., & Hassan, A. E. (2018). The impact of correlated metrics on defect models. *arXiv preprint arXiv:1801.10271*.

Jing, X., Wu, F., Dong, X., Qi, F., & Xu, B. (2015). Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 496–507).

Jing, X.-Y., Zhang, Z.-W., Ying, S., Wang, F., & Zhu, Y.-P. (2014). Software defect prediction based on collaborative representation classification. In *Companion proceedings of the 36th international conference on software engineering* (pp. 632–633).

Jojan, J., & Srivihok, A. (2013). Preprocessing of imbalanced breast cancer data using feature selection combined with over-sampling technique for classification. In *Advanced computer science and information systems (icacsis), 2013 international conference on* (pp. 407–412).

Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th international conference on predictive models in software engineering* (p. 9).

Kamarulzalis, A. H., Razali, M. H. M., & Moktar, B. (2018). Data pre-processing using smote technique for gender classification with imbalance hu's moments features. In *Proceedings of the second international conference on the future of asean (icofa) 2017–volume 2* (pp. 373–379).

Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., & Hassan, A. E. (2016). Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, *21*(5), 2072–2106.

Kamiran, F., & Calders, T. (2012). Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, *33*(1), 1–33.

Kang, Q., Chen, X., Li, S., & Zhou, M. (2017). A noise-filtered under-sampling scheme for imbalanced classification. *IEEE transactions on cybernetics*, *47*(12), 4263–4274.

Karayiannis, Y. A., Stojanovic, R., Mitropoulos, P., Koulamas, C., Stouraitis, T., Koubias, S., & Papadopoulos, G. (1999). Defect detection and classification on web textile fabric using multiresolution decomposition and neural networks. In *Icecs'99. proceedings of icecs'99. 6th ieee international conference on electronics, circuits and systems (cat. no. 99ex357)* (Vol. 2, pp. 765–768).

Karunaratne, T., Bostrom, H., & Norinder, U. (2010). Pre-processing structured data for standard machine learning algorithms by supervised graph propositionalization-a case study with medicinal chemistry datasets. In *Machine learning and applications (icmla), 2010 ninth international conference on* (pp. 828–833).

Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111–117.

Knab, P., Pinzger, M., & Bernstein, A. (2006). Predicting defect densities in source code files with decision tree learners. In *Proceedings of the 2006 international workshop on mining software repositories* (pp. 119–125).

Kononenko, I., & Bratko, I. (1991). Information-based evaluation criterion for classifier's performance. *Machine Learning*, *6*(1), 67–80.

Kontos, K., & Maragoudakis, M. (2013). Breast cancer detection in mammogram medical images with data mining techniques. In *Ifip international conference on artificial intelligence applications and innovations* (pp. 336–347).

Koru, A. G., & Liu, H. (2005). Building effective defect-prediction models in practice. *IEEE software*, *22*(6), 23–29.

Krawczyk, B. (2016). Cost-sensitive one-vs-one ensemble for multi-class imbalanced data. In *Neural networks (ijcnn), 2016 international joint conference on* (pp. 2447–2452).

Krawczyk, B., Woźniak, M., & Schaefer, G. (2014). Cost-sensitive decision tree ensembles for effective imbalanced classification. *Applied Soft Computing*, *14*, 554–562.

Kubat, M., Holte, R. C., & Matwin, S. (1998). Machine learning for the detection of oil

spills in satellite radar images. *Machine learning*, *30*(2-3), 195–215.

Kubat, M., Matwin, S., et al. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *Icml* (Vol. 97, pp. 179–186).

Kuhn, M., & Johnson, K. (2013). Data pre-processing. In *Applied predictive modeling* (pp. 27–59). Springer.

Kumar, N. S., Rao, K. N., Govardhan, A., Reddy, K. S., & Mahmood, A. M. (2014). Undersampled k k-means approach for handling imbalanced distributed data. *Progress in Artificial Intelligence*, *3*(1), 29–38.

Kwak, N., & Choi, C.-H. (2002). Input feature selection for classification problems. *IEEE Transactions on Neural Networks*, *13*(1), 143–159.

Lai, S.-T., & Leu, F.-Y. (2017). An iterative and incremental data preprocessing procedure for improving the risk of big data project. In *International conference on innovative mobile and internet services in ubiquitous computing* (pp. 483–492).

Landman, D., Serebrenik, A., Bouwers, E., & Vinju, J. J. (2016). Empirical analysis of the relationship between cc and sloc in a large corpus of java methods and c functions. *Journal of Software: Evolution and Process*, *28*(7), 589–618.

Landman, D., Serebrenik, A., & Vinju, J. (2014). Empirical analysis of the relationship between cc and sloc in a large corpus of java methods. In *IEEE international*

conference on software maintenance and evolution (icsme) (pp. 221–230).

Lane, P. C., Clarke, D., & Hender, P. (2012). On developing robust models for favourability analysis: Model choice, feature sets and imbalanced data. *Decision Support Systems*, *53*(4), 712–718.

Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, *58*, 388–402.

Laranjeira, L. A. (1990). Software size estimation of object-oriented systems. *IEEE Transactions on software engineering*, *16*(5), 510–522.

Larsen, K. (2005). Generalized naive bayes classifiers. *ACM SIGKDD Explorations Newsletter*, *7*(1), 76–81.

LaTonya Pearson. (2014). *The Rising Costs of Defects.* `https://www.seguetech.com/`. (Online; accessed 20 August 2019)

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, *34*(4), 485–496.

Li, J., Liu, L.-s., Fong, S., Wong, R. K., Mohammed, S., Fiaidhi, J., . . . Wong, K. K. (2017). Adaptive swarm balancing algorithms for rare-event prediction in imbalanced

healthcare data. *PloS one*, *12*(7), e0180830.

Li, K., Zhang, Z., & Liu, M. (2010). One data preprocessing method in high-speed network intrusion detection.

Li, Q., Rajagopalan, C., & Clifford, G. D. (2014). Ventricular fibrillation and tachycardia classification using a machine learning approach. *IEEE Transactions on Biomedical Engineering*, *61*(6), 1607–1613.

Li, Q., Yang, B., Li, Y., Deng, N., & Jing, L. (2013). Constructing support vector machine ensemble with segmentation for imbalanced datasets. *Neural computing and applications*, *22*(1), 249–256.

Li, W., Mo, W., Zhang, X., Squiers, J. J., Lu, Y., Sellke, E. W., . . . Thatcher, J. E. (2015). Outlier detection and removal improves accuracy of machine learning approach to multispectral burn diagnostic imaging. *Journal of biomedical optics*, *20*(12), 121305.

Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., & Ying, S. (2017). On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Transactions on Software Engineering*.

Liebchen, G., & Shepperd, M. (2016). Data sets and data quality in software engineering: eight years on. In *Proceedings of the the 12th international conference on predictive models and data analytics in software engineering* (p. 7).

Lima, R. F., & Pereira, A. C. M. (2015). A fraud detection model based on feature selection and undersampling applied to web payment systems. In *IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology (wi-iat)* (Vol. 3, pp. 219–222).

Lin, H.-Y., & Tzeng, W.-G. (2012). A secure erasure code-based cloud storage system with secure data forwarding. *IEEE transactions on parallel and distributed systems*, *23*(6), 995–1003.

Linda M. Laird. (2005). *Software Engineering Metrics.* `https://www.slideplayer.com/`. (Online; accessed 20 August 2016)

Liu, W., Hua, G., & Smith, J. R. (2014). Unsupervised one-class learning for automatic outlier removal. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3826–3833).

Liu, X.-Y., & Zhou, Z.-H. (2006). The influence of class imbalance on cost-sensitive learning: An empirical study. In *Sixth international conference on data mining, 2006. icdm'06.* (pp. 970–974).

Liu, Y., Khoshgoftaar, T. M., & Seliya, N. (2010). Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering*, *36*(6), 852–864.

Liu, Y., Yu, X., Huang, J. X., & An, A. (2011). Combining integrated sampling with

svm ensembles for learning from imbalanced datasets. *Information Processing & Management*, *47*(4), 617–631.

Longadge, R., & Dongre, S. (2013). Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*.

López, V., Fernández, A., Del Jesus, M. J., & Herrera, F. (2012). Cost sensitive and preprocessing for classification with imbalanced data-sets: Similar behaviour and potential hybridizations. In *Icpram (2)* (pp. 98–107).

López, V., Fernández, A., García, S., Palade, V., & Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, *250*, 113–141.

Lu, H., Cukic, B., & Culp, M. (2012). Software defect prediction using semi-supervised learning with dimension reduction. In *27th IEEE/ACM international conference on automated software engineering (ase)* (pp. 314–317).

Luengo, J., García, S., & Herrera, F. (2012). On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems*, *32*(1), 77–108.

Lusa, L., et al. (2010). Class prediction for high-dimensional class-imbalanced data. *BMC bioinformatics*, *11*(1), 523.

Ma, & Cukic, B. (2007). Adequate and precise evaluation of quality models in software engineering studies. In *Promise'07: Icse workshops 2007. international workshop on predictor models in software engineering* (pp. 1–1).

Ma, B., Zhang, H., Chen, G., Zhao, Y., & Baesens, B. (2014). Investigating associative classification for software fault prediction: an experimental perspective. *International Journal of Software Engineering and Knowledge Engineering*, *24*(01), 61–90.

Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology*, *54*(3), 248–256.

Mahmood, Z., Bowes, D., Lane, P. C., & Hall, T. (2015). What is the impact of imbalance on software defect prediction performance? In *Proceedings of the 11th international conference on predictive models and data analytics in software engineering* (p. 4).

Malaiya, Y. K., & Denton, J. (2000). Module size distribution and defect density. In *Software reliability engineering, 2000. issre 2000. proceedings. 11th international symposium on* (pp. 62–71).

Maldonado, S., Weber, R., & Famili, F. (2014). Feature selection for high-dimensional class-imbalanced data sets using support vector machines. *Information Sciences*, *286*, 228–246.

Malhotra, R., & Raje, R. (2014). An empirical comparison of machine learning techniques for software defect prediction. In *Proceedings of the 8th international conference on bioinspired information and communications technologies* (pp. 320–327).

Maneerat, N., & Muenchaisri, P. (2011). Bad-smell prediction from software design model using machine learning techniques. In *Eighth international joint conference on computer science and software engineering (jcsse)* (pp. 331–336).

Manek, A. S., Samhitha, M., Shruthy, S., Bhat, V. H., Shenoy, P. D., Mohan, M. C., . . . Patnaik, L. (2013). Repid-ok: Spam detection using repetitive pre-processing. In *2013 international conference on cloud & ubiquitous computing & emerging technologies* (pp. 144–149).

Manikandan, P., Ramyachitra, D., Kalaivani, S., & Rani, R. R. (2016). An improved instance based k-nearest neighbor (iibk) classification of imbalanced datasets with enhanced preprocessing. *International Journal of Applied Engineering Research*, *11*(1), 642–649.

Manly, B. F., & Alberto, J. A. N. (2016). *Multivariate statistical methods: a primer*. CRC Press.

Mao, W., Wang, J., He, L., & Tian, Y. (2017). Online sequential prediction of imbalance data with two-stage hybrid strategy by extreme learning machine. *Neurocomputing*.

Marston, T. M., Kennedy, J. L., & Marston, P. L. (2011). Coherent and semi-coherent

processing of limited-aperture circular synthetic aperture (csas) data. In *Oceans 2011* (pp. 1–6).

Masetic, Z., & Subasi, A. (2016). Congestive heart failure detection using random forest classifier. *Computer methods and programs in biomedicine*, *130*, 54–64.

Mausa, G., Grbac, T. G., & Basic, B. D. (2014). Software defect prediction with bug-code analyzer-a data collection tool demo. In *22nd international conference on software, telecommunications and computer networks (softcom)* (pp. 425–426).

Meadem, N., Verbiest, N., Zolfaghar, K., Agarwal, J., Chin, S.-C., & Roy, S. B. (2013). Exploring preprocessing techniques for prediction of risk of readmission for congestive heart failure patients. In *Data mining and healthcare (dmh), at international conference on knowledge discovery and data mining (kdd)* (Vol. 150).

Menardi, G., & Torelli, N. (2014). Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, *28*(1), 92–122.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, *33*(1).

Miyazaki, Y., Terakado, M., Ozaki, K., & Nozaki, H. (1994). Robust regression for developing software estimation models. *Journal of Systems and Software*, *27*(1), 3–16.

Moepya, S. O., Akhoury, S. S., & Nelwamondo, F. V. (2014). Applying cost-sensitive classification for financial fraud detection under high class-imbalance. In *2014 IEEE international conference on data mining workshop (icdmw)* (pp. 183–192).

Mohagheghi, P., Conradi, R., Killi, O. M., & Schwarz, H. (2004). An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the 26th international conference on software engineering* (pp. 282–292).

Mohd, F., Bakar, Z. A., Noor, N. M. M., & Rajion, Z. A. (2013). Data preparation for pre-processing on oral cancer dataset. In *Control, automation and systems (iccas), 2013 13th international conference on* (pp. 324–328).

Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M., & Matsumoto, K. (2013). Assessing the cost effectiveness of fault prediction in acceptance testing. *Software Engineering, IEEE Transactions on*, *39*(10), 1345–1357.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (Vol. 821). John Wiley & Sons.

Mori, T. (2015). Superposed naive bayes for accurate and interpretable prediction. In *2015 ieee 14th international conference on machine learning and applications (icmla)* (pp. 1228–1233).

Munková, D., Munk, M., & Vozár, M. (2013). Data pre-processing evaluation for text mining: transaction/sequence model. *Procedia Computer Science*, *18*, 1198–1207.

Muresan, S., Faloba, I., Lemnaru, C., & Potolea, R. (2015). Pre-processing flow for enhancing learning from medical data. In *IEEE international conference on intelligent computer communication and processing (iccp)* (pp. 27–34).

Nagappan, N., & Ball, T. (2005). Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on software engineering* (pp. 284–292).

Nagappan, N., Ball, T., & Murphy, B. (2006). Using historical in-process and product metrics for early estimation of software failures. In *Software reliability engineering, 2006. issre'06. 17th international symposium on* (pp. 62–74).

Nagappan, N., Ball, T., & Zeller, A. (2006). Mining metrics to predict component failures. In *Proceedings of the 28th international conference on software engineering* (pp. 452–461).

Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2017). Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*.

Nam, J., & Kim, S. (2015). Heterogeneous defect prediction. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 508–519).

Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. In *Proceedings of the 2013 international conference on software engineering* (pp. 382–391).

Nekooeimehr, I., & Lai-Yuen, S. K. (2016). Adaptive semi-unsupervised weighted oversampling (a-suwo) for imbalanced datasets. *Expert Systems with Applications*, *46*, 405–416.

Nelson, A., Menzies, T., & Gay, G. (2011). Sharing experiments using open-source software. *Software: Practice and Experience*, *41*(3), 283–305.

Ng, W. W., Zeng, G., Zhang, J., Yeung, D. S., & Pedrycz, W. (2016). Dual autoencoders features for imbalance classification problem. *Pattern Recognition*, *60*, 875–889.

Nguyen, N., Leclerc, A., & LeBlanc, G. (2013). The mediating role of customer trust on customer loyalty. *Journal of service science and management*, *6*(01), 96.

Nikulin, V., McLachlan, G. J., & Ng, S. K. (2009). Ensemble approach for the classification of imbalanced data. In *Australasian joint conference on artificial intelligence* (pp. 291–300).

Nithya, P., & Sumathi, P. (2012). An enhanced pre-processing technique for web log mining by removing web robots. In *2012 IEEE international conference on computational intelligence & computing research (iccic)* (pp. 1–4).

Nugroho, A., Chaudron, M. R., & Arisholm, E. (2010). Assessing uml design metrics for predicting fault-prone classes in a java system. In *7th IEEE working conference on mining software repositories (msr), 2010* (pp. 21–30).

Ohsaki, M., Wang, P., Matsuda, K., Katagiri, S., Watanabe, H., & Ralescu, A. (2017). Confusion-matrix-based kernel logistic regression for imbalanced data classification. *IEEE Transactions on Knowledge and Data Engineering*.

O'brien, R. M. (2007). A caution regarding rules of thumb for variance inflation factors. *Quality & quantity*, *41*(5), 673–690.

Palacios, A. M., Sánchez, L., & Couso, I. (2010). Preprocessing vague imbalanced datasets and its use in genetic fuzzy classifiers. In *IEEE international conference on fuzzy systems (fuzz), 2010* (pp. 1–8).

Panichella, A., Oliveto, R., & De Lucia, A. (2014). Cross-project defect prediction models: L'union fait la force. In *Software evolution week-IEEE conference on software maintenance, reengineering and reverse engineering (csmr-wcre)* (pp. 164–173).

Parthipan, S., Senthil Velan, S., & Babu, C. (2014). Design level metrics to measure the complexity across versions of ao software. In *International conference on advanced communication control and computing technologies (icaccct)* (pp. 1708–1714).

Pereira, G., Barbosa, R., & Madeira, H. (2016). Practical emulation of software defects in source code. In *Dependable computing conference (edcc), 2016 12th european* (pp. 130–140).

Pérez, J., Iturbide, E., Olivares, V., Hidalgo, M., Almanza, N., & Martínez, A. (2015). A data preparation methodology in data mining applied to mortality population

databases. In *New contributions in information systems and technologies* (pp. 1173–1182). Springer.

Peters, F., Menzies, T., & Layman, L. (2015). Lace2: Better privacy-preserving data sharing for cross project defect prediction. In *Proceedings of the 37th international conference on software engineering-volume 1* (pp. 801–811).

Petrić, J. (2016). Using different characteristics of machine learners to identify different defect families. In *Proceedings of the 20th international conference on evaluation and assessment in software engineering* (p. 5).

Petrić, J., Bowes, D., Hall, T., Christianson, B., & Baddoo, N. (2016). The jinx on the nasa software defect data sets. In *Proceedings of the 20th international conference on evaluation and assessment in software engineering* (p. 13).

Planning, S. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*.

Qiao, Y., JIANG, S., & ZHANG, Y. (2017). The performance stability of defect prediction models with class imbalance: An empirical study.

Rahman, F., Posnett, D., Herraiz, I., & Devanbu, P. (2013). Sample size vs. bias in defect prediction. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering* (pp. 147–157).

Rahman, M. H., Sharmin, S., Sarwar, S. M., & Shoyaib, M. (2016). Software defect prediction using feature space transformation. In *Proceedings of the international conference on internet of things and cloud computing* (p. 72).

Ramler, R., Himmelbauer, J., & Natschläger, T. (2014). Building defect prediction models in practice. In *Handbook of research on emerging advancements and technologies in software engineering* (pp. 540–565). IGI Global.

Ramya, R., Priyadarshini, S., & Karthik, S. (2012). An enhanced secure preserving for pre-processed data using dmi and pcrbac algorithm. In *Proceedings of the second international conference on computational science, engineering and information technology* (pp. 283–287).

Rhoads, G. B. (2011, May 3). *Processing data representing video and audio and methods related thereto.* Google Patents. (US Patent 7,936,900)

Rob Manser. (2016). *Impacts of Bad Data Lead to Negative Consequences.* `https://www.serviceobjects.com/`. (Online; accessed 20 August 2019)

Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., & Riquelme, J. C. (2014). Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 43).

Roy, A., Cruz, R. M., Sabourin, R., & Cavalcanti, G. D. (2018). A study on combining

dynamic selection and data preprocessing for imbalance learning. *Neurocomputing*, *286*, 179–192.

Ryu, D., Choi, O., & Baik, J. (2016). Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empirical Software Engineering*, *21*(1), 43–71.

Ryu, D., Jang, J.-I., & Baik, J. (2015). A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *Journal of Computer Science and Technology*, *30*(5), 969–980.

Ryu, D., Jang, J.-I., & Baik, J. (2017). A transfer cost-sensitive boosting approach for cross-project defect prediction. *Software Quality Journal*, *25*(1), 235–272.

Sáez, J. A., Luengo, J., Stefanowski, J., & Herrera, F. (2015). Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, *291*, 184–203.

Saleem, A., Asif, K. H., Ali, A., Awan, S. M., & Alghamdi, M. A. (2014). Pre-processing methods of data mining. In *Ieee/acm 7th international conference on utility and cloud computing (ucc)* (pp. 451–456).

Sayyad Shirabad, J., & Menzies, T. (2005). *The PROMISE repository of software engineering databases.".* School of Information Technology and Engineering, University of Ottawa, Canada". Retrieved from "`http://promise.site.uottawa.ca/`

SERepository"

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, *5*(2), 197–227.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel methods in computational biology*. MIT press.

Seidman, S. B. (2008). The emergence of software engineering professionalism. In *E-government ict professionalism and competences service science* (pp. 59–67). Springer.

Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2010). Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *40*(1), 185–197.

Shah, S. M. A., Morisio, M., & Torchiano, M. (2012). The impact of process maturity on defect density. In *Empirical software engineering and measurement (esem), 2012 acm-ieee international symposium on* (pp. 315–318).

Shanab, A. A., Khoshgoftaar, T. M., Wald, R., & Van Hulse, J. (2012). Evaluation of the importance of data pre-processing order when combining feature selection and data sampling. *International Journal of Business Intelligence and Data Mining*, *7*(1-2), 116–134.

Shashua, A. (2009). Introduction to machine learning: Class notes 67577. *arXiv preprint arXiv:0904.3664*.

Shepperd, M. (2015). How do i know whether to trust a research result? *IEEE Software*, *32*(1), 106–109.

Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *Software Engineering, IEEE Transactions on*, *40*(6), 603–616.

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215.

Shippey, T., Hall, T., Counsell, S., & Bowes, D. (2016). So you need more method level datasets for your software defect prediction?: Voilà! In *Proceedings of the 10th acm/ieee international symposium on empirical software engineering and measurement* (p. 12).

Shroff, K. P., & Maheta, H. H. (2015). A comparative study of various feature selection techniques in high-dimensional data set to improve classification accuracy. In *International conference on computer communication and informatics (iccci)* (pp. 1–6).

Siebra, C. A., & Mello, M. A. (2015). The importance of replications in software

engineering: a case study in defect prediction. In *Proceedings of the 2015 conference on research in adaptive and convergent systems* (pp. 376–381).

Singh, L. D., Das, P., & Kar, N. (2013). A pre-processing algorithm for faster convex hull computation.

Smith, M., Szongott, C., Henne, B., & Von Voigt, G. (2012). Big data privacy issues in public social media. In *6th ieee international conference on digital ecosystems technologies (dest)* (pp. 1–6).

Soda, P. (2011). A multi-objective optimisation approach for class imbalance learning. *Pattern Recognition*, *44*(8), 1801–1810.

Sommerville, I. (2004). *Software engineering, isbn:0-321-21026-3.* Pearson Education Limited.

Song, J., Huang, X., Qin, S., & Song, Q. (2016). A bi-directional sampling based on k-means method for imbalance text classification. In *2016 ieee/acis 15th international conference on computer and information science (icis)* (pp. 1–5).

Song, L., Li, D., Zeng, X., Wu, Y., Guo, L., & Zou, Q. (2014). ndna-prot: identification of dna-binding proteins based on unbalanced classification. *BMC bioinformatics*, *15*(1), 298.

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2010). A general software defect-

proneness prediction framework. *IEEE transactions on software engineering*, *37*(3), 356–370.

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, *37*(3), 356–370.

Stefanowski, J. (2016). Dealing with data difficulty factors while learning from imbalanced data. In *Challenges in computational statistics and data mining* (pp. 333–363). Springer.

Subasi, A., Alickovic, E., & Kevric, J. (2017). Diagnosis of chronic kidney disease by using random forest. In *Cmbebih 2017* (pp. 589–594). Springer.

Sullivan, M., & Chillarege, R. (1991). Software defects and their impact on system availability: A study of field failures in operating systems. In *Ftcs* (Vol. 21, pp. 2–9).

Sun, Y., Wong, A. K., & Kamel, M. S. (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, *23*(04), 687–719.

Sun, Z., Song, Q., Zhu, X., Sun, H., Xu, B., & Zhou, Y. (2015). A novel ensemble method for classifying imbalanced data. *Pattern Recognition*, *48*(5), 1623–1637.

Taba, S. E. S., Khomh, F., Zou, Y., Hassan, A. E., & Nagappan, M. (2013). Predicting bugs using antipatterns. In *2013 ieee international conference on software maintenance* (pp. 270–279).

Taipale, T., Qvist, M., & Turhan, B. (2013). Constructing defect predictors and communicating the outcomes to practitioners. In *2013 acm/ieee international symposium on empirical software engineering and measurement* (pp. 357–362).

Tang, Y., Zhang, Y.-Q., Chawla, N. V., & Krasser, S. (2009). Svms modeling for highly imbalanced classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(1), 281–288.

Tantithamthavorn, C. (2016). Towards a better understanding of the impact of experimental components on defect prediction modelling. In *Proceedings of the 38th international conference on software engineering companion* (pp. 867–870).

Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *arXiv preprint arXiv:1801.10269*.

Tantithamthavorn, C., McIntosh, S., Hassan, A., & Matsumoto, K. (2016b). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., & Matsumoto, K. (2015). The

impact of mislabelling on the performance and interpretation of defect prediction models. In *Ieee/acm 37th ieee international conference on software engineering (icse)* (Vol. 1, pp. 812–823).

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016a). Comments on "researcher bias: the use of machine learning in software defect prediction". *IEEE Transactions on Software Engineering*, *42*(11), 1092–1094.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, *43*(1), 1–18.

Tavares, T. R., Oliveira, A. L., Cabral, G. G., Mattos, S. S., & Grigorio, R. (2013). Preprocessing unbalanced data using weighted support vector machines for prediction of heart disease in children. In *Neural networks (ijcnn), the 2013 international joint conference on* (pp. 1–8).

Trafalis, T. B., Adrianto, I., Richman, M. B., & Lakshmivarahan, S. (2014). Machine-learning classifiers for imbalanced tornado data. *Computational Management Science*, *11*(4), 403–418.

Tsai, C.-F., & Chou, J.-S. (2011). Data pre-processing by genetic algorithms for bankruptcy prediction. In *Industrial engineering and engineering management (ieem), 2011 ieee international conference on* (pp. 1780–1783).

Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, *14*(5), 540–578.

Turhan, B., Mısırlı, A. T., & Bener, A. (2013). Empirical evaluation of the effects of mixed project data on learning defect predictors. *Information and Software Technology*, *55*(6), 1101–1118.

Uchigaki, S., Uchida, S., Toda, K., & Monden, A. (2012). An ensemble approach of simple regression models to cross-project fault prediction. In *13th acis international conference on software engineering, artificial intelligence, networking and parallel & distributed computing (snpd)* (pp. 476–481).

Van Hulse, J., Khoshgoftaar, T. M., & Napolitano, A. (2007). Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on machine learning* (pp. 935–942).

Verma, R., & Gupta, A. (2012). Software defect prediction using two level data pre-processing. In *Recent advances in computing and software systems (racss), 2012 international conference on* (pp. 311–317).

Vong, C.-M., Ip, W.-F., Chiu, C.-C., & Wong, P.-K. (2015). Imbalanced learning for air pollution by meta-cognitive online sequential extreme learning machine. *Cognitive Computation*, *7*(3), 381–391.

Wahono, R. S. (2015). A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks. *Journal of Software Engineering*, *1*(1), 1–16.

Wald, R., Khoshgoftaar, T. M., & Shanab, A. A. (2013). Comparison of two frameworks for measuring the stability of gene-selection techniques on noisy class-imbalanced data. In *Tools with artificial intelligence (ictai), 2013 ieee 25th international conference on* (pp. 881–888).

Wallace, B. C., Small, K., Brodley, C. E., & Trikalinos, T. A. (2011). Class imbalance, redux. In *Ieee 11th international conference on data mining (icdm)* (pp. 754–763).

Wang, H., & Wang, S. (2010). Mining incomplete survey data through classification. *Knowledge and information systems*, *24*(2), 221–233.

Wang, S., Li, Z., Chao, W., & Cao, Q. (2012). Applying adaptive over-sampling technique based on data density and cost-sensitive svm to imbalanced learning. In *Neural networks (ijcnn), the 2012 international joint conference on* (pp. 1–8).

Wang, S., & Yao, X. (2013). Relationships between diversity of classification ensembles and single-class performance measures. *IEEE Transactions on Knowledge and Data Engineering*, *25*(1), 206–219.

Watanabe, S., Kaiya, H., & Kaijiri, K. (2008). Adapting a fault prediction model to allow inter languagereuse. In *Proceedings of the 4th international workshop on predictor*

models in software engineering (pp. 19–24).

Wei, M.-H., Cheng, C.-H., Huang, C.-S., & Chiang, P.-C. (2013). Discovering medical quality of total hip arthroplasty by rough set classifier with imbalanced class. *Quality & Quantity*, *47*(3), 1761–1779.

Wei, W., Li, J., Cao, L., Ou, Y., & Chen, J. (2013). Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, *16*(4), 449–475.

Weiss, G. M., & Provost, F. (2003). Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, *19*, 315–354.

Westfall, L. (2013). *Defect density.*

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241–259.

Wong, G. Y., Leung, F. H., & Ling, S.-H. (2013). A novel evolutionary preprocessing method based on over-sampling and under-sampling for imbalanced datasets. In *Industrial electronics society, iecon 2013-39th annual conference of the ieee* (pp. 2354–2359).

Xiaoli, G., Ying, Y., Ling, W., Zhaoyang, Q., & Yongwen, W. (2015). Wind data preprocessing algorithm based on extracting isolated points. *International Journal*

of Multimedia and Ubiquitous Engineering, *10*(4), 53–62.

Xu, Z., Liu, J., Yang, Z., An, G., & Jia, X. (2016). The impact of feature selection on defect prediction performance: An empirical comparison. In *Ieee 27th international symposium on software reliability engineering (issre)* (pp. 309–320).

Xuan, X., Lo, D., Xia, X., & Tian, Y. (2015). Evaluating defect prediction approaches using a massive set of metrics: An empirical study. In *Proceedings of the 30th annual acm symposium on applied computing* (pp. 1644–1647).

Yang, J., Zhou, J., Zhu, Z., Ma, X., & Ji, Z. (2016). Iterative ensemble feature selection for multiclass classification of imbalanced microarray data. *Journal of Biological Research-Thessaloniki*, *23*(1), 13.

Yang, P., Xu, L., Zhou, B. B., Zhang, Z., & Zomaya, A. Y. (2009). A particle swarm based hybrid system for imbalanced medical data sampling. In *Bmc genomics* (Vol. 10, p. S34).

Yang, P., Yoo, P. D., Fernando, J., Zhou, B. B., Zhang, Z., & Zomaya, A. Y. (2014). Sample subset optimization techniques for imbalanced and ensemble learning problems in bioinformatics applications. *IEEE transactions on cybernetics*, *44*(3), 445–455.

Yang, X.-L., Lo, D., Xia, X., Huang, Q., & Sun, J.-L. (2017). High-impact bug report identification with imbalanced learning strategies. *J. Comput. Sci. & Technol*, *32*(1).

Yijing, L., Haixiang, G., Xiao, L., Yanan, L., & Jinling, L. (2016). Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data. *Knowledge-Based Systems*, *94*, 88–104.

Yin, Q.-Y., Zhang, J.-S., Zhang, C.-X., & Ji, N.-N. (2014). A novel selective ensemble algorithm for imbalanced data classification based on exploratory undersampling. *Mathematical Problems in Engineering*, *2014*.

Yousefi, S., & Modiri, N. (2011). Deployment of integrated design for the reduction of software complexity. In *7th international conference on networked computing and advanced information management (ncm)* (pp. 172–174).

Yu, L., & Mishra, A. (2012). Experience in predicting fault-prone software modules using complexity metrics. *Quality Technology & Quantitative Management*, *9*(4), 421–434.

Yun, J., Ha, J., & Lee, J.-S. (2016). Automatic determination of neighborhood size in smote. In *Proceedings of the 10th international conference on ubiquitous information management and communication* (p. 100).

Zhai, J., Zhang, S., & Wang, C. (2017). The classification of imbalanced large data sets based on mapreduce and ensemble of elm classifiers. *International Journal of Machine Learning and Cybernetics*, *8*(3), 1009–1017.

Zhang, C., Tan, K. C., & Ren, R. (2016). Training cost-sensitive deep belief networks

on imbalance data problems. In *Neural networks (ijcnn), 2016 international joint conference on* (pp. 4362–4367).

Zhang, D., Ma, J., Yi, J., Niu, X., & Xu, X. (2015). An ensemble method for unbalanced sentiment classification. In *11th international conference on natural computation (icnc)* (pp. 440–445).

Zhang, F., Hassan, A. E., McIntosh, S., & Zou, Y. (2017). The use of summation to aggregate software metrics hinders the performance of defect prediction models. *IEEE Transactions on Software Engineering*, *43*(5), 476–491.

Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y. (2014). Towards building a universal defect prediction model. In *Proceedings of the 11th working conference on mining software repositories* (pp. 182–191).

Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y. (2016). Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, *21*(5), 2107–2145.

Zhang, N. (2016). Cost-sensitive spectral clustering for photo-thermal infrared imaging data. In *Information science and technology (icist), 2016 sixth international conference on* (pp. 358–361).

Zhang, S., Zhang, C., & Yang, Q. (2003). Data preparation for data mining. *Applied artificial intelligence*, *17*(5-6), 375–381.

Zhang, Y., Lo, D., Xia, X., & Sun, J. (2015). An empirical study of classifier combination for cross-project defect prediction. In *Computer software and applications conference (compsac), 2015 ieee 39th annual* (Vol. 2, pp. 264–269).

Zhang, Z.-W., Jing, X.-Y., & Wang, T.-J. (2017). Label propagation based semi-supervised learning for software defect prediction. *Automated Software Engineering*, *24*(1), 47–69.

Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. In *Proceedings of the third international workshop on predictor models in software engineering* (p. 9).