



Investigating the Use of Ensemble Techniques in Predicting Object-Oriented Software Maintainability

By

Hadeel Abdullah Alsolai

A thesis submitted as partial fulfilment of the requirement of

Doctor of Philosophy

Faculty of Science, Computer and Information Sciences

University of Strathclyde


Glasgow, United Kingdom

August 2020

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Signed: 

Date: 17-08-2020

Abstract

Context: Prediction of the maintainability of classes in object-oriented systems is a significant factor for software success; however, it is a challenging task. Although prior object-oriented software maintainability literature acknowledges the role of machine learning techniques as valuable predictors of potential change, the most suitable technique that consistently achieves high accuracy remains undetermined and there is no clear indication of which techniques are more appropriate.

Objective: This thesis aims to empirically investigate the capability of ensemble models to provide an increased prediction accuracy, compared with individual models, by applying them on several software maintainability datasets using different base models and analysing the impact of parameter tuning.

Method: In the first part of this thesis, a systematic review of studies related to the prediction of the maintainability of object-oriented software systems using machine learning techniques is presented. In the remaining parts of this thesis, three empirical studies were performed to evaluate and compare different homogeneous and heterogeneous ensemble models against sets of individual models for predicting software maintainability of object-oriented systems at the class level. These models were employed on 14 datasets that were extracted from the maintenance of object-oriented software systems.

Results: The systematic literature review determined 56 relevant studies and indicated that the application of ensemble models is relatively rare, thus there is a need to perform studies using these models as well as others to an extensive variety of datasets. The results obtained from three empirical studies indicate that the proposed ensemble models yield improved prediction accuracy over most of the individual models. This improvement was significant only in the third empirical study, along with a few cases in the second empirical study. In most cases, k-nearest neighbours or support vector regression achieved the best prediction accuracy among individual models; moreover, these models as a base model in bagging and additive regression outperformed other prediction models, along with random forest.

Conclusion: The main finding is that ensemble models are effective for predicting software maintainability and they are more accurate than some individual models; their performance may be improved by using large datasets, or parameter tuning. Also, ensemble models improve the performance of weaker base models.

List of Publication

In the process of writing this thesis, five publications have been published as follows:

1. H. Alsolai, "Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques," in IEEE International Conference on Software Maintenance and Evolution, Madrid, Spain, 2018. (*Reference number:* [1], and appeared in Chapter 4).
2. H. Alsolai and M. Roper, "Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability," in the Proceedings of the Evaluation and Assessment on Software Engineering, Copenhagen, Denmark, 2019. (*Reference number:*[2], and appeared in Chapter 1).
3. H. Alsolai and M. Roper, " Determining the Best Prediction Accuracy of Software Maintainability Models Using Auto-WEKA, " in the Proceedings of the International Conference on Computing, Riyadh, Saudi Arabia, 2019. (*Reference number:*[3], and appeared in Chapter 5).
4. H. Alsolai and M. Roper, "A Systematic Review of Feature Selection Techniques in Software Quality Prediction," in the International Conference on Electrical and Computing Technologies and Applications, Ras Al Khaimah, UAE, 2019. (*Reference number:*[4], and appeared in Chapter 6).
5. H. Alsolai and M. Roper, "A Systematic Literature Review of Machine Learning Techniques for Software Maintainability Prediction," in Information and Software Technology Journal, vol. 119, p. 106214, 2020. (*Reference number:* [5], and appeared in Chapter 2).

Acknowledgment

In the name of Allah, the most Gracious and the most Merciful. First and foremost, praise is to Almighty Allah for his countless mercy and ultimate grace in my life. I thank God for giving me the strength and ability to accomplish this thesis.

I would like to extend my thanks and appreciation to my principal supervisor, Professor. Marc Roper, for careful guidance, advice, and helpful feedback. Under his dedicated supervision, I have learned how to be an independent researcher and creative scientist, and this will support my academic career.

I gratefully acknowledge Princess Nourah Bint Abdulrahman University, Riyadh, Saudi Arabia for sponsoring my PhD study and supporting my publications. In addition, an exceptional thanks to University of Strathclyde, Glasgow, United Kingdom, for providing several software products and useful courses during my PhD research.

Moreover, I would like to thank and express my deep and sincere gratitude to my parents (Abdullah and Badriah) for their endless love, encouragement and support. Thank you both for your continuous care, determination, and motivation throughout my life and even more during my PhD study. My gratitude also goes to my sisters (Leen and Reema) and brothers (Hatim, Nawaf and Faisal) who made this thesis an enjoyable and unforgettable experience. I would like to express my heartfelt gratitude to my lovely kids (my son Ibrahim and my daughter Jumaan), who shared and interacted with my postgraduate study from my pregnancy period until this moment. Being a mother has been one of the greatest gifts in my life.

Finally, I have written the last words of my thesis during a very tough and anxious time in the quarantine due to the COVID-19 outbreak. I am incredibly grateful for the massive efforts that have been made by health care workers around the world for saving the lives of patients. I sincerely hope that the Coronavirus pandemic will soon be over, and I wish everyone is staying safe and healthy.

Table of Contents

Declaration	i
Abstract	ii
List of Publication	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	xi
List of Tables	xiii
List of Acronyms	xvii
Chapter 1. Introduction	1
1.1. <i>Problem Statement</i>	5
1.2. <i>Motivation for the Thesis</i>	6
1.3. <i>Contributions</i>	6
1.4. <i>Research Objectives</i>	7
1.4.1 Objectives of the first empirical study	7
1.4.2 Objectives of the second empirical study	7
1.4.3 Objectives of the third empirical study	7
1.4.4 Research questions.....	8
1.5. <i>Scope of Work</i>	8
1.5.1 Maintainability measurement.....	8
1.5.2 Metrics	9
1.5.3 Datasets.....	10
1.5.4 Evaluation measures	10
1.5.5 Prediction models	11
1.6. <i>Thesis Organisation</i>	12
Chapter 2. A Systematic Literature Review of Machine Learning Models for Software Maintainability Prediction	13
2.1. <i>Introduction</i>	13
2.2. <i>Method</i>	14
2.2.1 Review protocol.....	15
2.2.2 Research questions for SLR.....	15

2.2.3	Search process.....	18
2.2.4	Inclusion and exclusion criteria	22
2.2.5	Quality assessment.....	23
2.2.6	Data extraction.....	24
2.2.7	Data synthesis	24
2.3.	<i>Results</i>	24
2.3.1	Selected primary studies	25
2.3.2	Publications years	25
2.3.3	Publication sources	26
2.3.4	Quality assessment result.....	27
2.4.	<i>Discussion</i>	27
2.4.1	Software maintainability measurement.....	28
2.4.2	Software maintainability metrics	33
2.4.3	Software maintainability datasets	38
2.4.4	Evaluation measures	42
2.4.5	Machine learning problem category.	46
2.4.6	Individual prediction models	47
2.4.7	Ensemble prediction models.....	50
2.5.	<i>Conclusion of Systematic Literature Review</i>	52

Chapter 3. Methodology for Empirical Studies in Software Maintainability Prediction

57

3.1.	<i>Individual Prediction Models</i>	58
3.1.1	Regression tree.....	60
3.1.2	Multilayer perceptron.....	60
3.1.3	K-Nearest neighbours	60
3.1.4	M5Rules.....	61
3.1.5	Support vector machine	61
3.1.6	Support vector regression	61
3.1.7	Naive Bayes	62
3.2.	<i>Ensemble Prediction Models</i>	62
3.2.1	Bootstrap aggregating (Bagging).....	64
3.2.2	Additive regression	65
3.2.3	Random forest.....	65
3.2.4	Stacking	66
3.2.5	Average probability ensemble	66
3.3.	<i>Parameters tuning</i>	67
3.3.1	Caret package.....	67
3.3.2	Auto-WEKA	69
3.3.3	Grid search.....	69
3.4.	<i>Datasets</i>	70
3.4.1	Change maintenance efforts.....	70
3.4.2	Bug prediction datasets.....	71
3.4.3	Refactoring datasets	72
3.5.	<i>Prediction Accuracy Measures</i>	73
3.5.1	Measures for the regression problem.....	73

3.5.2	Measure for the classification problem.....	75
3.5.3	Baseline.....	76
3.5.4	Criteria	76
3.5.5	Statistical tests and effect size.....	77
3.6.	<i>Validation</i>	78
3.7.	<i>Tools</i>	79
3.7.1	WEKA	79
3.7.2	R.....	79
3.8.	<i>Summary</i>	80
Chapter 4. First Empirical Study: Ensemble Techniques to Predict Change Maintenance Effort Using Well-Established Datasets.....		81
4.1.	<i>Introduction</i>	81
4.2.	<i>Motivation</i>	82
4.3.	<i>Research Method</i>	84
4.4.	<i>Experimental Data Setup</i>	86
4.4.1	Dependent variable: maintainability	86
4.4.2	Independent variables: metrics	87
4.4.3	Datasets pre-processing	87
4.4.4	Descriptive statistics	87
4.4.5	Correlation between metrics in the datasets.....	89
4.5.	<i>Results and Analysis</i>	91
4.5.1	Results of the first empirical study	91
4.5.2	Comparison of the best investigated model with the best model in selected studies.....	111
4.5.3	Impact of the parameters tuning using caret	114
4.5.4	Discussion and answers to research questions for the first empirical study	115
4.6.	<i>Threats to Validity</i>	118
4.6.1	Threats to external validity	118
4.6.2	Threats to internal validity	119
4.6.3	Threats to the construct validity.....	119
4.6.4	Threats to the conclusion validity	120
4.7.	<i>Conclusion of the first empirical study</i>	120
Chapter 5. Second Empirical Study: Ensemble Techniques to Predict Change Maintenance Effort Using More Recent and Larger Datasets		123
5.1.	<i>Introduction</i>	123
5.2.	<i>Motivation</i>	125
5.3.	<i>Research Method</i>	126
5.4.	<i>Experimental Data Setup</i>	129
5.4.1	Data pre-processing	130
5.4.2	Dependent variable: maintainability	135
5.4.3	Independent variable: metrics	137
5.4.4	Descriptive statistics	137

5.4.5	Correlation between metrics in the datasets.....	138
5.5.	<i>Results and Analyses</i>	139
5.5.1	Comparison between prediction models.....	140
5.5.2	Determining the best prediction accuracy using Auto-WEKA.....	164
5.5.3	Discussion and answers to research questions for the second empirical study.....	168
5.6.	<i>Threats to Validity</i>	174
5.7.	<i>Conclusion of the second empirical study</i>	176
Chapter 6. Third Empirical Study: Ensemble Techniques to Predict Change-Proneness Using Newest and Largest Datasets.....		179
6.1.	<i>Introduction</i>	179
6.2.	<i>Motivation</i>	180
6.3.	<i>Research Method</i>	184
6.4.	<i>Experimental Data Setup</i>	188
6.4.1	Evaluation of refactoring datasets.....	188
6.4.2	Dependent variable: change-proneness.....	189
6.4.3	Independent variables: source code metrics.....	190
6.4.4	Datasets analysis	190
6.4.5	Data pre-processing	194
6.5.	<i>Results and Analyses</i>	198
6.5.1	Results of feature selection	198
6.5.2	Results of sampling.....	199
6.5.3	Results of prediction models.....	201
6.5.4	Statistical tests of the third empirical study	207
6.5.5	Impact of parameter tuning for random forests.	208
6.5.6	Discussion and answers to research questions for the third empirical study	209
6.6.	<i>Threats to Validity</i>	212
6.6.1	External validity.....	212
6.6.2	Conclusion validity	212
6.6.3	Internal validity.....	213
6.6.4	Construct validity.....	213
6.7.	<i>Conclusion of the third empirical study</i>	214
Chapter 7. Conclusion and Contributions.....		216
7.1.	<i>Conclusion of the thesis</i>	216
7.2.	<i>Answers to Research Questions</i>	218
7.3.	<i>Contribution</i>	219
7.4.	<i>Recommendations for Practitioners</i>	221
7.5.	<i>Limitations and Possible Future Work</i>	223

References 225

Appendix A	241
Appendix B	246
Appendix C	247

List of Figures

Figure 2.1: The framework of SLR.....	15
Figure 2.2: Process of primary studies selection.....	21
Figure 2.3: Number of selected studies over the years.	26
Figure 2.4: The number of studies in each journal.....	26
Figure 2.5: The number of selected studies in each digital library database.	27
Figure 2.6: The number of selected studies in each software maintenance type.	33
Figure 2.7: The number of studies using metrics type and metrics level.....	37
Figure 2.8: The distribution of metrics.	37
Figure 2.9: The number of studies used each type of the dataset.	40
Figure 2.10: The number of datasets classified by the size of the dataset.	40
Figure 2.11: The distribution of programming language in each study.	41
Figure 2.12: The distribution of evaluation measures used by selected primary studies.....	45
Figure 2.13: Individual prediction models used in selected primary studies.	49
Figure 2.14: Ensemble prediction models used in selected primary studies.....	52
Figure 3.1: Structure of homogeneous and heterogeneous ensemble prediction models.	63
Figure 3.2: Ten- fold cross-validation.....	79
Figure 4.1: The process of the first empirical study.....	85
Figure 4.2: Framework of the first empirical study.	86
Figure 4.3: Boxplots of metrics in QUES dataset.	88
Figure 4.4: Boxplots of metrics in UIMS dataset.	89
Figure 4.5: The correlation between metrics.....	91
Figure 4.6: Pred(.25) of prediction models for QUES dataset.	95
Figure 4.7: Pred(.30) of prediction models for QUES dataset.	95
Figure 4.8: Boxplots of MRE for prediction models in QUES dataset.....	97
Figure 4.9: Boxplots of the residuals for prediction models in QUES dataset.	98
Figure 4.10: Plots of predicted and actual values for prediction models in the QUES dataset.	102
Figure 4.11: Pred(.25) of prediction models for UIMS dataset.	106
Figure 4.12: Pred(.30) of prediction models for UIMS dataset.	106
Figure 4.13: Boxplots of MRE for prediction models in UIMS dataset.	107
Figure 4.14: Boxplots of the residuals for prediction models in UIMS dataset.....	108
Figure 4.15: Plots of predicted and actual values for prediction models in UIMS dataset...	111
Figure 4.16: MMRE values obtained by the best model in the previous selected studies and the best model for the QUES dataset.	113
Figure 4.17: MMRE values obtained by the best model in the previous selected studies and the best model for the UIMS dataset.....	114
Figure 5.1: Framework of the research method.	128
Figure 5.2: Framework of data pre-processing.	131
Figure 5.3: Proportion of outliers removed during data cleaning.	134
Figure 5.4: Boxplots of CHANGE metric.....	136

Figure 5.5: Correlation between source code metrics in the software maintainability datasets.	139
Figure 5.6: Boxplot of MRE for prediction models on all datasets.	143
Figure 5.7: Boxplot of the residuals for prediction models on all datasets.	146
Figure 5.8: Pred(.25) for each prediction model on all datasets.	147
Figure 5.9: Pred(.30) for each prediction model on all datasets.	149
Figure 5.10: Multiple comparisons for prediction models using the residuals.	164
Figure 5.11: MMRE value for selected and ZeroR models in each dataset.	167
Figure 5.12: Box plot of MRE for selected and ZeroR models in each dataset.	167
Figure 6.1: Framework of the fourth scenario.	185
Figure 6.2: Framework of the research method.	188
Figure 6.3: Box plot of the AUC values for prediction models on the scenarios analysed. .	203
Figure 6.4: Ranking of the AUC values for prediction models on the scenarios analysed...	207
Figure 6.5: Multiple comparisons for prediction models using AUC.	208

List of Tables

Table 2.1: Research questions for SLR and their motivation.	17
Table 2.2: Selected journals and conferences.	19
Table 2.3: Data Extraction properties with their research question.	24
Table 2.4: The differences between software maintainability measurements and metrics.	28
Table 2.5: Summary for software maintainability measurement.	30
Table 2.6: Summary of metrics used in different maintenance types.	35
Table 2.7: Summary of software maintainability metrics.	36
Table 2.8: Summary of different types of dataset.	39
Table 2.9: The tool extraction independent metrics.	42
Table 2.10: The tools for dependent metrics extraction.	42
Table 2.11: Summary of evaluation measures used.	43
Table 2.12: Summary of validation types.	46
Table 2.13: Summary of machine learning problems.	47
Table 2.14: Summary of individual prediction models used with the best model in each study.	48
Table 2.15: Performance of MMRE value for some selected primary studies.	49
Table 2.16: Summary of the ensemble prediction models.	51
Table 3.1: Summary of selected individual prediction models.	59
Table 3.2: Similarities and Differences between homogeneous and heterogeneous ensemble models.	64
Table 3.3: Overview of packages and methods used to create prediction models.	69
Table 3.4: Definitions and description of L&H metrics [9].	71
Table 3.5: Summary of class level source code metrics [57].	72
Table 3.6: Metrics used as independent variables and their categories [156].	73
Table 3.7: Confusion matrix [163].	76
Table 4.1: Summary of selected paper using machine learning models to predict software maintainability.	83
Table 4.2: Correlations between the metrics in UIMS (upper right triangle) and QUES (lower lift triangle).	90
Table 4.3: Performance of the prediction models for the QUES dataset.	93
Table 4.4: One-way ANOVA for RT and ensemble models in QUES dataset using the residuals.	94
Table 4.5: One-way ANOVA for MLP and ensemble models in QUES dataset using the residuals.	94
Table 4.6: One-way ANOVA for KNN and ensemble models in QUES dataset using the residuals.	94
Table 4.7: One-way ANOVA for M5Rules and ensemble models in QUES dataset using the residuals.	94
Table 4.8: One-way ANOVA for SVR and ensemble models in QUES dataset using the residuals.	94
Table 4.9: Performance of the prediction models for the UIMS dataset.	104

Table 4.10: One-way ANOVA for RT and ensemble models in UIMS dataset using the residuals.....	104
Table 4.11: One-way ANOVA for MLP and ensemble models in UIMS dataset using the residuals.....	104
Table 4.12: One-way ANOVA for KNN and ensemble models in UIMS dataset using the residuals.....	105
Table 4.13: One-way ANOVA for M5Rules and ensemble models in UIMS dataset using the residuals	105
Table 4.14: One-way ANOVA for SVR and ensemble models in UIMS dataset using the residuals.....	105
Table 4.15: Performance of MMRE obtained by previous selected studies and proposed work for the QUES and UIMS datasets.	112
Table 4.16: Impact of the parameters tuning on QUES dataset.	115
Table 4.17: Impact of the parameters tuning on UIMS dataset.	115
Table 5.1: Summary of previous studies that applied pre-processing techniques on the software quality datasets.	125
Table 5.2: Summary of the bug prediction datasets [57].	132
Table 5.3: Summary result of pre-processing techniques.	135
Table 5.4: Descriptive statistics for the CHANGE metric.	136
Table 5.5: Baseline models and their corresponding MMRE, MAE and SA values.	150
Table 5.6: Individual models and their corresponding MMRE, MAE and SA values.	151
Table 5.7: Homogeneous ensemble models and their corresponding MMRE, MAE and SA values.....	153
Table 5.8: Heterogeneous ensemble models and their corresponding MMRE, MAE and SA values.....	154
Table 5.9: One-way ANOVA for RT and ensemble models in the Eclipse JDT Core dataset using the residuals.	155
Table 5.10: One-way ANOVA for MLP and ensemble models in the Eclipse JDT Core dataset using the residuals.....	155
Table 5.11: One-way ANOVA for KNN and ensemble models in the Eclipse JDT Core dataset using the residuals.....	155
Table 5.12: One-way ANOVA for M5Rules and ensemble models in the Eclipse JDT Core dataset using the residuals.....	156
Table 5.13: One-way ANOVA for SVR and ensemble models in the Eclipse JDT Core dataset using the residuals.....	156
Table 5.14: One-way ANOVA for RT and ensemble models in the Eclipse PDE UI dataset using the residuals.	156
Table 5.15: One-way ANOVA for MLP and ensemble models in the Eclipse PDE UI dataset using the residuals.	156
Table 5.16: One-way ANOVA for KNN and ensemble models in the Eclipse PDE UI dataset using the residuals.	156
Table 5.17: One-way ANOVA for M5Rules and ensemble models in the Eclipse PDE UI dataset using the residuals.....	156
Table 5.18: One-way ANOVA for SVR and ensemble models in the Eclipse PDE UI dataset using the residuals.	156
Table 5.19: One-way ANOVA for RT and ensemble models in the Equinox Framework dataset using the residuals.	157

Table 5.20: One-way ANOVA for MLP and ensemble models in the Equinox Framework dataset using the residuals.....	157
Table 5.21: One-way ANOVA for KNN and ensemble models in the Equinox Framework dataset using the residuals.....	157
Table 5.22: One-way ANOVA for M5Rules and ensemble models in the Equinox Framework dataset using the residuals.	157
Table 5.23: One-way ANOVA for SVR and ensemble models in the Equinox Framework dataset using the residuals.....	157
Table 5.24: One-way ANOVA for RT and ensemble models in the Lucene dataset using the residuals.....	157
Table 5.25: One-way ANOVA for MLP and ensemble models in the Lucene dataset using the residuals.....	157
Table 5.26: One-way ANOVA for KNN and ensemble models in the Lucene dataset using the residuals.....	157
Table 5.27: One-way ANOVA for M5Rules and ensemble models in the Lucene dataset using the residuals values.....	158
Table 5.28: One-way ANOVA for SVR and ensemble models in the Lucene dataset using the residuals.....	158
Table 5.29: One-way ANOVA for RT and ensemble models in the Mylyn dataset using the residuals.....	158
Table 5.30: One-way ANOVA for MLP and ensemble models in the Mylyn dataset using the residuals.....	158
Table 5.31: One-way ANOVA for KNN and ensemble models in the Mylyn dataset using the residuals.....	158
Table 5.32: One-way ANOVA for M5Rules and ensemble models in the Mylyn dataset using the residuals.....	158
Table 5.33: One-way ANOVA for SVR and ensemble models in the Mylyn dataset using the residuals.....	158
Table 5.34: Best model selected by Auto-WEKA in each dataset.....	165
Table 5.35: MMRE and MAE values for the selected models and ZeroR models.....	166
Table 6.1: Summary of FS, datasets and prediction models in software quality prediction.	182
Table 6.2: Scenarios in the empirical study.	184
Table 6.3: Summary of the datasets.	189
Table 6.4: Number of True and False values in the change-proneness attribute.	190
Table 6.5: Metrics with zero values that were removed using descriptive statistics	191
Table 6.6: Strong correlation metrics using Spearman correlation.....	192
Table 6.7: Number of metrics removed in each data analysis technique.....	193
Table 6.8: Parameters used in Weka for sampling techniques.....	198
Table 6.9: Best ten metrics using ensemble FS method.....	199
Table 6.10: Results before and after applying SMOTE.....	200
Table 6.11: Results before and after applying SpreadSubsample.....	200
Table 6.12: AUC values for performance evaluation of prediction models across seven datasets in the first scenario.	204
Table 6.13: AUC values for performance evaluation of prediction models across seven datasets in the second scenario.....	205
Table 6.14: AUC values for performance evaluation of prediction models across seven datasets in the third scenario.	206

Table 6.15: Performance of AUC for prediction models across seven datasets in the fourth scenario.	206
Table 6.16: One-way ANOVA results for prediction models using AUC.	208
Table 6.17: AUC values for performance evaluation of RF with default and Mtry parameter tuning.....	209

List of Acronyms

Acronym	Definition
ANOVA	Analysis of Variance
APE	Average Probability Ensemble
AUC	Area Under Curve
C&K	Chidamber and Kemerer
FPR	False Positive Rate
FS	Feature Selection
IQR	Inter Quartile Range
KNN	K-Nearest Neighbours
L&H	Li and Henry
MAE	Mean Absolute Error
MAR	Mean Absolute Residual
MI	Maintainability Index
MLP	Multilayer Perceptron
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
NB	Naive Bayes
OO	Object-Oriented
PROMISE	PRedictOr Models In Software Engineering
QA	Quality Assessment
QUES	QUality Evaluation System
RF	Random Forest
RQs	Research Questions
RT	Regression Tree
SA	Standardised Accuracy
SLR	Systematic Literature Review
SMOTE	Synthetic Minority Over-sampling Technique
SQA	Software Quality Assurance
Stdev	Standard deviation
SVM	Support Vector Machine
SVR	Support Vector Regression
TPR	True Positive Rate
UIMS	User Interface Management System
WEKA	Waikato Environment for Knowledge Analysis

Chapter 1. Introduction

Software quality assurance (SQA) is defined as a group of activities that guarantee that a software meets a certain quality level [6]. Quality is mainly affected by its attributes, which are divided into two groups: external and internal attributes. Internal attributes, such as class cohesion, are directly measured from the software, whereas external attributes, such as maintainability, need to be measured indirectly, and their prediction often relies on internal attributes [7]. Software maintainability is an essential attribute in evaluating SQA, and it is defined as the simplicity to make the modification of a software system in order to upgrade the performance, adapt to changes in the environment or to edit faults [8]. This definition tries to capture how easy it was for the developer to make a change in the software product. However, the information about the ease of change is a challenging task to get unless this information was observed by engineers during the maintenance process, such as time, effort, number of modules investigated. Therefore, proxy measures that compute the number of changes made in the classes (i.e., change maintenance effort measure) or whether or not a change has been made in the classes (i.e., change-proneness measure) are typically used to resolve this issue. These measurements (i.e., dependent variable) are considered a perfect predictor of software maintainability and have a powerful relationship with other metrics (i.e., independent variables) that capture the concept of maintainability [9]. Studies have acknowledged various types of software maintenance measurements: change maintenance effort [7, 9-18], change-proneness [16], adaptive maintenance effort, which calculates the effort spent on each phase of the adaptive maintenance process [19], corrective maintenance effort, which calculates the effort spent on each phase of the corrective maintenance process [20], maintainability index (MI), which is a single value of a composite metric that computes a function from of four metrics: cyclomatic complexity, percentage line of comments, Halsted volume and lines of code [21], and maintenance time, which calculates the time to implement the maintenance tasks [22].

This thesis has adopted change maintenance effort and change-proneness measures to predict software maintainability. Change maintenance effort is a well-known software maintainability measure that calculates the number of modifications made per class during the maintenance period [7, 9-18]. A larger number of changes requires higher maintenance effort, which implies a lower level of maintainability. Change-proneness is another software maintainability measure [16, 23], which is a dependent variable to indicate that changes (e.g., inserting, removing or editing) have been made in a given class. This dependent variable is a Boolean value that includes two values: TRUE if the change was made on the class or FALSE if the change was not made on the class, regardless of the type and number of changes [5]. A lower number of TRUE values or a lower number of change-proneness refers to better maintainability, that is, it requires low maintenance effort.

Maintenance is defined as “the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to an environment” [8]. Software maintenance consumes the largest amount of cost, time and effort during the software development life cycle [9]. Jones reported that maintenance consumes approximately 75% of the total project cost, and the cost of maintaining the source code is ten times higher than the cost of developing it [24].

Controlling costs, time and effort are the significant components for ensuring SQA. These are performed by determining appropriate measures: as T. DeMarco [25] stated, “you cannot control what you cannot measure”. Software metrics are quantitative measures that can be employed to evaluate the quality of the software. In particular, object-oriented (OO) metrics are used to measure aspects of the source code of software systems (e.g., cohesion, size and inheritance depth). Consequently, several studies have employed a variety of OO metrics to evaluate the concept of software maintainability, such as Chidamber and Kemerer (C&K) [26] and Li and Henry (L&H) metrics [9]. For example, the L&H metrics can be utilised as predictors of software maintenance effort, as they have exhibited a strong relationship with the number of changes in the source code [9-11, 13].

OO systems are structured around objects and classes that have different characteristics (i.e., encapsulation, coupling and inheritance). These systems are written in various programming languages such as Java, C++ and C#. Several OO systems are available in open-source projects (e.g., GitHub [27] or SourceForge [28]) and are commonly used by various

organisations. With the growing use of OO systems, organisations need to further develop and change systems, which in turn leads to an increase in their complexity [29], and consequently concerns regarding their effective maintenance.

Prediction is a core part of estimation, which is a crucial aspect of project planning [30] and involves the determination of a number of factors including duration, staff, size, costs and effort [20]. Prediction mainly depends on historical internal and external quality attributes from completed projects. The correlation between internal attributes, "independent variables", and external attributes, "dependent variables", is a recognised challenge for software maintainability prediction [7]. Considerable attention has been given to predicting software maintainability using machine learning techniques. Accurate predictions are increasingly important in software project management tasks: allocating developers, identifying resources, supporting decision-making, evaluating costs across different projects and performing maintenance processes [20]. This prediction can assist in gaining insights on likely future maintenance, and can help decrease the total cost and overall effort of the software project [31]. However, building an accurate prediction model is difficult to achieve. Several empirical studies have been performed to investigate various types of individual machine learning models, including neural networks [10], Bayesian networks [11], linear regression [32], multiple additive regression trees [13] and K-means clustering [33]. However, the prediction accuracy of these individual models is disappointing and does not meet the criteria suggested by MacDonnell and Kitchenham et al. [34, 35]. These criteria include the mean magnitude of relative error (MMRE) and $\text{Pred}(q)$, which is the proportion of instances in the dataset in which the MRE is less than or equal to a defined value (q) [35]. These measures should be $\text{Pred}(.30) \geq 0.70$ or $\text{Pred}(.25) \geq 0.75$ or/and $\text{MMRE} \leq 0.25$. The explanation and equations of these measures are provided in Section 3.5.1.

To improve the accuracy of individual models, an ensemble machine learning model is introduced. This ensemble model is created from individual models in a heterogeneous (integrating various types of individual models) or homogenous (integrating the same type of individual models) manner. One of the main advantages of the ensemble models is to reduce the prediction variance that is a common factor in machine learning models [36]. Ensemble models aim to control the variance factor by integrating several individual models and

producing high prediction accuracy. This integration helps ensemble models to capture the advantages of multiple individual models.

Researchers and software practitioners have realised the benefits of ensemble models and applied them in various areas of software engineering problems. Several studies have explored the application of ensemble models in fault prediction problems, such as Random Forest (RF) [37-39], voting feature intervals [40], combined defect predictor [41], bagging [42], stacking [43], and adaptive selection of classifiers [44], and have recorded high accuracy. More specifically, the study by Mısırlı et al. indicated that the ensemble models provide a significant improvement in terms of locating software defects [40]. In addition, an empirical study of cross-project defect prediction was employed combined defect predictor on 10 open-source software systems. The findings of the study evidenced that the combined defect predictor outperformed other individual models [41]. The study by Zhang et al. also investigated cross-project defect prediction using various ensemble models and revealed that bagging ensemble model achieved high accuracy [42]. The paper by Petri et al. used stacking ensembles to predict software defect and found that this model attained good performance [43]. One of the reasons for the success of stacking over other models is that stacking combines several types of individual models. Moreover, adaptive selection of classifier was proposed and compared with five individual models in the context of predicting defect proneness [44]. The results obtained from this study indicated that the proposed ensemble model performed better than the other five individual models. Previous studies have also stated the success of using ensemble models to predict effort estimation using techniques, for example: bagging [45] and ensembles of linear methods [46]. Among several ensemble models implemented in studies of software engineering field, a number of these studies emphasised that RF outperforms other prediction models and produced a high improvement in the prediction accuracy [37, 38, 47, 48]. Ensemble models have also been used to predict software maintainability and provided high prediction accuracy (e.g., bagging [16], majority voting [23] and RF [48]). Based on the findings obtained in this discussion, the ensemble models yield improved the prediction accuracy over individual models, and have been shown to be useful models in several studies of software engineering field. However, the use of ensemble models for predicting software maintainability is relatively limited compared to the use of

these models in other software engineering field. These findings motivate us to use ensemble models (and RF, bagging and stacking in particular) in this thesis.

The imbalanced class problem exists when the quantity of one class in the dataset (True) is far lower than the quantity of another class in the same dataset (False). This problem causes machine learning models to bias their predictions towards majority classes and ignore minority classes. As a result, machine learning models gain high prediction accuracy based on the majority class instead of both classes. To resolve this issue, sampling techniques have been used in various fields including telecommunications management [49], emerging patterns [50], medical diagnosis [51], and text categorisation [52]. Moreover, the research community in software engineering disciplines has made significant efforts to address this issue using various sampling techniques. For instance, the synthetic minority over-sampling technique (SMOTE) in refactoring prediction [53], random under-sampling in defect prediction [54] and condensed nearer neighbour in maintainability prediction [55].

In this thesis, the application of ensemble models for predicting software maintainability is investigated and recommendations for the proper use of these models are provided. This investigation is performed by using various sizes of datasets and several base models, along with an exploration of the impact of parameter tuning. Therefore, a systematic literature review (SLR) for software maintainability prediction using machine learning techniques is conducted in Chapter 2 to determine maintainability measurements, metrics, datasets, evaluation measures and prediction models in the current studies, along with their gaps. Additionally, three empirical studies are performed in Chapters 4, 5 and 6 to construct a range of advanced machine learning models (i.e., homogenous and heterogeneous ensemble models) from existing individual models to predict software maintainability at the class level. These models employed a number of representative datasets for software maintainability either from open source datasets hosted in PRedictOr Models In Software Engineering (PROMISE) [56] (for example) or dataset extraction from existing open-source projects proposed in repository projects (e.g., GitHub [27] or SourceForge [28]).

1.1. Problem Statement

Several research attempts have been made to construct different machine learning techniques for predicting software maintainability. However, creating an accurate model to predict

software maintainability is a challenging task to accomplish, and to date, there is no evidence in the current literature on which models are appropriate. Therefore, the goal of this thesis is to employ ensemble models, which might resolve the problem, and improve their prediction accuracy over individual models. Another goal is to apply these models to a wide variety of software maintainability datasets to validate the results.

1.2. Motivation for the Thesis

Early prediction of software maintenance using machine learning models is important in helping software project managers to control resources, establish the maintenance process, improve design or coding, evaluate productivity, and compare costs across different projects [20]. In addition, it helps to achieve SQA, decreasing the failure and future maintenance effort of the software project [11]. Furthermore, prediction software maintainability enables the support of decision-making, by scheduling future maintenance operations, and the selection of developers, assigning more experienced developers to classes with high maintenance requirements. To date, there are limited studies to determine an accurate and suitable model to predict software maintainability. Therefore, the motivation of this thesis is to empirically compare and evaluate the application of ensemble models to obtain more consistent and accurate prediction results by reducing variance.

1.3. Contributions

The investigation of software maintainability prediction using ensemble techniques provides a number of contributions. First, it enables the empirical exploration of the positive impact of ensemble models (heterogeneous and homogeneous) by using different types of base models, and assesses the extent to which these ensemble models provide an improvement in the prediction accuracy over individual models in the context of software maintainability. Second, it enables the comparison between the proposed ensemble models with previous studies conducted on the most popular software maintainability datasets to determine whether these models achieve higher accuracy than that obtained in previous studies. Third, it enables the critical validation of the proposed ensemble models by applying these models to several sizes of datasets of software maintainability extracted from open-source software projects or

gathered from public repositories. Fourth, it enables the investigation of the impact of parameter tuning on ensemble models.

1.4. Research Objectives

The fundamental objective of this thesis is to provide the ability to accurately predict software maintainability to software project managers. This objective is achieved by applying ensemble techniques on datasets with different sizes and using various base models, along with exploring the impact of parameter tuning. More specific objectives are associated with the following three empirical studies:

1.4.1 Objectives of the first empirical study

- Investigate the capability of ensemble models to predict change maintenance effort using well-established datasets [9];
- Identify the model that achieves the highest accuracy prediction and compare with the best model in selected studies that operated on the same datasets;
- Explore the impact of parameter tuning of software maintainability prediction models using the caret package in R.

1.4.2 Objectives of the second empirical study

- Study the ability of ensemble models to predict change maintenance effort accurately using more recent and larger datasets [57];
- Compare and evaluate the proposed models with the selected models using the Auto-Waikato environment for knowledge analysis (WEKA) tool.

1.4.3 Objectives of the third empirical study

- Explore the influence of the ensemble model, feature selection (FS), and sampling techniques in predicting change-proneness using newest and largest datasets [58];
- Evaluate the effect of the tuning Mtry parameter, which is the number of variables randomly sampled for splitting in RF using a grid search.

1.4.4 Research questions

The previous objectives can be achieved by providing an appropriate answer for the following research questions (RQs):

RQ1) How effective are individual models at predicting software maintainability?

RQ2) How do ensemble models perform in the context of predicting change maintenance efforts using well-established datasets when compared to the individual models?

RQ3) How do ensemble models perform in the context of predicting change maintenance efforts using more recent and larger datasets when compared to the individual models?

RQ4) How do ensemble models perform in the context of predicting change-proneness using the newest and largest datasets when compared to the individual models?

1.5. Scope of Work

The scope of the work includes five primary aspects: maintainability measurements, metrics, datasets, evaluation measures and prediction models. Their basic concepts and specific utilisation are described, as well as an overview of related studies.

1.5.1 Maintainability measurement

Maintainability is a dependent variable that may be determined by a wide variety of independent variables. The ISO/IEC 25010 standard [59] defined a software quality model as a collection of attributes that include efficiency, usability, suitability, compatibility, security, reliability, portability and maintainability. Therefore, maintainability is an essential attribute of software quality and is recognised as one of the most challenging measurements due to the problem of predicting activity in the future [60]. Software maintainability is described as the ability of a software system to be easily modified to develop, correct, adapt to changes in the environment, or meet particular requirements [8]. This description indicates that software maintainability relies on various aspects of software modification (i.e., adaptation, correction, improvement or prevention). Furthermore, the ISO/IEC 25010 standard categorizes software maintainability into five major sub-characteristics: reusability, to identify the level of the assets that can be used to construct other systems; modularity, to identify the level of component independence and the extent to which changes to one component impact on the rest of the

system; analysability, to identify the ease to analyse and investigate (for example) the consequence of changes or diagnose problems in the software; testability, to identify the degree to which test criteria for a system can be established and tests to meet the criteria developed; and modifiability, to identify the degree to which it is possible to modify the software product without degrading its quality. This thesis predicts software maintainability using this definition : “the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment” [8]. Change maintenance efforts measure is used in the first and second empirical studies, and change-proneness measures is performed in the third empirical study.

1.5.2 Metrics

Metrics are independent variables that capture the element of software maintainability. Most of these metrics focus on the quality features of a class and measure specific parts of software products in these systems, such as inheritance, cohesion and data abstraction. Prior software maintainability studies utilised a wide variety of OO metrics: C&K [26], Oman and Hagemeister [61], L&H [9], Coleman et al. [62], Welker and Oman [63], Genero and Piattini et al. [64], Misra [21], and Yuming and Baowen [32]. These studies confirmed a relationship between OO metrics and software maintainability. However, this relationship is considered nonlinear, complicated, and of low accuracy [10]. Several of these metrics have been validated only in a small number of studies, whereas some have been proposed but never used.

OO metrics, which primarily include the C&K [26] and L&H metrics [9], have been widely used owing to their strong relationship with software maintainability [10, 11, 13, 14, 23]. C&K includes six OO metrics: DIT, WMC, NOC, CBO, RFC and LCOM (abbreviations and detailed definition of these metrics are provided in Table 2.6 in Chapter 2). L&H metrics [9] involve all C&K metrics except CBO and also include further metrics: MPC, DAC, NOM, SIZE1, SIZE2 (abbreviations, and detailed definitions of these metrics are presented in Table 2.6 in Chapter 2). This thesis employs L&H metrics [9] in the first empirical study, C&K and other OO metrics in the second empirical study [57] and several OO metrics in the third empirical studies [58], where these metrics were extracted from open-source systems and can be used to capture the element of software maintainability.

1.5.3 Datasets

Datasets comprise several metrics that include independent and dependent variables. The datasets are divided into a testing set to evaluate a prediction model and a training set to construct the model [65]. Alternatively, N-fold cross-validation is used to compare and evaluate prediction models by equally dividing the dataset into ten folds. One of these folds is used to test the model, and the remainder is used to train the model, and this process is iterated N times with different folds [66]. Moreover, the datasets include three major types: a public dataset, which is available to use (i.e., Quality Evaluation System (QUES) and User Interface Management System (UIMS) [9]); a partial dataset, which is extracted from available open-source software but unavailable for public use, as the researcher does not provide the dataset to the public (i.e., datasets extracted from 148 open-source systems [32]); and a private dataset, which is extracted from a private system and unavailable to use (i.e., datasets extracted from private projects [20]). This thesis initially uses the QUES and UIMS datasets, which have been widely used for predicting software maintainability (i.e., CHANGE metric proposed in [9], making the results comparable and repeatable [9]. In addition, more recent, larger, and public datasets, namely bug prediction datasets, are investigated to validate and support the previous result [57]. Finally, this thesis uses the newest datasets, namely refactoring datasets, published in 2018 [58].

1.5.4 Evaluation measures

The evaluation of prediction models is a vital part of any machine learning problem to compare performance between several models and measure the accuracy of the model in predicting software maintainability. Several evaluation measurements have been proposed in the literature to assess prediction models in software engineering problems [67]. Usually, regression problems use residuals or prediction errors [68], whereas classification problems utilise confusion matrices [69]. Some of the most frequently used evaluation measurements have become de facto standards to measure the prediction accuracy of regression problems, namely MMRE, Pred(q) [35], mean absolute error (MAE), and standardised accuracy (SA). The formulas and issues related to these measures are provided in Section 3.5.1. Additionally, the area under curve (AUC), considered the most frequently used evaluation measure for the classification problem in software quality prediction, is performed to compare and evaluate

the performance of prediction models for the classification problem [4]. AUC is dependent on the receiver operating characteristic (ROC) curve that plots the false positive rate (FPR) on the x-axis against the true positive rate (TPR) on the y-axis at different threshold settings [70]. Furthermore, baseline measurements can be used to evaluate the performance of the predictors with the dependent variable (e.g., sample mean [71] or sample median [72]). MMRE, Pred(q), MAE, SA, AUC and baseline measures are used in this thesis to evaluate and compare OO software maintainability prediction models.

1.5.5 Prediction models

Several types of individual machine learning models, such as neural networks [10], Bayesian networks [11], linear regression [32], multiple additive regression trees [13], K-means clustering [33], SVR [73] and MLP [14] have been built to predict software maintainability. However, despite the large number of studies and models created, only a limited number of these achieved a reasonable level of predictive accuracy, but failed to meet the criteria of an accurate prediction model, which is $\text{Pred}(.30) \geq 0.70$ [34] or $\text{Pred}(.25) \geq 0.75$ or/and $\text{MMRE} \leq 0.25$ [67]. Furthermore, determining the best technique among individual models is difficult because the performance of these techniques relies on the dataset used. Therefore, it is clear that there is a considerable need to advance the performance of the individual models, which can be achieved by building ensemble models. As mentioned previously in Section 1, ensemble models have been successfully utilised in various areas of software engineering to decrease variance, which leads to improved prediction accuracy. The ensemble model may be heterogeneous, merging various types of individual models (i.e., software maintainability evaluation model based on multiple classifiers combination [74]), or homogenous, merging the same types of individual models (i.e., weighted voting, majority voting and hard instance [23]). This thesis evaluates and compares the application of bagging, additive regression and RF as examples of homogenous ensemble models, and stacking and Average Probability Ensemble (APE) as examples of heterogeneous ensemble models. Additionally, a range of individual models are used to predict software maintainability: regression tree (RT), multilayer perceptron (MLP), m5rules, k-nearest neighbours (KNN), support vector regression (SVR), naive Bayes (NB) and support vector machine (SVM). The description of the individual and ensemble models is presented in Sections 3.1 and 3.2, respectively.

1.6. Thesis Organisation

The remainder of this thesis is organised as follows:

Chapter 2 provides an SLR of machine learning models for software maintainability prediction.

Chapter 3 describes the research methodology and experimental design for the three empirical studies.

Chapter 4 presents the first empirical study, which employed various ensemble techniques on QUES and UIMS datasets to predict OO software maintainability (CHANGE metric). In addition, the impact of tuning parameters using the caret package is explored.

Chapter 5 reports the second empirical study, which used more recent and larger datasets, namely the bug prediction datasets and applied data pre-processing techniques on these datasets to improve their quality and to be suitable for software maintainability prediction. Furthermore, Auto-WEKA tools are used to determine the best prediction models.

Chapter 6 explores the effectiveness of the ensemble model, FS and sampling techniques in predicting change-proneness. The machine learning models in this chapter are applied to seven datasets recently extracted from open-source software systems, namely refactoring datasets. Additionally, the impact of tuning the Mtry parameter, which is the number of variables randomly sampled for splitting in an RF, is investigated in this chapter.

Chapter 7 concludes this thesis and summarises the main contributions, research limitations and some directions for future work.

Chapter 2. A Systematic Literature Review of Machine Learning Models for Software Maintainability Prediction

This chapter reports on a SLR of relevant journals and conference proceedings papers that focus on the topic of software maintainability prediction. This review investigates a set of RQs to comprehensively summarize, analyse and discuss various viewpoints: software maintainability measurements, metrics, datasets, evaluation measures, individual models and ensemble models. The search in this review was focused on the most common computer science digital database libraries between 1991 until 2018 and 56 relevant studies were surveyed in 35 journals and 21 conferences proceedings.

2.1. Introduction

The primary objective of this review is to investigate the current state of software maintainability prediction to discover the progress made, limitations and challenges, along with future research requirements. To the best of my knowledge, this is the first SLR of software maintainability prediction for OO systems that comprehensively evaluates a wide variety of important journal and conference proceedings with respect to specific defined research question. This review differs from the previous review studies [31, 75-79] because it includes a higher number of relevant journal and conference proceedings in the software maintainability field. Furthermore, a different analysis on the selected primary studies was applied and more additional detailed analysis of each paper was provided. Previous review studies focused on non-OO systems [77-79], or considered only fifteen studies [31], or concentrated on a single aspect such as the measurement of software maintainability, the models employed or the metrics implemented [75, 76]. In contrast, this study is classified the concept of software maintainability according to three dimensions: the measurement of maintainability (dependent variable), the metrics considered (independent variables) and the models employed. This study has applied the research method for conducting a SLR proposed by Kitchenham [80], and has analysed comprehensively each selected study. Therefore, I have

confidence my SLR is both novel and hope that software engineering researchers and practitioners will find it to be a useful resource. The key contributions of this Chapter are:

- This is the first SLR in the field of software maintainability prediction using machine learning techniques;
- This SLR is more extensive than previous review studies in software maintainability prediction and investigates a broader range of RQs that cover various viewpoints: software maintainability measurements, metrics, datasets, evaluation measures, individual models and ensemble models;
- Although a number of studies have used machine learning techniques, few have explored ensemble models and only one of the investigated models met the model accuracy criteria. So, there is scope for further research in the software maintainability prediction field.

2.2. Method

The SLR is a commonly and widely applied method in the software engineering field [80]. The review presented here aims to identify, evaluate and interpret all available research relevant to predicting software maintainability using machine learning models. This SLR is based on the quality reporting guidelines as proposed by Kitchenham for performing a SLR in software engineering [80], and also takes on board subsequent lessons learned and advice [81]. Three primary stages are established and adjusted to include appropriate steps, namely, planning, conducting and reporting the review. The planning stage involves the following steps: determining the needs for a systematic review, which was discussed in the introduction; evolving an appropriate review protocol to eliminate the possibility of researcher bias. The conducting stage involves the following steps: formulating RQs to focus on the central issues of this review; developing the search process to conduct search activities; identifying selection criteria to select appropriate studies; examining quality assessment (QA) to evaluate selected studies in terms of quality; applying data extraction to document the information obtained from the studies; performing data synthesis to accumulate and summarise the results. The final reporting stage involves only one step: presenting results and discussions to answer each research question. This process is illustrated in Figure 2.1.

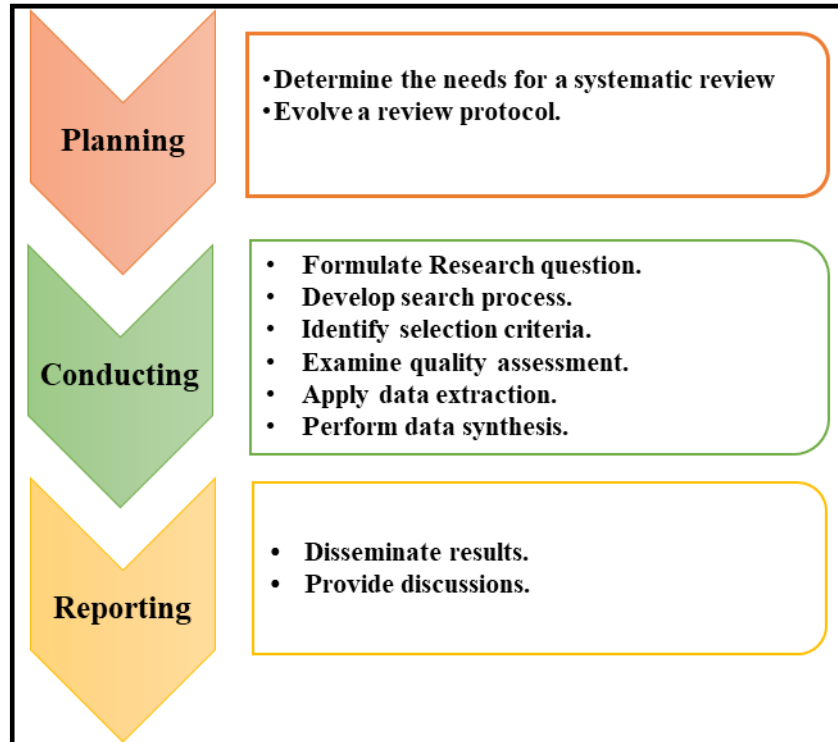


Figure 2.1: The framework of SLR.

2.2.1 Review protocol

The review protocol aims to direct the implementation of the review and minimise the possibility of researcher bias. The critical elements of this review protocol include RQs, the search process, inclusion and exclusion criteria, QA, data extraction and finally data synthesis. Furthermore, the review protocol was iteratively developed and evaluated during the conducting and reporting stages. Details of the protocol are explained in the following sections (2.2.2-2.2.7).

2.2.2 Research questions for SLR

The RQs were introduced to specify the research boundaries. They were formulated with the assistance of the (PICOC) criteria [80] which recognize RQs from four viewpoints as follows:

- **Population:** OO system, software system, application software, software project.
- **Intervention:** Software maintainability prediction, predicting software maintenance effort, change proneness, techniques, methods, models, process and product metrics, dataset.

- **Comparison:** N/A.
- **Outcomes:** Accuracy prediction of software maintainability, building good prediction models.
- **Context:** empirical or experimental studies in academia and industry, large and small size of the datasets.

The primary objective of this SLR is to collect and analyse appropriate evidence to answer the RQs for SLR. The motivation of this review is to answer a set of seven RQs to obtain insights into significant aspects of the research direction, including advancing my knowledge of software maintainability prediction for OO systems and identifying the limitations of research so as to define further research directions. The RQs and their motivation are documented in Table 2.1 below.

Table 2.1: Research questions for SLR and their motivation.

ID	Research question	Main motivation
RQ1	What are the definitions of software maintainability?	Identify different software maintainability definitions.
RQ1.1	How can the software maintainability be measured (dependent variable)?	Recognize the software maintainability measurements.
RQ2	What type of OO metrics have been proposed to measure software maintainability?	Identify OO proposed metrics that are commonly being used in software maintainability.
RQ2.1	What are the metrics used (independent variable)?	Determine various OO metrics.
RQ2.2	What is the level (e.g., class level, method level) of these metrics?	Identify the level of OO metrics.
RQ3	What software maintainability datasets in the literature have been used to build prediction models?	Determine various datasets commonly being used in the the software maintainability domain.
RQ3.1	What are the types of software maintainability datasets (e.g., public datasets, private datasets)?	Recognize the type of these datasets.
RQ3.2	What tools are used to extract metrics from open source projects?	Identify different tools to extract OO metrics.
RQ3.3	What software programming languages are used to write system code?	Determine various software programming languages commonly being used to collect OO metrics.
RQ3.4	What are the number of classes in the software system?	Identify the number of classes in the software system
RQ4	What are the evaluation measures used to assess the performance of software maintainability prediction models?	Explore evaluation measures commonly being used in each software maintainability datasets.
RQ4.1	What approach (e.g., cross-validation, holdout) is used to evaluate the performance of software maintainability prediction models?	Identify different validation approaches applied on software maintainability prediction models.
RQ5	What type of machine learning problem (e.g., classification and regression) software maintainability fall into?	Identify the type of machine learning problem.
RQ5.1	What are the categories of machine learning problem (e.g., supervised, unsupervised and semi-supervised)?	Determine various categories of machine learning problem.
RQ6	What are the individual prediction models (e.g., neural network, linear regression) used to predict software maintainability?	Investigate the individual prediction models commonly being used in software maintainability.
RQ6.1	What are the best performing individual prediction models?	Identify the best performing individual prediction models in each study.
RQ7	What ensemble prediction models (e.g., bagging, boosting) are used to predict software maintainability?	Investigate the ensemble prediction models commonly being used in software maintainability.
RQ7.1	What type of ensemble prediction models were performed to predict software maintainability?	Determine different type of ensemble prediction models.
RQ7.2	Do the ensemble models outperform the individual prediction models?	Investigate whether ensemble models represent an improvement over the performance over the individual prediction models.

2.2.3 Search process

The search process must be focused on a way that allows the identified RQs to be accurately investigated and includes four steps: choosing the digital libraries, identifying additional search sources, selecting the interval time of the published articles, and defining search keywords. The search was applied on five sources of the most popular and largest computer science online digital libraries that publish peer-reviewed articles:

- IEEE Xplore (ieeexplore.ieee.org)
- ACM Digital Library (dl.acm.org)
- Springer (springerlink.com)
- Elsevier (sciencedirect.com)
- Wiley online library (onlinelibrary.wiley.com)

Furthermore, manual research was applied to include relevant journal and conference proceedings in the software maintainability field. These journals and conferences were selected particularly since they involve empirical studies or literature reviews, and they are well-established and highly relevant software engineering publications. The selected journals and conferences are presented in Table 2.2 and the information of this table is collected from Web of Science (mjl.clarivate.com). The search was limited to articles published in the interval from 1991 to 2018. The search was restricted in this time interval since machine learning started to be applied to problems of this nature in the 1990s [82] and investigations into software maintenance began in earnest in 1985 [31]. Furthermore, research into software maintainability expanded dramatically with the usage of OO systems in 1990s [83] and no studies relevant to the identified RQs were found to exist before these dates.

Table 2.2: Selected journals and conferences.

Source	Acronym	Number of Studies	Published by	Impact factor on 5 years	Quarter category
IEEE Transactions on Software Engineering	TSE	8	IEEE	3.92	Q 1
Empirical Software Engineering	EMSE	20	Springer	3.49	Q 1
Information and Software Technology	IST	11	Elsevier	2.76	Q 1
Journal of Systems and Software	JSS	14	Elsevier	2.40	Q 1
IEEE Software	IEEE SW	3	IEEE	2.50	Q 1
Soft Computing	SC	5	Elsevier	2.20	Q 2
Software Quality Journal	SQJ	6	Springer	1.90	Q 2
Journal of Software Maintenance and Evolution: Research and Practice	JSME	6	Wiley	1.21	Q 2
IET Software	IST	2	IEEE	0.97	Q 3
International Journal of System Assurance Engineering and Management	IJSAEM	4	Springer	0.94	Q3
ACM SIGSOFT Software Engineering	ASSE	3	ACM	0.45	Q4
Conferences				H-index	
International Conference on Software Maintenance and Evolution	ICSME	4	IEEE	29	
International Conference on Software Engineering	ICSE	7	IEEE	68	
International Conference on Predictive Models and Data Analytics in Software Engineering	PROMISE	1	ACM	NA	

A list of search strings was created by integrating appropriate synonyms and alternative terms with the Boolean operator (AND has the effect of narrowing and limiting the search, while OR serves to broaden and expand the search) and the truncation symbol (*) which is used to identify words with a particular beginning (for example predict* will match with predict, prediction predicting and predicted) [84].

The following search terms were formulated in this SLR: (software maintainability OR maintenance effort) AND (predict* OR forecast* OR estimate*) AND (machine learning OR data mining OR artificial intelligence OR application OR Bayesian network OR neural network OR Regression OR support vector machine) AND (method OR model OR technique OR approach) AND (metric OR measure*).

The role of machine learning techniques has emerged as a recommended technique in several research fields, and these have often proven to be better than other techniques (e.g., human evaluation or statistical methods) [85]. The fact that worth mention is that some machine learning techniques based on statistical methods (e.g., NB) and they considered in this study. However, some studies were selected that do not use machine learning techniques because these studies answer some of my RQs. Nevertheless, this chapter focuses on a systematic summarisation of machine learning techniques used in software maintainability prediction and collects the empirical evidence from employing these techniques.

The endnote system was used to store and organize search results and a spreadsheet was used to create a table of data extracted from each selected paper. The initial search applied the search terms on each selected database as well as the journal and conference proceedings to include the full document. This procedure returned thousands of irrelevant studies, so it was decided to limit the search on the document title, abstract and content type (a conference or journal publication). Several duplicate papers were found in these databases which were subsequently removed.

Additional studies were determined by referring to the references of identified relevant studies. After collecting studies from the primary search, the relevant studies were selected by scanning the title and abstract. Further investigation was performed to determine appropriate studies by reading the full text. The candidate studies were selected if they meet the criteria in Section 2.2.4. Finally, the selected studies were identified after applying the QA criteria. The progress of the search process is presented in Figure 2.2 and shows the number of articles identified at each stage of the selection and filtering process. The steps were iterated until final agreement was reached. The SLR was completed at the end of July 2018 and 56 suitable studies were finally identified.

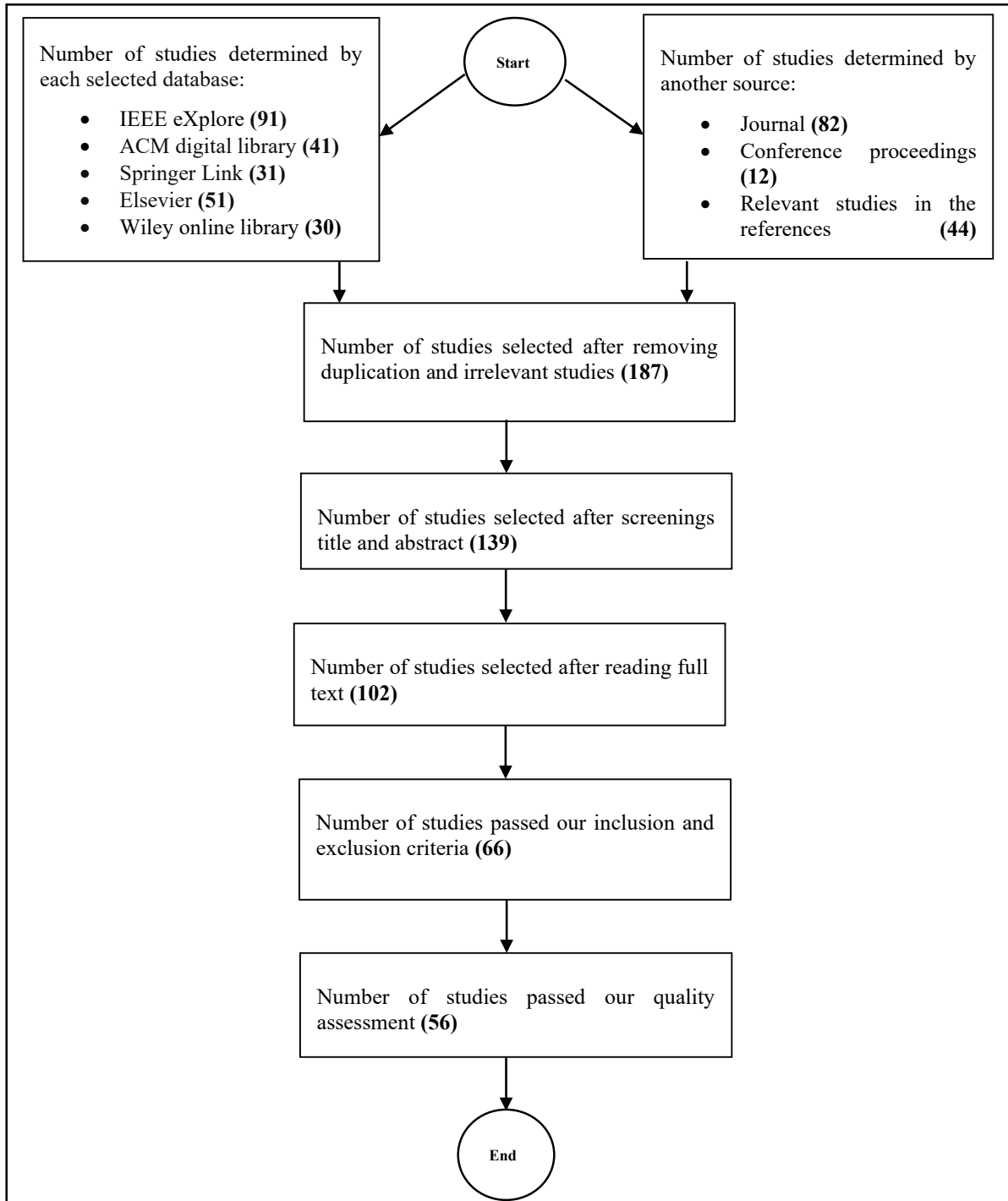


Figure 2.2: Process of primary studies selection.

2.2.4 Inclusion and exclusion criteria

The results from the previous steps yielded several irrelevant studies, so inclusion and exclusion criteria are defined in order to filter these out. The inclusion and exclusion criteria used in this SLR are outlined below:

1. Inclusion Criteria:

- Studies focus on software maintainability prediction and answer any of the RQs in this review.
- Studies are applied on OO systems.
- Studies consider machine learning techniques.
- Studies are written in the English language.
- Studies are published in either a journal or conference proceedings.
- Studies are peer reviewed articles.
- Studies report on comparisons between model predictions.
- Studies are the most recent and comprehensive (in the case where studies were repeated).

2. Exclusion Criteria

- Studies do not focus on software maintainability or answer the RQs in this review.
- Studies were applied on non-OO systems, such as service-oriented, aspect-oriented software, web applications or functional systems.
- Studies consider specific aspects of software maintainability, such as code smells, defects, or fault proneness. These studies were excluded because they did not predict any of software maintainability measurements (i.e., dependent variable).
- Studies are not written in English language.
- Studies do not include the full text.
- Studies fall outside the defined time frame i.e. from 1991 to 2018.
- Conference papers in the case where studies were published as both conference and journal versions.

Based on the criteria above, sixty-six studies were finally selected. Twenty-seven irrelevant studies were rejected that either did not answer the RQs or consider specific aspects of software maintainability. Five studies were rejected that focused on non-OO systems, and four conference papers [61, 86-88] were rejected because more recent journal versions of the work have been published.

2.2.5 Quality assessment

The QA stage is performed to evaluate each study identified in the previous step. The QA follows the defined quality checklist as proposed by Kitchenham [80]. The main objective of the QA is to evaluate studies and select studies that answer the RQs and to support more detailed analysis of inclusion and exclusion criteria. The QA questions are specified below:

- QA1: Does the study define a main research objective or problem?
- QA2: Does the study define software maintainability?
- QA3: Does the study determine the type of software maintainability measurement (dependent variable)?
- QA4: Does the study employ OO software metrics (independent variables)?
- QA5: Does the study indicate the source of the datasets?
- QA6: Does the study use a suitable tool for the extraction of the datasets?
- QA7: Does the study identify the programming language of the systems being analysed?
- QA8: Does the study identify the number of classes in software system?
- QA9: Does the study make the dataset publicly available?
- QA10: Does the study use appropriate evaluation measures?
- QA11: Does the study use suitable cross validation techniques?
- QA12: Does the study justify the prediction techniques?
- QA13: Does the study apply prediction models and identify the best performing model?
- QA14: Does the study present the results and findings clearly?
- QA15: Does the study identify research limitations or challenges?

The scoring procedure of the QA questions is constructed as a following:

- 1 represents Yes.
- 0.5 represents Partly.
- 0 represents No.

The scores rank the papers into four categories: excellent ($13.5 \leq \text{score} \leq 15$), good ($9.5 \leq \text{score} \leq 13$), fair ($5 \leq \text{score} \leq 9$), and fail ($0 \leq \text{score} \leq 4.5$). From applying the above QA criteria, ten studies fail in this QA. Finally, fifty-six primary studies were selected to conduct this SLR.

2.2.6 Data extraction

The data extraction step is performed to extract data from each selected primary study with the aim of collecting data that answers the RQs. Seven main properties were classified in Table 2.3 with respect to the RQs requirements.

Table 2.3: Data Extraction properties with their research question.

Properties	Research question
Software maintainability measurement	RQ1, RQ1.1
Software maintainability metrics.	RQ2, RQ2.1, RQ2.2
Software maintainability datasets.	RQ3, RQ3.1, RQ3.2, RQ3.3, RQ3.4
Software maintainability evaluation measures	RQ4, RQ 4.1
Machine learning problem to predict software maintainability.	RQ5, RQ5.1
Software maintainability individual models.	RQ6, RQ6.1
Software maintainability ensemble models.	RQ7, RQ7.1, RQ7.2

2.2.7 Data synthesis

Data synthesis is applied to extract both quantitative data and qualitative data that forms a body of evidence from the selected studies that address issues related to the RQs. The results are presented in the form of tables, pie charts, clustered bar charts, scatter charts and bar charts. These visualisations enable me to conduct a comparative analysis between selected studies and improve the quality of presentation.

2.3. Results

This section summarises the results obtained from selected primary studies and includes details about the search results, a visualisation of publication years and sources, and following on from this an overview of the QA outcomes.

2.3.1 Selected primary studies

In this SLR, fifty-six primary studies were selected to compare and evaluate the studies in the software maintainability prediction domain, and are summarised in Table A.1 in Appendix A. This table provides a brief description of each selected study and contains the following attributes: study ID; reference; the title of the publication; the authors of the articles (first author and co-author); the publication year; place published; publication name and publication type (journal or conference).

2.3.2 Publications years

The publication years of selected primary studies are between the year 1991 and 2018 and Figure 2.3 shows the numbers of studies published during these years. After 1993 L&H provided the QUES and UIMS datasets as an appendix to their paper, which motivated researchers to investigate prediction techniques on this dataset. Moreover, there is an indication of increased publications after 2005 when the PROMISE repository was launched [56], but in this year most of the datasets in the PROMISE repository were for software defect prediction and none for software maintainability. After 2005, the researchers may be recognised the importance and benefits of public datasets, which make the empirical study comparable and repeatable [9]. As a result, S55 was published relevant datasets of software maintainability prediction in the PROMISE repository and several studies (i.e., S25, S26, S30, S36, S38, S46, S47, S49 and S52) were used QUES and UIMS datasets, which published as an appendix in S2.

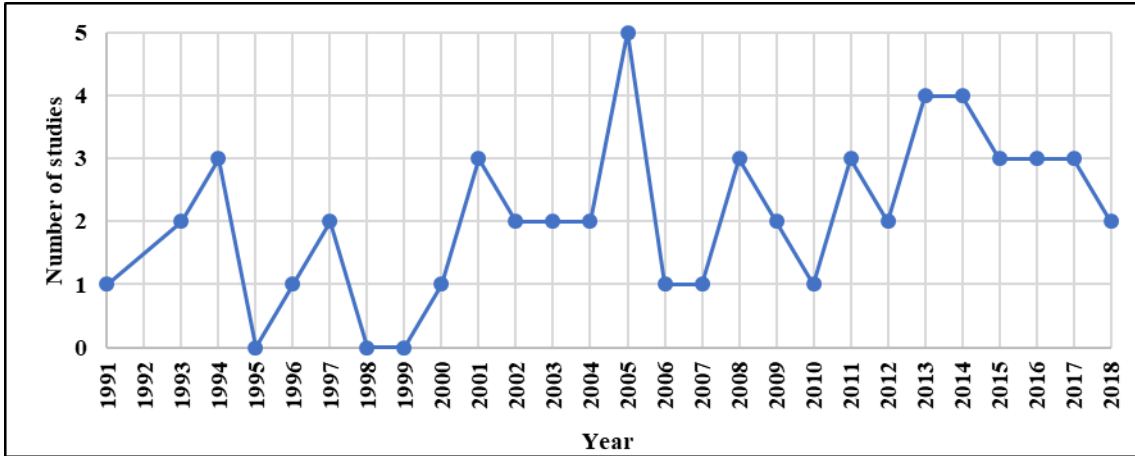


Figure 2.3: Number of selected studies over the years.

2.3.3 Publication sources

Of the 56 primary studies selected 35 appeared in journals and 21 in conferences. The most popular journal for papers associated with software maintainability prediction is the Journal of Systems and Software, followed by Information and Software Technology, then IEEE Transactions on Software Engineering. Figure 2.4 illustrates the number of selected primary studies in each journal.

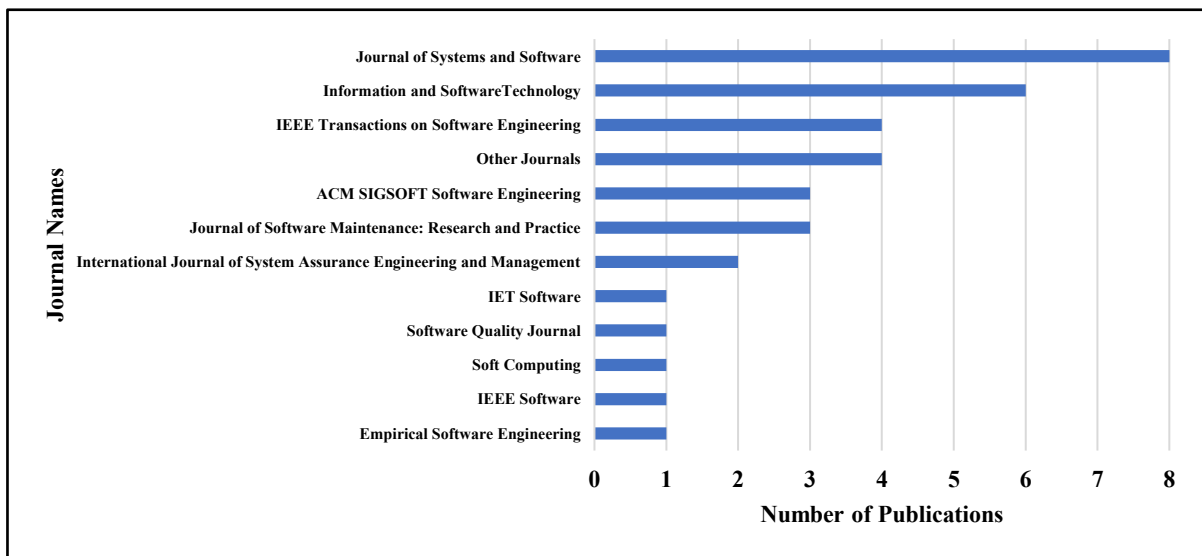


Figure 2.4: The number of studies in each journal.

Figure 2.5 shows the number of selected primary studies grouped by place of publication (i.e. digital library database). It can be seen that the most selected primary studies are chosen from the IEEE digital library, followed by Elsevier.

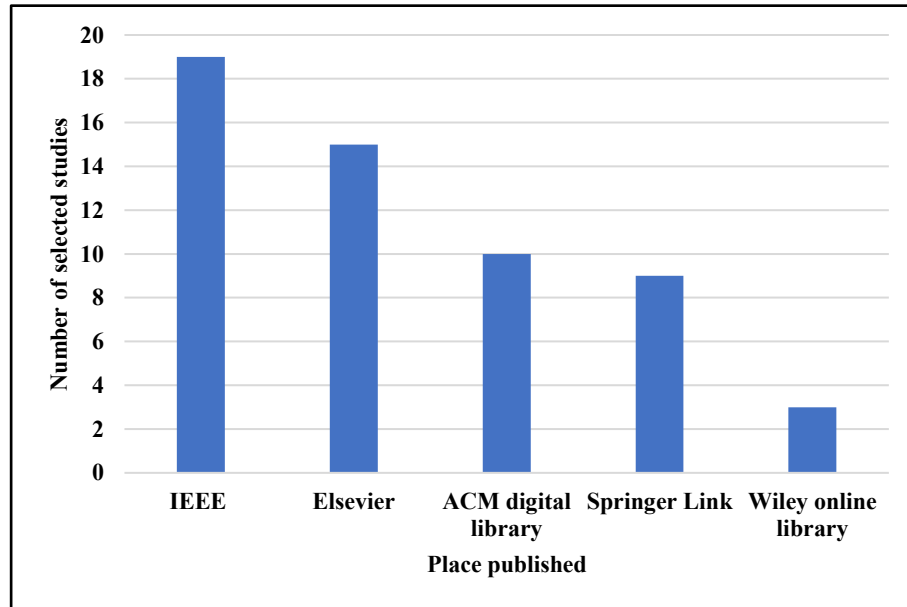


Figure 2.5: The number of selected studies in each digital library database.

2.3.4 Quality assessment result

The selected primary studies were evaluated according to the QA questions described in section 2.2.5 and present the results of this analysis in Table A.2 in Appendix A. This table shows that twelve of the selected studies obtained an excellent rating, thirty-two a good rating, followed by twelve which obtained a fair rating. However, ten studies were excluded that recorded a poor rating.

2.4. Discussion

This section aims to break down the analysis of the results based upon the RQs identified in Section 2.2.2. Each of the questions is considered in turn and the findings from all selected papers are considered to identify problems, solutions and to analyse the experimental results. Then, a comparison of the whole topic is conducted to comprehensively understand the topic and determine any limitations.

The first two questions consider software maintainability measurements (RQ1) and software maintainability metrics (RQ2) and the key differences between these are listed in Table 2.4 below.

Table 2.4: The differences between software maintainability measurements and metrics.

Software maintainability measurements	Software maintainability metrics
External attribute	Internal attribute
Dependent variable	Independent variable
Measured indirectly	Measured directly
Examines the software in an environment	Examines only the software
Difficult to extract	Easily extracted
Measured after or during execution	Measured without execution

2.4.1 Software maintainability measurement

In this section, the findings in relation to the following RQs are discussed:

RQ1: What are the definitions of software maintainability? and **RQ1.1:** How can the software maintainability be measured (dependent variable)?

As software maintainability is not something that can be measured directly, my motivation was to gain insight into the different software maintainability definitions and measures that are being employed.

A. Software maintainability definitions.

Software maintainability is defined in the IEEE Standard Glossary of Software Engineering Terminology [8] as “the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment”. This definition implies that software maintainability depends on various aspects of software modification (i.e., correction, improvement, adaptation or prevention). Moreover, maintainability is one substantial attribute proposed ([59]) of the set in the software quality model that involves: maintainability, efficiency, suitability, usability, reliability, security, portability and compatibility. However, several selected studies (e.g., S2, S16, S18) revealed that software maintainability is considered as one of the most challenging measurements of the software quality attributes, because there are several measurements and not all of them can be used to predict future activities. Some types of software maintainability (e.g., CHANGE metric, MI and change proneness) can be used as an indirect measure (i.e., dependent variable) based on an extensive variety of metrics (i.e., independent variables) and machine learning prediction models can be applied to make a prediction. On the other hand, other types of

software maintainability can be measured directly by observation during the maintenance process and record factors such as time, effort, or numbers of modules investigated. To explore these issues, different types of software maintenance measurements are presented and which type can be used as dependent variables to make a prediction and capture an element of maintainability.

B. Software maintainability measurement.

The key challenge with software maintainability measurement is that maintainability cannot be measured directly. Therefore, predictive models are based on indirect measures. Measuring software maintainability relies on a set of metrics that may be combined in a software maintainability function. The following Eq. (2.1) is a general software maintainability function that performs to collect metrics from A_1 to A_n [89]:

$$M = F(A_1, A_2, \dots, A_n) \quad (2.1)$$

The most obvious difficulty from the above function is to identify the appropriate metrics that can be measured from the source code directly. Various ways can be used to measure software maintainability, because there are different proxy definitions of maintainability. However, there is not any commonly agreed approach [89].

The selected primary studies for this SLR included nine types of software maintainability measurements (see Table 2.5). For each type of measurement, more details of the definition, a general form of the equation, value and interpretation of the value are presented. Among these software maintainability measurements shown in Table 2.5, maintainability is measured by counting the number of changes made in the code. L&H [9] define change metrics that capture the element of maintainability by counting the number of changes made in the source code, where the change could be insertion or deletion. This measurement is performed by several studies as shown in Table 2.5. Another measure of the software maintainability is based upon corrective maintenance. Lucia [20] calculated maintainability effort using four metrics that count the size and the number of tasks in the system under maintenance and are combined to produce the actual effort metric (dependent variable). Adaptive maintenance effort is used by three studies in Table 2.5 and is based on the effort (time) involved in adding, understanding and deleting in the source code in the system. The study by Oman and Hagemester [90] proposes the MI, a composite metric calculated from software complexity and quality metrics. Various selected primary studies have used this measurement as presented in Table 2.5. The

change proneness maintainability measure is employed by two selected primary studies [16, 23] as a part of their experiment, and takes the form of a Boolean variable to indicate if a change is made during maintenance, while the independent variables are the C&K metrics [26]. Three selected primary studies compute time effort to perform maintenance tasks, while only one selected primary study calculated the cost of maintenance tasks based on the time consumed to execute these tasks. The main goal of these studies was to determine maintenance time or cost directly rather than construct a new formula or compare the current system with other systems. Some studies describe software maintainability rather than providing a formula. This description may be classified depending on software maintainability attributes or components. Finally, four selected primary studies used other software maintainability measurements. The explanation of the metrics used in the maintainability equation in Table 2.5 appears in Section 2.4.2.

Table 2.5: Summary for software maintainability measurement.

Study ID	Type of maintenance	Maintainability definition	Maintainability equation	Measurement value	Maintainability interpretation
S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51, S52	Change maintenance effort	CHANGE, which is dependent metric, is computed according to the number of lines changed in a class where insertion or deletion are counted as 1, and modification of the contents is counted as 2	$CHANGE = F$ (WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2), the description of these metrics proposed in Table 2.6	Integer	Maintainability is interpreted as being inversely proportional to the number of changes made: a high value of change indicates low maintainability, while a low number represents high maintainability.
S18	Corrective maintenance	Corrective maintenance effort computes the effort spent on each phase of the corrective maintenance process (analyse, implement, produce, define, design).	Corrective maintenance effort = $b_1 NA + b_2 NB + b_3 NC + b_4 Size$	Time (person-hours)	A high number of corrective maintenance effort indicates low maintainability, while a low number of corrective maintenance effort represents high maintainability.
S12, S19, S22	Adaptive maintenance effort	Adaptive maintenance effort computes the effort spent on each phase of the adaptive maintenance process (adding, understanding, deleting, modification, change).	Maintainability Eff _{am} = Eff _{add} + Eff _{und} + Eff _{del} .	Time (hours or months)	A high number of adaptive maintenance effort indicates low maintainability, while a low number of adaptive maintenance effort represents high maintainability.
S5, S6, S8, S21, S28, S29, S33, S48, S54, S55	Maintenance evaluation by maintainability index	The maintainability index, which is dependent metric, is a single value of a composite metric that compute the function of four metrics: lines of code (lnLOC), cyclomatic complexity (CC), percentage line of comments (COM)	Maintainability Index = $171 - 5.2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LOC) + 50 \times \frac{\sin}{\sqrt{2.4 \times COM}}$	Decimal number between 0 and 1.	A number above 0.85 represents high maintainability, between 0.85 and 0.65 indicates medium maintainability, below 0.65 represents low maintainability

		and Halsted volume (HV).			
S47, S56	Maintenance evaluation by change proneness	Change proneness, which is dependent metric, is a Boolean variable indicating a change (addition, deletion or modification) has been made on a class.	Change Proneness = F (WMC, LCOM, CBO, DIT, RFC, NOC) IF (class change) Change proneness= TRUE ELSE Change proneness= FALSE	Boolean Variable TRUE or FALSE	Maintainability change proneness is TRUE if the change occurs in class or FALSE if the change does not occur.
S7, S17, S37	Maintenance time	Compute the time to perform maintenance tasks (including understanding, developing, and modifying the program)	NA	Time (hours)	The greater the amount of time spent, the lower the maintainability.
S9	Maintenance cost	Compute three types of the costs: testing (MMT), modifying (MMM) and understanding (MMU)	Maintenance Cost = $MM_T + MM_M + MM_U$	Time (person-hours)	Maintainability is directly related to maintenance cost.
S10, S11, S14, S15, S27, S31	Maintenance categorisation according to maintenance attributes	Categorise maintenance according to maintenance attributes: changeability, stability, analysability, maintainability and testability	Each study used a different equation	NA	NA
S1, S13, S16, S53	Maintenance categorisation according to maintenance components	Categorise maintenance according to maintenance components: corrective, adaptive and perfective	Each study used a different equation	NA	NA
S3, S24, S34, S40	Other measurements	NA	Each study used a different equation	NA	NA

It is apparent from Table 2.5 that there are several types of software maintenance measurements, these types can be divided into indirect measures and direct measures:

1. Indirect measures: As mentioned early, Software maintainability is defined as the ease with which a software system may be modified to correct faults, improve the performance, or adapt to changes in the environment [8]. This definition captures how easy it was for the developer to modify the software product. However, the information about the ease of change is a very difficult task to achieve and typically unavailable in the historical datasets. To resolve this issue, the proxy measures were used to calculate the number of changes made in the classes or whether or not a change has been made in the classes. There are three types that can be used as dependent variables to capture

the element of maintainability, which are the CHANGE metric (S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51 and S52), the MI (S5, S6, S8, S21, S28, S29, S33, S48, S54 and S55) and change proneness (S47 and S56). CHANGE metric is more about the numbers of change that is likely to be made to a class, MI is a single value of a composite metric, change proneness is a Boolean variable indicating a change (addition, deletion or modification) has been made on a class, whereas maintainability refers to the ease with which maintenance changes can be made and implemented. These measurements have proven to be a good indicator in predicting software maintainability in several selected primary studies in Table 2.5 and have strong relationships with other metrics (independent variables) [9]. Therefore, these measurements can be used as dependent variables to predict software maintainability.

2. Direct measures: the remaining types of software maintenance measurements are considered direct measures that are measured software during the maintenance process. These types include corrective maintenance in S18, adaptive maintenance effort in S12, S19 and S22, maintenance time in S7, S17 and S37 and maintenance cost in S9.

Moreover, Table 2.5 illustrates that relatively few software maintainability measurements are present in the current literature, and this finding is directly in line with previous studies [91].

Figure 2.6 illustrates the number of selected studies employing the different maintainability measures. From this figure, it can be seen that by far the most popular software maintainability measurement is change maintenance effort that used by twenty selected primary studies which may be attributable to the availability of QUES and UIMS datasets that use this measurement as the dependent variable. MI is recognised as the second most common measurement, being used by ten studies.

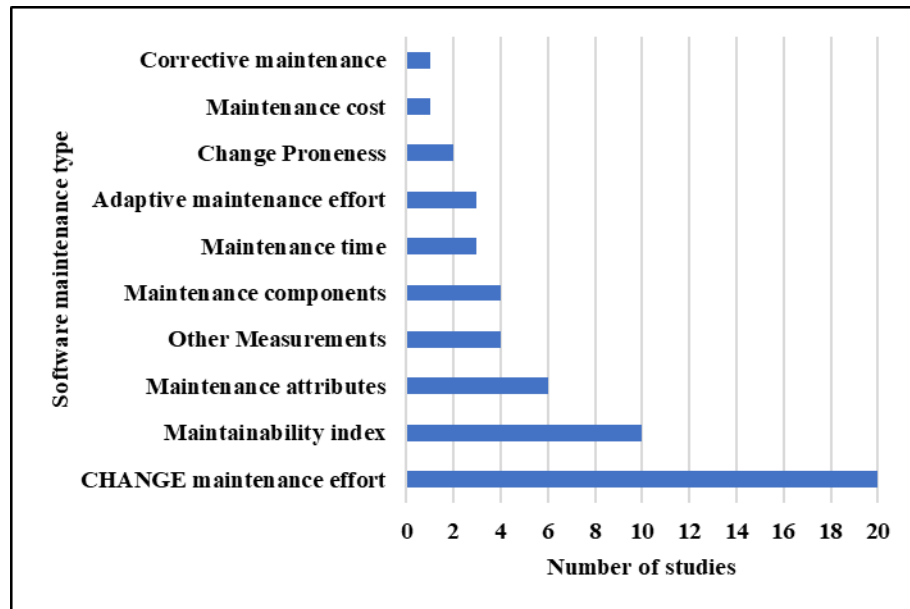


Figure 2.6: The number of selected studies in each software maintenance type.

2.4.2 Software maintainability metrics

In this section, the following research questions are addressed:

RQ2: What type of OO metrics have been proposed to measure software maintainability, **RQ2.1:** What metrics are used (independent variable)? and **RQ2.2:** What is the level of these metrics?

Software metrics play the most significant role in building a predictive model of software maintainability. OO software metrics are divided into two broad categories: product metrics to measure software maintainability that are based on the quality of software product (e.g., number of lines in source code) and process metrics to measure software maintenance that are based on the quality of software processes (e.g., number of hours to change code) [9].

In addition, software metrics can be categorized into those based on internal attributes that directly measure features of OO software such as inheritance or class cohesion, and external attributes that indirectly measure features of the software environment such as the Change metric that capture elements of maintainability from sets of internal attributes [7].

Furthermore, metrics can be categorised into different levels: method, class, and application. Method level is used mainly with traditional metrics that involve complexity and size metrics, such as Halstead metrics [92] and McCabe metrics [93]. Class level is used commonly and include those proposed by C&K [26] and L&H [9]. C&K presented a

theoretical work to define a set of software metrics for OO systems based on sound measurement theory. They defined six metrics as a base suite for OO designs, namely, DIT, WMC, NOC, CBO, RFC and LCOM (these abbreviations are expanded in Table 2.6). L&H constructed a model to predict maintenance effort based on 11 OO metrics, five metrics of which came from the C&K set (they excluded CBO) and six more proposed by themselves: MPC, ADT, NOM, LOC, SIZE1, SIZE2 (also expanded in Table 2.6) and the CHANGE metric, which is the dependent variable and based on the number of changes made in the source code. Application level metrics are extracted from the application as a whole e.g. error-prone subprograms [94] or total number of modules in each application [95].

There were too many different metrics used in selected primary studies, and it is so difficult to describe all these metrics. Therefore, I decided to present the description of metrics used in software maintenance type (Table 2.5). Table 2.6 presents the description of the metrics used to predict maintainability in my selected primary studies, grouped according to type of maintenance.

What stands out in Table 2.6 is the high number of selected primary studies that used the L&H metrics (20 studies). These studies reported evidence of a strong relationship between OO metrics and software maintainability. However, some studies performing FS techniques have not clearly specified the best OO metrics for software maintainability prediction. Moreover, the total number of selected primary studies that used the MI metrics, which are widely accepted in industry [96], is half that of the L&H metrics. It would seem likely that the high number of studies using the L&H metrics is due to the availability of QUES and UIMS datasets that include these metrics, while studies using the MI metrics may be due to the availability of tools to extract and measure these metrics.

Table 2.6: Summary of metrics used in different maintenance types.

Type of maintenance	Study ID	Metrics	Description	
Change maintenance effort	S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51, S52	L&H metrics		
		DIT	Depth of inheritance tree	
		NOC	Number of children	
		MPC	Message-passing coupling	
		RFC	Response for a class	
		LCOM	Lack of cohesion in methods	
		DAC	Data abstraction coupling	
		WMC	Weighted methods per class	
		NOM	Number of methods	
		SIZE1	Lines of code	
		SIZE2	Number of properties	
Corrective maintenance	S18	NA	Number of tasks requiring software modification	
		NB	Number of tasks requiring fixing of data misalignment	
		NC	Number of other tasks	
		N	Number of the total tasks (N=NA+NB+NC)	
		SIZE	Size of the system to be maintained	
Adaptive maintenance effort	S12, S19, S22	Eff _{add}	Effort for addition	
		Eff _{und}	Effort for understanding	
		Eff _{del}	Effort for deletion	
Maintenance evaluation by maintainability index	S5, S6, S8, S21, S28, S29, S33, S48, S54, S55	HV	Halstead volume metric	
		CC	Cyclomatic complexity metric	
		LOC	Counted as a line of code	
		COM	A percentage of comment lines	
Maintenance evaluation by change proneness	S47, S56	C&K metrics		
		DIT	Depth of inheritance tree	
		WMC	Weighted methods per class	
			NOC	Number of children
			CBO	Coupling between object classes
			RFC	Response for a class
			LCOM	Lack of cohesion in methods
Maintenance time	S7, S17, S37	IL	Interaction level	
		IS	Interface size	
		OAC	Operation argument complexity	
		ID	Inheritance depth	
Maintenance cost	S9	MM _T	Man-months to understanding	
		MM _M	Man-months to modifying	
		MM _U	Man-months to testing	

Table 2.7 provides a summary of the metrics used to predict maintainability in my selected primary studies, grouped according to type (product/process) along with an indication of the level at which the measurement is captured.

From Table 2.7, it can be seen that the majority of the selected primary studies used class level product metrics (related to the fact that most of the studies are attempting to predict classes changed in the source code). The table also shows that a large number of selected primary studies used process level metrics to predict application level maintainability, with product metrics only featuring in a small number of studies for this category of change.

Table 2.7: Summary of software maintainability metrics.

Metrics type	Metrics level	Study ID
Product metrics	Class level	S2, S4, S7, S10, S11, S14, S20, S25, S26, S27, S30, S32, S36, S38, S39, S40, S42, S43, S46, S47, S49, S50, S52, S56
	Application level	S3
	Class level, method level	S5, S6, S8, S12, S16, S21, S28, S29, S33, S41, S44, S48, S54, S55
	Class level and application level	S45, S51
Process metrics	Application level	S13, S17, S18, S22, S24, S31, S34, S37
	Application level, class level,	S15, S19
Product and process metrics	Application level	S9, S35

Figure 2.7 provides a visualisation of the data in Table 2.7, and aggregates the total number of selected primary studies using both metric type and metric level, and clearly illustrates the popularity of employing product metrics for class-level predictions, and process metrics for application level predictions.

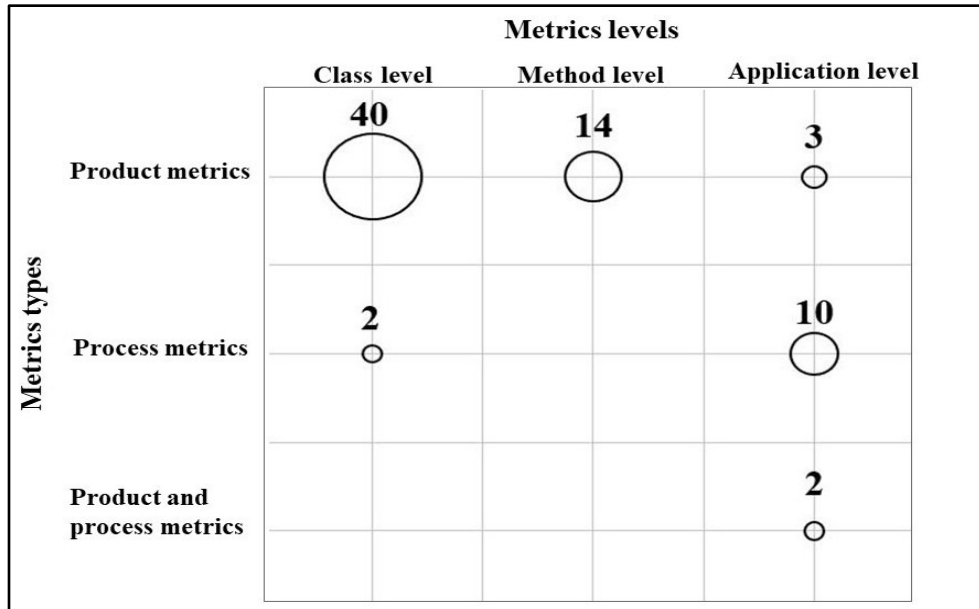


Figure 2.7: The number of studies using metrics type and metrics level.

Figure 2.8 presents the distribution of metrics of selected primary studies. 79% of the studies used product metrics and 17% of the studies used process metrics. Moreover, 4% of the studies integrated both product and process metrics. The evidence from this result suggests that product metrics are the majority of metrics used in software maintainability.

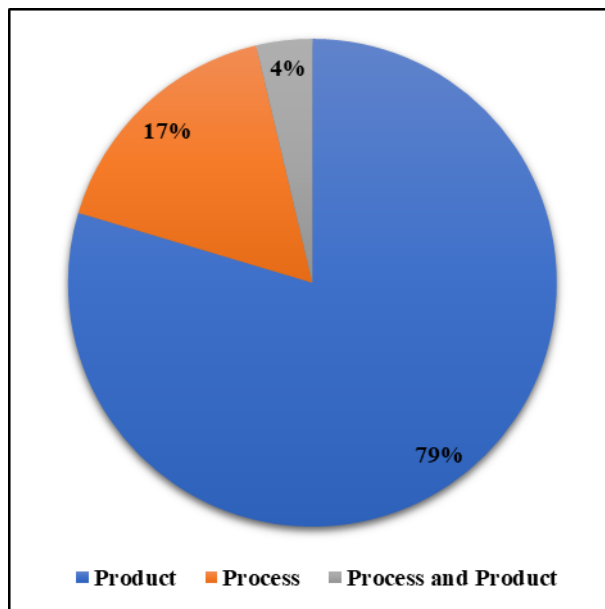


Figure 2.8: The distribution of metrics.

2.4.3 Software maintainability datasets

This section seeks to answer the following questions:

RQ3: What are the software maintainability datasets in the literature that were used to build prediction models? **RQ3.1:** What is the type of software maintainability datasets? **RQ3.2:** What tools are used to extract metrics from open source projects? **RQ3.3:** What programming languages are used to collect metrics? **RQ3.4:** What is the number of classes in the software system?

A dataset is a combination of related sets of the data that may be used to perform machine learning models, and it is considered the foundation of building prediction models. For the model building the dataset is divided into a training set, which is used as input to train model, and a testing set, which is used as input to evaluate the model [65].

A. Datasets used

The datasets used in the selected primary studies may be divided into three main types:

- **Public dataset:** The dataset is available as appendix or table in published paper or in a publicly accessible repository, such as the Promise repositories in S23. In 2018, Hegedus et al. in S55 provided their datasets via the PROMISE repository: one of the first regarded as suitable for software maintainability prediction [58]. These datasets were built from extracted OO metrics from seven open-source systems and include a calculated MI. Their contribution provides encouragement to the researcher community to develop more research in software maintainability prediction.
- **Partial dataset:** The dataset is not available, but has been extracted from open source software project repositories such as GitHub [27] or SourceForge [28].
- **Private dataset:** The dataset is not available and has been extracted from a private software system.

Table 2.8 illustrates a summary of different types of the datasets used in the selected primary studies. What can be clearly seen in Table 2.8 is most selected primary studies make use of two public datasets: QUES and UIMS proposed by L&H [9]. These datasets are derived from systems written in Classic-Ada as the OO programming language. A further notable finding is that most of the selected primary studies have been conducted using private datasets, as opposed to public or partial datasets, which makes many empirical studies of software maintainability prediction not repeatable.

Table 2.8: Summary of different types of dataset.

Public datasets				
Study ID	Datasets name	Dataset Source	Dataset size	Dataset link
S2, S20, S25, S26, S30, S36, S38, S46, S47, S49, S52	QUES	Commercial software products (Quality Evaluation System)	71 classes	Provided as an appendix in [9].
	UIMS	Commercial software products (User Interface System)	39 classes	
S11	UML class diagram	Bank information systems	27 classes	Provided as an appendix in [97].
S13	Bank	Bank information systems	55 classes	Proposed as a table in [95]
S34	Controlled experiment	Academic course	24 classes	Proposed as an appendix in [98].
S53	Spring, Edition, RxJava, Restlet, Hadoop, Camel, Kotlin, Elasticsearch, Hbase Drools, OrientDB	Open source system in GitHub [27]	1151 classes	https://zenodo.org/record/835534#.W123H9IzY2x
S55	Titan, mcMMO, junit, oryx, mct, antlr4, mapdb	Open source system in GitHub [27]	10,844 classes	https://zenodo.org/record/581651#.W129Y9IzY2w
Partial dataset				
Study ID		Dataset Source		
S27, S29, S33, S39, S41, S47, S50, S54		Open source system in SourceForge [28]		
S21		Industrial software		
S28, S35, S40, S44, S45, S48, S51, S56		Other open source projects		
Private dataset				
Study ID		Dataset Source		
S1, S3, S4, S5, S6, S7, S8, S9, S10, S12, S16, S17, S18, S19, S22, S24, S31, S32, S37, S42, S43		Private projects		

Figure 2.9 illustrates the number of selected primary studies in each dataset type. From the chart below which clearly illustrates the difference between the number of studies using private datasets compared with other types of datasets. However, there are new public datasets proposed in S53 and S55 that may encourage researchers to apply their models to predict software maintainability.

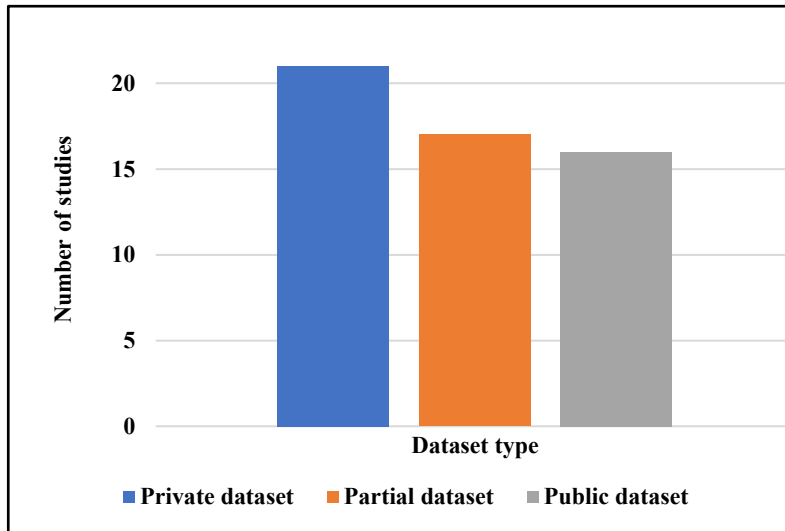


Figure 2.9: The number of studies used each type of the dataset.

B. dataset size

The dataset size may be classified into three main groups: small, where the number of classes less than 100, medium, where the number of classes less than 500, and large, where the number of classes is more than 500. Figure 2.10 provides the number of datasets classified by the size of the dataset. This result shows that selected primary studies were mostly performed on the larger sized datasets, which improves the validity of prediction models.

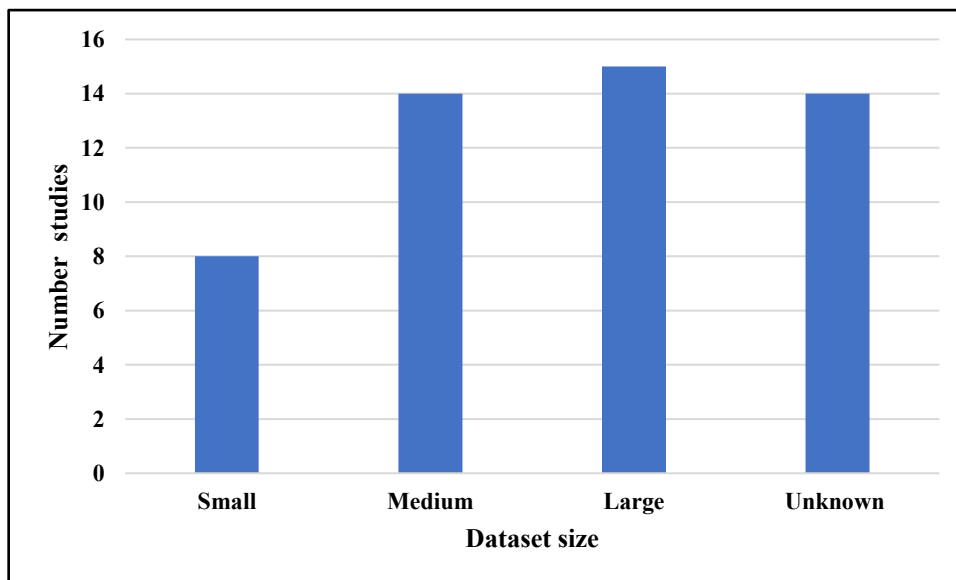


Figure 2.10: The number of datasets classified by the size of the dataset.

C. Programming language

Most of the datasets were extracted from systems written in Java, Classic-Ada, C#, or C++. Figure 2.11 presents the distribution of the programming language in each selected primary studies. Java is the most popular language used in the studies which may be due to the availability of open-source systems written in Java.

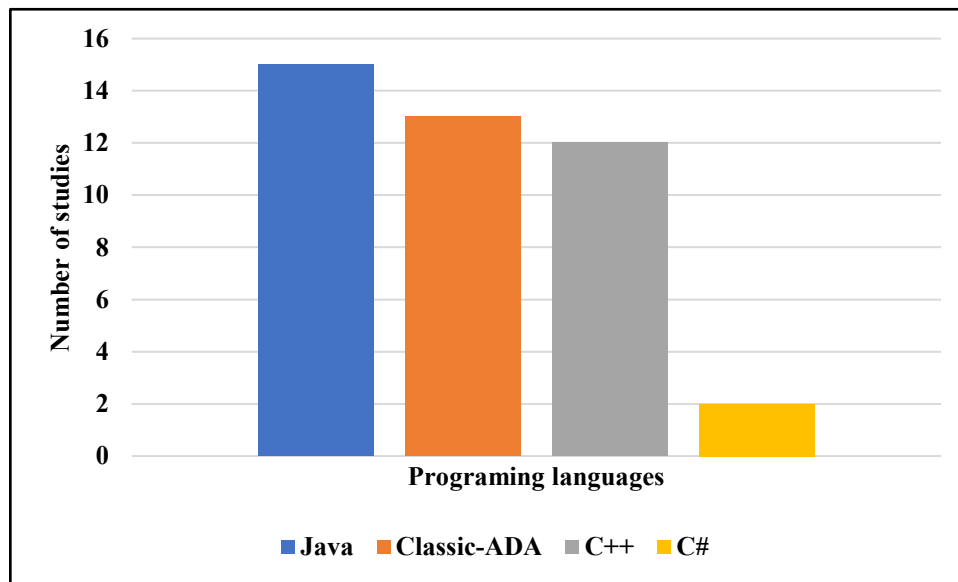


Figure 2.11: The distribution of programming language in each study.

D. Tools used for data extraction

The variables in the datasets were extracted from software source by using specific tools. Table 2.9 presents the tools used for extracting the independent variables of the selected primary studies. The dependent variables were typically collected from comparing the first and the last versions of source software (either manually or by using a tool).

As can be seen from Table 2.9, there is a broader range of tools available for Java which may be a reflection of the popularity of the language as illustrated in Figure 2.11. Extraction tools that work with C# are used less frequently. Some of the extraction tools can work with more than one language.

Table 2.9: The tool extraction independent metrics.

Programming language	Study ID	Tool name
Classic-Ada	S2, S20, S25, S26, S30, S36, S38, S46, S47, S49, S52	Classic-Ada metric analyser
C	S6	HPMAS prior to perfective maintenance modification
C++	S4, S15	An automated data collection tool
	S12	Log file
	S21	Krakatau metrics professional
C and C++	S5, S8, S11, S13 S17, S31, S34, S37	Survey questionnaire
	S35	Resource standard metrics
	S45	Logiscope software static analysis tool.
C++ and Java	S27, S41, S44	CCCC
Java and C#	S27, S41	OOMeter
C#	S44, S41	Visual studio
C, C++ and Java	S10	Reconfigurable automated metrics for OO software
	S16	KLOCwork tool
Java	S27, S41, S43, S46, S48, S50, S56	C&K java metric (CKJM) tool
	S51	ObjectAid UML explorer
		JHawk tool
		Classycle
	S54	COINS tool
	S55	SourceMeter static code analysis too
	S56	Defect collection and reporting system (DCRS) tool
	S27, S41	Analyst4j
		Dependency finder
	S28, S51, S41	JDepend
	S27, S28, S41	Understand for Java
	S41	Java coding standard checker

Table 2.10 shows the tools used to identify the changes that have taken place (although several of the studies did not mention the tool used to collect the data).

Table 2.10: The tools for dependent metrics extraction.

Study ID	Tool name	Software maintainability type
S11	Fuzzy prototypical knowledge discovery	Categorise maintainability into easy, medium or difficult maintain
S16	Distributed software development (DSD)	Categorise maintenance activities as perfective, adaptive or corrective
S39, S51	Version control system (VCS)	Track source code changes
S43	Eclipse compare plug-in	Compare changes
S45	Static code analysis tools	Calculate the code change history
S50	Beyond compare	Compare changes between two versions of software
S55	Quality gate source audit	Calculate relative maintainability index, similarly to the commonly maintainability index measurement

2.4.4 Evaluation measures

This section addresses the following questions:

RQ4: What are the evaluation measures used to assess the performance of software maintainability prediction models? **RQ4.1:** What are the validation approaches used to evaluate the performance of software maintainability prediction models?

Various evaluation measures have been used in software maintainability prediction to evaluate and compare the accuracy of model performance. The appropriate evaluation measure is usually based on problem type: regression, classification, or clustering. The evaluation measures identified in the selected studies are reported in Table 2.11 which also provides the definition and equation for each measure.

From Table 2.11, key findings emerge: the most popular prediction accuracy measures used for regression problems after R-squared are those based on the magnitude of relative error (MRE, and MMRE) [67] and (Pred) [11]. However, prior studies have pointed out that several evaluation measures (e.g., MMRE) have bias and instability issues in their results [99, 100]. To resolve these issues, a baseline or benchmark is recommended to compare and evaluate the performance of the prediction models [99, 100]. For classification problems, the most commonly used evaluation measures are recall and precision, followed by accuracy. However, the previous study stated that the accuracy measure provides misleading results with the problem of the imbalanced dataset because it tends to measure towards the majority class [55]. In the selected primary studies, only one study (S56) used accuracy with the imbalanced dataset. However, S56 resolved the problem of imbalanced dataset by using a threshold of 50% for balance values.

Table 2.11: Summary of evaluation measures used.

Evaluation measures for Regression problems				
Study ID	Name	Acronym	Definition	Equation
S18, S25, S26, S29, S34, S38, S44, S47, S54	Magnitude of relative error	MRE	The absolute difference between the actual value and predicted values divided by the actual value	$MRE = \text{actual value} - \text{predicted value} / \text{actual value}$
S25, S26, S30, S34, S38, S49, S38, S44, S47, S54	Mean magnitude of relative error	MMRE	It is the mean of MRE.	$MMRE = 1/n \sum_{i=1}^{i=n} MRE$
S18, S25, S26, S30, S34, S38, S42, S44, S47, S54	Pred(q)	PRED	It calculates the proportion of instances in the dataset where the MRE is less than or equal a specified value (q). The q is defined value, k is the number of states where MRE is less than or equal to q, and n is the whole number of views in the dataset.	$Pred(q) = k/n$
S20, S33, S38	Mean square error	MSE	It measures the average of the squares of the differences between the actual (Y_i) and predicted (\hat{Y}_i) values.	$MSE = 1/n \sum_{i=1}^{i=n} (Y_i - \hat{Y}_i)^2$
S26, S29	Absolute relative error	ARE	The Ab. Res. is the absolute value of residual, which is the difference between the actual value and predicted	$Ab.Res = \text{actual value} - \text{predicted value} $
S32, S42, S46, S52	Mean absolute relative error	MARE	It is the mean of the ARE	$MARE = 1/n \sum_{i=1}^{i=n} ARE$

S20, S36, S46, S49, S50, S52	Mean absolute error	MAE	It is the average of absolute values of the difference between $X'i$ and $X i$, where $X'i$ is the predicted value and $X i$ is the actual value	$MAE = 1/n \sum_{i=1}^n (X'i - X i)$
S46, S50	Root mean square error	RMSE	It is the square root between the square of predicted value and the actual divided by number of observations in the dataset, where $X'i$ is the predicted value and $X i$ is the actual value	$RMSE = \frac{\sqrt{\sum_{i=1}^n (X'i - X i)^2}}{n}$
S46, S52	Standard error of the mean	SEM	It is the standard deviation divided by root of the number of observations in the dataset.	$SEM = SD / \sqrt{n}$
S2, S5, S8, S12, S16, S17, S18, S19, S20, S21, S24, S29, S31, S32, S33, S34, S36, S48, S51	R square	R ²	It presents the proportion of the variance of the dependent variable that is explained by the model.	R-squared = 1 - (explained variance / total variance), where explained variation is the sum of squares of the residuals (i.e. actuals-predicted) and total variation is the residuals with respect to the average (i.e. actuals - average) [101].
Evaluation measures for Classification problems				
Study ID	Name	Acronym	Definition	Equation
S37, S45, S53, S56	Accuracy		It is the number of true predictions over all types of predictions.	$Accuracy = (TP + TN) / (TP + TN + FP + FN)$
S37, S40, S43, S53, S56	Recall		It is a proportion of actual positives that are correctly determined.	$Recall = TP / TP + FN$
S37, S40, S43, S53, S56	Precision		It is capability of a model to correctly identify relevant instances.	$Precision = TP / TP + FP$
S37, S39, S56	F-Measure		It integrates precision and recall/sensitivity in one equation.	$F-Measure = 2 * Precision * Recall / (Precision + Recall)$
S56	Specificity		It is a proportion of actual negative that are rightly determined.	$Specificity = TN / TN + FP$
S39, S40, S43, S47	Area under curve of ROC curve	AUC	It is a plot of two parameters: the true positive rate and false positive rate.	
Evaluation measures for Clustering problems				
Study ID	Name	Definition	Equation	
S11, S31	Mean cluster	It calculates how close the clustering (nearest clustering) is to the preidentified classes (e.g., low, medium and high maintainability) by average of all cross-cluster pairs, where $a_i = (a_1 + a_2 + \dots + a_n)$	$Mean = 1/n \sum_{i=1}^n (a_i)$	

**TN: True negatives, FP: False Positive, FN: False Negative, TP: True Positive.

Figure 2.12 shows the number evaluation measures in each machine learning problem used by selected primary studies. R-squared is the most frequently used evaluation measure for regression problems (19 studies), followed by MMRE and PRED (10 studies). For the classification problem Recall and Precision were applied by 5 studies. In the clustering problem, only one evaluation measure was employed.

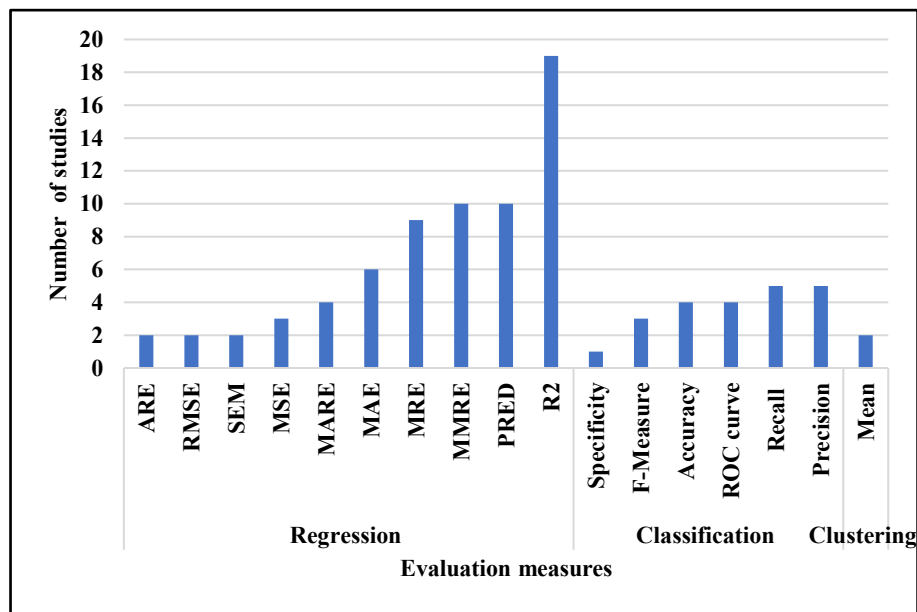


Figure 2.12: The distribution of evaluation measures used by selected primary studies.

Validation is used to evaluate the performance of a trained model on previously unseen data [65]. There are three major validation types used in the selected primary studies: k-fold cross-validation, leave-one-out and holdout. Leave-one-out is considered the most extreme way to perform cross-validation by using all instances. Both methods (i.e. k-fold cross-validation and Leave-one-out) has an advantage over the hold-out to make predictions on all datasets. For this reason, the main advantage of k-fold cross-validation and leave-one-out is to decrease variance and has lower variation than the hold-out method. Also, k-fold cross validation has a particular advantage over hold out, which every row in the dataset presents in the training and test set at least once. However, hold-out only needs one run, so it spends lower time than other methods. Table 2.12 illustrates the validation types used by the selected primary studies. A fact that worth mentioning is that this table contains fewer than half of the selected primary studies, the remainder of the studies measured software maintainability

directly without applying prediction models. Therefore, they did not need to validate the model's performance.

Table 2.12: Summary of validation types.

Study ID	Validation Types	Definition
S20, S37, S39, S40, S45, S46, S47, S49, S52, S53, S54, S56	K-fold cross-validation	The dataset is divided randomly into K folds (or partitions) of the same size, where one-fold is used to test the model and the remaining $k-1$ are used as training data. The process is repeated k times to select a new different fold at each iteration. The overall performance is based on the average over the k test folds.
S18, S26, S29, S30	Leave-one-out	It is a logical extreme version of k-fold cross validation where k is equal to the size of the data set. One observation of the total dataset is removed, and the model is constructed with the remaining dataset and tested in the removed observation. It then repeats the process by deleting a new different observation and so on for the whole data set.
S4, S17, S19, S25, S32, S36, S38, S42, S44	Holdout	It partitions dataset into two sets, where one partition is used for training the model and another partition for testing.

2.4.5 Machine learning problem category.

This section addresses following RQs: **RQ5:** What type of machine learning was performed in the software maintainability domain? **RQ5.1:** What are the categories of machine learning problem explored?

Machine learning approaches may be divided into three main groups: supervised, unsupervised and semi-supervised. Supervised learning encompasses two machine learning problem types: regression and classification, while unsupervised learning comprises clustering and association. The semi-supervised is a combination of supervised and unsupervised. The regression problem predicts a continuous number, where the classification predicts a category of two or more types. Clustering groups data together based upon attributes and distance measures, whereas association detects rules that explain a large amount of the data. Moreover, supervised learning builds machine learning models to predict output data from the input data, where the input data has a label. Unsupervised learning builds models from unlabelled input data. Semi-supervised learning builds models to predict output data from input data, where some data items have a label [65]. Table 2.13 shows the summary of machine learning problems used by the selected primary studies. As mentioned in the explanation of Table 2.12, most of the selected primary studies used direct measures to evaluate software during the maintenance process without performing any machine learning problems.

Table 2.13: Summary of machine learning problems.

Study ID	Machine learning approach	Machine learning problem type
S20, S25, S26, S29, S30, S38, S42, S44, S45, S46, S47, S48, S49, S50, S52, S54	Supervised	Regression
S37, S39, S40, S43, S47, S53, S56		Classification
S11, S31, S32	Unsupervised	Clustering

2.4.6 Individual prediction models

This section explores different individual prediction models used in the selected primary studies of software maintainability prediction and identifies the superior model in each study. This section addresses the following questions: **RQ6:** What are the individual prediction models used to predict software maintainability? **RQ6.1:** Which are the best performing individual prediction models?

Table 2.14 presents a summary of the studies that have used individual models to predict software maintainability, along with the best accuracy prediction model over the most evaluation measures. The selection of the best model relies on the evaluation measures that compare and evaluate the prediction accuracy of the individual prediction models used in each study. According to these measures, the best model is selected as a recommendation model in each selected study. Again as indicated in Table 2.12 and Table 2.13, Table 2.14 includes a subset of selected primary studies, since only these studies used indirect measures to predict software maintainability and apply machine learning models.

Table 2.14 is reported only the best model performance regardless of which evaluation measurements were used. As shown in Table 2.14, several selected primary studies have different recommended individual models to predict software maintainability. However, two studies have reported that SVR outperforms other selected models.

Table 2.14: Summary of individual prediction models used with the best model in each study.

Study ID	Individual prediction models	The best accuracy prediction model
S11	FDP	FDP
S20	WNN and GRNN	GRNN
S25	BN, RT, MLR (Backward elimination) and MLR (Stepwise selection)	BN
S26	MARS, MLR, SVR, ANN and RT	MARS
S29	LR	LR
S30	TreeNet, MARS, MLR, SVR, ANN, and RT	TreeNet
S31	K-means clustering	K-means cluster
S32	SVM	SVM
S36	MLP, WNN and GRNN	MLP
S37	LR, DT, CART, BFTree and LEGAL-Tree	DT
S38	FL, BN and MARS	FL
S39	Multivariate model	Multivariate model
S42	FF3LBPN, GRNN and GMDH	GMDH
S44	BPN, KN, FFNN, GRNN	KN
S45	SVR, MLP, IBk, M5Rules, KStar, GP, AR, REPTree, M5P, RF, RSS, IR, PLS classifier, GS, DS, PR, CR, RBD, LMS, LR, Decision table, LWL, RBFNN	SVR
S46	Neuro-GA Approach	Neuro-GA Approach
S48	MLR based on stepwise selection and backward elimination methods	Stepwise selection
S49	FLANN, GA, PSO, CSA, FLANN-GA, FLANN-PSO, FLANN-CSA	FLANN-GA
S50	GA, Decision Table, RBFNN, BN and SMO	GA
S52	Neuro-Fuzzy approach	Neuro-Fuzzy approach
S54	MLR, MLP, SVR, M5P and RT.	SVR

** FDP: Fuzzy Deformable Prototypes, WNN: Ward neural network, GRNN: General regression neural network, BN: Bayesian network, RT: Regression tree, MLR: multiple linear regression, MARS: Multiple adaptive regression splines, SVR: Support vector regression, ANN: Artificial Neural Network, LR: Linear regression, TreeNet: Multiple additive regression trees, SVM: Support vector machine, MLP: Multilayer Perceptron, GRNN: general regression neural network, LR: logistic regression, DT: decision tree, CART: Classification and Regression Trees, FL: fuzzy logic-based, FF3LBPN:Forward 3-Layer Back Propagation Network, GMDH: Group Method of Data Handling, BPN: Back Propagation Network, KN: Kohonen Network, FFNN: Feed Forward Neural Network, GP: Gaussian processes, AR: Additive regression, RF: Random forest, RSS: Random subspace, IR: Isotonic regression, GS: Grid search, DS: Decision stump, PR: Pace regression, CR: Conjunctive rule, RBD: Regression by discretization, LMS: LeastMedSq, LWL: locally weighted learning, FLANN: functional link artificial neural network, GA: Genetic algorithm, PSO: Particle swarm optimization, CSA: clonal selection algorithm, RBFNN: Radial Basis Function Neural Network, SMO: Sequential Minimal Optimization.

The results for an accurate prediction model are recognized if they meet the criteria of $Pred(.30) \geq 0.70$ [34] or $Pred(.25) \geq 0.75$ or/and $MMRE \leq 0.25$ [35]. Even though the suggested criteria are based on relatively old references [34, 35], recent studies have employed these criteria to evaluate prediction accuracy in the software engineering domain [102-104]. Also, several selected primary studies have used these criteria, such as S25, S26, S30, S38 and S54. However, S35 suggested that it is a challenging task to meet these criteria.

Table 2.15 presents the performance of the MMRE value for some selected primary studies; Boldface values in the table indicates the best results. However, several studies in Table 2.14 did not use MMRE or had not performed a regression problem, so I could not evaluate their performance against the criteria. The results in this table indicates that FLANN-GA: functional link artificial neural network - genetic algorithm in S49 is the only model that meets the criteria of MMRE in UIMS datasets, while FL model in QUES dataset in S38 and Neuro-Fuzzy approach in S52 are close to meeting the criteria to build an accurate effort prediction model.

Table 2.15: Performance of MMRE value for some selected primary studies.

Dataset name	Study ID	The best accuracy prediction model	MMRE
QUES	S25	Stepwise selection	0.39
	S26	MARS	0.32
	S30	MARS	0.32
	S38	FL model	0.27
	S46	Neuro-GA Approach	0.37
	S49	FLANN-GA	0.32
	S52	Neuro-Fuzzy approach	0.33
UIMS	S25	Bayesian network	0.97
	S26	SVR	1.86
	S30	TreeNet	1.57
	S38	FL model	0.53
	S46	Neuro-GA Approach	0.31
	S49	FLANN-GA	0.24
	S52	Neuro-Fuzzy approach	0.28
Five open source Java software systems	S44	KN	Mean MMRE: 0.44 for model 1. Mean MMRE: 0.32 for model 2.
Twenty-six open source Java software systems	S54	SVR	Mean MMRE: 0.91

The number of individual prediction models that has been used in selected primary studies is illustrated in Figure 2.13 (fifty-three in total). From the fifty-three models shown in Figure 2.13, the five most frequently employed individual models are MLR, SVR, GRNN, RT and FLANN. MLR is the most frequently used individual model for software maintainability (six studies).

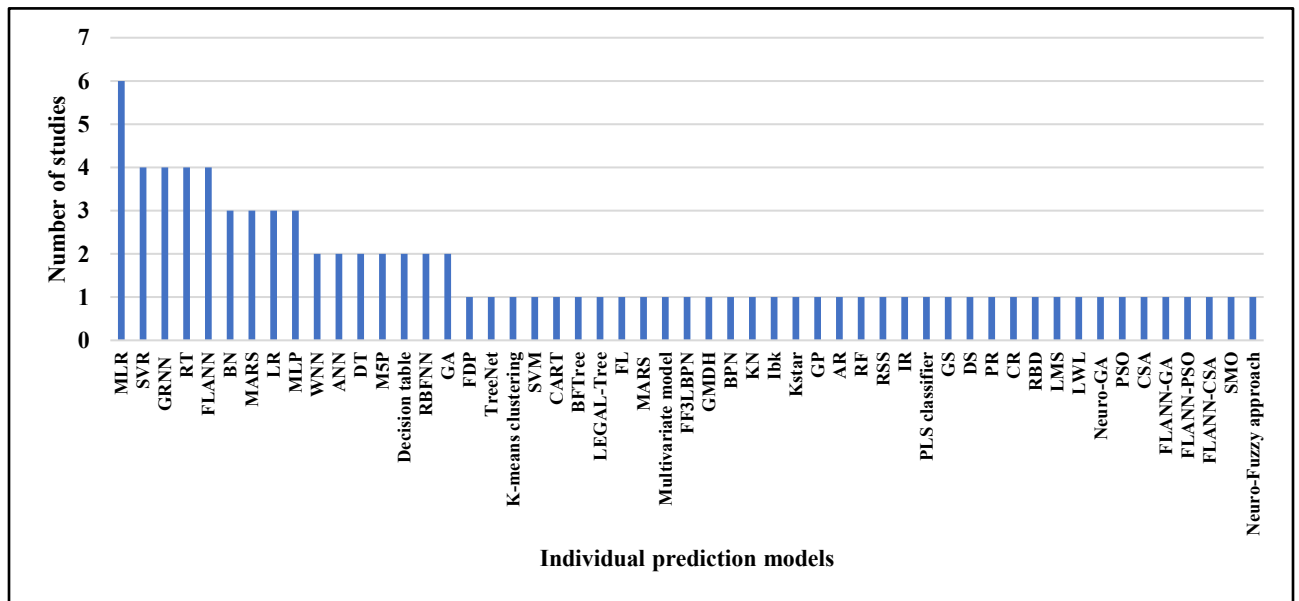


Figure 2.13: Individual prediction models used in selected primary studies.

2.4.7 Ensemble prediction models

This section investigates various ensemble prediction models used in the studies and compares the ensemble model performance against individual models. The aims of this section are to answer the following questions: **RQ7:** What type of ensemble prediction models were employed to predict software maintainability? And **RQ7.1:** Are the ensemble models able to improve over the performance of the individual prediction models?

An ensemble model is a combination several individual models designed to improve on the accuracy prediction of individual models. They can be classified into two major types: homogenous that uses the same type of individual models, and heterogenous that uses different types of individual models [105]. Moreover, the ensemble models can be categorized into linear ensembles, which combine the outputs of the base model in a linear manner (e.g., weighted averaging, averaging) and nonlinear ensembles, which combine the outputs of the base model in a nonlinear manner (e.g., decision tree forest) [106]. Five main selected primary studies employed ensemble models to predict software maintainability. These studies emphasised the positive impact of ensemble methods in predicting software maintainability compared with individual prediction models. Table 2.16 summarises the ensemble prediction models used.

As shown in Table 2.16 below, different types of ensemble models have been compared with individual prediction models. The ensemble prediction models in selected primary studies (i.e. S40, S43, S47, S53 and S56) improved the performance of individual models and increased their accuracy prediction over individual prediction models. Even though the ensemble models reported a superior result over individual prediction models, a limited number of selected primary studies applied the ensemble models to predict software maintainability (which may be due to the fact that ensemble models are relatively new [107]).

Table 2.16: Summary of the ensemble prediction models.

Study ID	Ensemble Type	Ensemble name	Base models	Combination rules	The best model	Does the ensemble model improve the performance of the base model?
S40	Heterogeneous	SMEM-MCC	ISMEM, DT, BPN, SMO	NA	Ensemble model (SMEM-MCC)	Yes
S43	Homogeneous	RF	Naïve Bayes, Bayes Network, Logistic, Multilayer Perceptron	Averaging	Ensemble model (RF)	Yes
S47	Heterogeneous	Linear ensemble	MLP, RBF, SVM, M5P	Averaging, weighted averaging and best in training	Ensemble model	Yes
	Homogeneous	Bagging and AdaBoost	MLP, RBF, SVM, DT	Averaging		Yes
	Heterogeneous	Linear ensemble and Non-linear	SVM, MLP, logistic regression, genetic programming, K-means	Best in training, majority voting and decision tree forest		Yes
S53	Homogeneous	RF and AdaBoost	J48	Averaging	Ensemble model (AdaBoost)	Yes
S56	Homogeneous	MVEC, WVEC, HIEC, WVHIEC	Seven individual particle swarm optimization (PSO)	Majority voting, weighted voting and hard instance	Ensemble model (HIEC and WVHIEC)	Yes
<p>** SMEM-MCC: Software Maintainability Evaluation Model based on Multiple Classifiers Combination, RF: Random forest, Bagging: Bootstrap aggregating, AdaBoost: Adaptive Boosting, MVEC: Majority Voting Ensemble Classifier, WVEC: Weighted Voting Ensemble Classifier, HIEC: Hard Instance Ensemble Classifier, WVHIEC: Weighted Voting Hard Instance Ensemble Classifier.</p>						

The number of ensemble prediction models used in selected primary studies is shown in Figure 2.14 (ten in total). The number of homogeneous ensemble models used exceeds the heterogeneous ones. Furthermore, the linear ensemble is the most frequently used heterogeneous model, and RF and AdaBoost are the most frequently used homogeneous models.

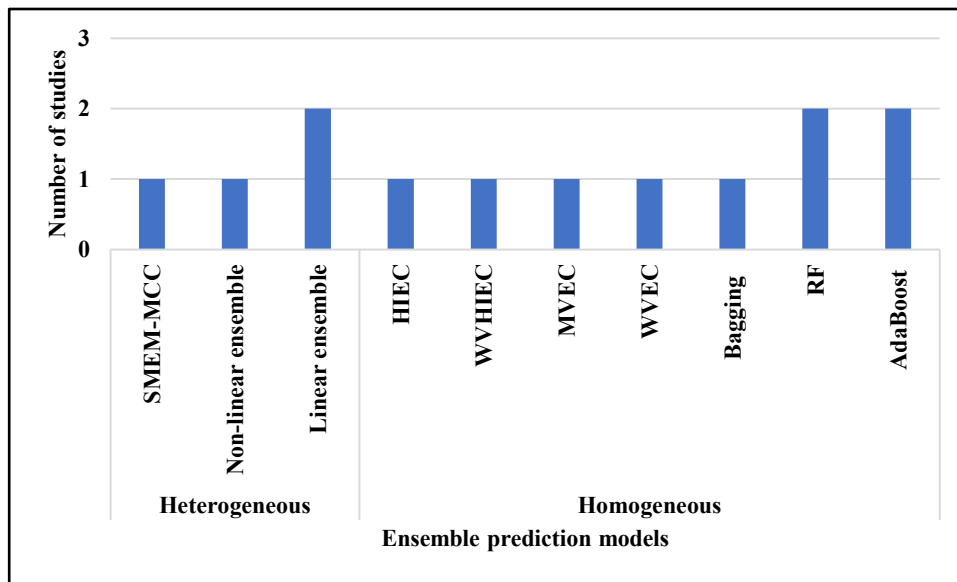


Figure 2.14: Ensemble prediction models used in selected primary studies.

2.5. Conclusion of Systematic Literature Review

This literature review aimed to identify and analyse the measurements, metrics, datasets, evaluation measures, machine learning problems, individual prediction models and ensemble prediction models employed in the field of software maintainability prediction. An extensive search was conducted in five online digital libraries to select peer-reviewed articles that publish in either journals or conferences. Fifty-six studies have been selected between 1991 and 2018, and seven main questions have been answered from each selected primary study. This literature review has been performed as a SLR to evaluate all relevant research evidence and identify the available studies in the field of software maintainability prediction, with the purpose to answer certain RQs.

The main answer for each research question in this literature review is reported in (Figure A.1 in Appendix A). This figure provides the mind of software maintainability prediction and highlights the most important answers. This visualisation enables us to break down the complex problem of software maintainability prediction into several solutions according to the RQs. As a result, this mind map integrates a whole literature review into one organised picture to respond to the RQs as the following:

- There are ten software maintainability measurements used in selected primary studies: CHANGE maintenance effort, corrective maintenance, adaptive maintenance effort, maintenance evaluation by MI, maintenance evaluation by change proneness, maintenance time, maintenance cost, maintenance attributes, maintenance components and other measurements.
- There are three main types of software maintainability metrics used in selected primary studies: product metrics, process metrics, and product metrics combined with process metrics.
- There are three main types of software maintainability datasets used in selected primary studies: public datasets, partial datasets and private datasets.
- The software maintainability evaluation measures are grouped by machine learning problem: regression problem-based models include MRE, MMRE, PRED, MSE, ARE, MARE, MAE, RMSE, SEM and R2, classification problems include Accuracy, Recall, Precision, F-Measure, Specificity and ROC curve, and cluster problems include the mean.
- Machine learning models to predict software maintainability involve two main categories: supervised learning for regression and classification, and unsupervised for clustering.
- Fifty-three individual models were constructed by selected primary studies.
- Ten ensemble models were created by selected primary studies (seven homogeneous ensemble models and three heterogeneous ensemble models).

Most notably, this is the first study to my knowledge that provides a SLR in software maintainability prediction. Therefore, I have confidence my proposed study will be a novel and hopefully valuable contribution to the area of software maintainability prediction. The findings obtained in this study can be used by the researchers to provide an overall overview of this area. The main findings derived from this SLR are:

- Relatively few studies have been conducted in software maintainability prediction compared with other software quality attributes such as defect prediction or fault prediction.

- Different types of software maintainability measurements were investigated that roughly equate to ten types. The change maintenance effort was the most common measurement used by 20 studies, followed by the MI, which is used by ten studies.
- Studies of software maintainability metrics were generally categorized into three types: product metrics, process metrics and product and process metrics. These types were applied on different levels: class level, method level and application level. Most studies used product metrics along with class level measurements (forty studies), followed by product metrics with method level measurements (fourteen studies), and finally process metrics with application level measures (ten studies). Overall, the total distribution of software maintainability metrics is as follows. 79% of studies used product metrics, 17% of studies used process metrics, while only 4% of studies used product and process metrics.
- The L&H metrics are utilised by twenty studies, and these studies confirmed the evidence of the power relationship between OO metrics and software maintainability. Ten studies used the MI metrics.
- Studies of software maintainability datasets were broadly divided into three types: public, private and partial. The most commonly used was private datasets (more than twenty studies).
- UIMS and QUES datasets were the most frequently used (eleven studies). Regarding the size of the datasets, the result reveal that most studies used either medium or large sized datasets, which improves the validity of prediction models.
- 36% of the datasets were collected from Java systems, most likely as a consequence of the availability of open-source systems written in Java.
- Even though there were several extraction tools used to collect metrics, most of these tools were designed for Java. However, some of these tools can work with more than one programming language.
- The evaluation measures are selected usually based on problem type: regression, classification or clustering. R-squared is the most repeatedly used in regression problem by nineteen studies, followed by MMRE and PRED (ten studies). Recall and

Precision were applied many times in the classification problem. In the clustering problem, only one evaluation measure was employed.

- Three primary validation types used in studies are k-fold cross-validation (48%), leave-one-out (36%) and holdout.
- The most popular machine learning problems were regression problems (62% of the total selected primary studies). 27% of the studies were related to classification problems, and only 11% of these studies were related to clustering problems.
- Some individual models achieved predictions that were close to meeting the criteria of an accurate prediction model, which are $\text{Pred}(.30) \geq 0.70$ [34] or $\text{Pred}(.25) \geq 0.75$ or/and $\text{MMRE} \leq 0.25$ [35], namely FL model in S38 and Neuro-Fuzzy approach in S52. FLANN-GA in S49 is the only model that meets the MMRE criteria.
- MLR, FLANN, RT, GRNN and SVR were the most frequently employed individual models in software maintainability prediction, with more than four studies using these models.
- Ensemble models were used by five (16%) of the selected primary studies (i.e. S40, S43, S47, S53 and S56), compared to more than three-quarters of the studies which used individual models. However, all the ensemble models used in studies yield improved accuracy over the individual models.
- Ten ensemble prediction models were applied to predict software maintainability (seven homogeneous ensemble models and three heterogeneous ensemble models).
- The linear ensemble was the most frequently used heterogeneous model, while RF and AdaBoost were the most frequently used homogeneous models.

The findings of this SLR revealed the following guidelines for the researchers in software maintainability prediction for future work:

- There is a need for more investigations to be carried out in the area of software maintainability prediction using machine learning techniques, as only 26 studies from the selected primary one using machine learning techniques.

- Only one model meets the model accuracy criteria mentioned earlier ($MMRE \leq 0.25$), so further studies are needed to empirically explore and aim to improve the performance of machine learning techniques.
- Selected primary studies have reported the success of using ensemble models to improve the performance of the prediction accuracy by reducing variance (see Table 2.16). However, there is no clear indication of which techniques are more suitable to predict software maintainability accurately and provide more consistent results. Also, a limited number of ensemble models (ten in total) were explored by five selected primary studies. Hence, this limitation is not enough to draw a satisfactory conclusion.
- The QUES and UIMS datasets are publicly available and used by most of the selected primary studies (ten studies) but are small datasets and quite old. There is a need to a larger number of more recent and more substantial datasets to become publicly available to encourage an increase in the number of experimental studies of machine learning techniques.

The next chapter proposes the methodology of this thesis to present an overview of the technical background used in the empirical studies in Chapter 4, Chapter 5 and Chapter 6.

Chapter 3. Methodology for Empirical Studies in Software Maintainability Prediction

This chapter describes the methodology for empirical studies in software maintainability prediction using ensemble techniques. The methodology of this thesis is based on three empirical studies proposed in the next three chapters to answer the RQs in Chapter 1. The methodology focuses specifically on empirical studies because in software engineering they are important in supporting decision-making in organisations and improving software development [108].

The first empirical study in Chapter 4 uses the UIMS and QUES datasets [9], which are the most frequently used, as seen in the SLR of Chapter 2, and this helps to compare and evaluate the investigated prediction models in this chapter with previous studies. The second empirical study in Chapter 5 utilises the bug prediction datasets [57], which are larger and more recent than the UIMS and QUES datasets. The third empirical study in Chapter 6 employs refactoring datasets [58]; according to Chapter 2, these datasets are the newest datasets for software maintainability prediction. To validate the results, statistical tests and effect size measurements were performed in all empirical studies. In addition, to improve the results, the impact of the parameters tuning is explored. The reason for selecting each element in this methodology is provided in each section below, whereas the research method and experimental dataset setup are provided in the next three chapters separately.

The design of this methodology is based on the findings obtained in Chapter 2. Therefore, this thesis particularly focuses on the study of the prediction of software maintainability, as there were relatively few activities in the area of software maintainability prediction compared with other software quality attributes. Additionally, ensemble models demonstrate increased prediction accuracy over individual models, and can be useful in predicting software maintainability. However, as their application in selected primary studies is limited, they should be applied, as well as other models, to an extensive variety of datasets to improve the accuracy and consistency of results. Furthermore, according to Chapter 2, there is a requirement to use publicly available datasets for prediction software maintainability to

make the predictive models refutable, confirmable and repeatable [109]. In addition, product metrics at the class level are the majority of metrics used in selected primary studies (see Figure 2.7); therefore, these metrics were used as the predictors for software maintainability in this thesis. Finally, there is a demand to predict change proneness, which is used by limited selected primary studies in Chapter 2.

In this chapter, the individual prediction models used in the empirical studies are described. The ensemble prediction models are also detailed, followed by a description of automatic parameter tuning. A brief overview of the datasets used in the empirical studies and prediction accuracy measures, along with baseline, criteria, statistical tests, and effect size measurement are presented. The validation technique performed to create the models and the tools used to implement and analyse the empirical studies are provided.

3.1. Individual Prediction Models

In general, the choice of individual models was based on popularity, good performance, and selection from different categories. In addition, each individual model has its own advantage. The models selected are commonly used for regression problems. Some of these models, such as RT, KNN, and SVR, are listed amongst the best ten data mining models [110]. RT is an unstable model because even small changes in the training set may lead to considerable changes in the model's prediction [111], and it has the ability to create correlated features. However, these selection features are not based on the effect of the independent variables on the dependent variable [112]. MLP is a simple neural network configuration that can effectively manage datasets that are not linearly separable; however, it requires long execution and cannot predict the minimum time to stop [113]. KNN is simple to perform and easy to understand, but it does not work properly if the datasets have outliers, noise, or missing values neighbours [114]. M5Rules is categorised from the tree and has some strengths and weaknesses, but it has demonstrated better performance than other tree models. M5Rules produces a series of M5 trees that contain sets of leaves or rules, and the best rule is maintained from each tree, whereas other trees, such as M5P, create only an individual decision tree [115]. SVR determines the preferable hyperplane without dependence on the dimensionality of the input space and has demonstrated superior performance with excellent accuracy prediction.

However, it does not work well with large datasets because it requires a long time to execute [116, 117].

Regarding classification problems, the most frequently observed individual models (i.e., NB, SVM and KNN) in Table 6.1, which summaries of FS, datasets and prediction models in software quality prediction were used. Additionally, these models are among the best five models for classification problems [118]. NB can estimate the parameters and a model from a smaller proportion of the dataset, and then, it produces means and variances of the variables for each class. Nevertheless, it assumes all features as independent from each other [119].

Each one of the individual models proposed has its advantages and drawbacks. However, there is no obvious evidence of which models are more suitable to predict software maintainability accurately. With the goal of improving prediction accuracy, this thesis used RT, MLP, KNN, M5Rules and SVR in Chapter 4 and Chapter 5, along with NB, SVM and KNN in Chapter 6 as the individual models to predict software maintainability. These models were created in the three empirical studies in this thesis using WEKA [120], and their parameters were initialised by applying the default values because this procedure was observed in several studies in software maintainability in WEKA [12, 13, 16, 47, 121]. In the empirical studies, these models were used as the base in the ensemble model. Individual prediction models from different categories were used with the aim of creating an effective stacking ensemble model [122]. In addition to the default parameter values used as a main method, the impact of automatically parameters tuning was assessed as a sub-section in each empirical study. Table 3.1 presents a summary of individual prediction models with their category and name in WEKA.

Table 3.1: Summary of selected individual prediction models.

Model Name	Model Acronym	Category in WEKA	Name in WEKA
Regression Tree	RT	Trees	REPTree
Multilayer Perceptron	MLP	Functions	MultilayerPerceptron
K-Nearest Neighbors	KNN	Lazy	IBK
M5Rules	M5Rules	Rules	M5Rules
Support Vector Regression	SVR	Functions	SMOreg
Support Vector Machine	SVM	Functions	SMO
Naive Bayes	NB	Bayes	NaiveBayes

3.1.1 Regression tree

RT is constructed using the binary recursive partitioning process. This is a repetitive process that recursively divides a dataset into partitions or branches by selecting at each stage the independent variables that have the lowest minimum sum of the squared deviations from the mean of the two separate partitions. RT measures the prediction error by calculating the squared difference between the predicted and actual values [112]. This process continues to divide nodes until the total sum of squared deviations from the average is equal to zero, which is called the terminal node [112].

3.1.2 Multilayer perceptron

MLP is an artificial neural network that includes input and output layers, along with hidden layers. Each layer comprises one or several nodes (neurons) that connect with each other by a specific weight. The hidden layers combine input data, which is a set of features, by utilising a linear combination [123]. The hidden layer converts the value from the previous layer (input layer) by using a weighted linear summation, whereas the output layer collects the value from the previous layer (hidden layer). The activation function of the hidden layer enables the capture of relationship between inputs and outputs and resolves the nonlinearity problem between them. Also, this function transfers weighted input to output. MLP uses the backpropagation algorithm to construct a neural model from historical training data [123]. One of the main characteristics of the backpropagation algorithm is to estimate the error rate in the output nodes and predict the results. This estimation performs by computing the total loss and determining the possibility of the loss into each node. Consequently, this algorithm changes the connected weights between the input and output by decreasing the loss between them and assigning lower weights to the nodes with higher error and vice versa [123].

3.1.3 K-Nearest neighbours

KNN, also called instance-based learning, is constructed by choosing the closest neighbours in the training data to predict the target data. First, the KNN algorithm stores all training instances by applying a linear transformation to normalise the values in the range from 0 to 1. In sequence, the algorithm categorises data by selecting the majority class of the k closest

neighbours or the nearest training instance. This process typically uses the Euclidean (or other specified) distance measure to determine the closest neighbours [114].

3.1.4 M5Rules

M5Rules is built by generating a series of M5 trees, which is a decision tree designed to predict numerical values for regression problems [124]. The M5 tree includes groups of leaves or rules using a separate-and-conquer strategy. This is an iterative process that constructs a model tree by utilising the M5 algorithm and selecting the best leaf to transform into a rule. Initially, the M5 algorithm constructs a tree by partitioning the data based on the values of the predictive attributes [115]. In sequence, the M5 algorithm calculates a linear model for each node by computing the average of the absolute difference between the actual and predicted values of every observations in the training set [115]. This process typically continues until all instances have one or more rules. M5Rules has a regression model in its nodes to predict the value, whereas the RT has only a constant fitted mean. Therefore, as M5Rules slightly differs from RT, they can be used to make a different prediction [115].

3.1.5 Support vector machine

SVM is designed to solve classification problems and is considered a category of generalised linear classifiers. SVM converts the original dataset training to a higher dimension using nonlinear mapping [125]. Then, it creates a linear optimal separating hyperplane to separate the dataset into two classes. Furthermore, there are two lines that create a maximal margin hyperplane, whose instances define the boundary line. Sequential minimal optimisation was used in Weka as the SVM implementation, and it has several features, such as the ability to manage very large datasets and faster model creation for sparse datasets and linear SVM [126]. Additionally, it can manage complex nonlinear decision boundaries and is less prone to overfitting than other models. These features help SVM reduce prediction error and improve overall prediction accuracy [125].

3.1.6 Support vector regression

SVR is a specific class of SVM designed to solve regression problems. SVR has all the major features of SVM used for the classification problem. Both algorithms aim to decrease

prediction error, increase the maximal margin, and build the hyperplane in the high-dimensional space. Additionally, two lines create a margin, namely boundary lines. SVR is implemented by initiating an ϵ -insensitive region around the function (i.e., ϵ -tube). Initially, SVR defines a symmetrical loss function (i.e., ϵ -insensitive loss function) to be reduced. Then, it determines the narrowest tube that includes most of the training instances. SVR is considered an optimisation problem because it addresses the convex optimisation using numerical optimisation algorithms [116, 117].

3.1.7 Naive Bayes

NB is a probabilistic algorithm that relies on Bayes' theorem to predict the class for each row. NB applies independence assumptions, which consider features to be independent of each other. This algorithm uses estimator classes, and this estimation is performed using the maximum likelihood method [119].

3.2. Ensemble Prediction Models

Ensemble prediction models aim to overcome the deficiencies of individual models (typically variance and instability) by combining a set of models to generate a final prediction. Typically, such a model can be of two types: homogeneous and heterogeneous. Homogeneous models combine individual models of the same type (e.g., bagging), whereas heterogeneous models combine individual models of different types (e.g., stacking). The combination rule depends on the problem type and can either be weighted averaging (regression problem) or majority voting (classification problem) [105]. The advantages of the ensemble prediction model are an increase in accuracy prediction compared to individual models and prevention of overfitting. Figure 3.1 illustrates the homogeneous and heterogeneous ensemble prediction model structures.

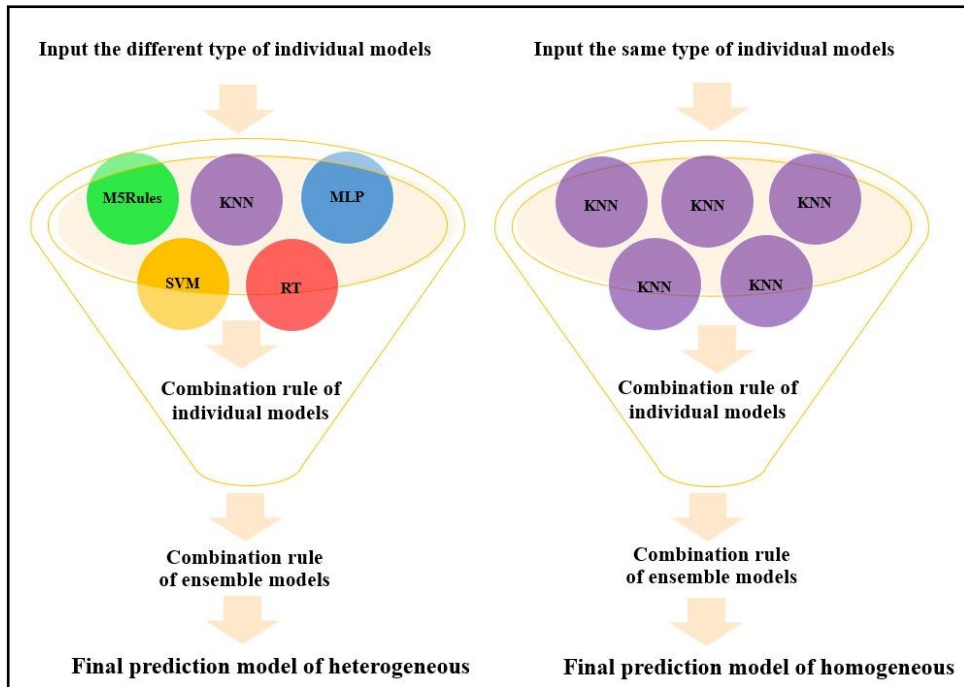


Figure 3.1: Structure of homogeneous and heterogeneous ensemble prediction models.

There are three main steps in creating an ensemble prediction model. First, several base models are created in sequential or parallel styles based on the ensemble type [105]. Second, all individual models are integrated using the combination rule [105]. If the output of individual models has equal weight, a simple average is performed to combine the models. Alternatively, if the output of individual models has different weights, the individual models accumulate their weight and assign the highest weight to the lowest root-mean-square error of the individual [105]. Third, the output of the previous combination is integrated to perform the ensemble combination [105, 127]. With respect to the design of the ensemble arbitrator, the ensemble model includes two types of classification: non-linear and linear ensemble models. In the linear ensemble, the arbitrator integrates the outputs of the base models in a linear method (e.g., averaging or weighted averaging), whereas in the non-linear ensemble, the arbitrator integrates the outputs of the base models in a non-linear method (e.g., MLP), thus this method does not require assumptions for collecting inputs [106]. Table 3.2 provides a comparison between the homogeneous and heterogeneous ensemble models.

Table 3.2: Similarities and Differences between homogeneous and heterogeneous ensemble models.

	Homogeneous	Heterogeneous
Similarities	<ul style="list-style-type: none"> Combine several base models. Aim to increase accuracy prediction of individual models. 	
Differences	Uses the same model type.	Uses different model types to improve prediction accuracy.
	Uses different training data to obtain a different model.	Uses the same training data.
	Uses the same FS method as a part of the homogeneous ensemble method.	Uses various FS methods as a part of different base models combined in the heterogeneous ensemble model.

In this thesis, the most popular and well-known ensemble models were selected, namely bagging [128], additive regression [129], stacking [130], APE [131], and RF [132]. RF was the most commonly used ensemble model in the SLR in Chapter 2 and Table 6.1 in Chapter 6, which summaries FS, datasets and prediction models in software quality prediction, followed by bagging in the SLR in Chapter 2. Additionally, RF produced the best performance in change-proneness prediction [47] and software fault prediction [37], along with APE in software fault prediction [131].

3.2.1 Bootstrap aggregating (Bagging)

Bagging is an ensemble technique that enhances the prediction accuracy of a model by creating separate models of the same type. Initially, the bagging algorithm builds the individual models, which have an equal weight by randomly sampling subsets of the training set iteratively with replacement. The bagging algorithm collects the results of these models by using voting with a classification problem and averaging with a regression problem [128]. The bagging algorithm improves the performance of unstable models such as RT[133], thus two types of trees were investigated when applying bagging. Moreover, bagging is recommended for use in small-sized training sets such as QUES and UIMS datasets to decrease the variance between the base models that cause the unstable model problem [133]. The following three major parameters must be determined in this algorithm:

- Base model: the individual model to be used in the bagging algorithm;
- Ensemble size: the number of individual models to be created in the bagging algorithm;
- Training set size: the size of the dataset used to construct the individual models [128].

3.2.2 Additive regression

Additive regression is an ensemble technique to increase the performance of a base regression model of the same type. This technique is considered a specific case of gradient boosting, as it begins with an empty ensemble and then augments an initial model with subsequent models that aim to correct the residuals (i.e., errors) in the predictions of the previous model (or models) using least-squares at each repetition. In each repetition, the ensemble model creates a model to the errors remaining using the model of the prior repetition. A part of the training data in each iteration is randomly selected without replacement, for later use as a complete sample [129]. Therefore, additive regression is created by adding the prediction of all models together, which leads to a more accurate model. Additive regression decreases the shrinkage parameter, which manages the shrinkage rate (learning rate) of the procedure (this parameter ranges from 0 to 1, where small values indicate better performance [134]), and avoids overfitting. However, it increases the learning time [129, 135]. The following three major parameters must be identified in this algorithm:

- Base model: the individual model used as the base model in the additive regression algorithm;
- Shrinkage rate: shrinkage rate or learning rate is a method to decelerate the learning model by performing a weighting factor. This rate should be minimised to improve performance and avoid overfitting;
- Ensemble size: the number of repetitions in the additive regression algorithm[129, 135].

3.2.3 Random forest

RF is an ensemble model that builds a forest of numerous unpruned decision trees from the training dataset. Then, it uses the mode of the classes of the individual trees on the testing dataset to make a prediction. RF is called random because a random sample of the training data is selected with repetition and forest because it involves several decision trees. Applying

this random selection of features leads to more error rates than in the AdaBoost [136]. However, the RF is better in terms of managing noisy data [132]. Moreover, this algorithm performs bagging on features based on majority voting and selects the dependent variables that have the highest votes [132]. In this study, RF integrates algorithms of the same types (i.e., decision trees), thus it can be classified as a homogenous ensemble model. The default parameters in Weka were applied, in which Weka creates a forest of several decision trees as the base models and initialises a forest to 100 tree instances [137]. RF depends on four parameters: the number of trees to construct, the subsample size common to each tree, the tree depth, and the number of variables randomly sampled for splitting [138].

3.2.4 Stacking

Stacking is an ensemble technique that improves prediction accuracy and decreases variance by integrating several models of different types. Stacking works effectively when the base models have significantly varied categories, such as a combination of neural network, tree-based, and support vector models. The stacking algorithm starts by applying an entire training set to train different model types. Each model at this level produces predictions with specific features. In addition, a meta-model is trained on these predictions to generate the final prediction in a second-level model [130]. Based on the stacking features suggested in the literature [139], linear regression was used as the combination method and RT, MLP, KNN, M5Rules and SVR as the base models. The remaining parameters were initialised by applying the default values in WEKA. The following four major parameters must be determined in this algorithm:

- Base model: the individual models to be used in the stacking algorithm;
- Meta model: the ensemble model that combines the results of the individual models;
- Ensemble size: the number of individual models in the stacking algorithm [130].

3.2.5 Average probability ensemble

APE is a type of heterogeneous ensemble technique that integrates a group of models from different types and produces a single output. This technique takes advantage of several individual models to remove prediction errors and improve accuracy prediction. The APE

calculates the average for all base models (i.e., individual models) and provides the result as a single output [131, 140].

3.3. Parameters tuning

Parameters tuning technique is the process of changing the parameter settings that control the features of the machine learning models, such as the size of each bag in RF or the number of nearest neighbours in KNN [141]. This process aims to improve performance and reach optimal results. However, limited studies in software quality prediction have investigated the impact of parameter tuning [142], and no studies discussed in the SLR in Chapter 2 have used parameter tuning. A possible explanation for this limitation might be that parameter tuning requires considerable time and effort. For example, tuning all the parameters of KNN requires at least 17000 different configurations [143]. To resolve this issue, parameter tuning is proposed to evaluate and compare various parameter configurations and select the best configurations that achieve the highest prediction accuracy. Therefore, three different methods of automatic parameter tuning were evaluated. The following subsections present a description of these methods.

3.3.1 Caret package

Caret, which is an abbreviation for classification and regression training, is a package available in R to create prediction models and automatically tune their parameters [144]. Various features have been developed and included in caret package, such as data pre-processing, data splitting, feature importance, feature selection, parallel processing, visualisation and model tuning [145]. In this thesis, visualisation was used to present the correlation between metrics in the datasets in Section 4.4.5 and Section 5.4.5 and model tuning was performed to investigate the impact of parameter tuning in Section 4.5.3 and Section 6.5.5, whereas other features of caret package were not used in this thesis. Caret package is one of the more practical ways of performing parameter tuning due to its ability to evaluate the impact of parameters, select an optimal model and estimate the prediction accuracy of models [145]. In particular, this package evaluates and compares several combinations of parameter tuning and chooses the optimised setting that performs with the highest prediction accuracy [141]. The caret

package creates and predicts models by removing the syntactical differences between several functions. Among these functions, *the train control* function evaluates the performance of a trained model using validation methods [144]. In this study, ten-fold cross validation was performed (i.e., CV in the caret package). Therefore, parameter tuning is based on the training sets, which are divided into a test set and a training set in the validation method. Additionally, *the train* function selects the *method* (i.e., prediction models) and *metrics* (i.e., evaluation measurements). The values of the parameters tuning in each prediction models were automatically selected by using a grid of tuning parameters. The grid size by default is 3^p , where p is the number of parameters tuning in a given model [146]. For instance, two parameters (gamma and lambda) are included in regularized discriminant analysis model, so the grid size in this model generates nine combinations of these two parameters [146]. Therefore, the parameter space in caret is not explored exhaustively. Also, the types of parameter are chosen automatically using default types proposed in the caret package. For example, Caret tries different values of the nearest neighbours (K) parameter in KNN; then, it selects the optimal model that records the highest prediction accuracy. Similarly, caret performs a combination of two parameters tuning in SVM, namely the scale function and the cost value to control radial basis function and the complexity of the boundary, respectively [147]. However, the values and types of parameters can be manually identified inside *the train* function. In this thesis, the individual models and the ensemble models that used these individual models as the base model are tuned automatically using the caret package. All prediction models in Chapter 4 were created again using the caret package to compare their performance with the default values created by WEKA. Furthermore, all prediction models except RT require other packages, along with caret. For example, the RWeka package is used with caret to initialise the base model parameter inside the ensemble model [148]. Table 3.3 lists the packages and methods used to create the prediction models in Chapter 4. In this table, all the models (i.e., base and ensemble models) and all packages (including models RWeka package) are created and tuned by R.

Table 3.3: Overview of packages and methods used to create prediction models.

Prediction model	Package	Methods	Base model parameter in ensemble model
RT	caret	rpart	NA
MLP	caret, RSNNS and Rcpp	mlp	NA
KNN	caret, kknn	knn	NA
M5Rules	caret, rJava, RWeka and RWekajars	M5Rules	NA
SVR	caret, e1071	svmLinear2	NA
Bagging	caret, rJava, RWeka and RWekajars	weka/classifiers/meta/Bagging	RT:weka.classifiers.trees.REPTree MLP:weka.classifiers.functions.MultilayerPerceptron KNN:weka.classifiers.lazy.IBk
Additive regression	caret, rJava, RWeka and RWekajars	weka/classifiers/meta/AdditiveRegression	M5Rules:weka.classifiers.rules.M5Rules SVR:weka.classifiers.functions.SMOreg
Stacking	caret and caretEnsemble	caretList	algorithmList=(rpart, mlp, knn, M5Rules, svmLinear2)

3.3.2 Auto-WEKA

Auto-WEKA, used in Chapter 5, is an automatic tool that implements several types of machine learning models with different integrated selected features and tuning parameters in WEKA [149, 150]. This tool tries different hyperparameter settings and selected features for several models and provides the best model performance using the Bayesian optimisation method [149, 150]. Recently, Auto-WEKA was combined with WEKA as a package and was constructed to perform regression algorithms, performance metrics and parallel runs [149].

3.3.3 Grid search

In Chapter 6, parameter tuning is performed on RF using a grid search with ten-fold cross-validation. Grid search is the process of exploring the search space of the hyperparameter values with the specification of parameter pairs and evaluation measurement (e.g., AUC). This process is iterated via ten-fold cross-validation until the optimal hyperparameters are determined, which results in the highest prediction accuracy [151]. Grid search is created using the tuneGrid function in R, along with the randomForest, mlbench, and caret packages to build the RF model. RF depends on four parameters: the number of trees to construct, the subsample size common to each tree, the tree depth, and the number of variables randomly

sampled for splitting [138]. However, only the last parameter was tuned (i.e., the number of variables randomly sampled for splitting), using `Mtry` variable in R. The rationale for focusing on just the `Mtry` parameter is because there is no clear indication or theory of which value of this parameter is more appropriate under most circumstances [138]. Therefore, the grid search used in Chapter 6 is considered a linear search, in which the vector of candidate values ranges from 1 to 15. These values were initialised because there is no recommendation to select the number `Mtry` parameter [138].

3.4. Datasets

This thesis used sets of three types of datasets that are publicly available and suitable for software maintainability prediction. Public datasets were selected to the empirical studies to enable comparison and reproducibility, which helps to improve research in the software engineering area. In this thesis, appropriate independent variables (metrics) were selected to predict software maintainability along with the dependent variable (maintainability). The details of the selected variables are explained in the experimental data setup in Chapters 4, 5, and 6, and the general overview of these datasets is provided in the next sections.

3.4.1 Change maintenance efforts

In Chapter 4, change maintenance efforts datasets (i.e., QUES and UIMS), proposed as an appendix in [9], were used. These datasets include the `CHANGE` metric collected from three years of software maintenance [9]. The QUES and UIMS datasets are publicly available, accurately validated, and widely used in software maintainability prediction studies [7, 11-13, 16, 18, 88, 152]. Therefore, they were selected to enable comparison with previous studies and contribute to the field of OO software maintainability prediction. Classic-Ada is an OO programming language developed by Software Productivity Solutions, Inc. QUES and UIMS are software systems written using the Classic-Ada language. Moreover, a Classic-Ada metric analyser tool was used to extract metrics and build datasets from these systems [9]. Class-level metric data of 71 and 39 classes were collected for the QUES and UIMS datasets, respectively. Table 3.4 lists the definitions and descriptions of the OO metrics selected in this study.

Table 3.4: Definitions and description of L&H metrics [9].

Independent variable	Definition	Description
DIT	Depth of inheritance tree	This metric determines the depth of a class in the hierarchy by calculating the length of the path from the root class, and the root class is zero in the class hierarchy.
NOC	Number of children	This metric calculates the total number of child classes that inherit directly from a given class.
MPC	Message-passing coupling	This metric calculates the total number of messages sent out from a class.
RFC	Response for a class	This metric calculates the sum of the total number of local methods, along with the number of methods called by local methods in the class
LCOM	Lack of cohesion in methods	This metric calculates the total number of disjoint sets of local methods via at least one instanced variable and one member of the disjoint set, whereas the disjoint sets are a group of sets that do not intersect with each other [9].
DAC	Data abstraction coupling	This metric calculates the total number of abstract data types that are instances of another class declared within a class.
WMC	Weighted methods per class	This metric calculates the total number of McCabe's cyclomatic complexity of all methods in a class.
NOM	Number of methods	This metric calculates the total number of local methods defined in a class.
SIZE1	Lines of code	This metric calculates the total number of semicolons in a class.
SIZE2	Number of properties	This metric calculates the total number of attributes and local methods defined in a class.

3.4.2 Bug prediction datasets

In Chapter 5, five datasets publicly available, primarily designed to support the problem of bug prediction, were used. These datasets were extracted from five open-source software systems: Eclipse JDT Core (997 classes), Eclipse PDE UI (1,562 classes), Equinox framework (439 classes), Lucene (691 classes) and Mylyn (2,196 classes) [57]. They were collected at the class level and included a collection of several metrics, bug changes, and version information about the system [57]. These datasets are composed of the data collected from the CVS change log at biweekly intervals, and include classified post-release defect counts extracted from each class in the system, along with a collection of 17 source code metrics (OO and CK metrics) and 15 metrics calculated from CVS change log data for each class in the system [153]. These datasets (initially proposed in 2010) have been primarily used in software defect prediction studies [154, 155]. However, no previous study has investigated these datasets for the prediction of software maintainability. Table 3.5 lists a brief description of each metric (independent variable) used in this study.

Table 3.5: Summary of class level source code metrics [57].

CK metrics	
LCOM	Lack of cohesion in methods
NOC	Number of children
DIT	Depth of inheritance tree
CBO	Coupling between objects
RFC	Response for class
WMC	Weighted method count
OO metrics	
NOMI	Number of methods inherited
NOPM	Number of public methods
LOC	Number of lines of code
NOPRA	Number of private attributes
NOA	Number of attributes
FanIn	Number of other classes that reference the class
NOPRM	Number of private methods
NOM	Number of methods
NOAI	Number of attributes inherited
NOPA	Number of public attributes
FanOut	Number of other classes referenced by the class

3.4.3 Refactoring datasets

In Chapter 6, seven publicly available datasets published in [58] and collected from class level were used, namely antlr4 (436 classes), junit (657 classes), MapDB (439 classes), mcMMO (301 classes), mct (2162 classes), oryx (536 classes), and titan (1486 classes). The datasets were initially collected to investigate the impact of code refactoring (changes made to the structure of the internal source code which don't affect the functionality or external behaviour of the code [53]) on maintainability, and the original datasets contain source code metrics including refactoring metrics, along with a score for maintainability at both method and class levels [58]. These datasets were collected from a total of 37 subsequent releases of systems from seven open-source Java systems located in GitHub [27] and were combined into one manually validated dataset for each of the seven systems [58]. To the best of the author's knowledge, these datasets are considered the newest datasets in software maintainability prediction and have not been utilised in previous studies to predict change-proneness. Table

3.6 lists the metrics used as independent variables and their category, and the description of their abbreviations is provided in Table C. 1 in Appendix C.

Table 3.6: Metrics used as independent variables and their categories [156].

Category	Metrics
Cohesion	LCOM5
Complexity	NL, NLE and WMC
Coupling	CBO, CBOI, NII, NOI and RFC
Documentation	AD, CD, CLOC, DLOC, PDA, PUA, TCD and TCLOC
Inheritance	DIT, NOA, NOC, NOD and NOP
Size	LOC, LLOC, NA, NG, NLA, NLG, NLM, NLPA, NLP, NLS, NM, NPA, NPM, NS, NOS, TLOC, TLLOC, TNA, TNG, TNLA, TNLG, TNLM, TNLPA, TNLPM, TNLS, TNM, TNPA, TNPM, TNS and TNOS
Code duplication	CCL, CCO, CC, CI, CLC, CLLC, LDC and LLDC
Warning	WarningBlocker, WarningCritical, WarningInfo, WarningMajor and WarningMinor
Rules	Android, Basic, Brace, Clone implementation, Clone metric, Code size, Cohesion metric, Comment, Complexity metric, Controversial, Coupling metric, Coupling, Design, Documentation metric, Empty code, Finalizer, Import statement, Inheritance metric, J2EE, JUnit, Jakarta commons logging, Java logging, JavaBean, MigratingToJUnit4, Migration, Migration13, Migration14, Migration15, Naming, Optimization, Security code guideline, Size metric, Strict exception, String and StringBuffer, Type resolution, Unnecessary and Unused Code and Vulnerability
Refactoring	REMOVE_PARAMETER, ADD_PARAMETER, REPLACE_MAGIC_NUMBER_WITH_CONSTANT, REMOVE_ASSIGNMENT_TO_PARAMETERS, INTRODUCE_EXPLAINING_VARIABLE, INLINE_TEMP, REMOVE_CONTROL_FLAG, CONSOLIDATE_COND_EXPRESSION, CONSOLIDATE_DUPLICATE_COND_FRAGMENTS, REPLACE_NESTED_COND_WITH_GUARD_CLAUSES, INLINE_METHOD, EXTRACT_METHOD, REPLACE_EXCEPTION_WITH_TEST, INTRODUCE_ASSERTION, RENAME_METHOD, REPLACE_METHOD_WITH_METHOD_OBJECT, MOVE_METHOD, HIDE_METHOD, INTRODUCE_NULL_OBJECT, INTRODUCE_LOCAL_EXTENSION, EXTRACT_SUPERCLASS, EXTRACT_INTERFACE and MOVE_FIELD

3.5. Prediction Accuracy Measures

This section presents the prediction accuracy measures used in the empirical studies, which include measures for regression and classification problems.

3.5.1 Measures for the regression problem

Prediction accuracy measures are applied to evaluate and compare OO software maintainability prediction models. These measures are adopted from the standard measures; generally, in the scope of regression problems and specifically in the field of OO software maintainability prediction, the following evaluation measures are used.

MRE [35] is calculated by the absolute difference between actual and predicted values, and further dividing the difference by the actual values, as seen in Eq. (3.1).

$$MRE = \frac{|\text{actual value} - \text{predicted value}|}{\text{actual value}} \quad (3.1)$$

MMRE is the mean of MRE, where n represents the number of observations over a dataset, as seen in Eq. (3.2).

$$MMRE = \frac{1}{n} \sum_{i=1}^{i=n} MRE \quad (3.2)$$

Pred [67] is the proportion of all the instances in the dataset where the MRE is less than or equal to a specified value, usually 25% or 30%, as recommended in software effort prediction studies [34, 67]. The pred value is calculated as shown in Eq. (3.3).

$$Pred(q) = \frac{k}{n} \quad (3.3)$$

where q is a defined value, k is the number of instances where the MRE is smaller than or equal to q , and n is the number of instances in the whole dataset.

MAE [65] is the average of the absolute values of the difference between $X'i$ and Xi , where $X'i$ is the predicted value and Xi is the actual value, as seen in Eq. (3.4).

$$MAE = \frac{1}{n} \sum_{i=1}^{i=n} (|X'i - Xi|) \quad (3.4)$$

SA [99] is proposed based on the mean absolute residual (MAR), as seen in Eq. (3.5).

$$SA_{pi} = 1 - \frac{MAR_{pi}}{MAR_{p0}} \times 100 \quad (3.5)$$

MAR_{pi} is the mean absolute residual of the prediction model, and $\overline{MAR_{p0}}$ is the mean value of random guessing. This random guessing is repeated a considerable number of times; Shepperd and MacDonell [99] suggested 1000 runs. However, they stated that a considerable number of random guessing would produce the same result as using the sample mean [99]. Therefore, $\overline{MAR_{p0}}$ was computed from the baseline, presented in Section 3.5.3.

These measures have emerged as the de facto common accuracy measures, namely MMRE, Pred (q) and MAE. According to Chapter 2, MMRE and Pred (q) were used by ten selected primary studies, whereas MAE was used in six studies. Moreover, MMRE and Pred (q) measures are usually applied in empirical software engineering studies [20, 157] and were employed in past [71], recent [18], and several other studies on software maintainability prediction [7, 11-13, 16]. However, MMRE has a bias issue towards models that provide underestimated results; it is also an unreliable measure and does not always determine the most accurate model [158, 159].

Although $\text{Pred}(q)$ is based on the MRE measure, it is considered reliable and less sensitive to the variance and outliers of MRE values [160]. However, Kitchenham et al. stated that MMRE and Pred evaluate only the spread and the kurtosis of the residuals' values (i.e., predicted divided by actual values). Hence, they also recommended evaluating the central location and skewness of these values [35]. Korte and Port reported that the spread and kurtosis parameters are logical and good indicators for the prediction accuracy [160]. To date, there are no commonly accepted alternative measurements to measure software effort. To overcome these limitations, researchers have suggested further measurements, such as creating a baseline of the predicted values [99], visualising the boxplots of the residuals [35, 161] and performing statistical tests based on the residuals and effect size [100]. MAE is also suggested to be used because it is unbiased and does not depend on ratios, as the MMRE [99]. However, MAE is based on residuals, which are not standardised, thus it is difficult to explain and evaluate among several datasets [99]. SA was suggested as a response to this issue, as it depends on MAR, which is the same as MAE [99]. All these suggestions are considered in the empirical studies in Chapters 4 and 5.

3.5.2 Measure for the classification problem

Many measures have been published in the literature to estimate the prediction accuracy of models in software engineering problems [67]. In Chapter 6, only one prediction accuracy measure, AUC, is performed to compare and evaluate the performance of prediction models, and it ranges from 0 to 1. AUC is based on the ROC that graphs the FPR on the x-axis against the TPR on the y-axis at various threshold settings [70].

Eq. (3.6) calculates the value of AUC [70], where i represents observations, $(1 - \beta)$ represents $(\text{TPR}) = \frac{TP}{TP+FN}$, α represents $(\text{FPR}) = \frac{FP}{FP+TN}$, and these values are extracted from the confusion matrix presented in Table 3.7.

$$AUC = \sum_i \left\{ (1 - \beta_i) \cdot \Delta\alpha + \frac{1}{2} [\Delta(1 - \beta) \cdot \Delta\alpha] \right\} \quad (3.6)$$

where $\Delta(1 - \beta)$ indicates $(1 - \beta_i) - (1 - \beta_{i-1})$, and $\Delta\alpha$ indicates $\alpha_i - \alpha_{i-1}$.

According to a systematic review of FS techniques [4], AUC was the main and most frequently used evaluation measure for classification problems in software quality prediction.

In addition, it is considered well-known and commonly employed in software maintainability prediction [48, 74, 162], along with change-proneness prediction [16].

Table 3.7: Confusion matrix [163].

	PREDICTED CLASS (NO)	PREDICTED CLASS (YES)
ACTUAL CLASS (NO)	True negatives (TN)	False Positive (FP)
ACTUAL CLASS (YES)	False Negative (FN)	True Positive (TP)

3.5.3 Baseline

The baseline measure is used as a benchmark to evaluate the performance of the predictors with the dependent variable only (e.g., CHANGE metric or change-proneness) that disregards other independent variables. The ZeroR model, which is implemented to determine a baseline, relies on the mean of the predicted values for the regression problem or majority category (i.e., the mode values) for the classification problem [164]. Therefore, this model is performed as a reference to investigate the improvements of the prediction models, but it does not contribute to the prediction values [165]. Several studies used the baseline [35, 72, 166, 167], which relies on either the mean [167], median [72], boxplots for the residuals [35, 161], or the ZeroR model [71, 166]. Fernández-Delgado et al. performed 179 classifiers on 121 datasets and used ZeroR as a percentage of the majority class in the classification problem to evaluate the classifiers [166].

In this thesis, baseline is used in the empirical studies to predict the mean value for the regression problem in Chapters 4 and 5. The majority classes for the classification problem in Chapter 6 and the boxplots of the residuals, which are based on actual predicted values, are used as further measures to evaluate the predicted models. Overall, the use of all these measurements (i.e., MMRE, Pred(q), MAE, SA and AUC) as evaluation measures, along with the baseline and the boxplots ensures an increase in conclusion stability and avoids choosing inferior models [100, 168].

3.5.4 Criteria

The suggested criteria to build an accurate effort prediction model for regression problems are $\text{Pred}(0.30) \geq 0.70$ [34] or $\text{Pred}(0.25) \geq 0.75$ or/and $\text{MMRE} \leq 0.25$ [67]. However, if the prediction models do not meet the proposed criteria, the prediction model is still acceptable

because it is hard to construct an accurate model for software maintainability prediction that satisfies these criteria [20]. Although the suggested criteria were proposed in old studies [34, 35], several recent studies are still being performed to evaluate prediction accuracy in the software engineering problem [102-104]. In terms of classification problems, AUC extends from 0 to 1, a higher value indicates better results and 1 is the optimal result (a perfect classifier). Additionally, 0.5 indicates no discrimination, a value from 0.7 to 0.8 indicates an acceptable result, a value from 0.8 to 0.9 is recognised as excellent, and any values higher than 0.9 are considered outstanding results [169].

3.5.5 Statistical tests and effect size

The test of significance is used to validate the results according to a defined hypothesis. The one-way analysis of variance (ANOVA) F test [170] was carried out using the residuals in Chapters 4 and 5 and AUC values in Chapter 6. ANOVA was selected because there were more than one pair of variables analysed. This test was performed to investigate whether the group population means (i.e., performance of the prediction models) were significantly different between each individual model and ensemble models. Factor A indicates the prediction models, grouped by each individual model, and ensemble models. For example, the prediction accuracy difference (better or worse) was analysed to evaluate whether it was significant between RT as the individual model and bagging, additive regression, stacking and APE as the base model. ANOVA evaluates the relevance of the evidence against the null hypothesis. The null hypothesis (H_0) states that there is no statistically significant difference in all the group population means, whereas the alternative hypothesis (H_1) states that there is a statistically significant difference in at least one pair of means. When H_0 is rejected, H_1 is accepted. A statistically significant result for the ANOVA experiment is typically defined as $\alpha = 0.05$, and the p-value was evaluated and compared with this defined value. If the p-value is smaller than α , H_0 is rejected. Consequently, a smaller p-value obtained by the results provides evidence against H_0 . However, a larger p-value is not evidence that H_0 is true, which may sometimes happen due to the data analysed.

Therefore, to further understand the strength of a result, the effect size is also used, which is considered an essential component for understanding the results of empirical studies [171]. Among the various effect sizes introduced in the literature, eta-squared (η^2) was selected

because it is a suitable measure for ANOVA [171]. Cohen proposed the standard classifications of the effect sizes, which are small (≈ 0.01), medium (≈ 0.06), and large (≈ 0.14) [172]. Eq. (3.6) computes the value of η^2 .

$$\eta^2 = \frac{SS_{\text{effect}}}{SS_{\text{total}}} \quad (3.7)$$

SS_{effect} is the sum of squares of the effect, and SS_{total} is the total sum of squares [171]. Additionally, if H_0 is rejected, multiple comparisons are applied using a plot chart of Tukey's confidence intervals [170] to identify which pairs of Factor A are significantly different. If a confidence interval does not include 0, then the pair is significantly different.

3.6. Validation

An essential rule in building machine learning models is not to test against the datasets used in training [142]. Therefore, the ten-fold cross-validation method is employed in all the empirical studies in Chapters 4, 5, and 6 to build and predict models. This method is widely used across different machine learning problems. Furthermore, most selected primary studies (i.e., 48%) in Chapter 2 performed k-fold cross-validation. It separates the whole dataset into ten equal folds, where one fold is utilised to train, and the remaining are utilised for testing. This procedure was repeated ten times to select different folds for the test. Finally, the results of the iterations were averaged. Ten-fold cross-validation decreases the variance by averaging the validation accuracy for all ten partitions. Therefore, the final accuracy is less sensitive and has lower variation than that provided by other validation methods, such as the single hold-out method. The main advantage of this approach is the estimation of accurate performance [66, 125]. Figure 3.2 presents the ten-fold cross-validation implementation.

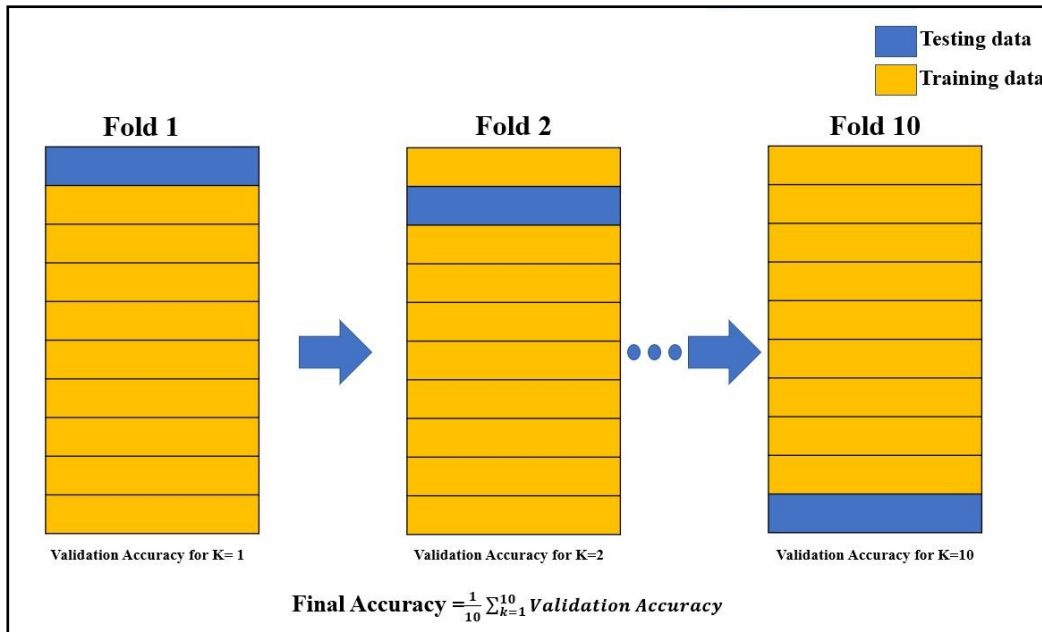


Figure 3.2: Ten- fold cross-validation.

3.7. Tools

Two tools were used, namely WEKA and R, to produce the results of the empirical studies. These tools are open source and freely available, and this motivates researchers to conduct research and replicate results. The description of these tools is presented in the following subsections.

3.7.1 WEKA

WEKA is a Java-based suite of machine learning software [120]. It has a collection of Java class libraries that implement several machine learning techniques. WEKA is an open source software that includes several features with a provision for data preparation, regression, classification, association rules, clustering and visualisation [120]. The empirical studies in this thesis use this tool to build and evaluate OO software maintainability prediction models by using default parameters.

3.7.2 R

R is a language for graphics and statistical computing that includes a different collection of statistical and graphical techniques for the analysis and visualisation of data [173]. It is a free

tool created by Ihaka and Gentleman and developed by the R Development Core Team [174]. The empirical studies in this thesis use this tool to visualise the correlation of the datasets and to investigate the impact of parameter tuning in Chapters 4 and 6.

3.8. Summary

This chapter presents the research methodology applied to predict software maintainability using ensemble techniques. The basic concepts of the prediction models, which include both individual and ensemble models, were provided. Subsequently, different methods of parameter tuning were introduced. An explanation of the prediction accuracy measures was demonstrated, with baseline and criteria used for comparison. Then, the validation used to build and predict models was determined. Finally, the tools used to perform the empirical studies were identified. The proposed research methodology provides the foundation for conducting empirical studies in this thesis. The next three chapters will show the application of this methodology in three different empirical studies, including various datasets, and a description of the experimental data setup.

Chapter 4. First Empirical Study: Ensemble Techniques to Predict Change Maintenance Effort Using Well-Established Datasets

This chapter empirically evaluates the performance of homogeneous (i.e., bagging and additive regression) and heterogeneous (i.e., stacking) ensemble models against a range of individual models (i.e., RT, MLP, M5Rules, KNN, SVR) when applied to the QUES and UIMS datasets that extracted from OO systems [9]. The primary objective is to investigate the capability of ensemble models to increase or decrease prediction accuracy over individual models. Furthermore, another objective is to identify the model that achieves the highest prediction accuracy and compare it with previous studies that operated on the same datasets. In addition, this chapter aims to investigate the impact of parameter tuning of the software maintainability prediction models using the caret package in R.

4.1. Introduction

Various software maintenance measurements have been used in selected primary studies in Chapter 2. The most critical observation is that most of these studies used the change maintenance effort measurement for predicting software maintainability, which uses the CHANGE metric as a dependent variable to capture the elements of software maintainability [9]. Therefore, this chapter used the CHANGE metric as the most selected in primary studies in Chapter 2, which measures maintainability of OO systems by calculating the total number of lines added and removed in each class during the maintenance process. The higher the number of CHANGE metrics, the higher the maintenance effort and the lower the maintainability [7, 11-13, 15-18, 88, 152]. Furthermore, Chapter 2 stated that several selected primary studies focused on employing machine learning on the QUES and UIMS datasets that include ten independent metrics (i.e., L&H metrics), along with one dependent metric (i.e., CHANGE metric). Although the QUES and UIMS datasets are old (i.e., since 1993) and small (i.e., contain 71 and 39 classes, respectively), they were used to make the prediction models

repeatable and comparable [9]. The present chapter makes several noteworthy contributions as follows:

- This empirical study used well-established datasets and provided additional evidence which indicates that the homogeneous models improved prediction accuracy compared to most investigated individual models in both datasets, whereas the heterogeneous model improved prediction accuracy compared to most investigated individual models in the QUES dataset only;
- This empirical study compared the best proposed model with the best model in the selected studies and found that KNN as the individual model, or as the base model in additive regression, achieved the best prediction accuracy not only amongst all investigated models in both datasets but also against the best model in the selected previous studies in the QUES dataset;
- This is the first study exploring the influence of parameter tuning in the QUES and UIMS datasets.

4.2. Motivation

For the purposes of comparison and to gain advantages of replicability, the potential papers are selected, which considered in the review according to the following selection criteria: a) Papers must be published with at least one of the major computer science libraries: IEEE, Elsevier, ACM, or Springer. b) Papers should aim to predict software maintainability of OO systems. c) The study in the paper must utilize machine learning models; either individual or ensemble. d) The study in the paper must be applied to both QUES and UIMS datasets. e) The study in the paper must present MMRE as an evaluation measure to enable effective comparisons because MMRE measure was used frequently in Chapter 2 more than other measures, such as MAE (see Table 2.11). Criteria (d) and (e) tend to be very restrictive criteria; however, I provide these since the performance of the models differ with different datasets [16] and with different evaluation measures as well [159], and without them the results will be difficult to compare and generalise. Table 4.1 summarises the selected papers which meet these criteria. All these papers created machine learning models to predict software maintainability using default parameters.

Table 4.1: Summary of selected paper using machine learning models to predict software maintainability.

ID	Author	Year	Ref	Prediction model	The best prediction model
S1	Koten and Gray	2006	[11]	Bayesian network, Regression tree, Backward elimination, Step-wise selection	Bayesian network
S2	Zhou and Leung	2007	[12]	MARS, MLR, SVR ANN and RT	MARS
S3	Elish and Elish	2009	[13]	TreeNet, MARS, MLR, SVR, ANN, and RT	TreeNet
S4	Aljamaan and Elish et al.	2013	[88]	MLP, RBF, SVM, M5P and ensemble model	Ensemble model
S5	Ahmed and Al-Jamimi	2013	[7]	FL model, BN model, MARS model	FL model
S6	Kumar and Rath	2015	[152]	Hybrid Neural Network	Hybrid Neural Network
S7	Elish and Aljamaan et al.	2015	[16]	MLP, RBF, SVM, M5P and multi-model ensembles	Multi-model ensembles
S8	Kumara and Naikb et al.	2015	[175]	Neuro-Genetic	Neuro-Genetic
S9	Kumar and Rath	2017	[18]	Neuro-Fuzzy approach	Neuro-Fuzzy approach

A key observation from the aforementioned in Table 4.1 is that the utilisation of individual machine learning models has been investigated in several studies to predict software maintainability accurately. Even though a wide range of techniques have been employed by previous studies, there is no consistent best model emerging from their results. Therefore, a much more meaningful comparison of these studies with the prediction models are provided to determine the best model using the MMRE evaluation measure.

Ensemble models have been applied across a wide range of software engineering problem domains such as fault prediction to increase accuracy prediction over individual models [176]. However, as mentioned in Chapter 2, less attention has been given to the usage of ensemble models in the software maintainability domain. Additionally, the most obvious shortcoming in Table 4.1 is that a model that achieves consistently high software maintainability prediction accuracy fails to emerge. These studies used a wide variety of individual models, along with heterogenous ensemble models in two of the studies. No homogeneous ensemble models were used in any study.

Parameters tuning have been proposed in software quality prediction studies to improve the performance of machine learning models [141, 142, 166]. However, very few studies have investigated the impact of tuning parameters (e.g., only 20% of the papers in the defect

prediction literature) [142]. Recently, investigators have examined the effects of caret package in R, which automatically tunes parameters and requires minimal researcher knowledge [166]. Their results demonstrated a very effective improvement of the prediction accuracy of models with tuning parameters [166]. Additionally, a recent study acknowledged that the use of the caret package in R in prediction models provided results 40% better than using default parameter settings [141].

4.3. Research Method

The primary objective of this chapter is to evaluate the performance of ensemble models to predict software maintainability as compared to individual models. A secondary objective is to determine the model with the best prediction performance among the investigated models and compare it with the best prediction models in the previous selected studies in Table 4.1. A third objective is to investigate the impact of the parameters tuning of the software maintainability prediction models using the caret package in R. To perform these objectives, the following RQs for the first empirical study are provided:

RQ4.1) How effective are individual models at predicting change maintenance effort?

RQ4.2) How do homogenous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

RQ4.3) How do heterogeneous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

RQ4.4) Which prediction models (the best-proposed model in this empirical study or the best-model in the selected studies) provide the best prediction accuracy?

RQ4.5) What are the effects of parameter tuning on the performance of the prediction models?

The overall process of the empirical study is shown in Figure 4.1. This figure illustrates three main experiments to compare model performance. These models are used on two well-known public datasets collected from OO system maintenance: QUES and UIMS. The first experiment was conducted to select the best individual model performance among a set of commonly used regression models, i.e., RT, MLP, KNN, M5Rules, and SVR. The second experiment assessed the performance of homogenous ensemble models (bagging and additive regression) in comparison to individual models. The third experiment evaluated the performance of a heterogeneous ensemble model (stacking) against both the individual models

and the homogenous ensemble models in the second experiment. Finally, a comparison was performed between the best prediction model from this empirical study and those from previous selected studies. Finally, the impact of the parameters tuning of the proposed models is investigated using the caret package in R.

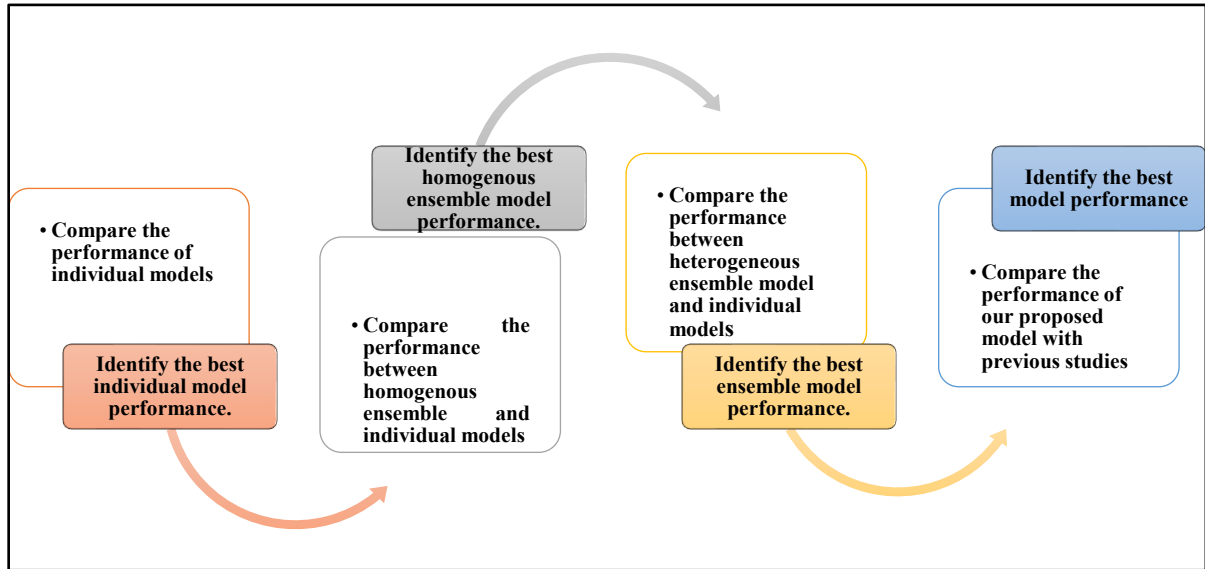


Figure 4.1: The process of the first empirical study.

The framework of the empirical study is illustrated in Figure 4.2 below.

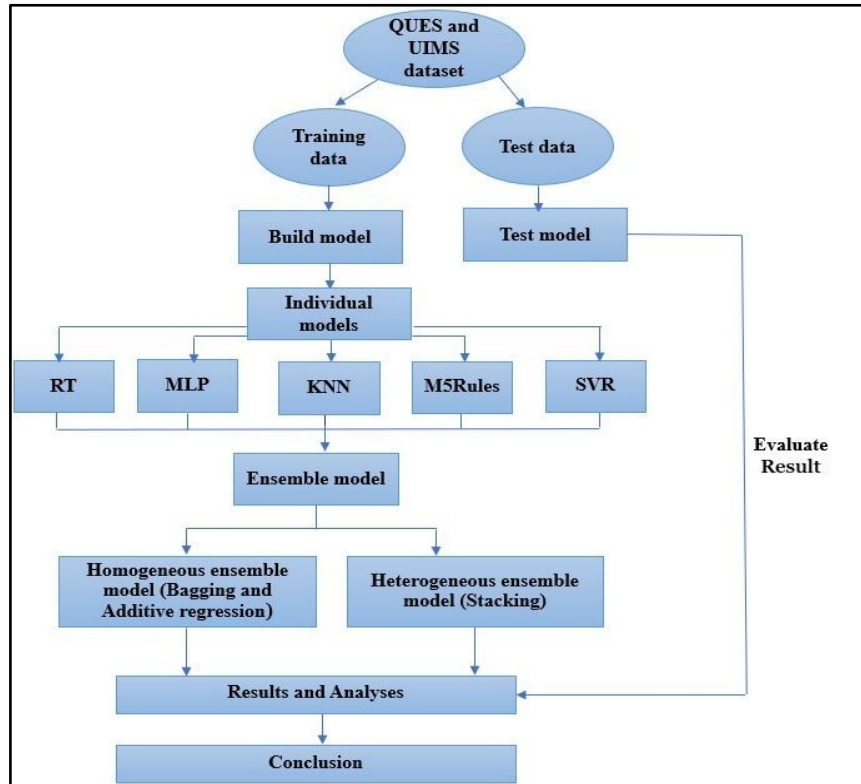


Figure 4.2: Framework of the first empirical study.

4.4. Experimental Data Setup

The following subsections identify the fundamental elements for the experimental setup: maintainability, metrics, and data pre-processing.

4.4.1 Dependent variable: maintainability

In this study, maintainability is defined by the CHANGE metric, which calculates the total number of lines added and deleted in each class during maintenance period [9]. This metric can be ‘addition’ or ‘deletion’ or ‘content changes’. ‘Addition’ or ‘deletion’ are counted as 1, whereas any ‘content change’ is counted as 2 [12]. Therefore, classes that have many changed lines are considered to have a low maintainability value, i.e., require high maintenance effort, whereas those with few changed lines are considered to have high maintainability value, i.e., require little maintenance effort. Eq. (4.1) presents a functional relationship between software maintainability and OO metrics that will present in next section.

$$\text{Software maintainability} = \text{Change metric} = f(\text{DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1, SIZE2}) \quad (4.1)$$

4.4.2 Independent variables: metrics

Metrics are independent variable measures that aim to capture the concept of software maintainability. This study uses L&H metrics [9] that have been used widely owing to their strong relationship with software maintainability [10-13, 21, 73]. These metrics include ten independent variables to measure specific parts of the system and one dependent variable to capture the concept of software maintainability. The definitions and descriptions of the OO metrics was provided in Table 3.4 in Section 3.4.1.

4.4.3 Datasets pre-processing

The datasets were assessed according to recommended pre-processing techniques [125]. The fundamental advantage of QUES and UIMS datasets is that they do not involve any missing values and incomplete or noisy cases, reflecting their high quality. For this reason, the data cleaning technique was not required to apply. Furthermore, both datasets have a small number of records, (39 for UIMS and 71 for QUES); therefore, it was not necessary to apply any data reduction or FS techniques. Previous studies [7, 11-13, 15-18, 88, 152] indicate that these datasets have been used without the application of any pre-processing techniques.

4.4.4 Descriptive statistics

The primary objective of preliminary statistical analyses is to characterise the datasets (range and distribution of values) and also determine the relationship between independent variables (OO metrics) and the dependent variable (CHANGE metric). Figure 4.3 and Figure 4.4 present boxplots of metrics in QUES and UIMS datasets, respectively.

As shown in these figures, NOM and SIZE2 have approximately the same median, mean and Stdev in both datasets, which suggests that the systems are of approximately equal size in terms of numbers of classes and methods. In contrast, the median and mean of SIZE1 in QUES have considerably higher values than those in UIMS. The most notable metric is that NOC has a zero value for all the data in QUES dataset. Consequently, there is no inheritance in the QUES dataset, and NOC values have no influence on predicting maintainability. However,

NOC was not removed from QUES dataset, since several studies used this metric [7, 11-13, 15-18, 88, 152]. Furthermore, DIT and DAC metrics have small values for the median and mean in both datasets, which indicates that an inherited class and data abstraction are rarely used in both systems. The CHANGE metric in the QUES dataset has a higher median and mean compared with the UIMS dataset, and this variation indicates that more maintenance has been performed in the QUES than in the UIMS dataset. Moreover, RFC and MPC in QUES dataset have a greater median and mean compared to UIMS dataset, thereby emphasising that the coupling between classes in QUES dataset is higher than that in UIMS dataset. However, LCOM has similar medians and means in both datasets, indicating that both datasets have the same cohesion. Finally, this table indicates that UIMS and QUES datasets have different characteristics. Consequently, this finding agrees with a previous finding that the characteristics of the UIMS dataset differ distinctly from those of QUES dataset, so the datasets are recognised as heterogeneous; therefore, a software maintainability prediction model is constructed separately for each dataset [7, 11-13, 15-18, 88, 152].

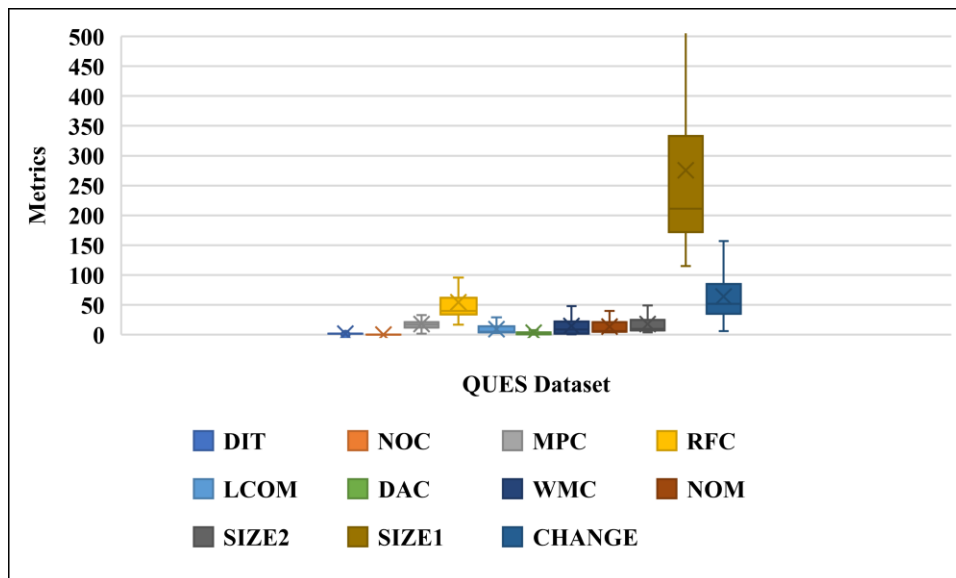


Figure 4.3: Boxplots of metrics in QUES dataset.

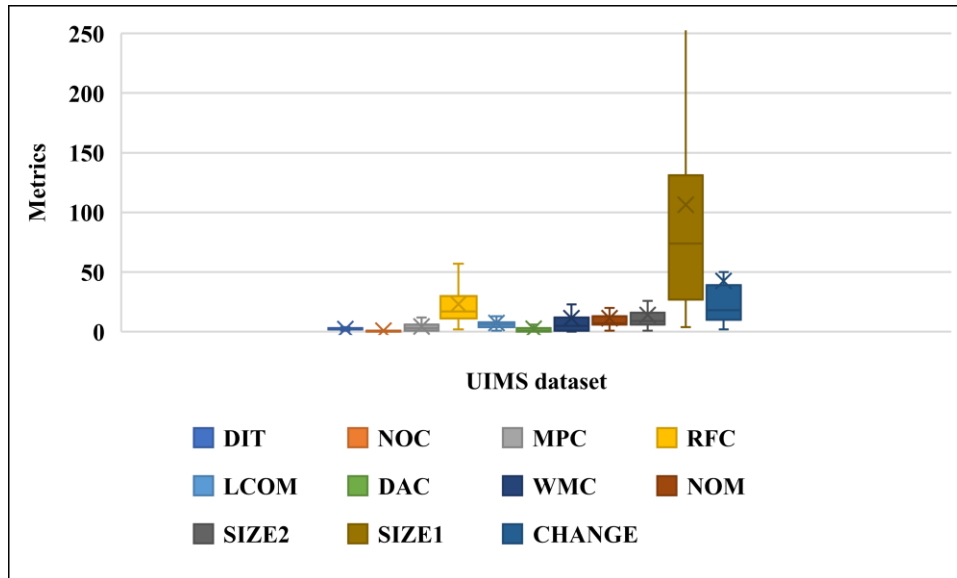


Figure 4.4: Boxplots of metrics in UIMS dataset.

4.4.5 Correlation between metrics in the datasets

The Pearson correlation, which is the well-known statistics [125] measures the strength of a linear relationship between two variables. This correlation is either a strong positive correlation (i.e. the value close to +1) or a strong negative correlation (i.e. the value close to -1); also it can be uncorrelated (i.e. the value equal to zero) [177]. Furthermore, the sign of correlation coefficient determines the direction of the relationship between variables, which is either a positive relationship (+) or a negative relationship (-). The positive relationship occurs when the source code metrics increases, the change metric increases, and vice versa [48]. According to Hopkins [125], these values may be interpreted as follows: any value equal to 0 is trivial, 0.1 is small, 0.3 is moderate, 0.5 is large, 0.7 is very large, 0.9 is nearly perfect, and 1 is perfect.

Table 4.2 lists the results of Pearson's correlation for each metric in QUES and UIMS datasets. This table is divided into two triangular matrices: the upper right triangular matrix represents the correlations between the metrics in the UIMS dataset, whereas the lower left triangular matrix represents the correlations between the metrics in QUES dataset. The results obtained from Table 4.2 highlights the correlations between the metrics in the UIMS and QUES datasets. This result provides evidence of a strong relationship between almost all metrics except DIT and NOC with the CHANGE metric. Furthermore, there are strong

correlations between some metrics with each other (e.g., Size2 with NOM (0.98) and Size1 with WMC (0.96)).

Table 4.2: Correlations between the metrics in UIMS (upper right triangle) and QUES (lower lift triangle).

	DIT	NOC	MPC	RFC	LCOM	DAC	WMC	NOM	SIZE2	SIZE1	CHANGE
DIT	1	-.47	0.05	-0.22	-0.19	-0.43	-0.22	-0.35	-0.40	-0.18	-0.33
NOC	NA	1	.03	0.20	0.12	0.32	0.22	0.23	0.26	0.17	0.47
MPC	0.01	NA	1	0.74	0.50	0.43	0.62	0.54	0.55	0.67	0.60
RFC	0.10	NA	0.33	1	0.79	0.62	0.90	0.93	0.89	0.90	0.79
LCOM	0.12	NA	-.10	0.82	1	0.36	0.79	0.75	0.67	0.81	0.66
DAC	0.39	NA	0.01	0.63	0.56	1	0.44	0.75	0.86	0.51	0.72
WMC	-.13	NA	0.13	0.73	0.57	0.57	1	0.83	0.76	0.96	0.77
NOM	0.12	NA	-0.11	0.81	0.88	0.80	0.70	1	0.98	0.87	0.75
SIZE2	0.20	NA	-0.08	0.80	0.83	0.88	0.68	0.98	1	0.81	0.78
SIZE1	0.01	NA	0.37	0.79	0.53	0.63	0.89	0.69	0.70	1	0.75
CHANGE	-0.08	NA	0.46	0.38	0.04	0.08	0.42	0.14	0.14	0.63	1

A visualization of correlation between metrics on QUES and UIMS datasets is shown in Figure 4.5. The scale in this figure shows how the colour maps to the strengths of the relationship. The value from 1 represents a strong positive correlation, whereas the values at -1 indicate a strong negative correlation. It is evident from Figure 4.5 that most metrics have light blue colour, with more dark blue in UIMS than in QUES which indicates that most metrics have a moderate to large positive correlation. It is noteworthy that metrics were not eliminated because one of the objectives of this chapter is to compare this study with previous studies and most previous studies used all metrics without performing FS [7, 11-13, 15-18, 88, 152]. Also, previous studies did not indicate that the datasets suffer from any problems, such as high dimensional, irrelevant or redundant features that require applying FS [7, 11-13, 15-18, 88, 152]. Additionally, these metrics have been validated and shown as a good predictor of software maintainability [9].

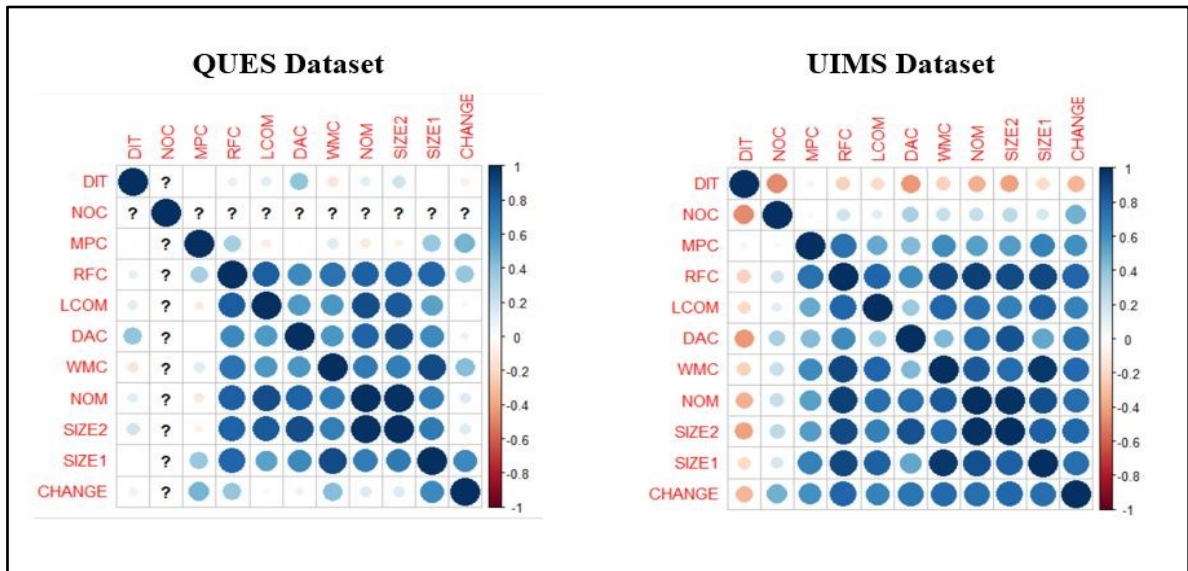


Figure 4.5: The correlation between metrics.

4.5. Results and Analysis

This section provides the results and analysis of the empirical study conducted. First, the performance of the investigated models was compared: individual models (RT, MLP, KNN, M5Rules, SVR), homogeneous ensemble model (bagging and additive regression), and heterogeneous ensemble model (stacking) applied on the QUES and UIMS datasets. In addition, the statistical tests to explore the difference between individual and ensemble models for each dataset were applied. Then, the best performing model was compared with the best model in the selected previous studies. Finally, the impact of parameter tuning of the ensemble models was investigated.

4.5.1 Results of the first empirical study

Table 4.3 and Table 4.9 present the results of the prediction accuracy measures obtained by applying prediction models on QUES and UIMS, respectively. Boldface values (highlighted in light green) in the table indicate the best results for each experiment and boldface with underline (highlighted in dark green) refer to the best results in all experiments that are the lowest (MMRE and MAE) or the highest (Pred(.25), Pred(.30), and SA) depending on the measure.

Moreover, this section provides the statistical tests of the first empirical study using one-way ANOVA, as proposed in Section 3.5.5 (see Table 4.4, Table 4.5, Table 4.6, Table 4.7 and Table 4.8 for QUES dataset and Table 4.10, Table 4.11, Table 4.12, Table 4.13 and Table 4.14 for UIMS dataset). ANOVA was performed using the residuals values of the prediction models to investigate whether the performance difference between the group population means is significant or not. Factor A refers to each individual model, along with these individual models as the base models in the ensemble models. For example, Factor A in Table 4.4 includes the results of the residual values for RT as the individual model and as the base model in bagging, additive regression, and stacking.

A. Results of QUES dataset

First, Table 4.3 summarises the results of the prediction models on the QUES dataset. The baseline measure in this table depends on the dependent variable only (i.e., CHANGE metric) and calculates the mean value of this variable. The results of the baseline show that all the prediction models have better results than the baseline. For example, the MMRE of the baseline is equal to 0.99, and all the prediction models have values lower than this.

Among the individual models, KNN achieved the best result in all accuracy prediction measurements. SVR was the second best model in all accuracy prediction measurements except Pred(.25), which recorded the highest values (0.42) in RT, whereas SVR recorded (0.41). Because of the minor difference between SVR and RT, SVR was considered the second-best individual model. This finding is consistent with another study which states that KNN improved the prediction accuracy compared to other models [178]. In addition, previous study stated that SVR achieved high prediction accuracy and considered a strong model [179].

After building a bagging ensemble on each individual model, it clear that this model improved the accuracy prediction for only MLP, M5Rules and SVR. Although bagging had a negative impact on KNN, KNN outperformed all other models in all accuracy predictions. However, bagging had a minor impact on the accuracy prediction of SVR. According to the statistical tests presented below, there were no significant differences in terms of the residual values among all the individual and bagging ensemble models.

It was evident that the additive regression ensemble model improved the accuracy prediction of individual models except RT and did not impact KNN; KNN showed the same result with this model and as an individual model. Additionally, as mentioned in the bagging

case, KNN as the base model in the additive regression model outperformed all other base models in all accuracy predictions; SVR was the second best after KNN. Finally, it was evident that the prediction accuracy of additive regression ensemble models in most cases was better than that of bagging ensemble models.

The stacking ensemble model improved the performance of individual models, except KNN. Moreover, it performed better than both bagging and additive regression on all their base models except KNN. Stacking improves accuracy prediction if the individual models are chosen from various categories such as RT from tree and KNN from lazy, as was performed in this experiment. Finally, the prediction accuracy of heterogeneous (stacking) ensemble models, in general, was better than that of homogeneous (bagging and additive regression) ensemble models.

KNN as an individual model or as the base model in additive regression is the only model that was close to meeting the criteria of an accurate prediction mentioned in Chapter 3 because it achieved an MMRE value of 0.26, and the criterion for MMRE is $MMRE \leq 0.25$ [67]. Pred values of 0.65 and 0.68 were obtained for Pred(.25) and Pred(.30), respectively, and the criteria for Pred are: $Pred(.30) \geq 0.70$ or $Pred(.25) \geq 0.75$ [34].

Table 4.3: Performance of the prediction models for the QUES dataset.

QUES Dataset	MMRE	Pred(.25)	Pred(.30)	MAE	SA
Baseline	0.99	0.30	0.32	32.71	0
Individual models					
RT	0.45	0.42	0.48	26.24	19.79
MLP	0.50	0.30	0.42	28.71	12.22
KNN	0.26	0.65	0.68	19.75	38.85
M5Rules	0.49	0.39	0.41	23.39	27.45
SVR	0.38	0.41	0.52	20.33	37.86
Homogeneous ensemble model – Bagging					
RT	0.48	0.37	0.41	22.61	30.87
MLP	0.39	0.48	0.56	19.89	39.21
KNN	0.30	0.51	0.58	19.04	41.80
M5Rules	0.45	0.37	0.45	28.72	32.23
SVR	0.38	0.44	0.54	20.42	37.59
Homogeneous ensemble model – Additive Regression					
RT	0.47	0.39	0.44	26.23	19.80
MLP	0.52	0.35	0.45	28.43	13.07
KNN	0.26	0.65	0.68	19.75	39.63
M5Rules	0.47	0.42	0.45	23.85	27.07
SVR	0.35	0.48	0.54	19.85	39.32
Heterogeneous ensemble model – Stacking					
Stacking (RT, MLP, KNN, M5Rules, SVR)	0.32	0.48	0.54	19.80	39.47
<p>Dark green: represents the best results in all experiments. Light green: represents the best results for each experiment.</p>					

Second, Table 4.4, Table 4.5, Table 4.6, Table 4.7 and Table 4.8 list one-way ANOVA results in the QUES dataset using the residual values for RT, MLP, KNN, M5Rules, and SVR and ensemble models, respectively. From these tables, the results of the p-values were higher than the defined value ($\alpha = 0.05$). Therefore, H_0 is accepted and all the group population means (Factor A) are the same in all tables. This indicates that the performance of the individual and ensemble models in terms of the residual values was not significantly different from each other for Factor A. Furthermore, the results of the eta-squared reveal that the effect sizes in all tables are small because all the eta-squared values are close to 0.01, which is considered small according to the standard classifications published in [180].

Table 4.4: One-way ANOVA for RT and ensemble models in QUES dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2076.45	3.00	692.15	1.27	0.29	0.01
Error	153018.62	280.00	546.50			
Total	155095.07	283.00				

Table 4.5: One-way ANOVA for MLP and ensemble models in QUES dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	5413.90	3.00	1804.63	2.54	0.06	0.03
Error	198690.89	280.00	709.61			
Total	204104.79	283.00				

Table 4.6: One-way ANOVA for KNN and ensemble models in QUES dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	28.27	3.00	9.42	0.01	1.00	0.00
Error	196680.56	280.00	702.43			
Total	196708.84	283.00				

Table 4.7: One-way ANOVA for M5Rules and ensemble models in QUES dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	700.86	3.00	233.62	0.54	0.65	0.01
Error	120132.87	280.00	429.05			
Total	120833.72	283.00				

Table 4.8: One-way ANOVA for SVR and ensemble models in QUES dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	21.46	3.00	7.15	0.02	1.00	0.00
Error	121832.52	280.00	435.12			
Total	121853.99	283.00				

Figure 4.6 and Figure 4.7 illustrate a bar chart of the Pred values (Pred(.25) and Pred (.30), respectively) to compare the investigated models for the QUES dataset. A higher score indicates a better performance, and the proposed criteria for Pred was $\text{Pred}(.30) \geq 0.70$ or $\text{Pred}(.25) \geq 0.75$ [34]. The results in the figures provide confirmatory evidence that bagging

increased the performance of Pred values over all individual models except RT, KNN and the Pred(.25) value for M5Rules. Additive regression increased this performance over all individual models except RT. KNN, when used as an individual or the base model in additive regression, achieved the best accuracy prediction, reaching approximately 70. The stacking ensemble model also increased the performance of Pred values over all the individual models; however, it reported lower values than KNN. The statistical tests indicate that the improvement of the ensemble models over the individual models was not significantly different.

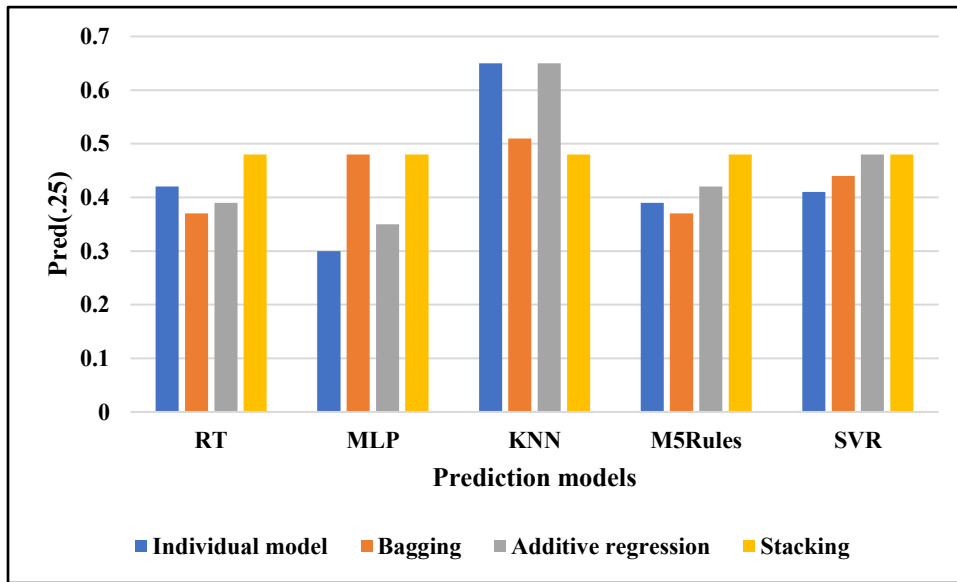


Figure 4.6: Pred(.25) of prediction models for QUES dataset.

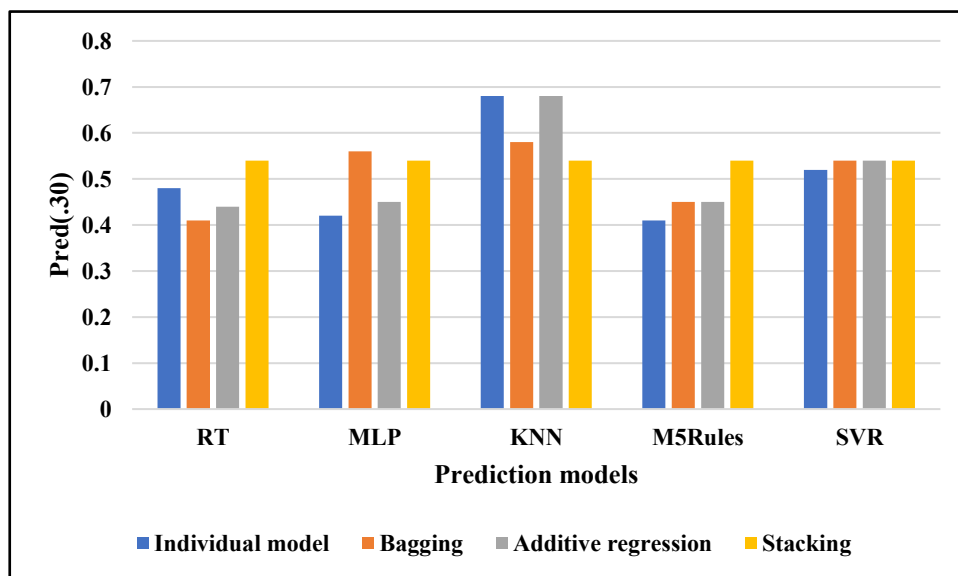


Figure 4.7: Pred(.30) of prediction models for QUES dataset.

Figure 4.8 presents a boxplot of MRE for the visual comparison of different prediction models based on the MMRE values. Ten-fold cross-validation was used, and the number of observations was equal to the size of the dataset, which was 71 residuals values for the QUES. The mean value is indicated by an 'X' and the upper and lower lines of the box represent 'whiskers', where the middle horizontal line across the box represents the middle quartile. The impact of applying both homogeneous and heterogeneous ensemble models on various individual models is shown in the figure. The results indicate that most ensemble models improved the accuracy prediction of individual models because they had the smallest whiskers, the narrowest box, and the lowest MMRE value compared with individual models. KNN as an individual model or as the base model in additive regression outperformed all other prediction models. It was followed by KNN as the base model for bagging and then stacking. Additionally, the results of the statistical tests reveal that the improvement in all ensemble models was not significantly different from that in the individual models.

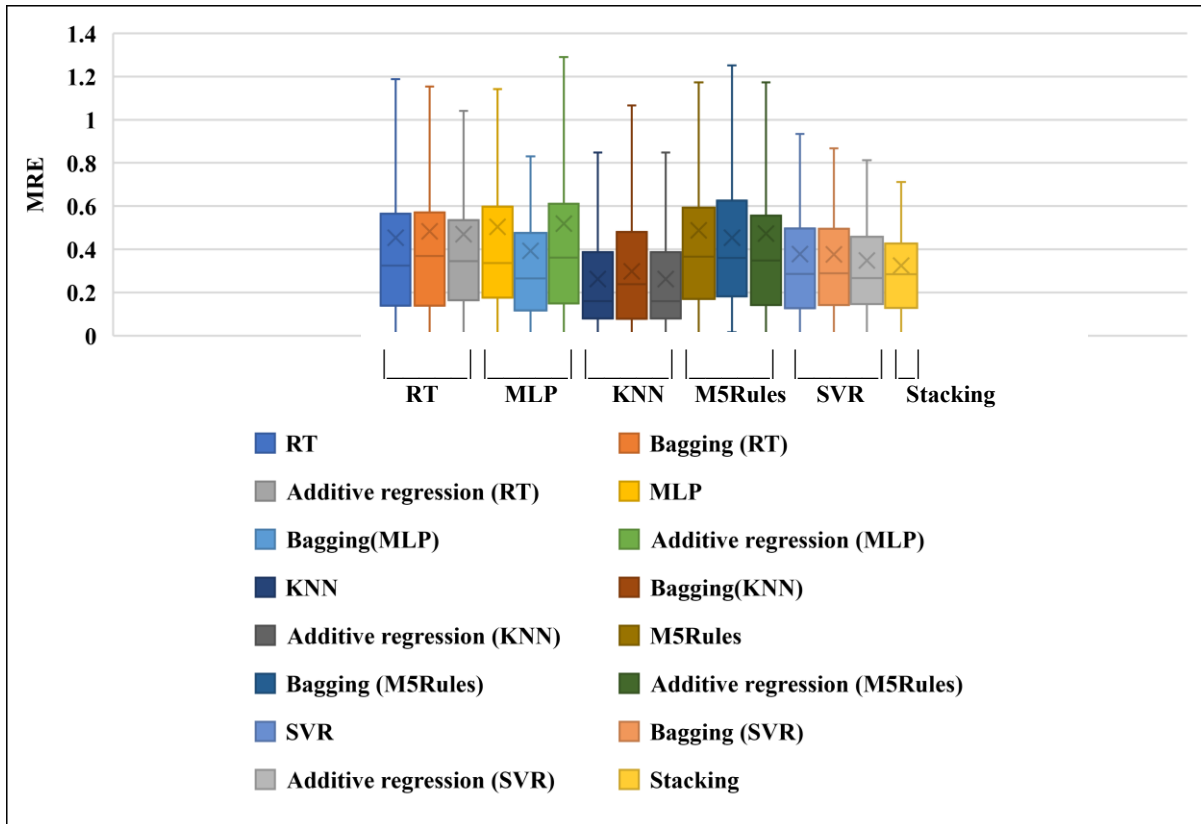


Figure 4.8: Boxplots of MRE for prediction models in QUES dataset.

Figure 4.9 compares the results obtained from the residuals of the prediction models for the QUES dataset. The MAE value is defined by an 'X'; the lower score of MAE, the small whiskers, and the tight box indicate a better performance. First, all the prediction models achieved better prediction accuracy than the baseline. Second, the ensemble models improved the prediction accuracy over all the individual models except for M5Rules as the base model in bagging and additive regression, and SVR as the base model in bagging, whereas KNN as the individual model reached the same result as KNN as the base model in additive regression. Third, KNN as the base model in bagging was the best model, followed by KNN as an individual model or as the base model in additive regression and then stacking. Furthermore, none of these improvements obtained by the ensembles were significantly different.

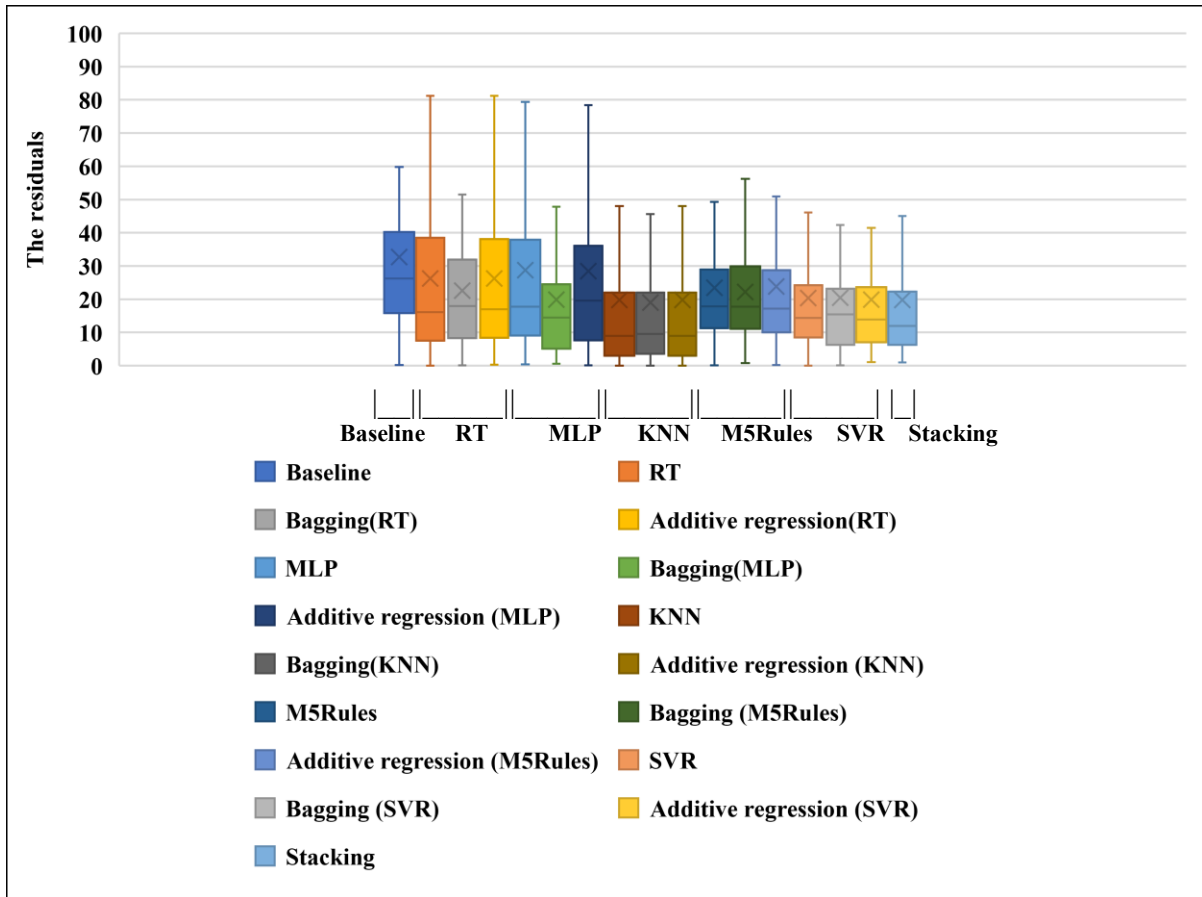


Figure 4.9: Boxplots of the residuals for prediction models in QUES dataset.

Figure 4.10 illustrates the plots of the predicted and the actual values, along with , a baseline for each prediction model investigated in the QUES dataset. The main idea of these plots is to present the behaviour of the prediction models in term of the overestimated and underestimated observations from the actual values rather than the performance of the models (e.g., MMRE and Pred). Thus, these graphs depend on the requirements of the project managers, if they need to overestimate, and thus lose control, or underestimate, and thus lose quality. Therefore, these plots were interpreted by the project managers as follow:

- Overestimation is the number of observations that the predicted values are higher than the actual values.
- Underestimation is the number of observations that the predicted values are lower than the actual values.

The actual values include 71 observations of the dependent variable (CHANGE) sorted in ascending order. The baseline represents the mean of the actual values, whereas the

predicted values represent the observations by each prediction model. This figure indicates the following findings. First, most observations of predicted values for each prediction model were higher than actual values, which means that these prediction models have more overestimated observations than underestimated ones. Second, KNN as an individual model or as the base model in additive regression tended to have equally overestimated and underestimated observations and remained relatively steady compared to other models. Third, the predicted values were spread around the baseline (equal to approximately 64), which means that the prediction models added value and made changes in the observations. Additionally, the mean of the prediction value is typically lower than the value of the baseline except for some models such as RT and M5Rules, which have a minor increase over the baseline, ranging from 64.5 to 66.0.

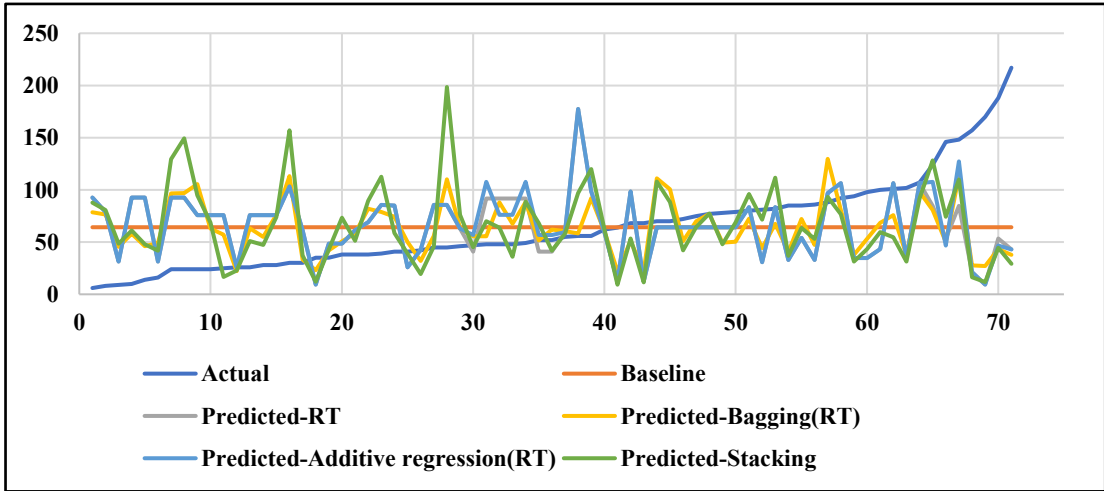


Figure 4.10.A: Plots of predicted and actual values for RT in the QUES dataset.

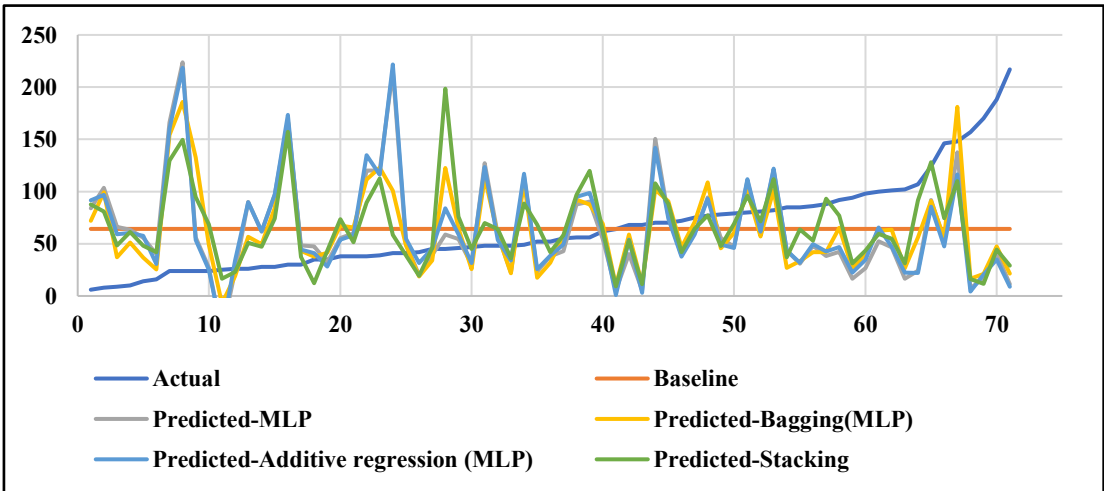


Figure 4.10.B: Plots of predicted and actual values for MLP in the QUES dataset.

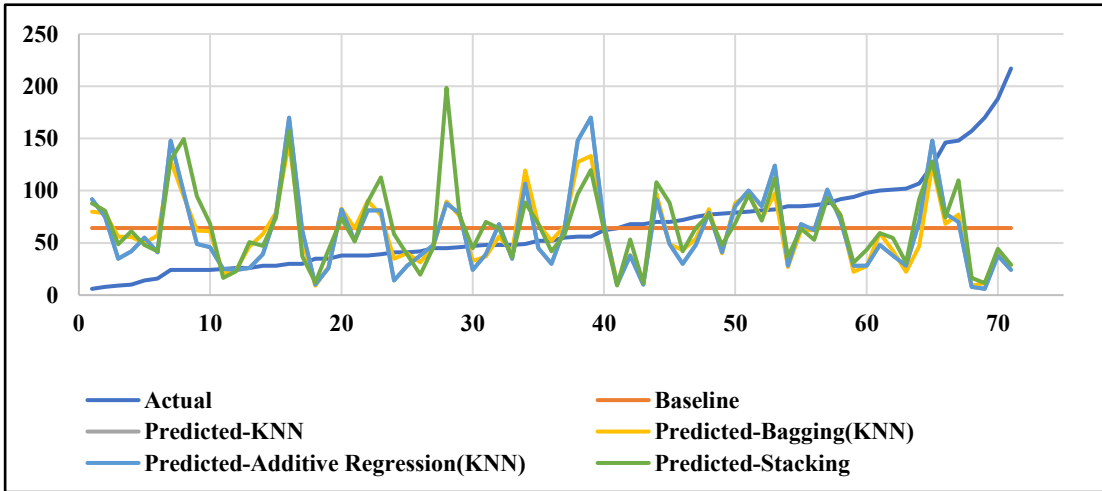


Figure 4.10.C: Plots of predicted and actual values for KNN in the QUES dataset.

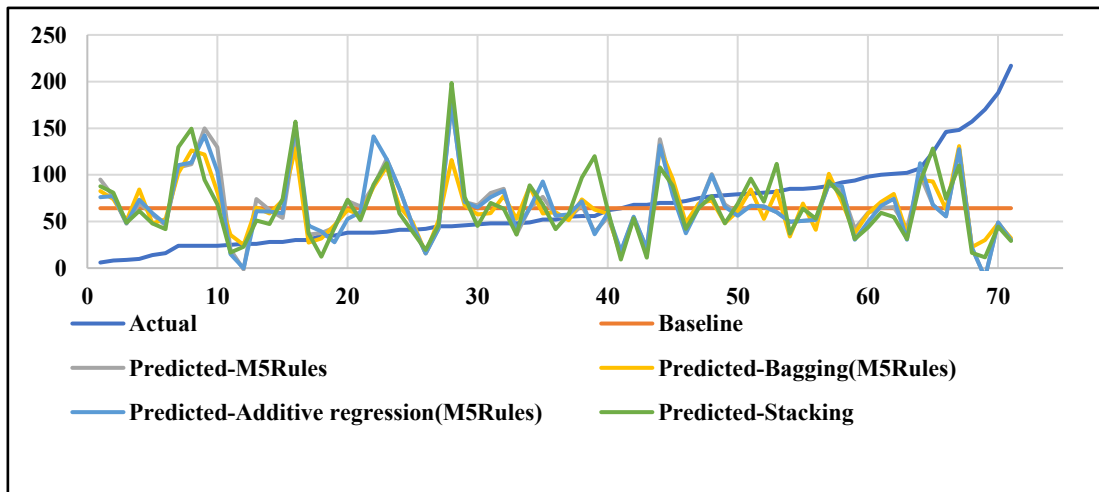


Figure 4.10.D: Plots of predicted and actual values for M5Rules in the QUES dataset.

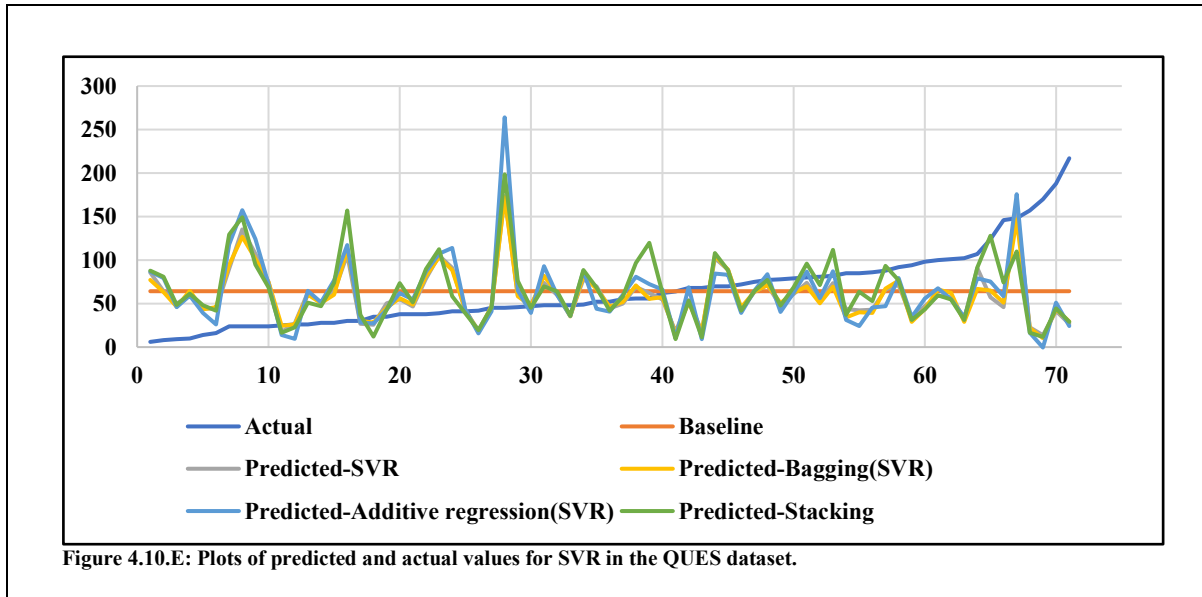


Figure 4.10: Plots of predicted and actual values for prediction models in the QUES dataset.

B. Results of UIMS dataset

Table 4.9 lists the results of the experiment on the UIMS dataset, reported similarly as the QUES findings. The results show that all the prediction models have better results than the baseline. For example, the MMRE of the baseline is equal to 5.43, where all the prediction models have results lower than this value.

Among the individual models, KNN, as in the QUES dataset, outperformed other prediction models in all prediction accuracy measures except MAE, which achieved the best results by MLP. M5Rules was the second best among all prediction accuracy measures, except MAE.

In terms of MMRE and MAE, application of the bagging ensemble model on each individual model clearly indicated that this model improved the performance of MMRE and MAE measures for all the individual models. However, it either decreased the performance of the Pred values of all individual models or produced the same results. It seems possible that these results are due to the spread of the residual boxplots of MRE increasing after applying this ensemble (see Figure 4.13). However, it is usual to obtain contradictory results in empirical studies of the software engineering field [99]. Furthermore, this model decreased the performance of some prediction accuracy measures (i.e., KNN and M5Rules). Although the bagging ensemble model using KNN produced the highest accuracy prediction compared to other models in terms of MMRE value, it had a negative impact on KNN itself. These results

are consistent with a prior study, which showed that bagging ensemble models have higher potential to contribute to more accurate predictions than individual models [107].

Additive regression increased most prediction accuracy measures for all the individual models except MLP. As mentioned during the analysis of the QUES dataset, the additive regression ensemble model produced the same accuracy prediction for KNN as an individual model. KNN, as the base model in additive regression, provided superior improvement in most prediction accuracy measurements, followed by M5Rules and SVR. This result indicates that KNN as either the base model in additive regression or as an individual model, achieved the best accuracy prediction, which is consistent with the findings for the QUES dataset. In conclusion, the prediction accuracy of additive regression ensemble models in most cases was better than that of bagging ensemble models.

A stacking ensemble model did not increase the performance of individual models in the UIMS dataset with the exception of the RT model. However, the performance of stacking in the UIMS dataset was lower than that in the QUES dataset. This occurs because the QUES and UIMS datasets have different characteristics, as mentioned in Section 4.4.4. Moreover, stacking failed to satisfy the criteria for accurate prediction, as mentioned in Chapter 3. Additionally, stacking showed worse results compared to the homogeneous ensemble model. In conclusion, the prediction accuracy of homogeneous (bagging and additive regression) ensemble models was generally better than that of heterogeneous (stacking) ensemble models. None of the prediction models applied on UIMS fulfilled the criteria of prediction accuracy described in Chapter 3.

Table 4.9: Performance of the prediction models for the UIMS dataset.

UIMS Dataset	MMRE	Pred (.25)	Pred (.30)	MAE	SA
Baseline	5.43	0.13	0.13	40.71	0
Individual models					
RT	4.52	0.18	0.21	41.74	-2.52
MLP	1.32	0.31	0.31	23.39	42.54
KNN	0.74	0.41	0.41	23.41	42.50
M5Rules	1.24	0.36	0.38	26.64	34.57
SVR	1.84	0.28	0.33	26.63	34.58
Homogeneous ensemble model – Bagging					
RT	3.08	0.13	0.21	31.63	22.31
MLP	1.17	0.26	0.31	20.10	50.63
KNN	0.83	0.26	0.33	20.80	48.91
M5Rules	1.31	0.21	0.26	21.65	46.82
SVR	1.65	0.21	0.26	24.79	39.12
Homogeneous ensemble model – Additive Regression					
RT	4.56	0.21	0.23	42.14	-3.51
MLP	1.87	0.23	0.28	26.66	34.51
KNN	0.74	0.41	0.41	23.41	42.50
M5Rules	1.16	0.33	0.36	27.23	33.12
SVR	1.20	0.33	0.36	24.57	39.65
Heterogeneous ensemble model – Stacking					
(RT, MLP, KNN, M5Rules, SVR)	2.45	0.23	0.23	33.51	17.68
<p>Dark green: represents the best results in all experiments. Light green: represents the best results for each experiment.</p>					

Table 4.10, Table 4.11, Table 4.12, Table 4.13 and Table 4.14 show one-way ANOVA results for the UIMS dataset using the residuals values for RT, MLP, KNN, M5Rules and SVR and ensemble models, respectively. These results indicate that the p-values were higher than the defined value ($\alpha = 0.05$). Then, H_0 is accepted and all the group population means (Factor A) are the same in all tables. Therefore, the performance of individual and ensemble models in terms of residual values was not significantly different from each other for Factor A. Additionally, the results of the eta-squared indicate that the effect sizes in all tables are small because all the eta-squared values are close to 0.01, which is considered small according to the standard classifications published in [180].

Table 4.10: One-way ANOVA for RT and ensemble models in UIMS dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	3494.67	3.00	1164.89	0.58	0.63	0.01
Error	304355.14	152.00	2002.34			
Total	307849.81	155.00				

Table 4.11: One-way ANOVA for MLP and ensemble models in UIMS dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	3841.22	3.00	1280.41	1.23	0.30	0.02
Error	158728.81	152.00	1044.27			
Total	162570.02	155.00				

Table 4.12: One-way ANOVA for KNN and ensemble models in UIMS dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	3699.69	3.00	1233.23	0.92	0.43	0.02
Error	204486.42	152.00	1345.31			
Total	208186.11	155.00				

Table 4.13: One-way ANOVA for M5Rules and ensemble models in UIMS dataset using the residuals .

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2768.62	3.00	922.87	0.69	0.56	0.01
Error	203993.82	152.00	1342.06			
Total	206762.45	155.00				

Table 4.14: One-way ANOVA for SVR and ensemble models in UIMS dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2059.56	3.00	686.52	0.59	0.62	0.01
Error	176247.08	152.00	1159.52			
Total	178306.64	155.00				

Figure 4.11 and Figure 4.12 illustrate a bar chart of the Pred values (Pred (.25) and Pred (.30)) to compare prediction models for the UIMS dataset. It is evident that a bagging ensemble had a negative or little impact on all the individual models. The additive regression ensemble model produced superior Pred values in RT and SVR only, whereas stacking improved the Pred values of RT only. Similar to the findings for the QUES dataset, no difference was found between KNN as an individual model and as the base model in additive regression; however, both achieved the best Pred value of 0.41. They were followed by M5Rules as an individual model and as the base model in additive regression. As mentioned previously in the QUES dataset, there was no difference in the performance between the ensemble and the individual models (see Table 4.10, Table 4.11, Table 4.12, Table 4.13, Table 4.14).

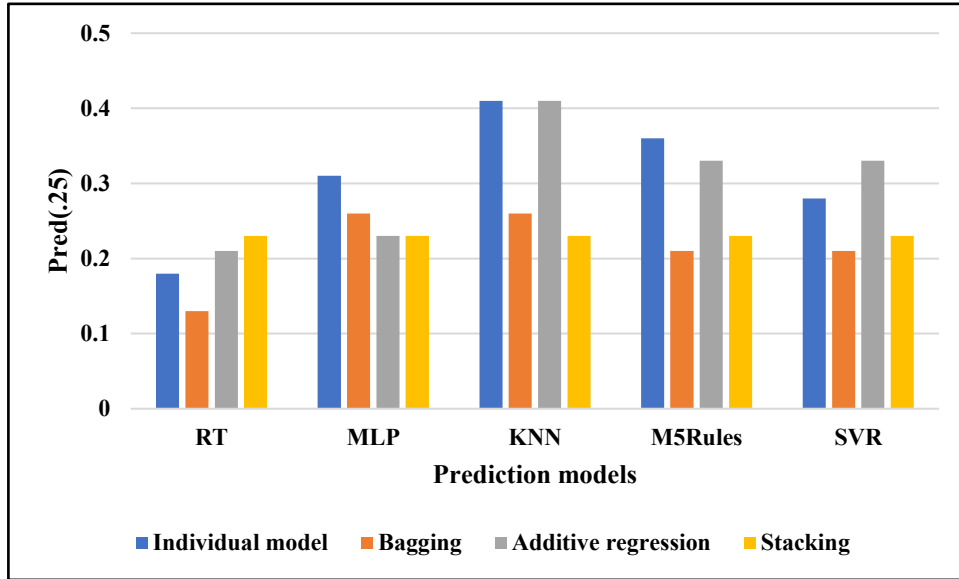


Figure 4.11: Pred(.25) of prediction models for UIMS dataset.

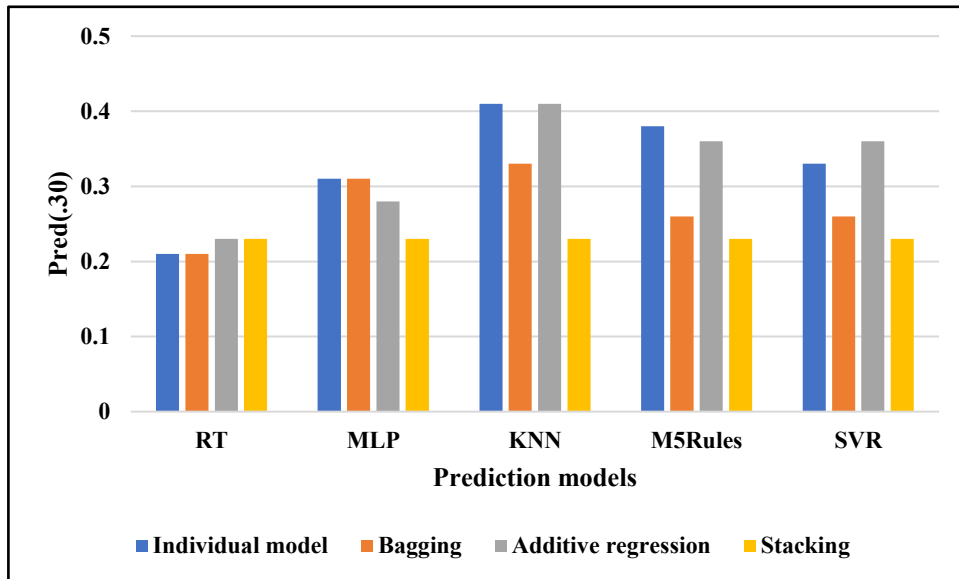


Figure 4.12: Pred(.30) of prediction models for UIMS dataset.

Figure 4.13 shows the residual boxplots of MRE to enable a comparison between prediction models in terms of the MMRE value that is specified by an 'X'. After applying bagging ensemble models, there is an obvious tendency for the mean (i.e., MMRE) to decrease. Ten-fold cross-validation was performed, and the number of observations is similar to the size of the dataset, which is 39 residuals values for the UIMS dataset. The spread increases after applying this ensemble model to all individual models, except for KNN and M5Rules. The

impact of applying additive regression ensemble models on M5Rules and SVR is evident as they have the smallest whiskers, the narrowest box, and the lowest MMRE value. Additive regression decreased the accuracy prediction of RT and MLP, whereas it produced the same accuracy prediction in KNN. Stacking reported the lowest accuracy prediction compared to all individual models except RT. Overall, KNN as an individual model or as the base model in additive regression outperformed all other prediction models, followed by KNN as the base model for bagging. According to Table 4.10, Table 4.11, Table 4.12, Table 4.13, Table 4.14, the performance between the ensemble and the individual models was not significantly different.

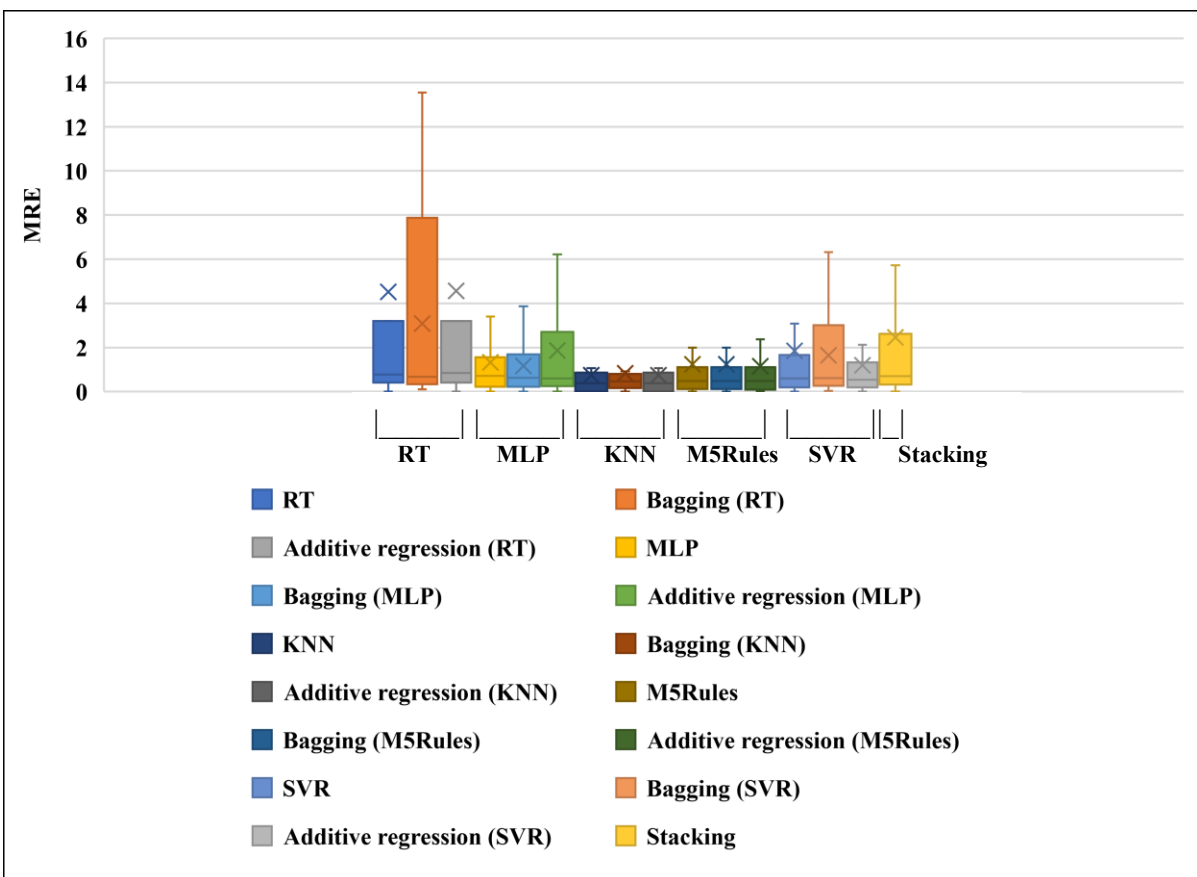


Figure 4.13: Boxplots of MRE for prediction models in UIMS dataset.

Figure 4.14 summarises the comparison between the residuals of prediction models for the UIMS dataset. The MAE value is defined by an ‘X’; a lower score refers to better performance, along with the small whiskers and the tight box. It can be observed that all the prediction models were better than the baseline in terms of MAE, except RT, as the individual model and as the base model in additive regression. Additionally, bagging ensemble models

increased the prediction accuracy over all the individual models, whereas additive regression and stacking only improved the SVR and RT, respectively. Finally, MLP and KNN as the base models in bagging and as individual models achieved the best prediction accuracy, whereas KNN as an individual model recorded the same result as the base model in additive regression (23.41). Moreover, Table 4.10, Table 4.11, Table 4.12, Table 4.13, Table 4.14 indicate that there were no significant differences between the ensemble and individual models.

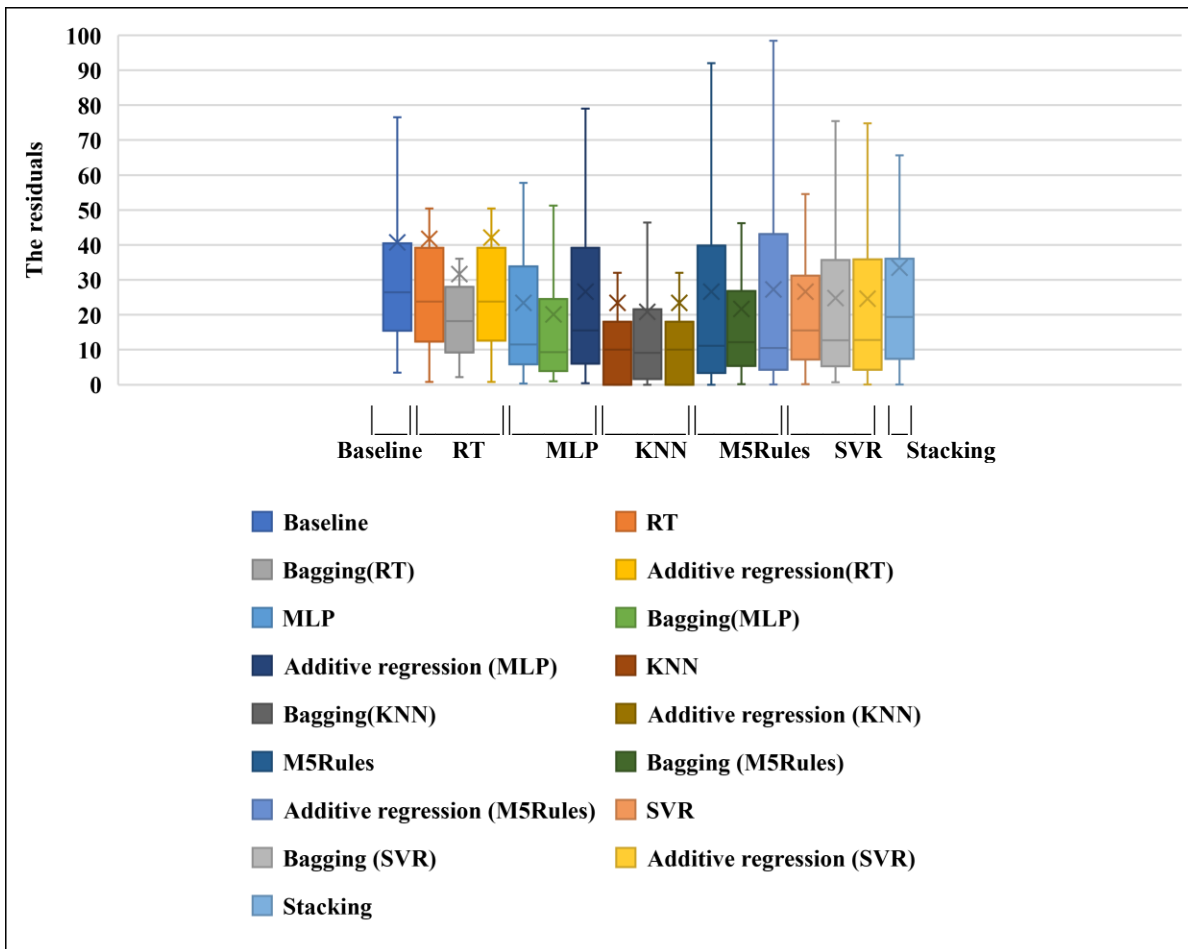


Figure 4.14: Boxplots of the residuals for prediction models in UIMS dataset.

Figure 4.15 presents the plots of the actual values, a baseline, and the predicted values for each prediction model for the UIMS dataset. The actual values include 39 observations of the dependent variable (CHANGE) sorted in ascending order. The baseline indicates the mean of the actual values, whereas the predicted values indicate the observations attained by each prediction model. The findings obtained from this figure reveal the following. First, most

observations of predicted values for each prediction model were higher than actual values, which indicates that these prediction models have more overestimated observations than underestimated ones. Second, the KNN as an individual model or as the base model in additive regression tended to have equal overestimated and underestimated observations and remained relatively steady compared to other models, which is a conclusion similar to that reached for the QUES dataset. Third, the predicted values are spread around the baseline (equal to approximately 42), which means that the prediction models added value and made changes to the observations. Fourth, the stacking model had several underestimated observations and several observations under the baseline. This finding may be interpreted as the reason for the low accuracy prediction in stacking compared to other models. Additionally, the mean of the prediction values was typically lower than the value of the baseline, except for a few models that had a minor increase over the baseline, for example, bagging with MLP as the base model and additive regression with RT as the base model, ranging from 42.5 to 46.0.

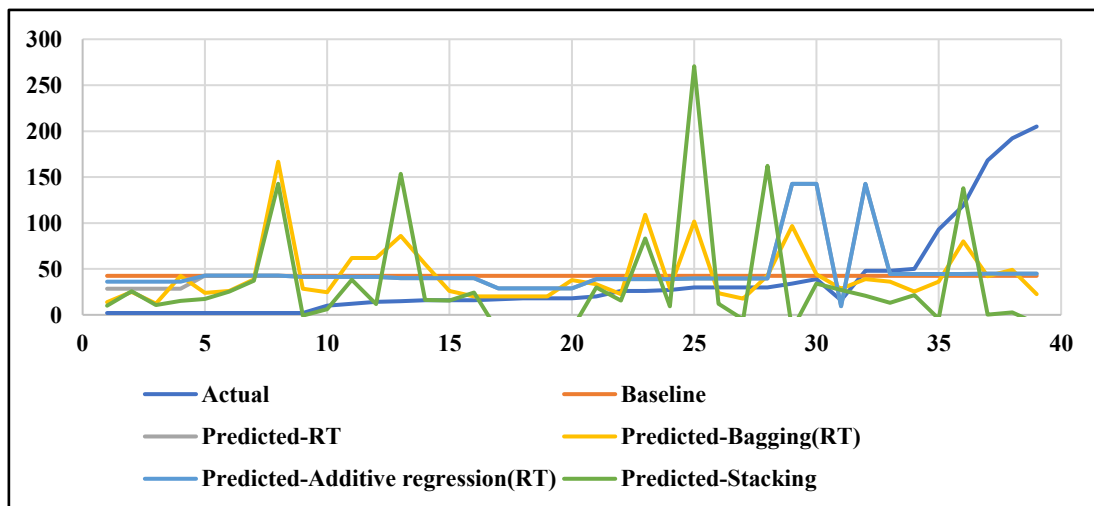


Figure 4.15.A: Plots of predicted and actual values for RT in UIMS dataset.

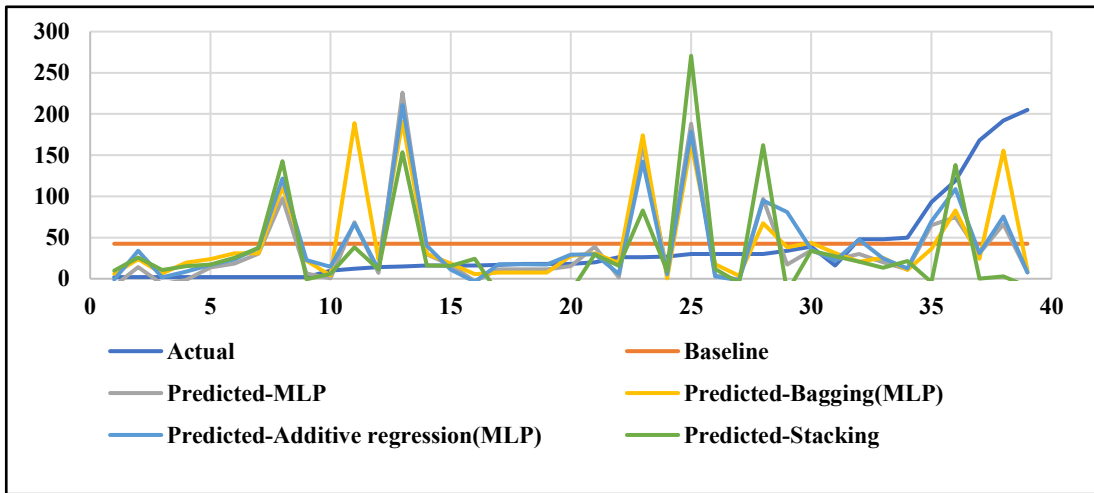


Figure 4.15.B: Plots of predicted and actual values for MLP in UIMS dataset.

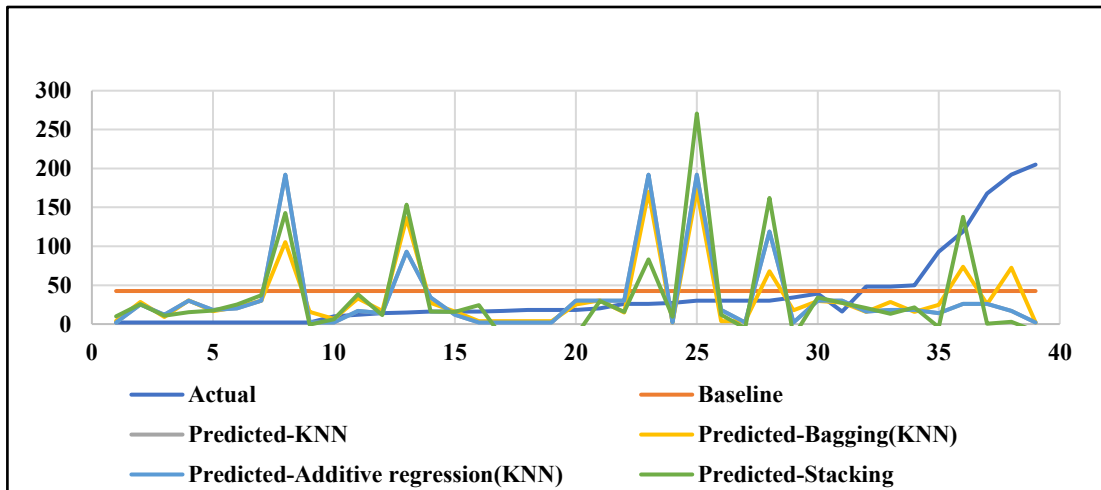


Figure 4.15.C: Plots of predicted and actual values for KNN in UIMS dataset.

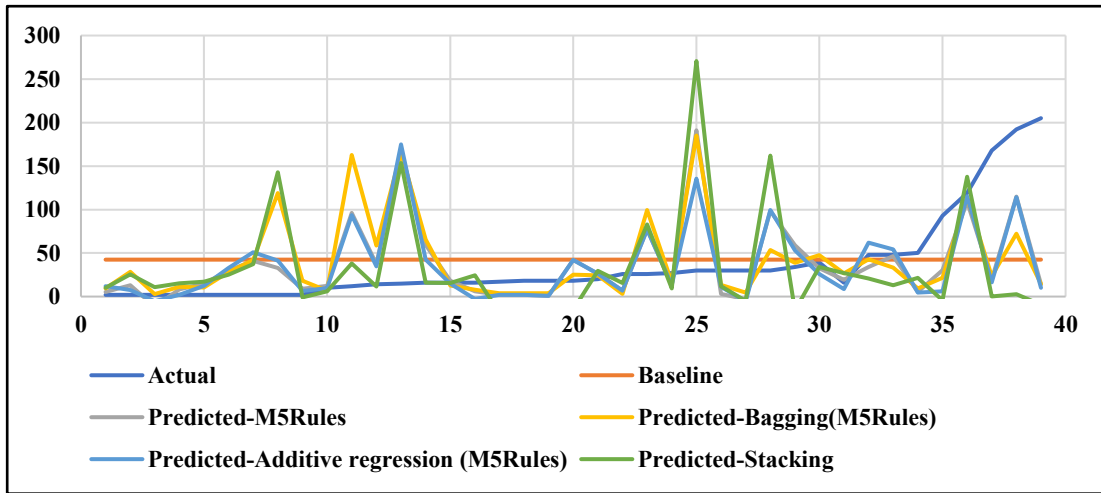


Figure 4.15.D: Plots of predicted and actual values for M5Rules in UIMS dataset.

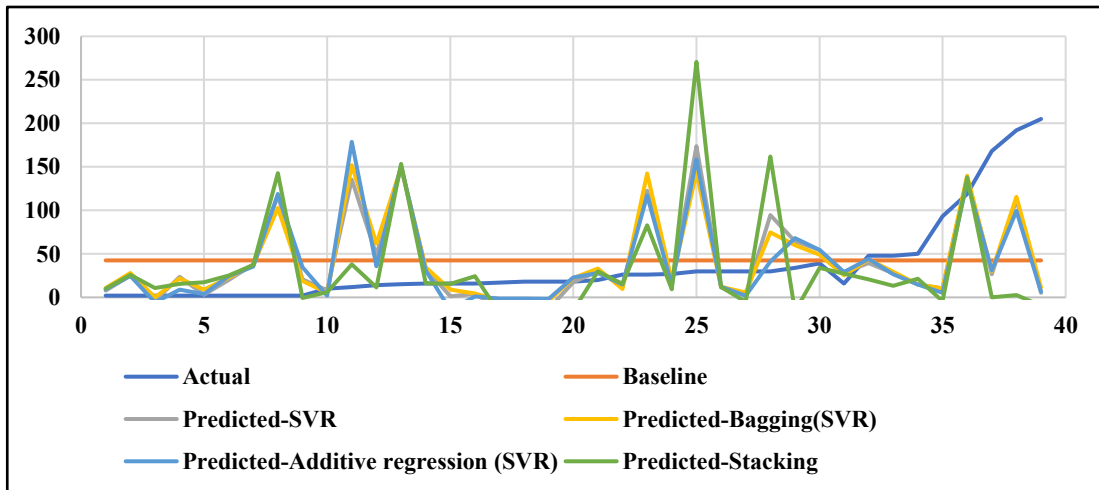


Figure 4.15.E: Plots of predicted and actual values for SVR in UIMS dataset.

Figure 4.15: Plots of predicted and actual values for prediction models in UIMS dataset.

4.5.2 Comparison of the best investigated model with the best model in selected studies

The best model performance in this empirical study was evaluated and compared in terms of MMRE with the best model determined by selected previous studies in Table 4.1. Table 4.15 presents the performance of MMRE obtained by the best model determined in previous selected studies and the proposed model for the QUES and UIMS datasets. Boldface values in the table indicate the best results (i.e., the lowest MMR). This table demonstrates that the

proposed model, that is, KNN as an individual model or as the base model in additive regression, achieved the best MMRE value against all selected previous studies for the QUES dataset and proved to be the 7th best model in the UIMS dataset. Only MMRE was considered, as most of the studies used this measure, and other measures (e.g., Pred) were not used in some studies.

Table 4.15: Performance of MMRE obtained by previous selected studies and proposed work for the QUES and UIMS datasets.

ID	Prediction model	QUES	UIMS
S1	Bayesian network	0.45	0.97
	Regression tree	0.49	1.53
	Backward elimination	0.40	2.58
	Step-wise selection	0.39	2.47
S2	MARS	0.32	1.86
	MLR	0.42	2.70
	SVR	0.43	1.68
	ANN	0.59	1.95
	RT	0.58	4.95
S3	TreeNet	0.42	1.57
	MARS	0.32	1.86
	MLR	0.42	2.70
	SVR	0.43	1.68
	ANN	0.59	1.95
	RT	0.58	4.95
S4	MLP	0.71	1.39
	RBF	0.96	3.23
	SVM	0.44	1.64
	M5P	0.54	1.67
	Ensemble model	0.41	0.97
S5	FL model	0.27	0.53
	BN model	0.45	0.97
	MARS model	0.32	1.86
S6	Hybrid neural network - A1	0.37	0.31
	Hybrid neural network - A2	0.35	0.47
	Hybrid neural network – A3	0.38	0.18
S7	MLP	0.71	1.39
	RBF	0.96	3.23
	SVM	0.44	1.64
	M5P	0.54	1.67
	Ensemble model(AVG)	0.58	1.46
	Ensemble model (WT)	0.49	1.21
	Ensemble model (BT)	0.41	0.97
S8	Neuro-GA	0.37	0.31
S9	Neuro-fuzzy approach	0.33	0.28
Proposed model (this study)	Additive regression (KNN)	0.26	0.74

Figure 4.16 summarises the comparison of the MMRE values between the best model in the previous selected studies and the best model in this proposed work for the QUES dataset. The superior MMRE value (0.26) was obtained for the proposed model: KNN as an individual model or as the base model in additive regression, followed by the FL model in S5 (0.27) and the MARS model (0.32) in S2, S3 and S5. Notably, only the present model nearly meets the criteria of accurate prediction that states that MMRE should be equal to or lower than 0.25 [67].

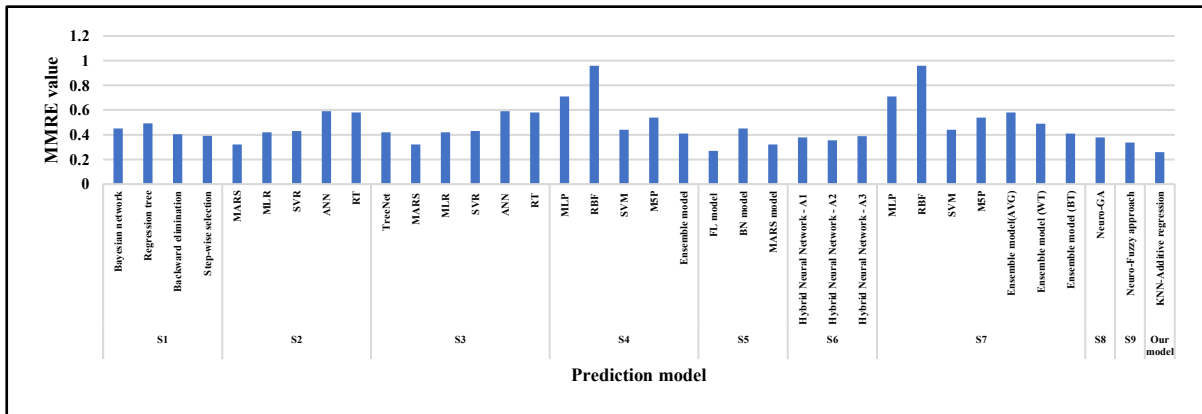


Figure 4.16: MMRE values obtained by the best model in the previous selected studies and the best model for the QUES dataset.

Figure 4.17 illustrates the comparison of the MMRE values between the best model in the previous selected studies and the best model in this proposed work for the UIMS dataset. The main finding is that the hybrid neural network – A3 has an MMRE value of (0.18) in S6, whereas the neuro-fuzzy approach has an MMRE value of (0.28) in S9. The proposed model proved to be the 7th best model compared with those in the previous selected studies. However, only the hybrid neural network, namely, A3 fulfils the criteria for accurate prediction that states that MMRE should be equal to or lower than 0.25 [67].

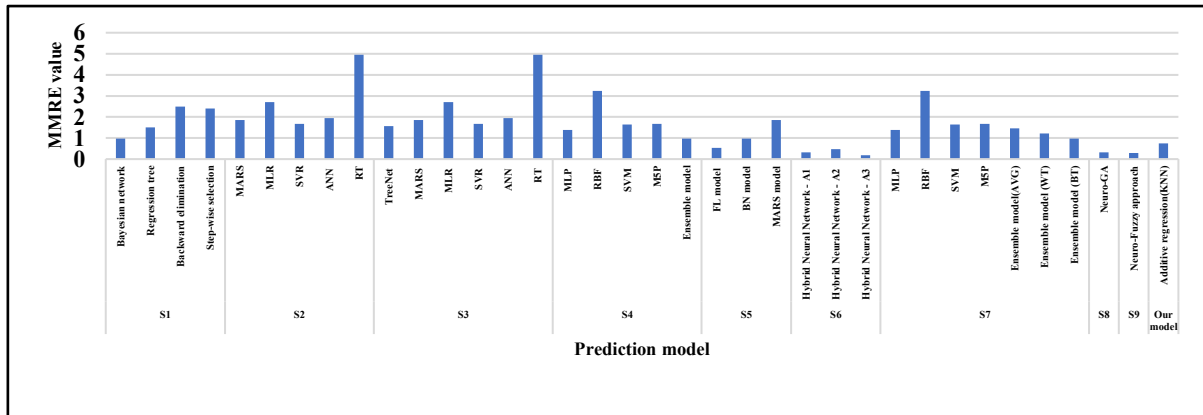


Figure 4.17: MMRE values obtained by the best model in the previous selected studies and the best model for the UIMS dataset.

4.5.3 Impact of the parameters tuning using caret

Considering the values of MMRE and MAE for the default and tuned parameters with respect to each prediction model, Table 4.16 and Table 4.17 demonstrate the impact of parameter tuning on the QUES and UIMS datasets, respectively. From these tables, it may be observed that the prediction accuracy was improved for most prediction models with parameter tuning (i.e., 22 Yes out of 32 prediction models). However, most individual models with default parameters performed better than individual models with parameter tuning (i.e., 3 Yes out of 10 individual models). Additionally, ensemble models were more sensitive to their parameter settings in terms of improvement than individual models. With respect to parameter tuning, the prediction accuracy of the ensemble models in most cases was better than that of the individual models. The most interesting finding was that KNN as the base model in the additive regression with parameter tuning achieved the best prediction accuracy in both datasets. Finally, the results obtained from Table 4.16 and Table 4.17 show that parameter tuning improved the prediction accuracy of most models with default parameters, and this improvement is clearly observed in the ensemble models.

Table 4.16: Impact of the parameters tuning on QUES dataset.

Prediction model	MMRE		MAE		Does the parameters tuning improve the performance of the default parameters model?
	Default parameters	Parameters tuning	Default parameters	Parameters tuning	
Individual models					
RT	0.45	0.68	26.24	30.64	No
MLP	0.50	1.19	28.71	40.54	No
KNN	0.26	0.43	19.75	24.29	No
M5Rules	0.49	0.55	23.39	25.12	No
SVR	0.38	0.27	20.33	20.17	Yes
Homogeneous ensemble model – Bagging					
RT	0.48	0.30	22.61	14.96	Yes
MLP	0.39	0.20	19.89	11.11	Yes
KNN	0.30	0.10	19.04	7.19	Yes
M5Rules	0.45	0.46	28.72	17.96	Conflicting results
SVR	0.38	0.30	20.42	17.16	Yes
Homogeneous ensemble model – Additive regression					
RT	0.47	0.47	26.23	16.23	Yes
MLP	0.52	0.21	28.43	8.54	Yes
KNN	0.26	0.00	19.75	0.00	Yes
M5Rules	0.47	0.74	23.85	41.89	No
SVR	0.35	0.29	19.85	15.46	Yes
Heterogeneous ensemble model – Stacking					
Stacking	0.32	0.42	19.80	17.80	Conflicting results

Table 4.17: Impact of the parameters tuning on UIMS dataset.

Prediction model	MMRE		MAE		Does the parameters tuning improve the performance of the default parameters model?
	Default parameters	Parameters tuning	Default parameters	Parameters tuning	
Individual models					
RT	4.52	1.20	41.74	28.98	Yes
MLP	1.32	2.85	23.39	43.43	No
KNN	0.74	1.20	23.41	26.51	No
M5Rules	1.24	1.11	26.64	27.87	Conflicting results
SVR	1.84	0.98	26.63	25.33	Yes
Homogeneous ensemble model – Bagging					
RT	3.08	1.97	31.63	19.91	Yes
MLP	1.17	0.65	20.10	10.81	Yes
KNN	0.83	0.19	20.80	7.36	Yes
M5Rules	1.31	1.17	21.65	16.19	Yes
SVR	1.65	1.30	24.79	17.34	Yes
Homogeneous ensemble model – Additive regression					
RT	4.56	0.40	42.14	10.37	Yes
MLP	1.87	1.08	26.66	7.45	Yes
KNN	0.74	0.01	23.41	0.26	Yes
M5Rules	1.16	1.08	27.23	15.39	Yes
SVR	1.20	0.66	24.57	11.90	Yes
Heterogeneous ensemble model – Stacking					
Stacking	2.45	0.63	33.51	19.64	Yes

4.5.4 Discussion and answers to research questions for the first empirical study

This section discusses and answers RQs for the first empirical study.

RQ4.1) How effective are individual models at predicting change maintenance effort?

KNN produced superior results for most accuracy prediction measurements in both the QUES and UIMS datasets. Moreover, KNN as an individual model or as the base model in additive regression achieved the best accuracy prediction among all investigated models in both datasets, and neither homogeneous nor heterogeneous models improved its prediction accuracy. Additionally, this model in the QUES dataset is the only model that nearly fulfils the criteria of accurate prediction proposed in Chapter 3. These results provide substantial evidence of the effectiveness of KNN in the prediction of software maintainability. This evidence is in agreement with the findings of Chen and Shah [178], which state the success of KNN in predicting either regression or classification problems. KNN is considered stable model that implements simply. Also, KNN performs well with small datasets [181], such as QUES and UIMS datasets.

RQ4.2) How do homogenous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

Bagging improved the prediction accuracy over almost all individual models except RT in the QUES dataset, M5Rules in the UIMS dataset, and KNN in both datasets. The findings of the statistical tests indicate that there were no significant differences in terms of the residual values among all the individual models and bagging ensemble models, and the effect sizes were small. These findings are also consistent with a previous study, thereby indicating that the bagging ensemble models have high potential for producing more accurate predictions than individual models [107]. The notable improvement of prediction accuracy in individual models verified that the bagging ensemble model effectively improves performance when applied to datasets with a limited amount of data (i.e., QUES and UIMS contain 71 and 39 classes, respectively) [133]. However, this was not observed in two datasets. Bagging aims to decrease variance by randomly selecting different training sets with replacement [128]. For this reason, applying bagging to stable models (e.g., KNN) has little value as their output results in few changes in the training data from sampling [128, 182]. However, KNN becomes unstable for a small number of nearest neighbours (K) [183], and Caprile et al. reported that this number should be higher than one [184]. In addition, KNN is considered a simple model and performs well with small datasets. In contrast, RT is unstable and suffers from variance; therefore, bagging improves the prediction accuracy of RT in the UIMS dataset [128]. Additionally, bagging had

a minor influence on SVR; this result agrees with the previous literature, showing that an ensemble model had little improvement on SVM [179].

The additive regression ensemble model had a positive impact on the prediction accuracy for most individual models except RT in the QUES dataset and MLP in the UIMS dataset, and no significant differences in terms of residual values and effect sizes were small. The additive regression ensemble model had no impact on KNN in either dataset. A possible explanation for this is that KNN is an instance-based rather than model-based approach. Additive regression starts with a null ensemble and sequentially adds the KNN predictions. The second and subsequent models are aimed at predicting the residuals (errors), and if no instances able to predict these residuals are found, KNN will be unable to make any improvements to the initial predictions. Overall, KNN as the individual model or the base model in additive regression was the best model to predict software maintainability. A similar conclusion was reached by Zahara et al., who stated that KNN was the best model to predict reusability evaluation of OO software components [185].

RQ4.3) How do heterogeneous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

The stacking ensemble model improved the prediction accuracy over all the individual models except KNN in the QUES dataset. The results obtained by statistical tests reported no significant difference between the individual models and the stacking ensemble model, and the effect sizes were small. This finding is in agreement with those of other studies, and it suggests that stacking achieved a better result when selecting a collection of several models from different types [122]. The relative advantage of stacking is to combine five individual models from different types and gain the strengths and weaknesses of this combination. A potential explanation for this result may be the low performance of other individual models (RT, MLP, M5Rules and SVR). For this reason, the performance of the stacking ensemble model was worse than that of KNN, but better than that of the other models. However, stacking decreased the accuracy prediction of all the individual models except RT in the UIMS dataset. Another possible explanation for this is that RT produced lower prediction accuracy. For this reason, when RT was integrated with other models in stacking, a lower prediction accuracy was reported. When comparing the results of stacking ensemble models with those of previous

studies that used different ensemble models, the prediction accuracy of ensemble models was different for various datasets [16].

RQ4.4) Which prediction models (the best-proposed model in this empirical study or the best-model in the selected studies) provide the best prediction accuracy?

KNN as an individual model or the base model in additive regression exhibited the best performance compared with previous selected studies. In addition, it nearly met the criterion of accurate prediction (i.e., $MMRE \leq 0.25$ [67]) in QUES dataset. With respect to the tuning parameters, KNN as the base model in additive regression outperformed all models in this empirical study.

RQ4.5) What are the effects of parameter tuning on the performance of the prediction models?

The parameter tuning increased the prediction accuracy over most of the investigated models with default parameters. This is consistent with what has been found in previous studies indicating that parameter tuning using the caret package in R enhanced the performance of the machine learning models [141, 166]. However, ensemble models were more influenced by parameter tuning than the individual models. In addition, the prediction accuracy of ensemble models with parameter tuning was better than that of all the individual models. Therefore, it is recommended to create ensemble models with parameter tuning to increase the performance of software maintainability. Further work needs to be done to use a statistical test to investigate the performance difference between the individual and ensemble models using parameter tuning.

4.6. Threats to Validity

The threats to validity usually appear in any empirical software engineering study that uses open-source software projects [186]. The following threats to validity exist in this empirical study:

4.6.1 Threats to external validity

External validity indicates a limited generalisation of the results outside the empirical study settings [186], and is based on the dependent variable used in this empirical study. A well-

known and common dependent variable (i.e., change metric) applied in several studies [7, 11-13, 15-18, 88, 152, 175] was used. A higher value of this metric indicates a higher maintenance effort or lower maintainability. The CHANGE metric is related to the number of changes that are likely to be made to a class, whereas maintainability refers to the ease to implement maintenance changes. The change metric has proven to be a perfect indicator in predicting the maintenance effort and has strong relationships with other metrics (independent variables) [9]. Therefore, the dependent variable is acceptable and there is no threat to external validity.

4.6.2 Threats to internal validity

Internal validity is the capability to present the results with different experimental variables [187]. To avoid these threats, datasets that have already been investigated in the literature were used [7, 11-13, 15-18, 88, 152, 175]. However, these datasets include only two types (i.e., QUES and UIMS) and contain a limited number of classes. Consequently, this limitation may negatively affect the performance of the prediction models. Moreover, these datasets were extracted from real-world systems that were designed in the ADA language, which limits the depiction of all software systems. To control these threats, this study can be extended by considering more recent and large datasets that are collected from various open-source systems. Additionally, further research can investigate real-world systems designed by other programming languages (e.g., C++, Java, C or C#).

4.6.3 Threats to the construct validity

Construct validity measures the relationship between the dependent and independent variables [188]. To prevent these threats, ten metrics that have been widely performed and validated by previous studies as predictors for software maintainability were applied [7, 11-13, 15-18, 88, 152, 175]. However, machine learning models were employed without applying FS techniques. These techniques help to determine the best subset metrics to accurately predict software maintainability. Regarding parameter tuning, the caret package in R, which automatically applies parameter tuning in each model, was used. However, the performance of these models may improve with the manual application of parameter tuning in each model. For example, a specific number for the K parameter in the KNN model was defined. Thus, in a future work, automated and manual parameter tuning will be compared. In addition, the

performance difference between individual and ensemble models using parameter tuning will be explored.

4.6.4 Threats to the conclusion validity

Conclusion validity relates to the statistical relationship between the results and the output of the experiment, which impacts the capability to reach the right conclusion [187]. Ten-fold cross-validation was used to prevent the threat of conclusion validity. This method aims to decrease biased results by selecting ten different tests from the dataset. However, the present results were compared with those of previous selected studies in Table 4.1, and some of them, namely, S2, S3, S5, S6 and S9 did not use ten-fold cross-validation. Therefore, there exists a conclusion validity threat that the results may not be entirely comparable. Moreover, the ANOVA test was used in this empirical study to explore if there are any statistically significant differences between the means of four groups (i.e., individual model and this individual model as the base model in bagging, additive regression and stacking). This empirical study includes more than two pairs (i.e., four groups) and continuous datasets; therefore, a parametric test (ANOVA) is appropriate to implement, which has an advantage to produce more reliable results than non-parametric statistical test. However, the parametric statistical test, requiring some assumptions (e.g., normally distributed and independent observations in the datasets). Although these assumptions are fulfilled, and this method is based on only ten runs. As a result, there is a threat to conclusion validity.

4.7. Conclusion of the first empirical study

Prior studies have documented the effectiveness of machine learning models in predicting software maintainability of the OO system. However, these studies used a wide variety of individual models and had limited focus on ensemble models. Moreover, the prediction accuracy of their models was low according to the proposed criteria.

This chapter empirically evaluated the application of homogeneous (bagging and additive regression) and heterogeneous (stacking) ensemble models in predicting software maintainability of OO systems and investigated their accuracy prediction over individual

models (RT, MLP, M5Rules, KNN and SVR). All these models were run on the QUES and UIMS datasets that were obtained from two different OO systems.

The results obtained from both datasets provides several insights as follows:

- Among the individual models, KNN achieved the best prediction accuracy in both datasets. It seems possible that these results are due to the small size of the datasets, as KNN performs well with smaller datasets [181];
- Although bagging improved the prediction accuracy over all individual models except RT and KNN on the QUES dataset and KNN and M5Rules on the UIMS dataset, the differences were not statistically significant between bagging and individual models, and the effect sizes were small. The potential reason for the improvement is that bagging reduced variance by randomly selecting different training sets with replacement [128]. However, bagging did not improve the prediction accuracy of stable models, such as KNN or strong models, such as SVR;
- The additive regression ensemble model also increased the prediction accuracy over all individual models except RT on the QUES dataset and MLP on the UIMS dataset. Again, there were no significant differences between additive regression and individual models, and the effect sizes were small;
- The stacking ensemble model increased the prediction accuracy of the individual models in the QUES dataset because these individual models produced low prediction accuracy and when they were integrated together, stacking became better than these models. However, stacking decreased the prediction accuracy of all individual models except RT on the UIMS dataset because RT did not perform well as the individual models, and when it was integrated with other models in stacking, it reduced the performance of stacking. Regarding the statistical tests, the differences were not statistically significant between the group population means (i.e., individual and stacking models) in both datasets, and the effect sizes were small;
- Although there were no significant differences, the homogeneous ensemble model exhibited better performance results than the heterogeneous ensemble model on UIMS and QUES datasets. Additive regression showed superior prediction accuracy compared to the bagging ensemble model in both datasets;

- KNN as the individual model or as the base model in the additive regression ensemble model achieved the best prediction accuracy compared to all the investigated models. In terms of the proposed criteria and when compared to selected previous studies, this model showed the best improvement in the QUES dataset, whereas it proved to be the 7th best model in the UIMS dataset;
- The parameter tuning improved the prediction accuracy of the ensemble models but not that of the individual models in most cases. KNN as the base model in additive regression in the parameter tuning achieved the best prediction accuracy.

These findings extend the current knowledge regarding the capability of ensemble models to improve the prediction accuracy of individual models. However, there were no significant differences between the individual and ensemble models. It is difficult to explain this result, but it might be related to the limited size of the dataset. To resolve this issue, this empirical study can be extended by using more recent and larger datasets for software maintainability prediction. Therefore, the next chapter will replicate this empirical study across various large representative datasets extracted from open-source software systems with various programming languages. This replication will allow further investigation on the impact of advanced machine learning models (homogenous and heterogeneous ensemble models) over existing individual models to predict software maintainability.

Chapter 5. Second Empirical Study: Ensemble Techniques to Predict Change Maintenance Effort Using More Recent and Larger Datasets

In this chapter, the impact of the same prediction models used in Chapter 4 is empirically investigated in the context of predicting change maintenance effort of OO systems, along with one more heterogeneous ensemble model, namely APE. These models are applied on five different datasets (i.e., bug prediction datasets [57]) extracted from real-world software systems. This chapter aims to further explore the application of ensemble models on more recent and larger datasets, and to compare and evaluate the proposed models with the selected models using the Auto-WEKA tool.

5.1. Introduction

The performance of machine learning techniques is closely related to the size and characteristics of the dataset on which they are trained and tested. As mentioned in Chapter 4, considerable uncertainty still exists in relation to the interpretation of machine learning techniques applied to the problem of software maintainability because most of them were built and evaluated on relatively limited and old datasets [7, 11-13, 15-18, 88, 152, 175], namely QUES and UIMS datasets [9], and few studies used ensemble models [16, 88]. Additionally, there is no clear evidence of which models provide high prediction accuracy. In Chapter 4, the impact of ensemble models using these datasets was empirically evaluated and obtained, and the results indicated improved accuracy over individual models. However, Chapter 4 used the aforementioned old and small datasets that contained only a few rows. Consequently, there is a clear need to explore the performance on more recent and larger datasets.

Data pre-processing techniques are essential for the production of high-quality datasets and have a positive impact on building accurate prediction models [189]. Prior studies have employed a wide variety of data pre-processing techniques on different software quality datasets (e.g., NASA datasets) to achieve reliable model prediction [190-195]. These studies

emphasised that the implementation of an accurate model relies on high-quality datasets. The datasets were used in this chapter collected for the purposes of the bug prediction. However, these datasets include metrics that suitable to predict change maintenance effort. Therefore, pre-processing techniques are performed on the bug prediction datasets [57] to achieve the following objectives:

- Select appropriate source code metrics (independent variables) as predictors of software maintainability;
- Determine the CHANGE metric (dependent variable) by calculating the number of lines changed (added or deleted) per class during the maintenance period;
- Evaluate the quality of the bug prediction datasets and convert them into software maintainability prediction datasets using pre-processing techniques.

To build an accurate maintainability prediction model, prior studies have applied several types of machine learning models with configurations manually set (i.e., parameter tuning [196] or selected features [18]). However, this approach requires considerable time and effort. This study demonstrates the application of Auto-WEKA as a new, rapid, and automated tool to identify the best accurate prediction model among sets of models, with different parameters and features, using Bayesian optimisation [149, 150].

The main contributions of this chapter as follows:

- Pre-processing was applied on the bug prediction datasets to convert and ensure their suitability for software maintainability prediction;
- Recent, large public datasets that have not been applied before in the prediction of software maintainability, were used;
- This empirical study investigated the capabilities of both homogeneous and heterogeneous ensemble models and found that the evaluated ensemble models increased the prediction accuracy over most of the individual models. Also, there were significant differences between some of the individual and heterogeneous ensemble models. In the homogeneous ensemble models, the performance of bagging was better than additive regression, whereas in the heterogeneous ensemble models, APE achieved better prediction accuracy than stacking. However, in most cases, neither homogeneous nor heterogeneous ensemble models increased the performance of SVR and KNN;

- The Auto-WEKA tool was used. To the author’s knowledge, no previous work used this tool to predict software maintainability.

5.2. Motivation

Several studies have investigated different pre-processing techniques for software quality datasets [190-195]. However, these pre-processing techniques depend on the different problems posed by particular datasets. Although most of the pre-processing techniques are performed using a specific algorithm, some of the pre-processing techniques are manually implemented, such as integrating attributes from multiple sources or aggregating two attributes. Table 5.1 presents previous studies that applied preprocessing techniques on a software quality dataset. Many of these studies used public datasets from NASA or the PROMISE repository [190-194], and only one study extracted a dataset from an open-source system [195]. It should be mentioned that creating software quality datasets, such as defect datasets, is a challenging and time-consuming task [191, 192]

Table 5.1: Summary of previous studies that applied pre-processing techniques on the software quality datasets.

Author	Ref	Dataset name	Dataset Problem	Pre-processing techniques to fix problem
Sunghun Kim et al. (2011)	[195]	Defect dataset from SWT and Debug projects in Eclipse 3.4 system	Data noise	Algorithm to detect and eliminate noises
David Gray et al. (2011)	[191]	NASA datasets	Repeated data, noise, incorrect data	Meticulously documented data cleansing process
David Gray et al. (2012)	[192]	NASA datasets	Repeated data, noise, incorrect data	Data cleansing process to prepare dataset for binary classification and remove noise
Martin Shepperd et al. (2013)	[193]	NASA datasets	Implausible, duplicate instances, missing and conflicting values	Algorithm used to transform and preprocess the data
Jean Petri 'c et al.(2016)	[190]	NASA datasets	Inconsistent data	Introduce integrity checks for cleaning dataset
Baljinder Ghotra et al. (2017)	[194]	18 datasets from NASA and PROMISE	Low classification accuracy and misclassification rates	FS techniques

The most apparent gap from previous studies in Table 5.1 is that no studies performed pre-processing techniques on the software maintainability dataset. This may be due to the limited availability of public datasets for software maintainability prediction [5]. There is limited research in the software maintainability prediction area [13], whereas defect prediction is a relatively popular research field in software engineering [195].

Although the use of parameters tuning is recommended by Fu et al. [142] to improve the performance of the prediction model, a limited number of studies have applied this approach

for software maintainability prediction. For example, a study by Dahiya et al. [196] applied a genetic algorithm to optimise the parameters of a fuzzy logic-based maintainability metrics system. Moreover, research studies have investigated FS as an alternative method to improve the prediction accuracy. Kumar and Rath enhanced the performance of the software maintainability prediction model using one method of FS, namely rough set analysis [18], whereas Reddy and Ojha [83] used sets of seven FS: best first, linear forward selection, greedy stepwise, evolutionary search, genetic algorithm, PSO, and Tabu search. However, tuning parameters and selecting features require several attempts and significant effort to obtain optimal results. Therefore, Auto-WEKA has been recently proposed to resolve this problem and to provide mechanisms for both tuning parameters and selecting features [149, 150]. Finally, the main motivation of this chapter is to use recent, large, and high-quality datasets that are suitable for the software maintainability prediction, investigate the application of several ensemble models on these datasets to determine the best model to predict software maintainability, and compare the results with models selected by Auto-WEKA tools.

5.3. Research Method

This chapter aims to increase the prediction accuracy of software maintainability in OO systems by exploring the impact of ensemble models (homogeneous and heterogeneous). The second empirical study consists of two studies, namely 5.A and 5.B. RQs for the second empirical Study 5.A are provided as follows:

RQ5.A.1) What are the suitable metrics (independent variables) in the bug prediction datasets to predict software maintainability?

RQ5.A.2) How can the dependent variable calculate the CHANGE metric from the bug prediction datasets?

RQ5.A.3) How to evaluate the quality of the bug prediction datasets using preprocessing techniques?

RQ5.A.4) How much can prediction models increase or decrease the performance compared to a baseline (i.e., ZeroR)?

RQ5.A.5) How effective are individual models at predicting change maintenance effort?

RQ5.A.6) How do homogenous ensemble models perform in the context of predicting change maintenance effort when compared to individual models?

RQ5.A.7) How do heterogeneous ensemble models perform in the context of predicting change maintenance effort when compared to individual models?

Figure 5.1 depicts an overview of the research method performed to predict software maintainability of OO systems. This method consists of the following steps.

Step 1. Evaluate the bug prediction datasets [57] extracted from Java systems to determine suitable metrics. The potentially selected datasets contain several changes to fix maintenance issues. These datasets were collected on the first released version and multiple updated versions of the systems.

Step 2. Extract source code metrics (independent variables) from the bug prediction datasets: the result from this step are the C&K [26] and OO metrics. These metrics are extracted using various tools, including infusion, Moose and Churrasco [57].

Step 3. Calculate the change metric (dependent variable): the result from this step is performed by manually summing two metrics from the bug prediction datasets, namely lines added until and lines removed until. These metrics are calculated between subsequent versions by evaluating system log files, counting the number of lines changed in each class (insertion or deletion were counted as 1), and modifications (inserting and deleting were counted as 2).

Step 4. Create the datasets by combining the source code metrics from Step 2 with the change metric from Step 3 into one file. The number of records in the dataset was equal to the number of classes in the selected system.

Step 5. Divide datasets into ten sets using ten-fold cross-validation: the dataset is divided into a training set to create machine learning models and a test set to evaluate the performance of machine learning models.

Step 6. Construct sets of individual models: these models are selected from different categories (see Table 3.1).

Step 7. Construct ensemble models from these individual models. The ensemble models include two main types: homogeneous (i.e., bagging and additive regression) and heterogeneous ensemble models (i.e., stacking and APE).

Step 8. Predict software maintainability: the output of the previous steps is evaluated and compared to identify the most accurate prediction model.

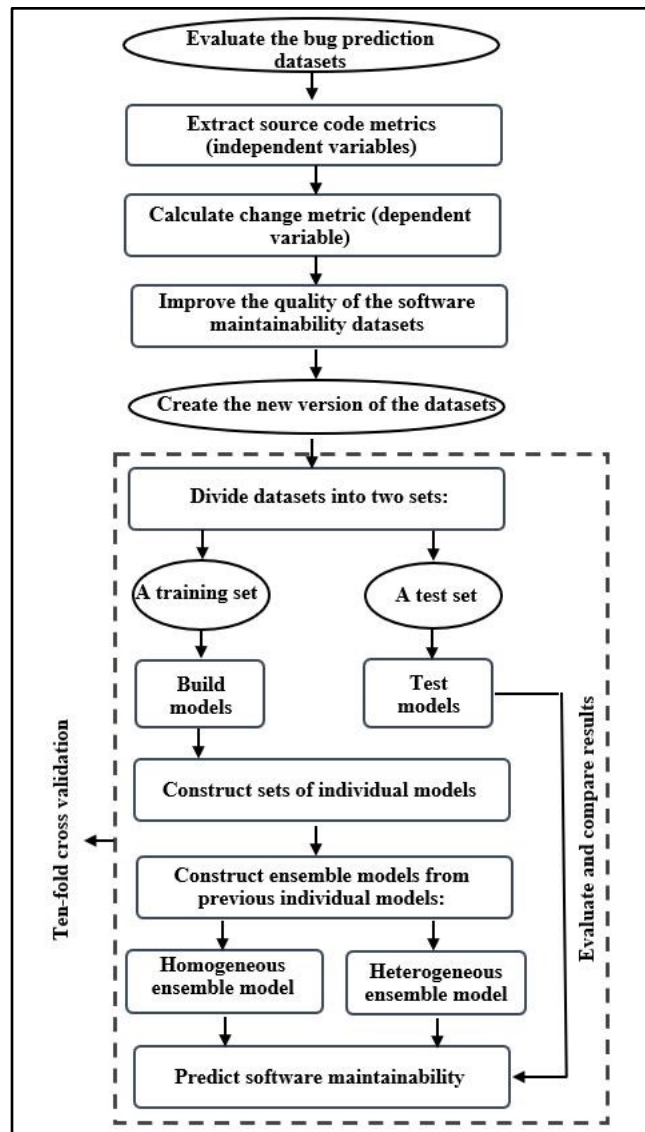


Figure 5.1: Framework of the research method.

Four main experiments were performed to answer these RQs for Study 5.A as described below:

Experiment 1 (RQ5.A.4). The ZeroR model was applied to each dataset to determine a baseline; then, it was used as a benchmark to compare the performance of the prediction models.

Experiment 2 (RQ5.A.5). Sets of individual models (i.e., RT, MLP, KNN, M5rules and SVR) were built, and the best one for predicting software maintainability in each dataset was identified.

Experiment 3 (RQ5.A.6). Homogeneous ensemble models (i.e., bagging and additive regression) were constructed and the results were compared with the performance of individual models.

Experiment 4 (RQ5.A.7). Heterogeneous ensemble models (i.e., stacking and APE) were constructed and the results were compared with the performance of individual models.

This chapter also aims to use the Auto-WEKA tool to identify the best model to predict software maintainability by applying it to previous datasets. To achieve the highest prediction accuracy, this tool tries several models with different tuning parameters and selected features.

The second part of the empirical study is performed to respond to the following RQs for Study 5.B:

RQ5.B.1) What is the best model selected by Auto-WEKA to predict software maintainability in each dataset?

RQ5.B.2) How many configurations are attempted to select the best model?

RQ5.B.3) What are the tuning parameter settings in the selected model?

RQ5.B.4) What are the selected features in the selected model?

RQ5.B.5) What are the MAE and MMRE values for the selected models?

RQ5.B.6) What is the performance of the model selected by Auto-WEKA compared with that of the baseline (i.e., ZeroR)?

RQ5.B.7) What is the performance of the model selected by Auto-WEKA compared with that of performance of the best model in Study 5.A?

5.4. Experimental Data Setup

The following subsections provide information about details of the data pre-processing, and an explanation of dependent and independent variables used for the software maintainability datasets. Furthermore, they present the descriptive statistics and correlation between the metrics in the datasets.

5.4.1 Data pre-processing

This section aims to apply data pre-processing techniques to bug prediction datasets to produce a new and high-quality version of these datasets suitable for software maintainability prediction. Therefore, this section aims to answer RQ5.A.1, RQ5.A.2 and RQ5.A.3 for Study 5.A.

The bug prediction datasets have several irrelevant metrics and do not provide a direct measure for software maintainability (i.e., CHANGE metric). Additionally, they may contain some issues, such as incomplete, missing values, outliers or inconsistencies that negatively affect the data quality and prediction models. Therefore, data pre-processing is introduced to create a new version of the bug prediction datasets. The data pre-processing techniques include the following steps [125]:

- **Data reduction:** established to decrease the data size by employing aggregation, eliminating redundant features, selecting attribute subsets or clustering;
- **Data integration:** applied to integrate data from various sources into a separate coherent source. This technique is established to remove redundant data (where the same attribute appears again under a different name) and inconsistent naming (where the same attribute values appear under different names);
- **Data cleaning:** applied to clean noise, missing values and inconsistencies of data;
- **Data transformation:** performed to improve the accuracy and efficiency of data mining by transformation of the data. This technique is designed to aggregate, generalise or normalise the data.

Figure 5.2 depicts the framework of the pre-processing techniques used in Study 5.A. First, the original version of the bug prediction datasets [57] was evaluated. Second, sets of four primary pre-processing techniques were applied in the order proposed in [125]: data reduction, data integration, data cleaning and data transformation. Finally, a new version of the high-quality datasets was produced.

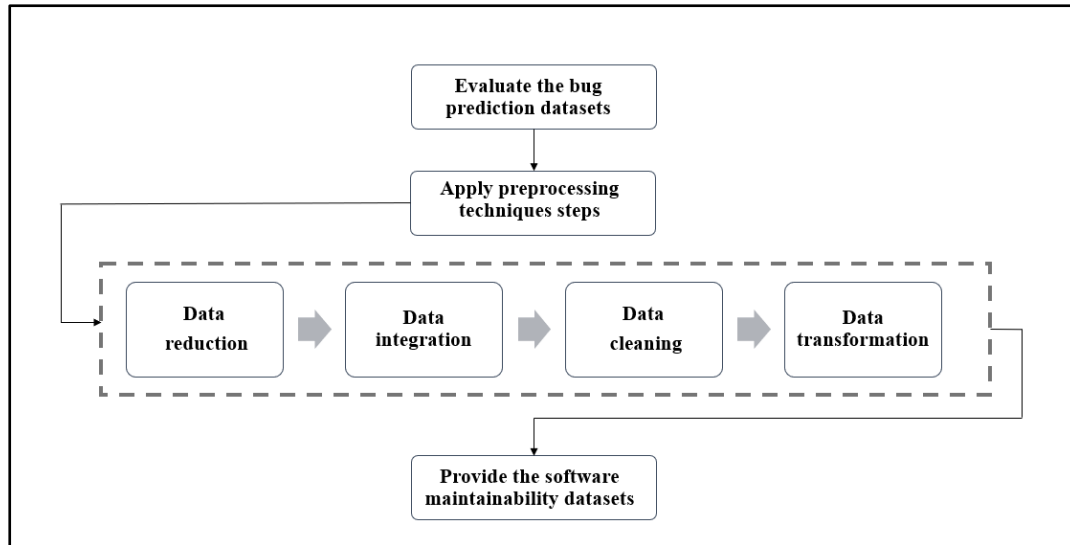


Figure 5.2: Framework of data pre-processing.

A. Evaluate the bug prediction datasets

This chapter uses the bug prediction datasets proposed by Marco Ambros et al. [57]. The original version of the bug prediction datasets includes five datasets extracted from open-source software systems. These datasets collected changes during a 3-year maintenance period and consist of 439 and 2,196 classes with the same number of files and metrics. Table 5.2 presents a summary of bug prediction datasets [57]. Each dataset includes eight files: biweekly-ck-values, biweekly-oo-values, bug-metrics, change-metrics, churn, complexity-code-change, entropy and single-version-ck-oo. However, several of these files are inappropriate and include irrelevant metrics for software maintainability prediction. Thus, pre-processing techniques were applied to determine suitable metrics (independent variables) to predict software maintainability (RQ5.A.1) and to calculate the CHANGE metric (dependent variable) to capture the element of maintainability (RQ5.A.2). In addition, these techniques aim to evaluate the quality of the bug prediction datasets and produce a new version of the dataset (i.e., software maintainability prediction datasets) (RQ5.A.3).

Table 5.2: Summary of the bug prediction datasets [57].

Dataset name	Time period	# Class	#Versions	#Transactions	#Features in single-version-ck-oo	#Features in change-metrics
Eclipse JDT Core	1.1.2005 - 6.17.2008	997	91	9,135	22	20
Eclipse PDE UI	1.1.2005 - 9.11.2008	1562	97	5,026	22	20
Equinox framework	1.1.2005 - 6.25.2008	439	91	1,616	22	20
Lucene	1.1.2005 - 10.8.2008	691	99	1,715	22	20
Mylyn	1.17.2005 - 3.17.2009	2196	98	9,189	22	20

B. Data reduction

The primary objective of data reduction is to decrease the size of the dataset by either removing or aggregating variables [125]. The bug prediction datasets have problems that require data reduction: the datasets have several unnecessary files and irrelevant metrics (independent variables) for the software maintainability prediction, and have not clearly defined the CHANGE metric (dependent variable) to predict software maintainability. Therefore, three methods were applied to address these problems. **First**, six unnecessary files of software maintainability were manually removed: biweekly-ck-values, biweekly-oo-values, bug-metrics, churn, complexity-code-change and entropy, and only single-version-ck-oo and change-metrics files remained. **Second**, five bug metrics were manually removed from the single-version-ck-oo file, namely bugs, nonTrivialBugs, majorBugs, criticalBugs and highPriorityBugs, because these metrics are not related to software maintainability. As a result, the remaining 17 metrics (i.e., OO and CK) may be performed as predictors for software maintainability (independent variable); the description of these metrics is provided in Section 5.4.3. **Third**, all metrics were removed from the change-metrics file except two: lines added until and lines removed until, as these metrics are necessary to calculate the CHANGE metric value. **Fourth**, lines added until and lines removed until were aggregated into the CHANGE metric (dependent variable), which captures the element of interest of software maintainability. The results from applying data reduction are two files: the single-version-ck-oo file, which has 17 metrics (independent variables), and a change-metrics file that has the CHANGE metric (dependent variable), calculated by aggregating lines added until and lines removed until metrics.

C. Data integration

Data integration aims to integrate data from many sources into a single coherent source [125]. This pre-processing technique assists in preventing inconsistencies and redundancies in the datasets. The results of the datasets from previous pre-processing techniques include two files: the single-version-ck-oo file, which has 17 metrics (independent variables), and a change-metrics file that has the CHANGE metric (dependent variable). These two files were combined into a single file with 17 OO and CK metrics (independent variables) and only one CHANGE metric (dependent variable) for software maintainability prediction datasets.

D. Data cleaning

The primary objective of data cleaning is to detect and remove missing, noisy, and outlier values from the dataset source [125]. The results of the datasets from the previous pre-processing techniques are one file with 17 OO and CK metrics (independent variables) and one CHANGE metric (dependent variable). First, these datasets were evaluated in terms of missing data, but none was found due to accurate data collection. Second, the software maintainability prediction datasets were assessed in terms of outliers by applying the Inter Quartile Range (IQR) filter in WEKA. This filter divides the dataset into three quartiles: *first quartile* (Q1), which is the centre value between the median and the lowest value of the dataset that divides the lowest 25% of the dataset from the highest 75%; *second quartile* (Q2), which is the middle value that divides the dataset into halves; and *third quartile* (Q3), which is the middle value between the highest and the median values of the dataset that divides the highest 25% of the dataset from the lowest 75% [197]. The IQR filter relies on interquartile ranges, which recognise any value outside the interval as outliers, $(Q1 - x(Q3 - Q1), Q3 + x(Q3 - Q1))$ [197], where x is a constant number. IQR is the difference between *third* and *first* quartiles $(IQR = Q3 - Q1)$ [197]. Consequently, this filter identifies the outliers by creating a new attribute, namely the outlier index. Third, the *Removewithvalues* filter was applied in WEKA to remove outliers in the datasets. The parameter values in this filter were changed as follows: the outlier index was inserted in the attribute index to determine the outlier attribute, which was inserted in the nominal index to determine outlier values. Finally, the result from applying data cleaning is that the software maintainability prediction datasets are free from outliers. Figure 5.3 shows the proportion of outliers removed during data cleaning.

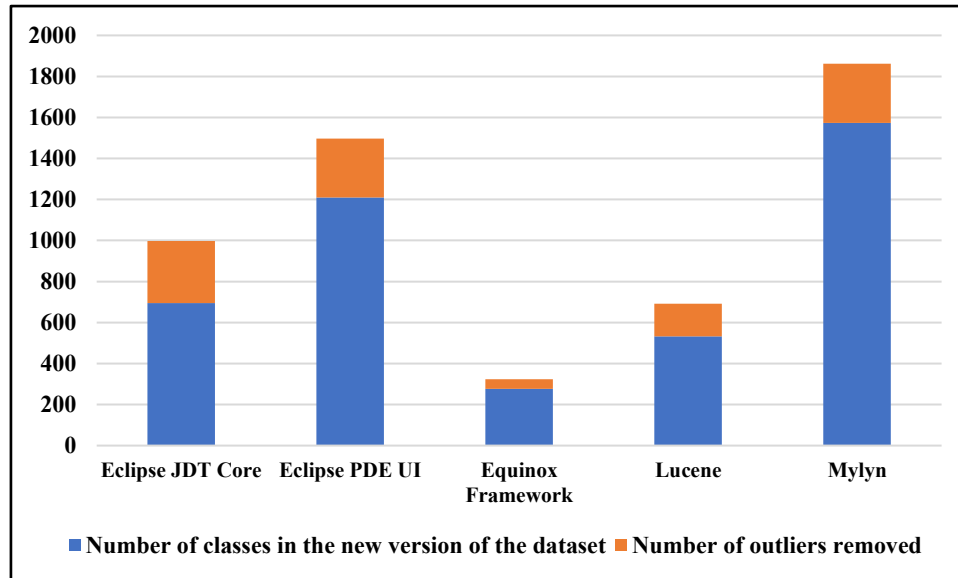


Figure 5.3: Proportion of outliers removed during data cleaning.

E. Data transformation

The main objective of data transformation is to convert data into format suitable for mining by smoothing, aggregation or normalisation. As the software maintainability prediction datasets do not have any issues requiring data transformation, applying these steps to the datasets is not necessary [125].

Table 5.3 presents summary results from applying pre-processing techniques on the bug prediction datasets. The final version after applying these techniques is called software maintainability prediction datasets. This version can be downloaded using the following link: <https://zenodo.org/record/4256386#.X6aMzogzY2w>

Table 5.3: Summary result of pre-processing techniques.

Pre-processing techniques	Problem	Method	Result
Data reduction	The datasets contain several unnecessary files of software maintainability	Manually delete six unnecessary files from the bug prediction datasets.	The datasets have only two files: single-version-ck-oo and change-metrics
	The datasets contain several irrelevant metrics of software maintainability	Manually delete five bug metrics from the single-version-CK-OO file and delete all metrics from the change-metrics file except two metrics.	The datasets have only two files: single-version-CK-OO, which has 17 CK and OO metrics, and change-metrics, which has two lines added until and lines removed until metrics
	The datasets need to identify dependent metric (CHANGE)	Calculate dependent metric (CHANGE) by aggregating lines added until and lines removed until metrics.	The datasets have only two files: single-version-CK-OO, which has 17 CK and OO metrics, and change-metrics, with only one metric: CHANGE
Data integration	The datasets include two separate files, one contains independent variables and the other the dependent variable.	Combine two files into a single file and modify the name of the file to software maintainability prediction.	The datasets have only one file (software maintainability prediction datasets) that has 17 OO and CK metrics (independent variables) and one CHANGE metric (dependent variable)
Data cleaning	The datasets contain outliers	Apply InterQuartileRange filter to identify outliers, then perform removewithvalues filter to remove outliers.	The software maintainability prediction datasets do not have outliers or missing values.
Data transformation	The datasets do not require data transformation	NA	NA

5.4.2 Dependent variable: maintainability.

Maintainability is a dependent variable that may be identified from several independent variables (metrics). In this study, maintainability is defined as the number of changes made in the class during the maintenance process. These changes are determined by calculating the number of added or deleted lines in each class during the maintenance period [10, 11, 13, 14] and correspond to the CHANGE metric described by L&H [9]. A higher value of CHANGE refers to higher maintenance effort, which implies lower maintainability. In the software maintainability prediction datasets, the dependent variable (CHANGE) was calculated by summing lines added until and lines removed until metrics, which indicate the lines added to or removed from the classes during the maintenance period, respectively [57]. The descriptive statistics of the CHANGE metric after removing outliers for each dataset of the software maintainability prediction datasets are shown in Table 5.4.

Table 5.4: Descriptive statistics for the CHANGE metric.

Dataset name	# Classes	Min.	Mean	Median	Max.	Standard deviation
Eclipse JDT Core	695	0	825.25	366	31245	2013.10
Eclipse PDE UI	1209	0	230.41	100	6824	424.56
Equinox Framework	276	0	242.59	56	5684	520.87
Lucene	532	0	106.97	19	3089	247.50
Mylyn	1573	0	110.80	29	9000	316.04

Figure 5.4 illustrates the residual boxplots of the CHANGE metric for each dataset of the software maintainability prediction datasets. This diagram depicts the first quartile (Q1), median, third quartile (Q3), and whiskers values. The line across the middle of the solid body of the boxes indicates the median value, which is the middle 50% of the data (between Q1 and Q3). Q1 and Q3, which are inside the solid body of the boxes, present values of 25% and 75% of the data, whereas the vertical lines (“whiskers”) represent the spread of values that fall within 1.5 times the inter-quartile range [197]. From the data in Figure 5.4, it is apparent that the median value of CHANGE in Eclipse JDT Core is higher than that of the other datasets, indicating that Eclipse JDT Core has the highest maintenance effort, which implies the lowest maintainability. In contrast, the median value of CHANGE in Lucene is lower compared to other datasets, which indicates that Lucene has the lowest maintenance effort, which implies the highest maintainability.

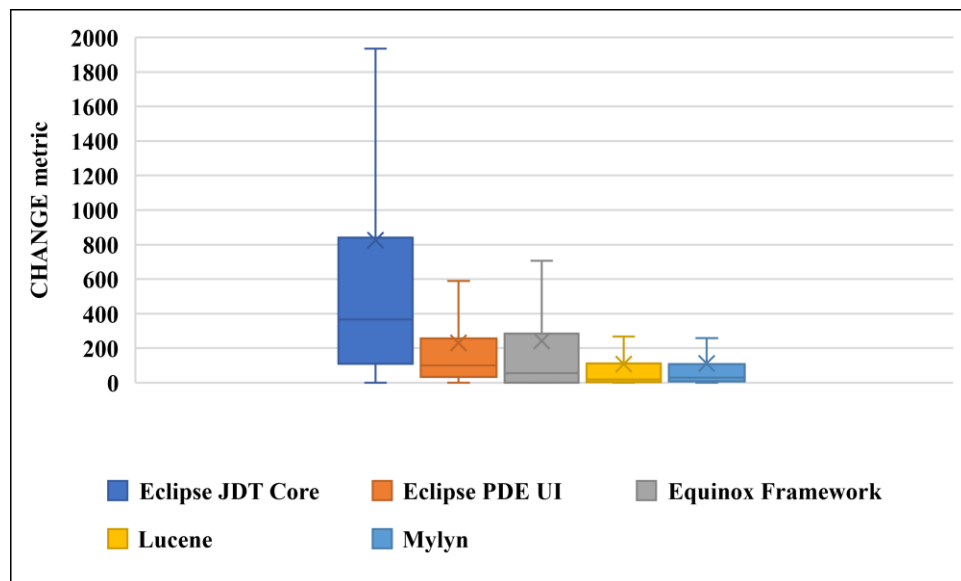


Figure 5.4: Boxplots of CHANGE metric.

5.4.3 Independent variable: metrics

The independent variables consist of 17 source code metrics that include 6 CK metrics [26] and 11 OO metrics. The details of the extraction of these metrics were described by Ambros et al. [57]. A brief description of each metric (independent variable) used in this chapter is presented in Table 3.5 in Section 3.4.2.

5.4.4 Descriptive statistics

Table B.1 in the Appendix compares the descriptive statistics, namely min, max, median, mean and Stdev for all the metrics (CK and OO metrics) across all software maintainability datasets. The results obtained from this table can be summarised as follows:

1. LOC metrics, which refers to lines of code, range from 0 to 2475;
2. In all datasets, the median value of the NOC metrics is zero. This implies that there are very few sub-classes in all datasets;
3. The mean values of the NOA metric in all datasets range from 3.0 to 4.9, which indicates that the average number of attributes is relatively close;
4. The mean values of DIT range from 1.22 to 2.59, which refers to all the datasets with almost the same depth of inheritance tree;
5. The median of NOPRM and NOPA in all datasets has zero values. This suggests that the number of private methods and the number of public attributes are very low;
6. The median value of the coupling metric (i.e., RFC) in Eclipse JDT Core and Eclipse PDE UI is higher than those in other datasets;
7. The median value of the number of methods (i.e., NOM metric) in all datasets ranged from 4 to 7. This refers to the number of methods that are almost similar in all systems;
8. The remaining metrics have a different number of median values. Accordingly, it is challenging to conclude the overall results.

Overall, there are no zero values for each metric in all datasets. Therefore, all metrics are considered relevant to measure software maintainability. Briand et al. [198] demonstrated that metrics that have zero values for all descriptive statistics should be removed from the analysis because they have no potential to be good predictors.

5.4.5 Correlation between metrics in the datasets

Figure 5.5 shows Pearson's correlation between source code metrics (independent variables), along with the change metric (dependent variable). From the summary data across software maintainability datasets in Figure 5.4, it seems that there are no red coloured circles, which implies no negative correlation between metrics. In addition, there are several weak positive linear correlations between metrics (e.g., LCOM with CBO), and this indicates that it is not necessary to eliminate any metrics because they all measure different elements of the code. Furthermore, there are multiple white circles that refer to uncorrelated metrics (e.g., NOC with other metrics). However, there are few strong positive linear correlations between some metrics, such as RFC and LOC in Eclipse JDT Core dataset and NOA and NOPA in Eclipse PDE UI dataset, but they were not removed. These results suggest that most of the metrics in software maintainability datasets are uncorrelated, and the characteristic of these datasets are different, hence maintainability prediction models are constructed for each dataset separately.

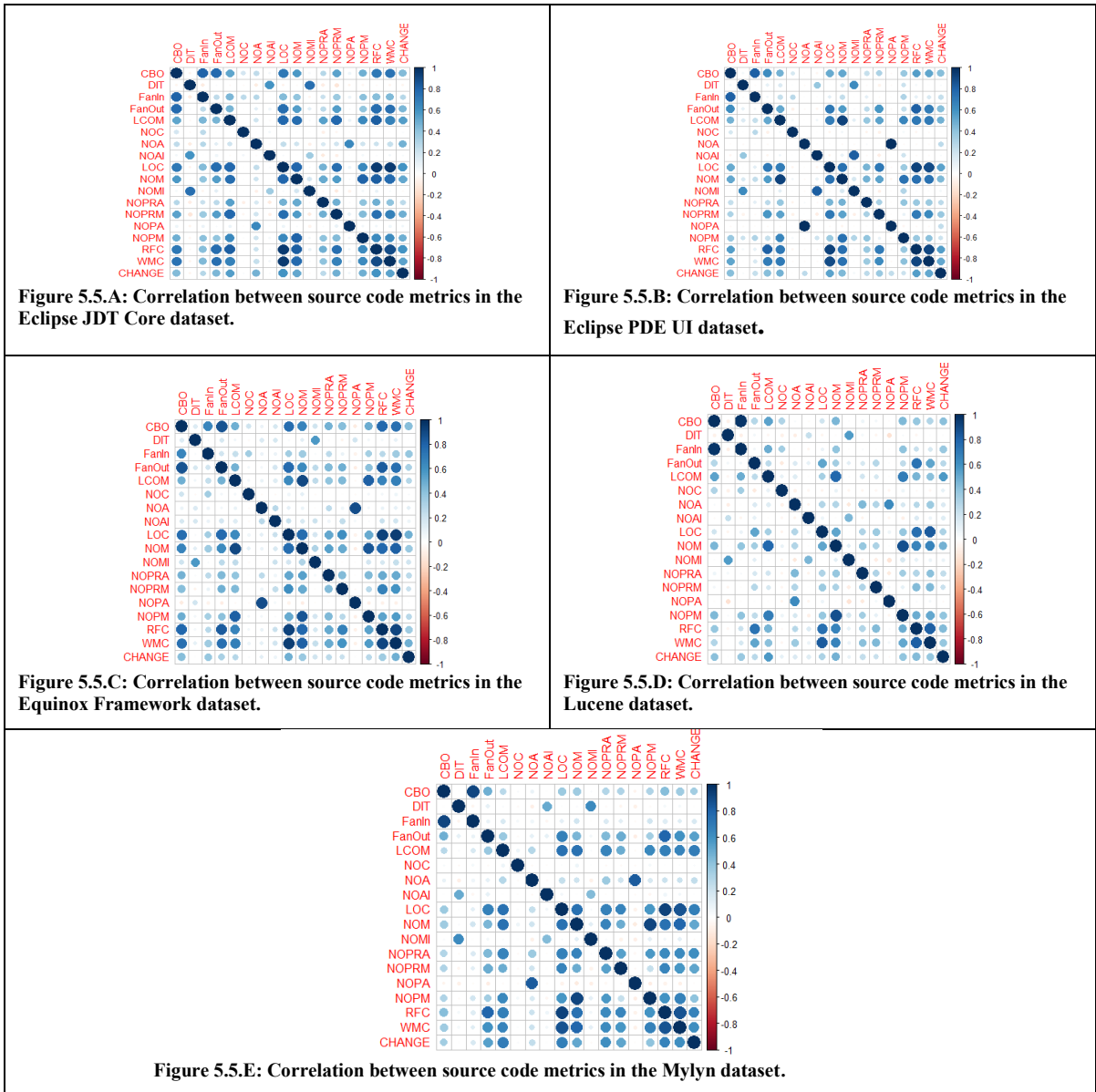


Figure 5.5: Correlation between source code metrics in the software maintainability datasets.

5.5. Results and Analyses

In this section, the results and analyses performed in Study 5.A are presented, which includes four experiments, each one addressing one research question. The prediction models were built using WEKA tools presented in Chapter 3 and the default values in their parameters [120]. In sequence, the results of the statistical tests are provided. Furthermore, this section presents the results and analyses performed in Study 2.B using the Auto-WEKA tool, described in Section 3.3.2.

Moreover, this section provides a one-way ANOVA test to examine H_0 , which states that all the group population means are equal, and H_1 , which states that at least one pair of means is different. Factor A includes the residuals values of the prediction models, grouped by each individual model used as the base model in the ensemble models. Therefore, 25 tables (5 individual models \times 5 datasets) are produced to compare the performance among the five prediction models (one individual model and four individual models as the base model in bagging, additive regression, stacking and APE).

5.5.1 Comparison between prediction models

To visualise the data distribution using quartiles, Figure 5.6 shows a boxplot of MRE for every prediction model investigated. This visualisation is based on the MMRE value, which refers to “X”. The prediction model, which has the lowest MMRE value, the narrowest box and the smallest range is considered preferable. In Figure 5.6, there are clear cases of decrease in the MMRE values indicated by “X” in the diagram, and reduction in the boxes spread, which are considered to have high prediction accuracy. These cases are summarised across all datasets in general as follows:

- SVR as an individual model or a base model in bagging and additive regression outperformed all other prediction models, followed by KNN. It seems that SVR and KNN performed very similarly because they had smaller box and lower IQR;
- Bagging ensemble models substantially increased the accuracy over RT and MLP. However, these models slightly improved the accuracy of M5Rules and had a minor or negative impact on KNN and SVR;
- Applying bagging ensemble models to the base models positively influenced the overall prediction accuracy compared to additive regression ensemble models;
- The prediction accuracy of APE ensemble models was better than that of stacking ensemble models.

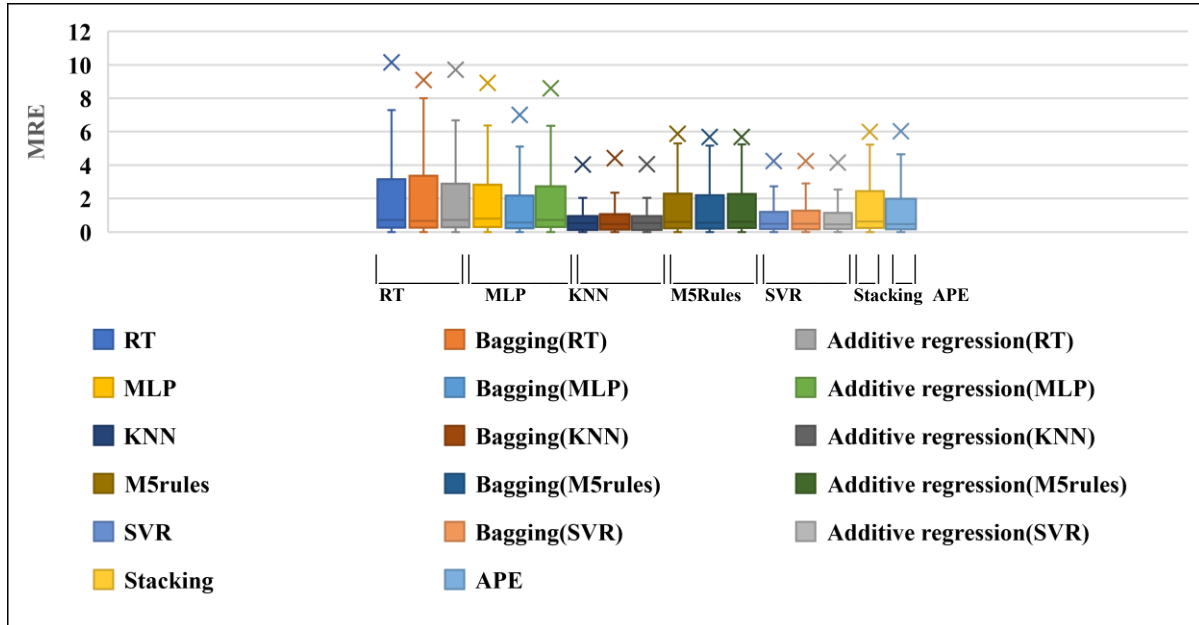


Figure 5.6.A: Boxplot of the MRE for prediction models on the Eclipse JDT Core dataset.

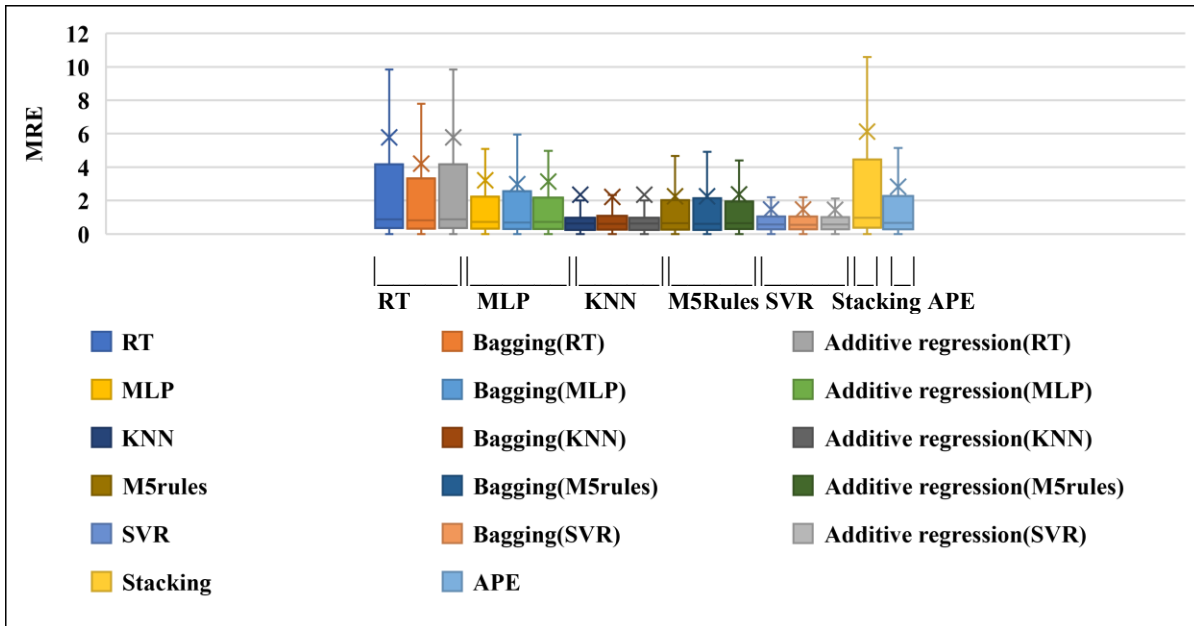


Figure 5.6.B: Boxplot of MRE for prediction models on the Eclipse PDE UI dataset.

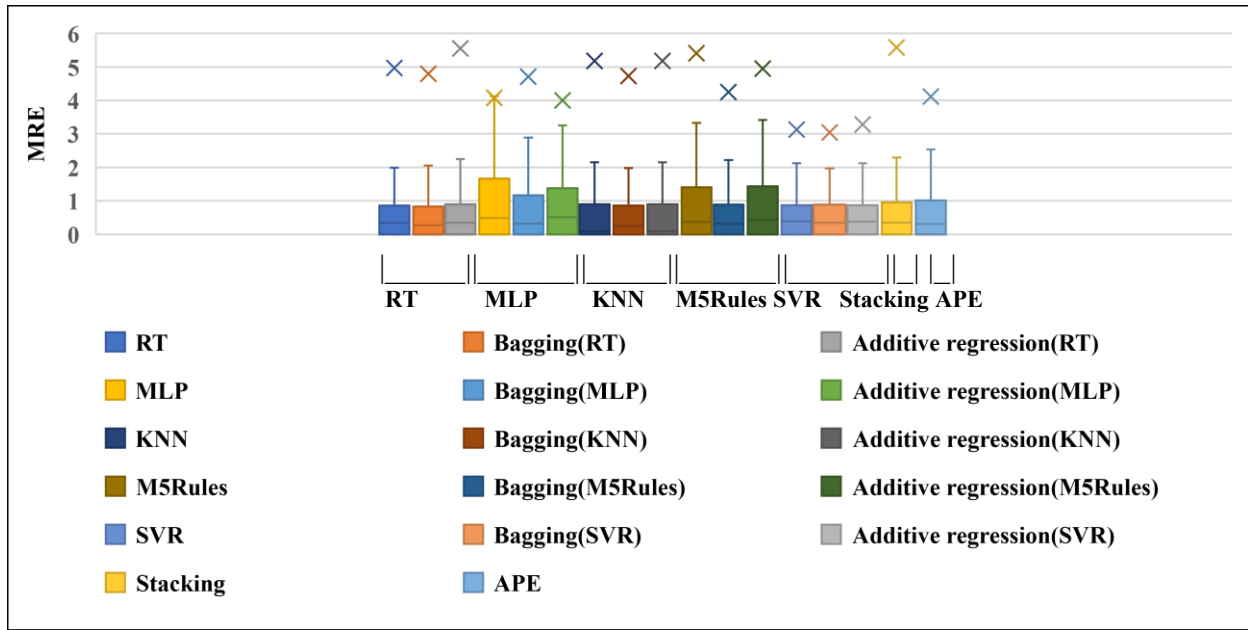


Figure 5.6.C: Boxplot of MRE for prediction models on the Equinox Framework dataset.

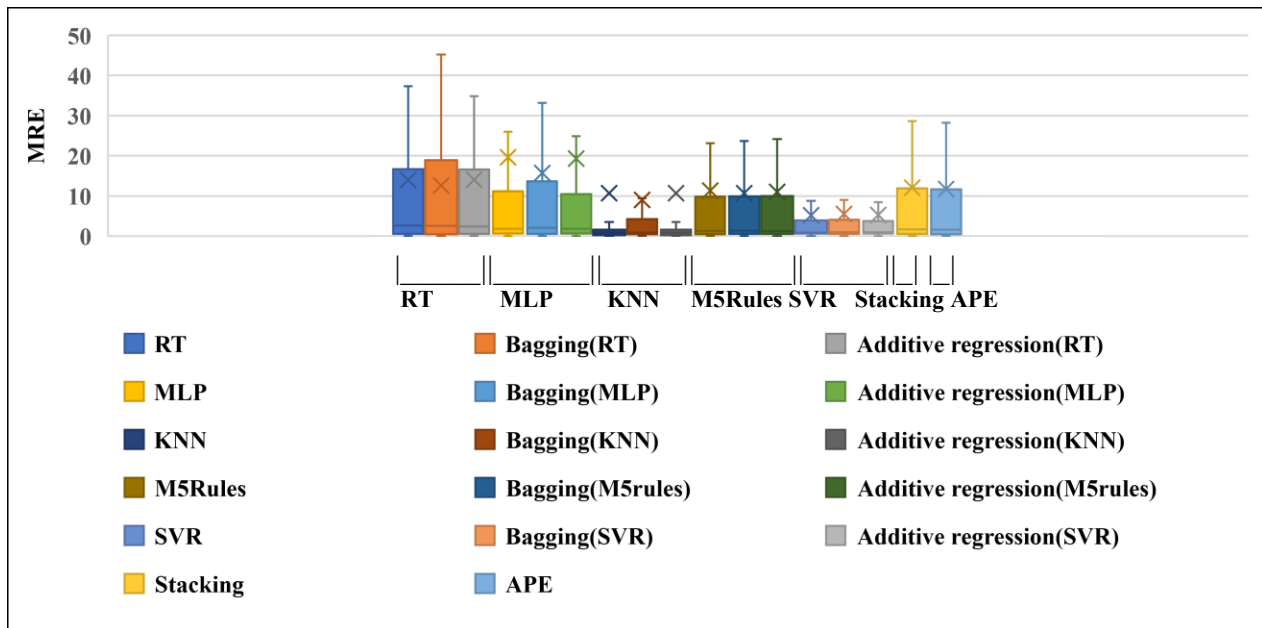


Figure 5.6.D: Boxplot of MRE for prediction models on the Lucene dataset.

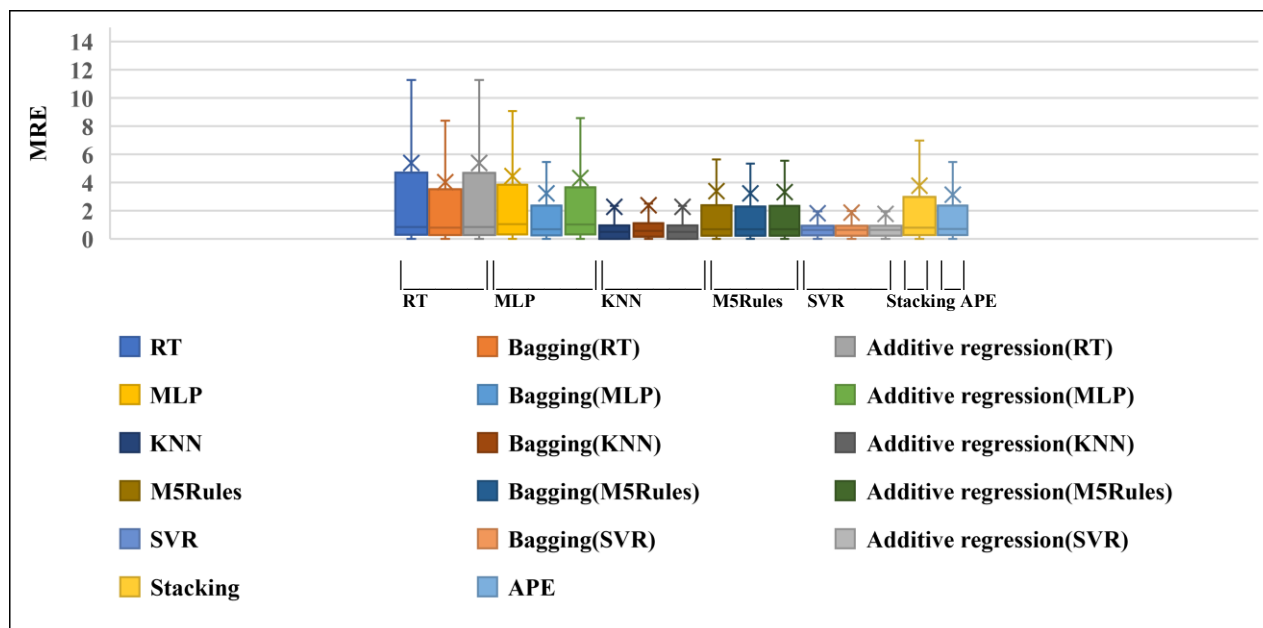


Figure 5.6.E: Boxplot of MRE for prediction models on the Mylyn dataset.

Figure 5.6: Boxplot of MRE for prediction models on all datasets.

Figure 5.7 compares the results obtained from the boxplot of the residuals for the prediction models across all datasets. The MAE value is defined by an ‘X’; a lower score, small whiskers, and narrow box refer to better performance. The following are the most evident findings from this figure:

- The prediction models performed better than the ZeroR model in almost all situations;
- The Mylyn dataset achieved the highest total prediction accuracy compared with other datasets; this result may be explained by the fact that this dataset was extracted from a large system that includes 2,196 classes;
- The ensemble models yield enhanced prediction accuracy over most of the investigated individual models. However, the statistical tests will be further investigated, as the difference between the ensemble and individual models may not be significant;
- SVR as the individual model or the base model in bagging and additive regression outperformed other prediction models and recorded the highest prediction accuracy across all datasets.

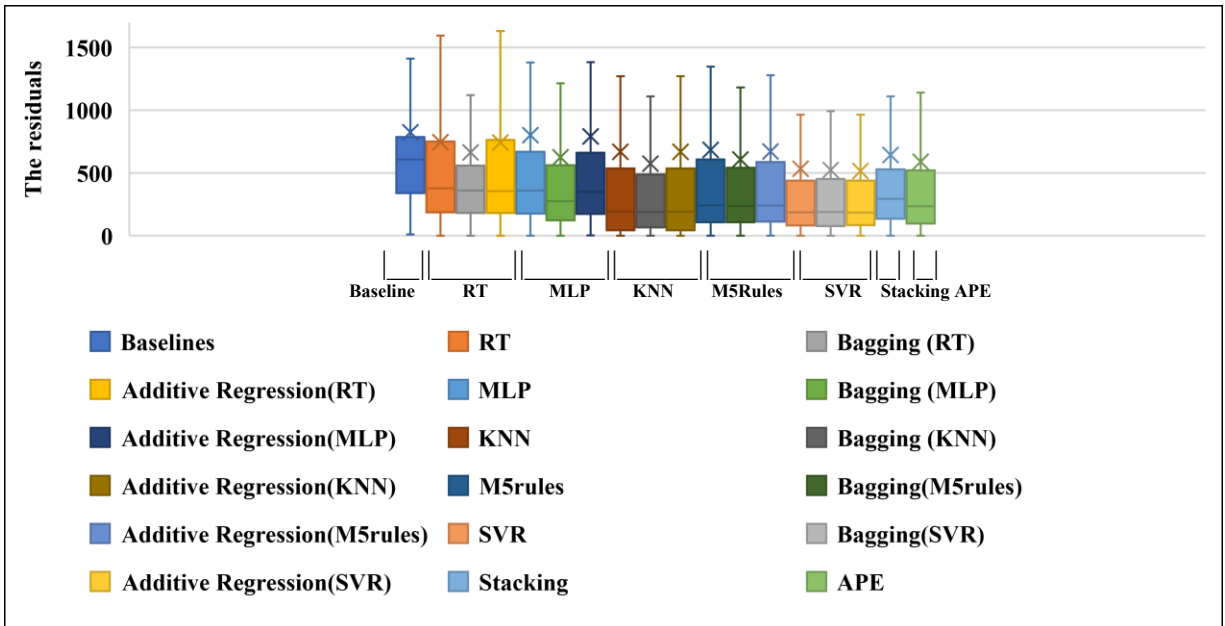


Figure 5.7.A: Box plot of the residuals for prediction models on the Eclipse JDT Core dataset.

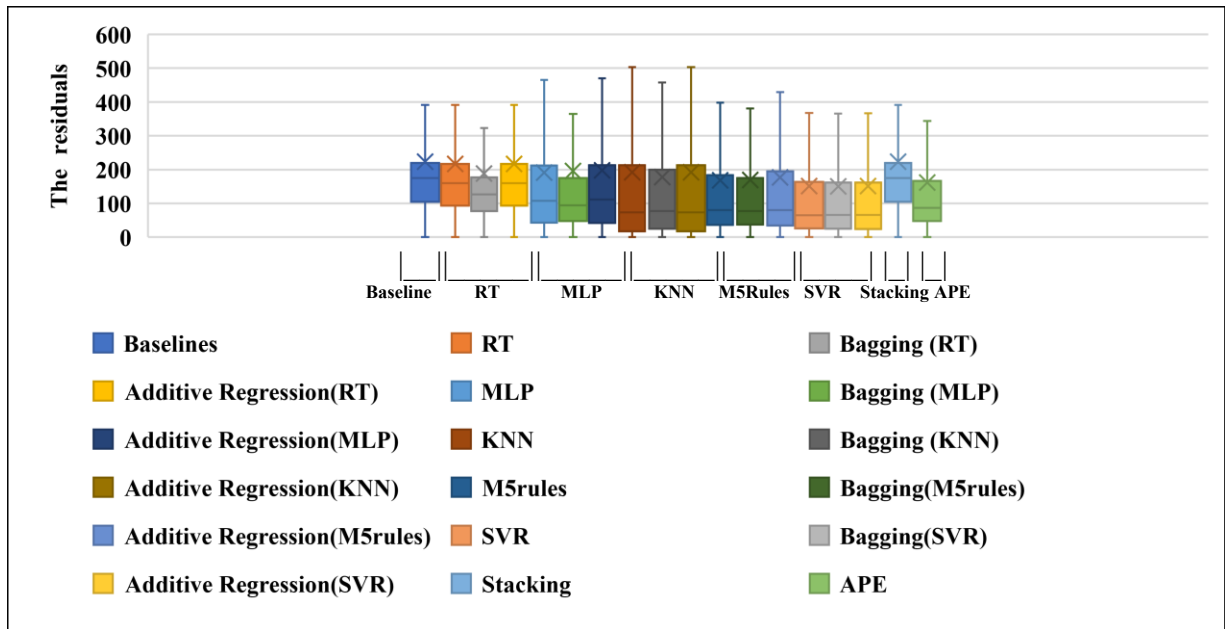


Figure 5.7.B: Box plot of the residuals for prediction models on the Eclipse PDE UI dataset.

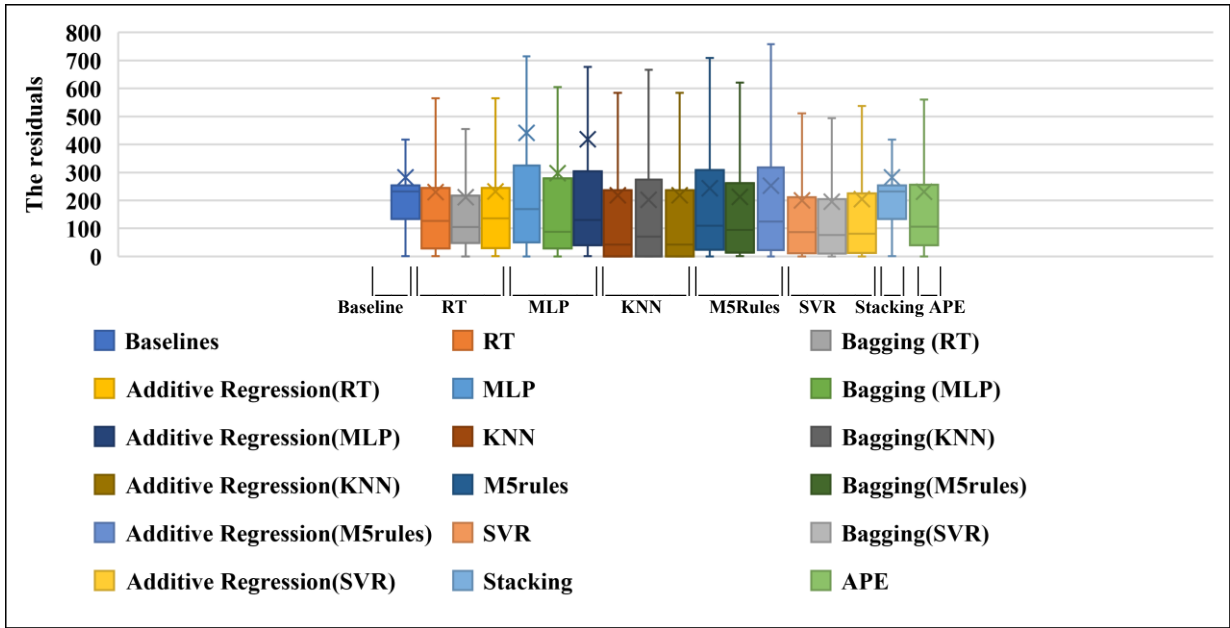


Figure 5.7.C: Box plot of the residuals for prediction models on the Equinox Framework dataset.

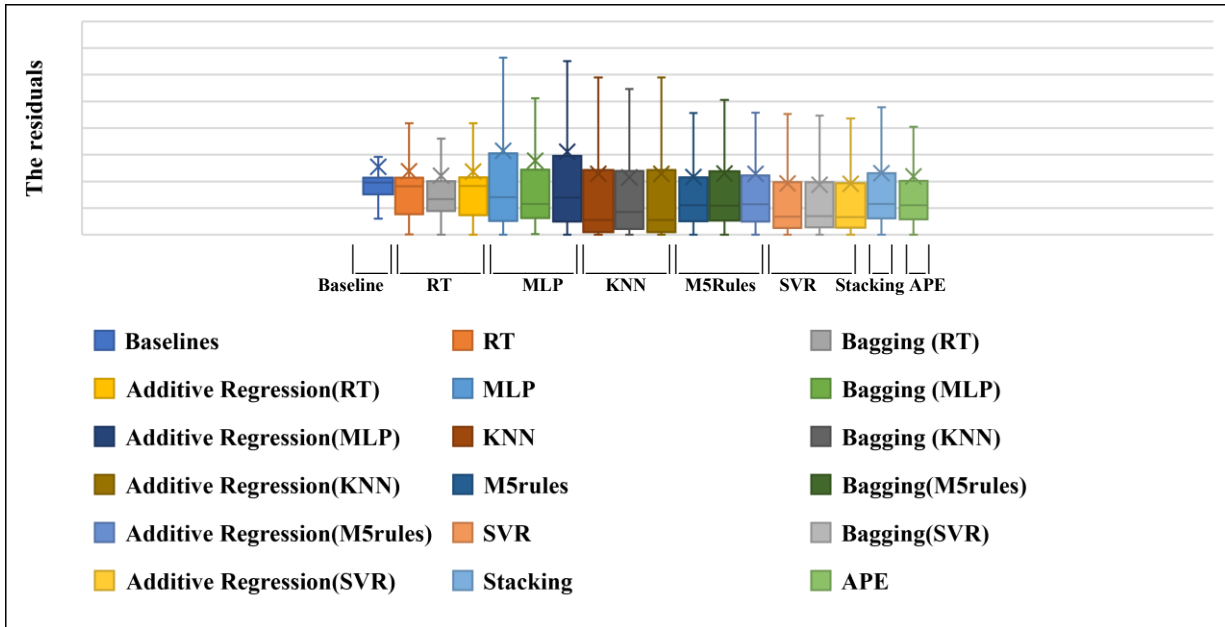


Figure 5.7.D: Box plot of the residuals for prediction models on the Lucene dataset.

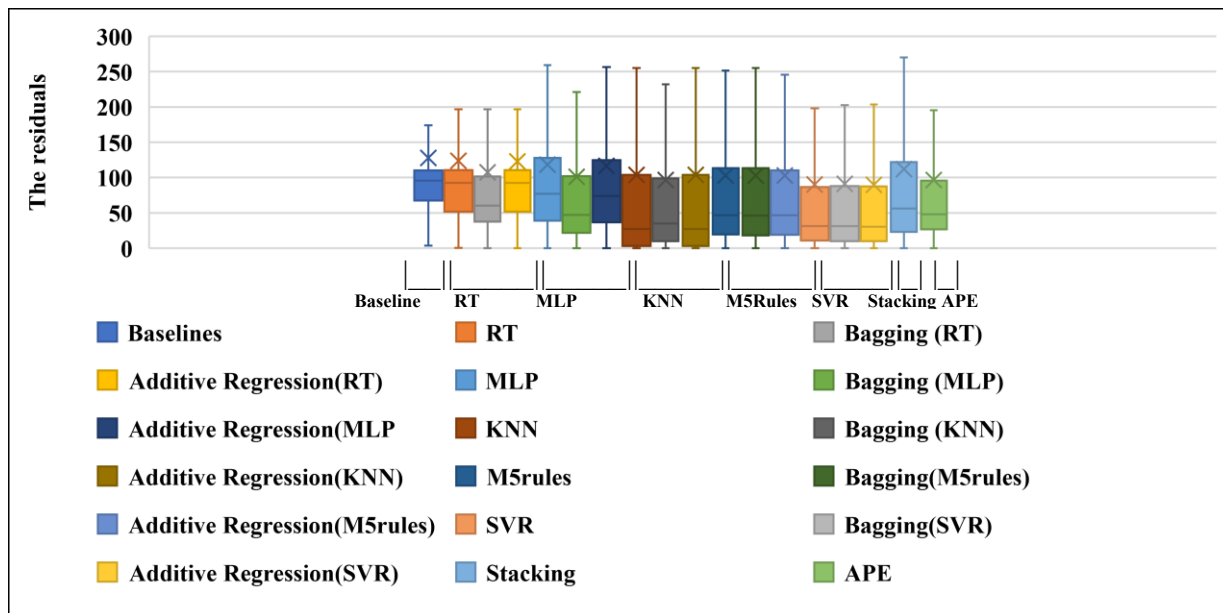


Figure 5.7.E: Box plot of the residuals for prediction models on the Mylyn dataset.

Figure 5.7: Boxplot of the residuals for prediction models on all datasets.

Figure 5.8 shows a histogram of the Pred (.25) values to compare the accuracy prediction between models. The higher the value in this diagram, the better the performance achieved by the prediction model, and the model accuracy criteria mentioned earlier are $\text{Pred}(.30) \geq 0.70$ or $\text{Pred}(.25) \geq 0.75$ [34]. From Figure 5.8, the following observations are made:

- KNN as an individual model or a base model in bagging and additive regression outperformed all other prediction models across the datasets. The potential reason why KNN produced better performance than SVM in term of Pred is that KNN predicts by selecting the closest neighbours of the instances in the training set, whereas the calculation of Pred is based on the proportion of all the instances in the dataset where the MRE is less than or equal to 25 or 30. Consequently, it is easy to determine these instances from the prediction of KNN. However, this difference may not be significant;
- APE achieved the best prediction accuracy with KNN in the Eclipse JDT Core dataset, the second best in Eclipse PDE UI and Mylyn datasets, and the third best in the remaining datasets.

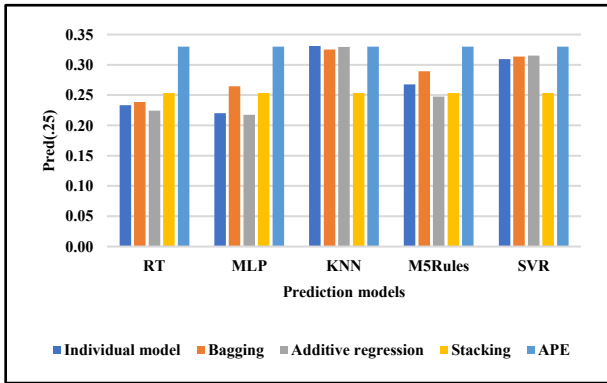


Figure 5.8.A: Pred(.25) for each prediction model on the Eclipse JDT Core dataset.

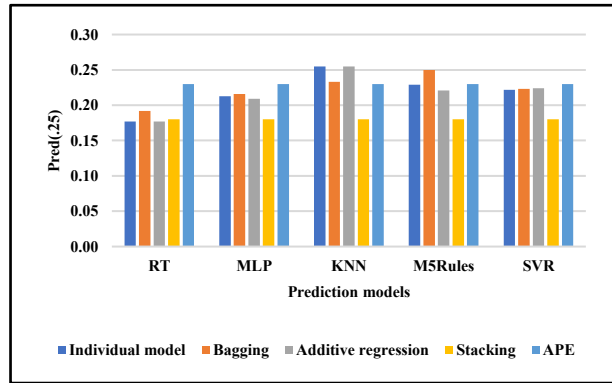


Figure 5.8.B: Pred(.25) for each prediction model on the Eclipse PDE UI dataset.

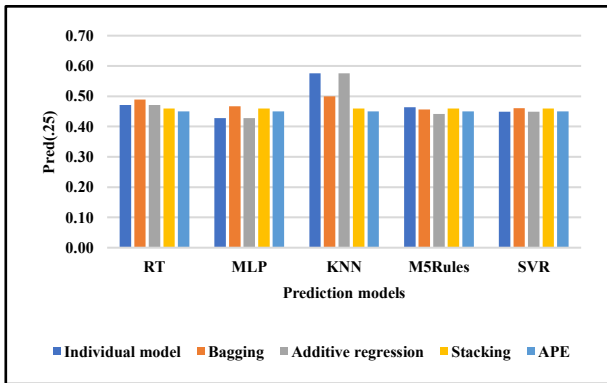


Figure 5.8.C: Pred(.25) for each prediction model on the Equinox Framework dataset.

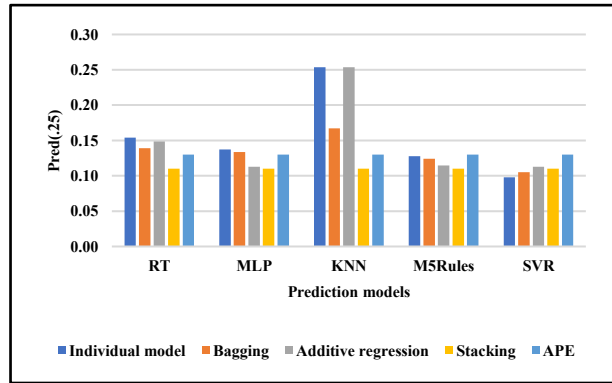


Figure 5.8.D: Pred(.25) for each prediction model on the Lucene dataset.

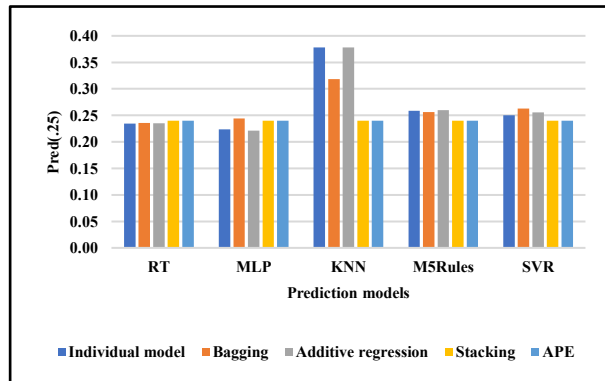


Figure 5.8.E: Pred(.25) for each prediction model on the Mylyn dataset.

Figure 5.8: Pred(.25) for each prediction model on all datasets.

Figure 5.9 presents a histogram of the Pred (.30) values to compare the accuracy prediction between models. The higher the value in this diagram, the better the accuracy achieved by the prediction model. From Figure 5.9, the following observations are made:

- APE and KNN as a base model in bagging achieved the best prediction accuracy compared to all prediction models in Eclipse JDT Core, followed by KNN as an individual model or base model in the additive regression ensemble model;
- KNN as an individual model or a base model in bagging and additive regression outperformed all other prediction models in the accuracy prediction of the remaining datasets;
- M5Rules as a base model in bagging and SVR as a base model in additive regression produced the highest prediction accuracy in the Eclipse PDE UI dataset, which is the same result as KNN.

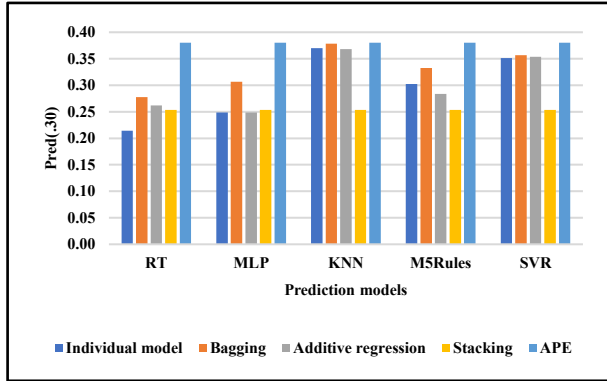


Figure 5.9.A: Pred(.30) for each prediction model on the Eclipse JDTCore dataset.

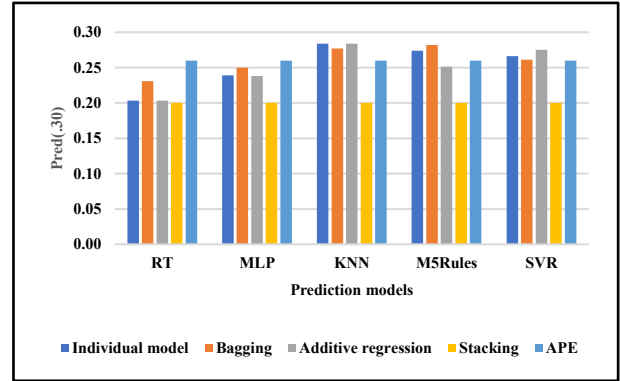


Figure 5.9.B: Pred(.30) for each prediction model on the Eclipse PDE UI dataset.

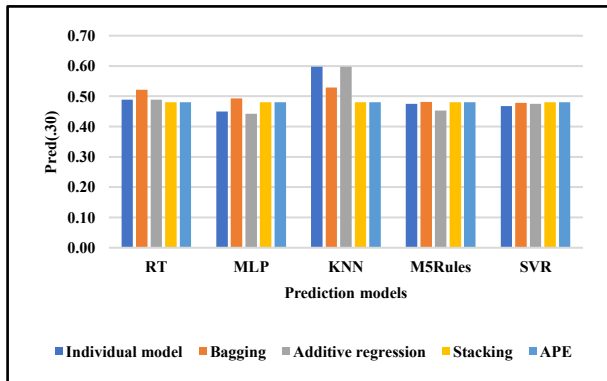


Figure 5.9.C: Pred(.30) for each prediction model on Equinox Framework dataset.

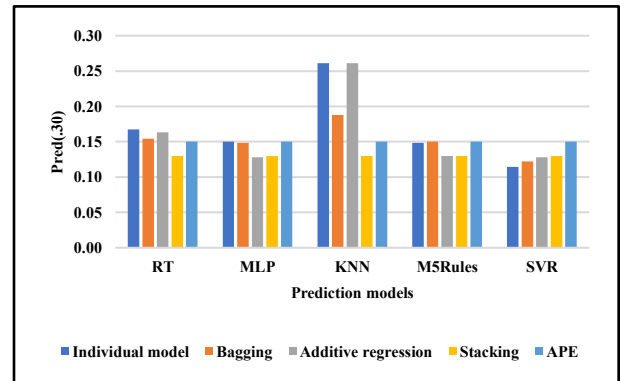


Figure 5.9.D: Pred(.30) for each prediction model on the Lucene dataset.

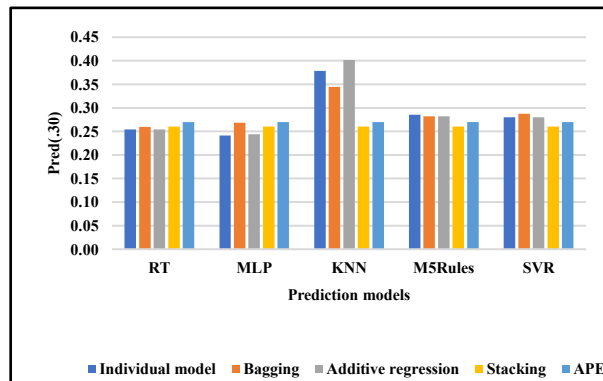


Figure 5.9.E: Pred(.30) for each prediction model on the Mylyn dataset.

Figure 5.9: Pred(.30) for each prediction model on all datasets.

Table 5.5, Table 5.6, Table 5.7 and Table 5.8 present the MMRE, MAE and SA values achieved by each prediction models for the software maintainability prediction datasets. Bold values (highlighted in light green) in the tables show the best results among each experiment in each dataset, whereas bold together with underlined values (highlighted in dark green) indicate the best results among each experiment in all datasets. The lowest MMRE and MAE and highest SA refer to the best results depending on the measure.

A. Experiment 1: comparison between prediction models and baseline, RQ5.A.4

ZeroR depends on the dependent variable only (i.e., CHANGE metric) and predicts the mean value of this metric. It is implemented to determine a baseline and is used as a benchmark to evaluate the performance of the prediction models. When we compare the results of ZeroR in Table 5.5 against the results of the prediction models in Table 5.6, Table 5.7 and Table 5.8, all machine learning models performed better than the ZeroR model, except in the case of MLP as the individual model and as the base model in additive regression in the Mylyn dataset. Moreover, other models (i.e., stacking in Eclipse PDE UI and Equinox Framework datasets) produce the same performance as the ZeroR model.

Table 5.5: Baseline models and their corresponding MMRE, MAE and SA values.

Model	Eclipse JDT Core	Eclipse PDE UI	Equinox Framework	Lucene	Mylyn
ZeroR	MMRE				
	13.79	6.12	5.58	15.18	5.92
	MAE				
	823.90	223.46	282.04	127.15	127.67
	SA				
	0	0	0	0	0

Dark green: represents the best results in all experiments.

B. Experiment 2: comparison between individual models, RQ5.A.5

As shown in Table 5.6, the results of the five individual models indicate that SVR outperformed the individual models in the prediction accuracy for all datasets except the Eclipse JDT Core dataset, which achieved the best result using KNN (indicated by the bold values), followed by SVR. Furthermore, the performance of SVR in the Eclipse PDE UI dataset is better than all prediction models across the software maintainability datasets in Experiment 2 (i.e., bold and underlined values), with a value of 1.47. However, MLP attained negative values in SA measurement in the Equinox Framework and Lucene datasets, indicating that their results were worse than the baseline and did not produce meaningful predictions in this empirical study [99]. However, the MLP performed better than the baseline in the remaining datasets.

Table 5.6: Individual models and their corresponding MMRE, MAE and SA values.

Individual models	MMRE				
	Eclipse JDT Core	Eclipse PDE UI	Equinox Framework	Lucene	Mylyn
RT	10.15	5.79	4.96	13.96	5.40
MLP	8.93	3.22	4.08	19.59	4.45
KNN	4.04	2.35	5.18	10.71	2.28
M5Rules	5.88	2.26	5.41	11.30	3.39
SVR	4.24	1.47	3.13	5.19	1.83
	MAE				
RT	746.59	216.23	230.00	119.08	123.95
MLP	801.51	191.24	441.03	157.47	118.49
KNN	678.86	197.03	238.36	114.69	111.61
M5Rules	683.62	168.44	243.79	108.30	102.94
SVR	533.78	151.41	200.33	96.17	90.30
	SA				
RT	9.38	3.24	18.45	6.35	2.91
MLP	2.72	14.42	-56.37	-23.84	7.19
KNN	18.88	14.14	22.52	10.26	18.56
M5Rules	17.03	24.62	13.56	14.83	19.37
SVR	35.21	32.24	28.97	24.36	29.27

Dark green: represents the best results in all experiments.
Light green: represents the best results for each experiment.

C. Experiment 3: comparison between homogeneous ensemble models, RQ5.A.6

- **Bagging**

After applying homogeneous ensemble models (i.e., bagging) on each individual model, as shown in Table 5.7, the bagging ensemble model increased the prediction accuracy over most of the individual models. SVR as a base model in the bagging ensemble produced the highest accuracy in all software maintainability datasets (i.e., bold values), and achieved the best result in the Eclipse PDE UI dataset, reaching 1.46 (i.e., bold values underlined). However, the positive influence of the bagging ensemble on SVR is considered low, as the change is only 0.01 (0.68%) in the Eclipse PDE UI dataset and 0.08 (2.62%) in the Equinox Framework, and in the remaining datasets the performance of SVR decreased. However, the bagging ensemble model increased the prediction accuracy of RT, M5Rules, and MLP more than the KNN and SVR models. Overall, the bagging ensemble model increased the performance of all the individual models except SVR and KNN in the Eclipse JDT Core and Mylyn datasets, and SVR as the base model in bagging achieved the best results.

- **Additive regression**

Table 5.7 shows the results after employing homogeneous ensemble models (i.e., additive regression) on each individual model. It is apparent from these results that the additive regression ensemble model improved the prediction accuracy over most of the individual models. SVR as a base model in the additive regression ensemble achieved the best accuracy prediction in all software maintainability datasets except for the Eclipse JDT Core dataset,

where the best result was achieved by KNN (as indicated by the bold values). Furthermore, SVR in the Eclipse PDE UI dataset outperformed all other models across the software maintainability datasets in Experiment 3 (bold values with underline), reaching 1.46. This result is slightly better compared to that of the SVR as the individual model, which was 1.47. Interestingly, additive regression had a positive influence on MLP across all datasets, whereas it had a negative influence on RT and SVR in the Equinox Framework and Lucene datasets. Nevertheless, additive regression had no impact on the performance of KNN across all datasets and performed the same result as the KNN individual models. Finally, the additive regression ensemble models increased the accuracy prediction over most individual models, or produced results similar to the individual models, as is the case of KNN. Regarding SA measurement, all homogeneous ensemble models outperformed the baseline except MLP in the Equinox Framework and Lucene datasets, which recorded negative values. Similarly, MLP as the individual model achieved similar values in these datasets.

Table 5.7: Homogeneous ensemble models and their corresponding MMRE, MAE and SA values.

<i>Bagging Models</i>	MMRE				
	Eclipse JDT Core	Eclipse PDE UI	Equinox Framework	Lucene	Mylyn
RT	9.08	4.20	4.79	12.56	4.01
MLP	7.00	2.98	4.71	15.69	3.23
KNN	4.43	2.21	4.73	8.99	2.39
M5Rules	5.69	2.26	4.25	10.67	3.22
SVR	4.25	1.46	3.05	5.53	1.86
	MAE				
RT	662.77	188.29	211.23	110.58	107.26
MLP	624.67	195.80	297.15	139.09	100.89
KNN	573.45	177.46	204.27	107.70	96.72
M5Rules	608.98	169.25	212.29	115.03	102.71
SVR	522.94	150.35	195.08	93.86	90.93
	SA				
RT	19.56	15.74	25.11	13.04	15.99
MLP	24.18	12.38	-5.36	-9.39	20.98
KNN	30.40	20.62	28.43	15.30	24.37
M5Rules	26.08	24.26	24.73	9.54	19.55
SVR	36.53	32.72	30.83	26.19	28.77
<i>Additive regression models</i>	MMRE				
RT	9.70	5.79	5.55	14.00	5.39
MLP	8.60	3.14	4.00	19.28	4.32
KNN	4.06	2.35	5.18	10.71	2.28
M5Rules	5.69	2.37	4.95	11.02	3.32
SVR	4.15	1.46	3.28	5.23	1.78
	MAE				
RT	743.77	216.23	232.12	118.62	122.97
MLP	790.43	197.40	418.33	155.71	116.67
KNN	678.86	197.03	238.36	114.69	111.61
M5Rules	672.06	177.20	252.74	114.10	102.77
SVR	515.58	151.18	204.67	95.68	89.67
	SA				
RT	9.72	3.24	17.70	6.71	3.68
MLP	4.06	11.66	-48.32	-22.46	8.61
KNN	18.88	14.14	22.52	10.26	18.56
M5Rules	18.43	20.70	10.39	10.27	19.50
SVR	37.42	32.35	27.43	24.76	29.77

Dark green: represents the best results in all experiments.
Light green: represents the best results for each experiment.

D. Experiment 4: comparison between heterogeneous ensemble models, RQ5.A.7

- Stacking

In Table 5.8, the heterogeneous ensemble model (i.e., stacking) combined five individual models (i.e., RT, MLP, KNN, M5Rules and SVR) and used linear regression as a metamodel to integrate their outputs. The results reveal that the stacking performance did not improve compared to that of the five individual models. Stacking increased the performance of RT and MLP in three datasets (i.e., Eclipse JDT Core, Lucene and Mylyn), and showed a slight decrease in performance in the remaining individual models across datasets. Stacking achieved the highest prediction accuracy in Mylyn, followed by the Equinox Framework dataset, achieving 3.78 and 5.58, respectively. Additionally, stacking performed well in terms of SA

measurement; however, it recorded zero values in the Eclipse PDE UI and Equinox Framework datasets, which indicates no improvement over the baseline in these datasets.

- **APE**

Another heterogeneous ensemble model shown in Table 5.8, namely APE, was performed by considering the average of the five individual models (i.e., RT, MLP, KNN, M5Rules and SVR), thus, it provides a single output value [140]. The findings demonstrate that the APE advanced the prediction accuracy of RT and MLP in all datasets except the Equinox Framework. It also improved the prediction accuracy of M5Rules in three datasets (i.e., Equinox Framework, Lucene and Mylyn). However, APE achieved a lower prediction accuracy than SVR in all datasets as well as a lower prediction accuracy than KNN in all datasets except the Equinox Framework. The finding of APE confirms previous finding of stacking that produced a lower performance because it integrated the good and bad performance of the individual models. APE achieved the highest prediction accuracy in the Eclipse PDE UI, followed by the Mylyn dataset, reaching 2.83 and 3.14, respectively.

Finally, the most evident finding from previous tables is that bagging achieved better performance than additive regression, whereas APE achieved better performance than stacking. Furthermore, the prediction models, which were performed on large datasets (i.e., Eclipse PDE UI and Mylyn), achieved better prediction accuracy than the same prediction models performed on small datasets, ranging from 276 to 695. This suggests that such prediction models are robust for large datasets. Furthermore, in most cases, SVR as an individual model or a base model in bagging and additive regression is the best choice to predict software maintainability.

Table 5.8: Heterogeneous ensemble models and their corresponding MMRE, MAE and SA values.

	Eclipse JDT Core	Eclipse PDE UI	Equinox Framework	Lucene	Mylyn
<i>Stacking models</i> (RT, MLP, KNN, M5Rules, SVR)	MMRE				
	5.99	6.12	5.58	12.02	3.78
	MAE				
	644.21	223.46	282.04	115.51	111.81
	SA				
	21.81	0	0	9.16	12.42
<i>APE models</i> (RT, MLP, KNN, M5Rules, SVR)	MMRE				
	6.03	2.83	4.11	11.7	3.14
	MAE				
	590.91	161.82	231.21	109.48	96.60
	SA				
	28.28	27.59	19.79	13.90	24.33
<p style="margin: 0;">Dark green: represents the best results in all experiments. Light green: represents the best results for each experiment.</p>					

One-way ANOVA results for different datasets using the residual values for RT, MLP, KNN, M5Rules, and SVR and ensemble models are listed as follows: Table 5.9 to Table 5.13 for Eclipse JDT Core dataset; Table 5.14 to Table 5.18 for Eclipse PDE UI dataset; Table 5.19 to Table 5.23 for Equinox Framework dataset; Table 5.24 to Table 5.28 for Lucene dataset; Table 5.29 to Table 5.33 for Mylyn dataset.

The results obtained from these tables indicate that, in most cases, the performance of the prediction models was not significantly different in terms of the residual values from each other for Factor A because their p-values are larger than the significance level ($\alpha = 0.05$), so H_0 is accepted. However, the p-values in the Eclipse PDE UI dataset (see tables from Table 5.14 to Table 5.18) and other tables (Table 5.20, Table 5.25,

Table 5.29 and Table 5.30) are lower than the significance level. Therefore, the performance of the prediction models in these tables was significantly different in terms of the residual values from each other for Factor A, and H_0 is rejected and H_1 is accepted, meaning that for Factor A, at least two group means significantly differ from each other. According to the standard classifications of Cohen proposed in Section 3.5.5, the results of eta-squared indicate that the effect sizes were small in all tables [180].

Table 5.9: One-way ANOVA for RT and ensemble models in the Eclipse JDT Core dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	12501422.11	4.00	3125355.53	0.92	0.45	0.00
Error	11774121071.13	3470.00	3393118.46			
Total	11786622493.24	3474.00				

Table 5.10: One-way ANOVA for MLP and ensemble models in the Eclipse JDT Core dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	26898330.99	4.00	6724582.75	1.02	0.39	0.00
Error	22834275291.33	3470.00	6580482.79			
Total	22861173622.32	3474.00				

Table 5.11: One-way ANOVA for KNN and ensemble models in the Eclipse JDT Core dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	5467853.84	4.00	1366963.46	0.40	0.81	0.00
Error	11712791648.27	3470.00	3375444.28			
Total	11718259502.10	3474.00				

Table 5.12: One-way ANOVA for M5Rules and ensemble models in the Eclipse JDT Core dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	4391944.36	4.00	1097986.09	0.33	0.86	0.00
Error	11579332739.53	3470.00	3336983.50			
Total	11583724683.88	3474.00				

Table 5.13: One-way ANOVA for SVR and ensemble models in the Eclipse JDT Core dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	8388497.23	4.00	2097124.31	0.82	0.51	0.00
Error	8921762704.87	3470.00	2571113.17			
Total	8930151202.10	3474.00				

Table 5.14: One-way ANOVA for RT and ensemble models in the Eclipse PDE UI dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	3221762.01	4.00	805440.50	6.88	0.00	0.00
Error	706719092.59	6040.00	117006.47			
Total	709940854.59	6044.00				

Table 5.15: One-way ANOVA for MLP and ensemble models in the Eclipse PDE UI dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2328640.49	4.00	582160.12	3.57	0.01	0.00
Error	984602758.75	6040.00	163013.70			
Total	986931399.24	6044.00				

Table 5.16: One-way ANOVA for KNN and ensemble models in the Eclipse PDE UI dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2511471.04	4.00	627867.76	5.18	0.00	0.00
Error	732218899.51	6040.00	121228.29			
Total	734730370.55	6044.00				

Table 5.17: One-way ANOVA for M5Rules and ensemble models in the Eclipse PDE UI dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	2994170.89	4.00	748542.72	6.10	0.00	0.00
Error	741644965.84	6040.00	122788.90			
Total	744639136.73	6044.00				

Table 5.18: One-way ANOVA for SVR and ensemble models in the Eclipse PDE UI dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	4815664.03	4.00	1203916.01	11.49	0.00	0.01
Error	632989845.26	6040.00	104799.64			
Total	637805509.29	6044.00				

Table 5.19: One-way ANOVA for RT and ensemble models in the Equinox Framework dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	772577.21	4.00	193144.30	1.08	0.36	0.00
Error	245045622.90	1375.00	178215.00			
Total	245818200.11	1379.00				

Table 5.20: One-way ANOVA for MLP and ensemble models in the Equinox Framework dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	9160880.15	4.00	2290220.04	2.94	0.02	0.01
Error	1072285162.54	1375.00	779843.75			
Total	1081446042.69	1379.00				

Table 5.21: One-way ANOVA for KNN and ensemble models in the Equinox Framework dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	1039249.78	4.00	259812.44	1.21	0.30	0.00
Error	294068457.62	1375.00	213867.97			
Total	295107707.40	1379.00				

Table 5.22: One-way ANOVA for M5Rules and ensemble models in the Equinox Framework dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	743122.31	4.00	185780.58	1.00	0.41	0.00
Error	255494191.82	1375.00	185813.96			
Total	256237314.13	1379.00				

Table 5.23: One-way ANOVA for SVR and ensemble models in the Equinox Framework dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	1430312.90	4.00	357578.22	1.94	0.10	0.01
Error	253852788.18	1375.00	184620.21			
Total	255283101.07	1379.00				

Table 5.24: One-way ANOVA for RT and ensemble models in the Lucene dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	42242.60	4.00	10560.65	0.26	0.90	0.00
Error	108220661.59	2655.00	40761.08			
Total	108262904.20	2659.00				

Table 5.25: One-way ANOVA for MLP and ensemble models in the Lucene dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	1053595.89	4.00	263398.97	2.47	0.04	0.00
Error	282988904.28	2655.00	106587.16			
Total	284042500.17	2659.00				

Table 5.26: One-way ANOVA for KNN and ensemble models in the Lucene dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	24425.70	4.00	6106.42	0.12	0.97	0.00
Error	131135243.95	2655.00	49391.81			
Total	131159669.65	2659.00				

Table 5.27: One-way ANOVA for M5Rules and ensemble models in the Lucene dataset using the residuals values.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	23797.49	4.00	5949.37	0.11	0.98	0.00
Error	137478923.08	2655.00	51781.14			
Total	137502720.57	2659.00				

Table 5.28: One-way ANOVA for SVR and ensemble models in the Lucene dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	201463.28	4.00	50365.82	1.25	0.29	0.00
Error	106913813.36	2655.00	40268.86			
Total	107115276.63	2659.00				

Table 5.29: One-way ANOVA for RT and ensemble models in the Mylyn dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	820433.52	4.00	205108.38	3.22	0.01	0.00
Error	500539699.16	7860.00	63681.90			
Total	501360132.68	7864.00				

Table 5.30: One-way ANOVA for MLP and ensemble models in the Mylyn dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	591892.35	4.00	147973.09	2.65	0.03	0.00
Error	438668440.81	7860.00	55810.23			
Total	439260333.16	7864.00				

Table 5.31: One-way ANOVA for KNN and ensemble models in the Mylyn dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	253506.93	4.00	63376.73	0.98	0.41	0.00
Error	506131215.02	7860.00	64393.28			
Total	506384721.95	7864.00				

Table 5.32: One-way ANOVA for M5Rules and ensemble models in the Mylyn dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	185713.32	4.00	46428.33	0.79	0.53	0.00
Error	459562662.82	7860.00	58468.53			
Total	459748376.14	7864.00				

Table 5.33: One-way ANOVA for SVR and ensemble models in the Mylyn dataset using the residuals.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-value	Eta-Squared
Factor A	548317.21	4.00	137079.30	2.35	0.05	0.00
Error	458132781.24	7860.00	58286.61			
Total	458681098.45	7864.00				

Multiple pairwise comparison tests for Factor A were also conducted to determine which pairs were significantly different from each other. This test was performed using Tukey's confidence intervals [170] for tables that rejected H₀ (from Table 5.14 to Table 5.18, along with Table 5.20, Table 5.25, Table 5.29 and Table 5.30). The comparison results are presented in Figure 5.10, which involves nine subfigures. Figure 5.10 reveals that if a confidence interval

does not include 0, then the pair is significantly different and H_0 is rejected. The following cases show the pairs that significantly differ from each other:

- (RT – APE) and (RT – APE) pairs were significantly different, as shown in Figure 5.10.A, whereas APE performed better than these models in the Eclipse PDE UI dataset;
- (Stacking – APE) pair was significantly different in the Eclipse PDE UI dataset (see Figures 5.10.A, 5.10.B, 5.10.C, 5.10.D and 5.10.E), whereas APE outperformed stacking in the Eclipse PDE UI dataset;
- (MLP – APE) pair was significantly different, as shown in Figure 5.10.F, whereas APE was better than MLP in the Equinox Framework dataset;
- Although the P-values in Table 5.20 and Table 5.30 were lower than the significance level ($\alpha = 0.05$), there were no significant differences between each pair (see Figure 5.10.G and Figure 5.10.I).
- (SVR – Stacking), (Additive regression (SVR) – Stacking), (Bagging (SVR) – Stacking) pairs were significantly different, as seen in Figure 5.10.E, in which stacking performed worse than these models in the Eclipse PDE UI dataset;
- (RT – APE) and (Additive regression (RT) – APE) pairs were significantly different, as seen in Figure 5.10.H, in which APE was better than RT and additive regression (RT) in the Mylyn dataset.

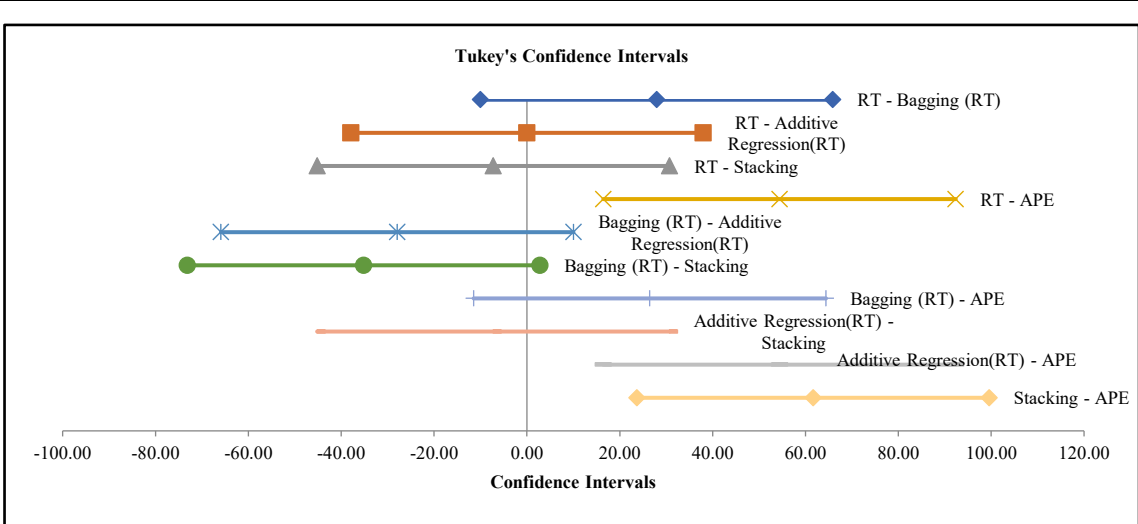


Figure 5.10.A: Multiple comparisons for RT and ensemble models in the Eclipse PDE UI dataset using the residuals.

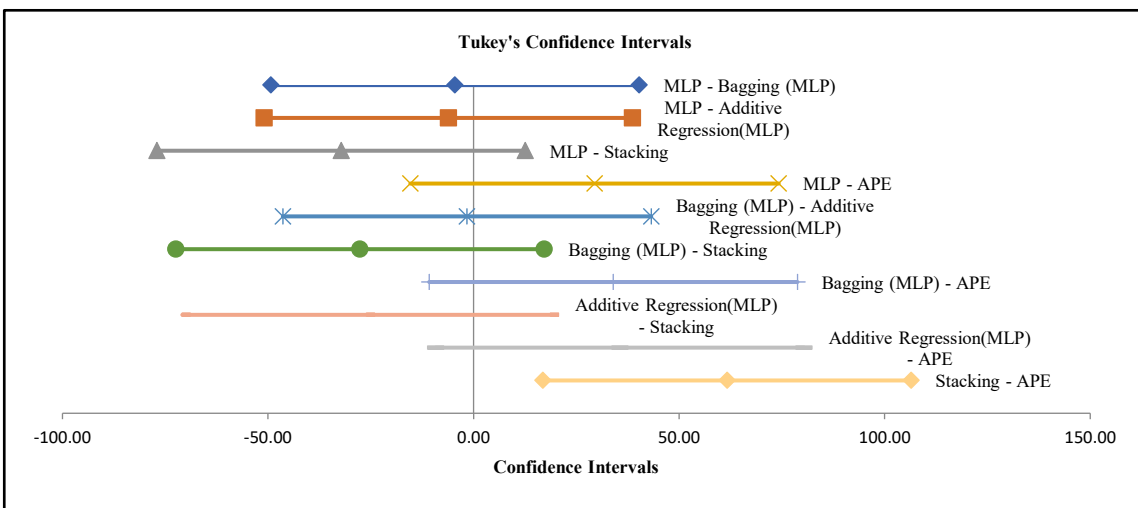


Figure 5.10.B: Multiple comparisons for MLP and ensemble models in the Eclipse PDE UI dataset using the residuals.

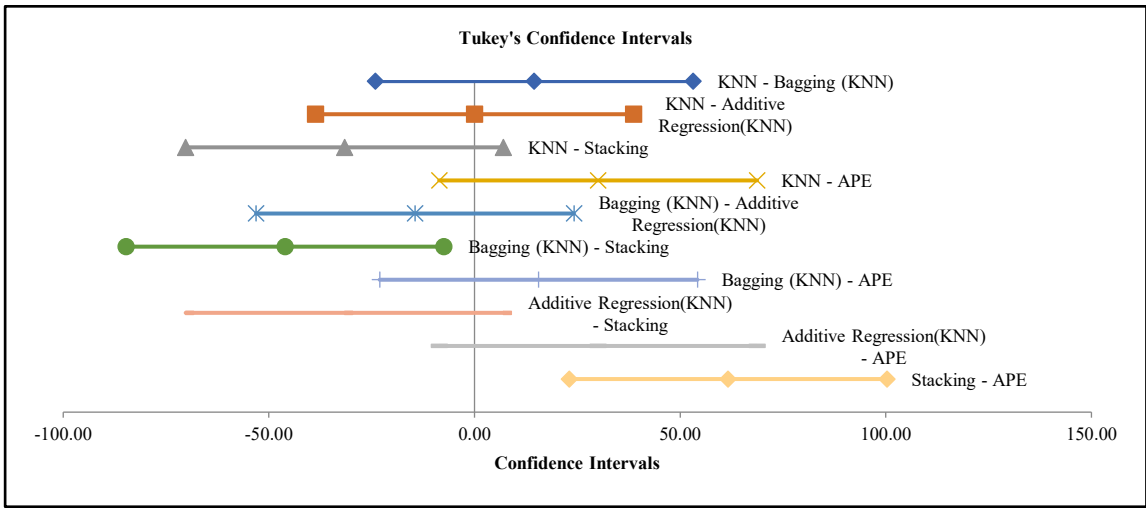


Figure 5.10.C: Multiple comparisons for KNN and ensemble models in the Eclipse PDE UI dataset using the residuals.

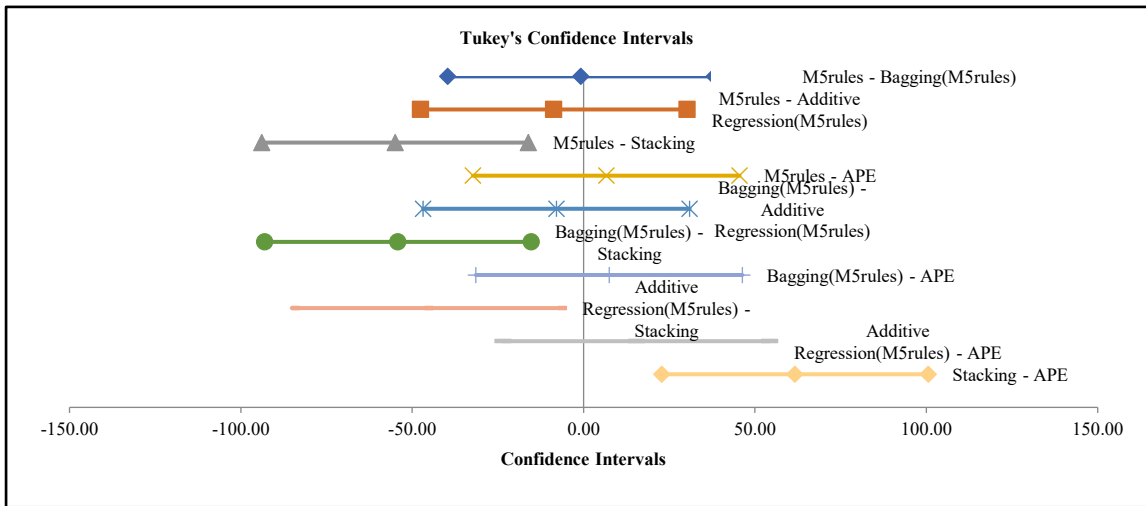


Figure 5.10.D: Multiple comparisons for M5Rules and ensemble models in the Eclipse PDE UI dataset using the residuals.

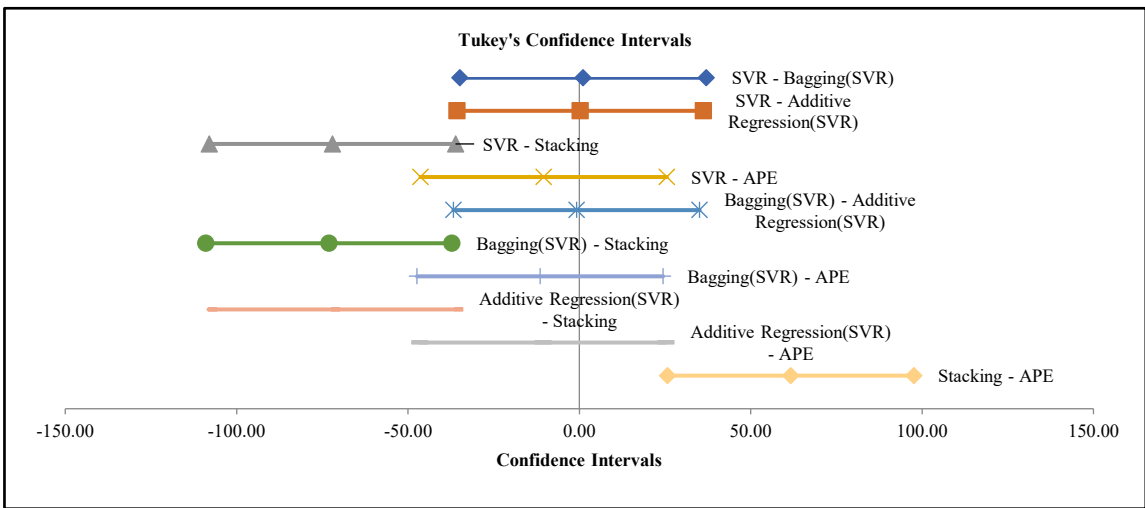


Figure 5.10.E: Multiple comparisons for SVR and ensemble models in the Eclipse PDE UI dataset using the residuals.

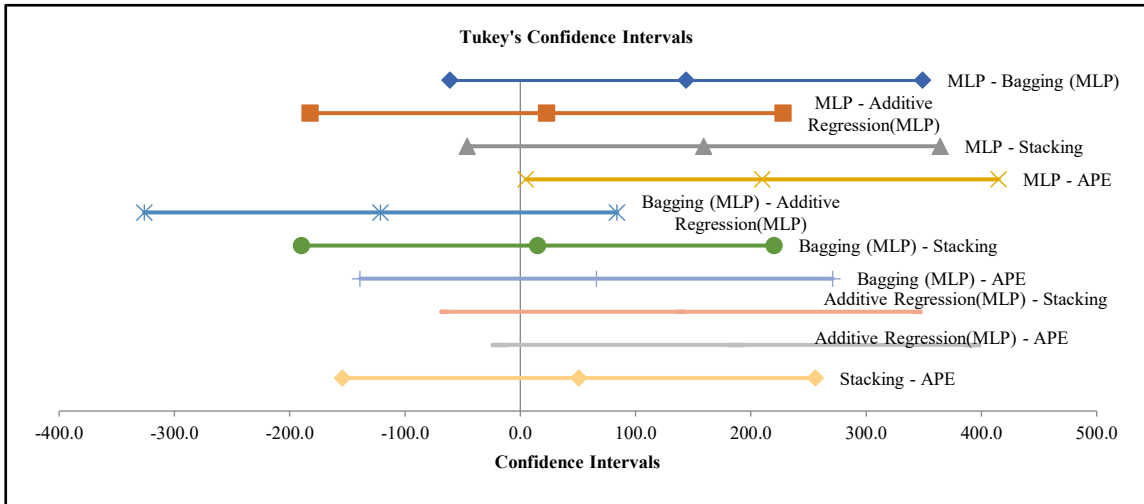


Figure 5.10.F: Multiple comparisons for MLP and ensemble models in the Equinox Framework dataset using the residuals.

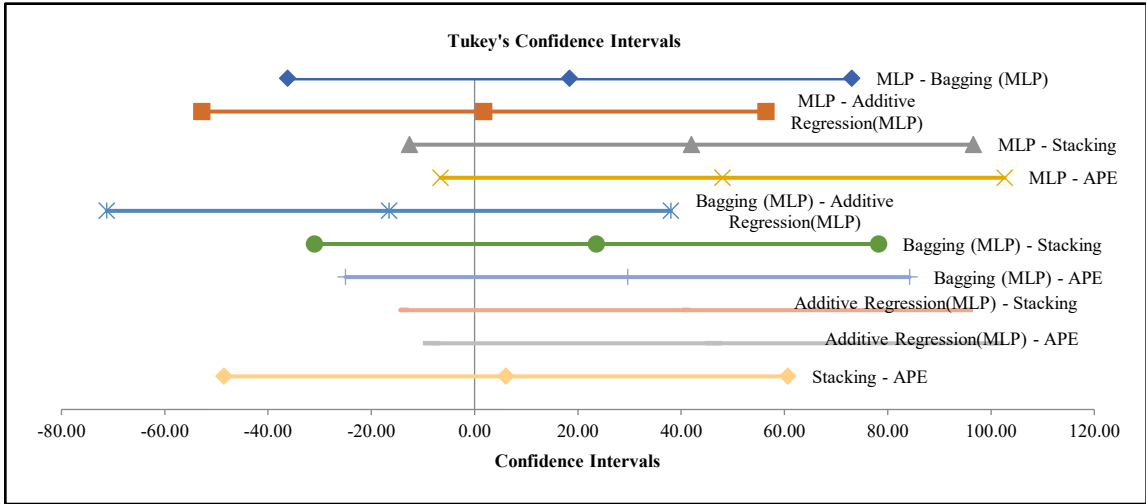


Figure 5.10.G: Multiple comparisons for MLP and ensemble models in the Lucene dataset using the residuals.

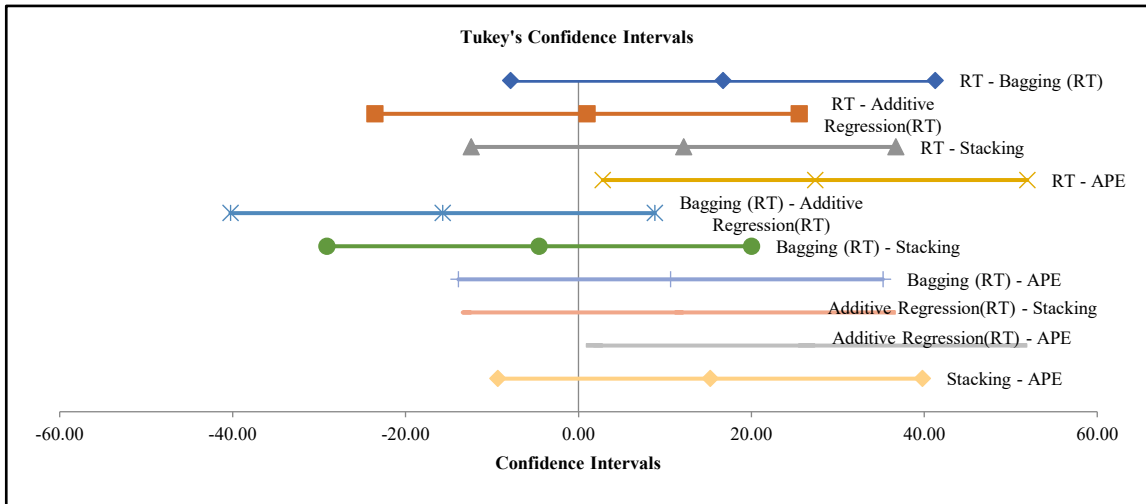


Figure 5.10.H: Multiple comparisons for RT and ensemble models in the Mylyn dataset using the residuals.

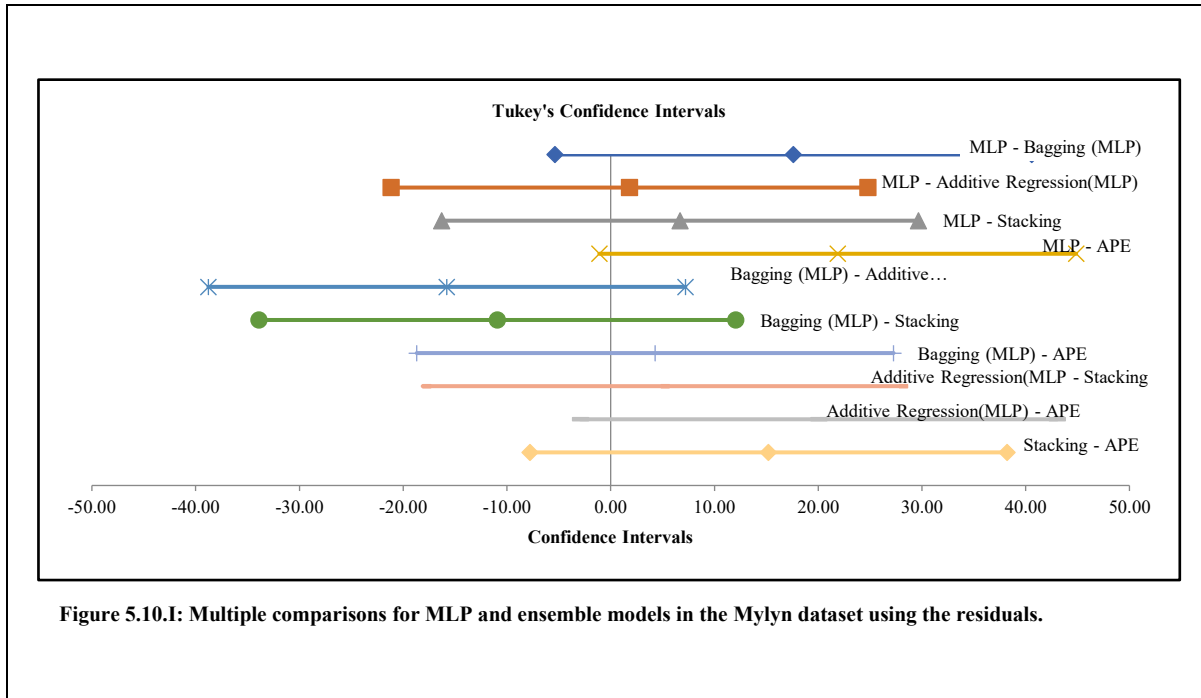


Figure 5.10: Multiple comparisons for prediction models using the residuals.

5.5.2 Determining the best prediction accuracy using Auto-WEKA

The development of an accurate prediction model may involve attempting several types of machine learning models with different configurations, including tuning parameters and selected features. However, this is a difficult and time-consuming task to implement. In this section, a new, rapid and automated tool was used to identify the best prediction accuracy of a software maintainability model, namely Auto-WEKA, applied to sets of different models with various configurations. Auto-WEKA is applied to previous datasets to select the best model, along with the best choice features and parameters. The MMRE and MAE values were used to evaluate the accuracy of the predictive models, along with the ZeroR model, to compare selected model performance with the baseline.

Table 5.34 provides the results of the best-selected model, along with the number of attempted configurations, selected features and tuning parameters. The null in the selected features refers to the selection of the entire dataset without applying the selected features. This table answers **RQ5.B.1**, **RQ5.B.2**, **RQ5.B.3** and **RQ5.B.4**, and reveals several findings. **First**, two individual models were selected as the best prediction accuracy models, namely SMOreg in Eclipse JDT Core, which is an SVR model, and KStar in the Equinox Framework, which is

an instance-based model. **Second**, two meta-models (i.e., ensembles) were chosen as the best prediction accuracy models, namely RandomSubSpace in Eclipse PDE UI, which creates a decision tree-based model, and RandomForest in Lucene and Mylyn, which creates a forest of random trees. **Third**, the number of attempted configurations ranged from 134 to 428. **Fourth**, each selected model had specifically defined tuning parameters, whereas the selected features were applied only for the Equinox Framework and Lucene to determine the BestFirst filter for attribute search and the CfsSubsetEval filter for attribute evaluation.

Table 5.34: Best model selected by Auto-WEKA in each dataset.

Dataset ID	Best-selected model / Number of configurations attempted	Configurations	
		Selected features	Tuning parameters
Eclipse JDT Core	SMOreg / 295	Null	[-C, 1.3565252749701955, -N, 0, -I, weka.classifiers.functions.supportVector.RegSMOImproved, -K, weka.classifiers.functions.supportVector.NormalizedPolyKernel -E 2.8518299249980115 -L]
Eclipse PDE UI	RandomSubSpace / 134	Null	[-I, 53, -P, 0.37299422237345936, -S, 1, -W, weka.classifiers.functions.MultilayerPerceptron, --, -L, 0.6520314185757002, -M, 0.6694968982868784, -B, -H, i, -R, -D, -S, 1]
Equinox Framework	KStar / 428	Attribute search: BestFirst attribute evaluation: CfsSubsetEval	[-B, 38, -M, n]
Lucene	RandomForest / 321	Attribute search: BestFirst attribute evaluation: CfsSubsetEval	[-I, 96, -K, 1, -depth, 12]
Mylyn	RandomForest / 162	Null	[-I, 136, -K, 7, -depth, 14]

Table 5.35 presents the results of the MMRE and MAE values to determine the prediction accuracy achieved by the best-selected model, along with the baseline (i.e., ZeroR). This table responds to RQ5.B.5 and RQ5.B.6, and it is apparent that the selected models performed better than the baseline. In addition, there was a large difference between the MMRE values for the selected models and the MMRE values for ZeroR. Interestingly, the selected model in the Equinox Framework achieved the best accuracy prediction, with a change of 89.78%.

Finally, the results in Table 5.35 were compared with the best prediction model in Table 5.6, Table 5.7 and Table 5.8 to answer RQ5.B.7. The results indicate that all the models

selected by the Auto-WEKA tool outperformed the best model prediction except for KNN and SVR in Eclipse JDT Core and Lucene, respectively.

Table 5.35: MMRE and MAE values for the selected models and ZeroR models.

Datasets ID	MAE for the best-selected model	MMAE for ZeroR	MMRE for the best-selected model	MMRE for ZeroR	% of change between MMRE values
Eclipse JDT Core	638.37	823.90	6.36	13.79	53.88%
Eclipse PDE UI	185.25	223.46	0.79	6.12	87.09%
Equinox Framework	35.54	282.04	0.57	5.58	89.78%
Lucene	50.43	127.15	5.21	15.18	65.68%
Mylyn	46.24	127.67	1.54	5.92	73.99%

Figure 5.11 shows a histogram of the MMRE value to compare the prediction accuracy between the models selected by Auto-WEKA, the best model in the previous section and the ZeroR models, and answer RQ5.B.5 and RQ5.B.6. The low value of this diagram indicates better accuracy achieved by the prediction model. The results obtained from Figure 5.11 indicate the positive impact of using the Auto-WEKA tool to select the best model in prediction software maintainability. This yields an improvement over the baseline (i.e., ZeroR) in the range of 53.88% to 89.78%. The best-selected model by Auto-WEKA in the Equinox Framework dataset achieved the best result (lowest MMRE value), followed by the best-selected model by Auto-WEKA in the Eclipse PDE UI and Lucene datasets. Additionally, all the models selected by the Auto-WEKA tool performed better than the best model prediction in the previous section except for KNN and SVR in Eclipse JDT Core and Lucene, respectively.

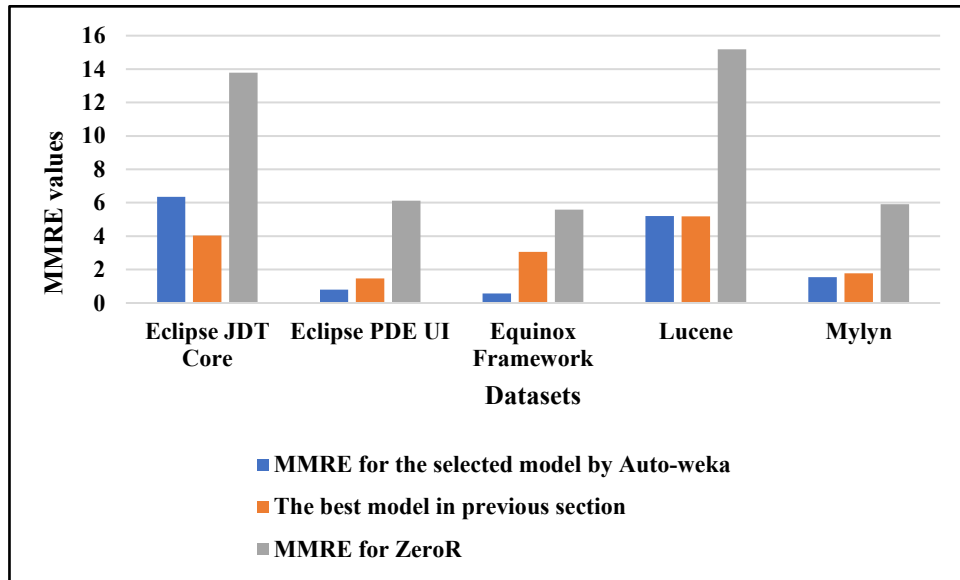


Figure 5.11: MMRE value for selected and ZeroR models in each dataset.

Figure 5.12 illustrates the residual boxplots of the MRE values for the selected and ZeroR models in each dataset. In Figure 5.12, there is a clear tendency of a decrease in the MMRE value indicated by “X” in the diagram, after applying the Auto-WEKA tool to all datasets. Moreover, each selected model had a reduction in the box spread. The results obtained from this figure indicated the positive impact of employing the Auto-WEKA tool on all datasets. This yields an improvement between 53.88% and 89.78%, which is considered a high performance in the software maintainability prediction.

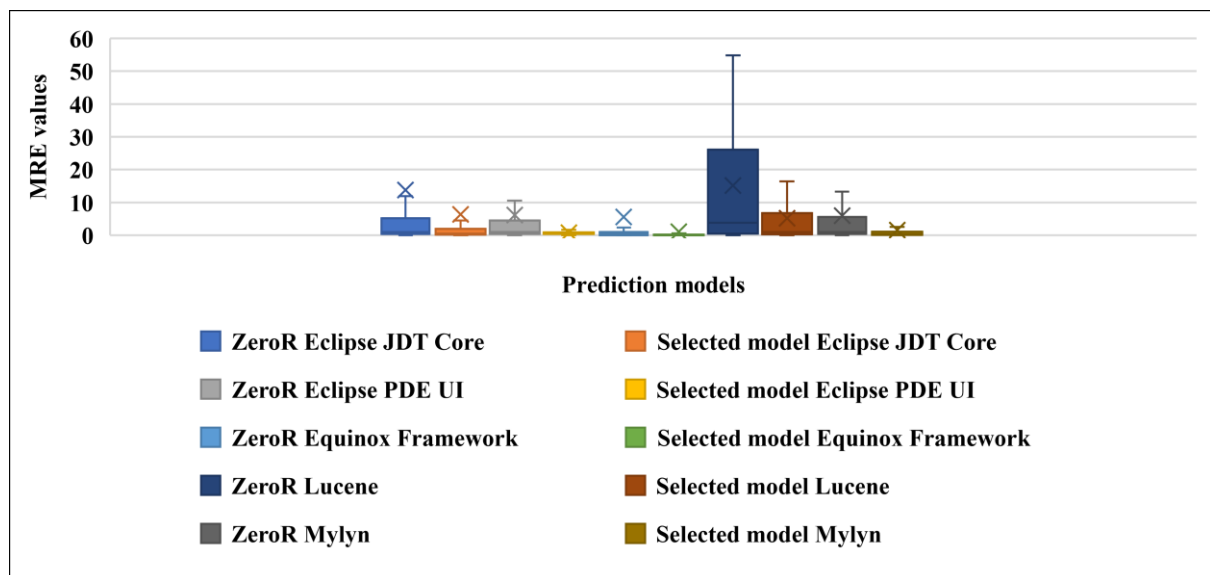


Figure 5.12: Box plot of MRE for selected and ZeroR models in each dataset.

5.5.3 Discussion and answers to research questions for the second empirical study

This section presents the details of the discussion and answers RQs for the second empirical study, which involves two studies, namely 5.A and 5.B. First, the following RQs for Study 5.A are answered and discussed.

RQ5.A.1) What are the suitable metrics (independent variables) in the bug prediction datasets to predict software maintainability?

In the data reduction step, five bug metrics from the single-version-ck-oo file were removed. Consequently, 17 metrics were used as predictors for software maintainability (i.e., independent variables), including 6 CK metrics [26] and 11 OO metrics (see Table 3.5). The metrics were not removed for the following reasons:

- In most cases, in most cases, there was no perfect correlation between two independent variables (see Figure 5.5).
- In few cases, some of the metrics exhibited perfect correlation, but it is not a common problem in all datasets. For example, NOA and NOPA were perfectly correlated in the Eclipse PDE UI dataset (dark blue circle), but not in the remaining datasets.
- Almost all metrics have the same correlated with the dependent variable, which are considered good predictor.

RQ5.A.2) How can the dependent variable calculate the CHANGE metric from the bug prediction datasets?

The CHANGE metric was calculated in two steps of the data preprocessing techniques, namely data reduction and data integration. In the data reduction, all metrics were excluded from the change-metrics file except two fundamental metrics, namely lines added until and lines removed until. Then, these metrics were integrated to compute the CHANGE metric, which is similar to the CHANGE metric proposed by L&H [9]. In the data integration, two files (i.e., single-version-ck-oo and change-metrics files) were integrated into one file, namely software maintainability datasets.

RQ5.A.3) How to improve quality of the software maintainability prediction datasets using preprocessing techniques?

The previous two data pre-processing techniques aim to choose suitable source code metrics (independent variables) from the bug prediction datasets and identify the CHANGE metric

(dependent variable). As a result, a new version of the bug prediction datasets was produced, called the software maintainability datasets, which includes 17 OO and CK metrics (independent variables) and only one CHANGE metric (dependent variable). The objective of the other data pre-processing techniques (i.e., data cleaning and data transformation) is to improve the quality of the software maintainability datasets. According to the data cleaning step, the software maintainability datasets only required the removal of outliers to eliminate variance, and these datasets did not include any noisy or missing values. This is in line with previous studies that used the original version of the bug prediction datasets without finding any missing values problem [154, 155, 199]. Regarding the data transformation step, previous studies performed a pre-processing algorithm to resolve the duplicate instances problems [193] or the normalisation technique to rescale the large range in the datasets [53]. However, these problems were not observed in the software maintainability datasets. Therefore, as the application of all four data pre-processing techniques on the datasets is not mandatory, the transformation step was not necessary [125].

Finally, the findings of applying the data pre-processing techniques suggest that these techniques are very useful for providing a new version of the datasets that can be used for other purposes or to solve problems of the datasets. The findings obtained are compatible with those of previous studies, in which data pre-processing techniques were applied on software quality datasets [190-194].

RQ5.A.4) How much can prediction models increase or decrease the performance compared to a baseline (i.e. ZeroR)?

All the prediction models increased their performance over the baseline, except in rare cases. Additionally, the prediction accuracy of the baseline is considered very low. For example, the result of MMRE ranges from 5.58 to 15.18, and this range was very far from acceptable considering the specified criteria $MMRE \leq 0.25$ [67]. However, if the prediction model is less accurate than ZeroR or if ZeroR has a bad result, this may indicate that the prediction is difficult to achieve [200].

RQ5.A.5) How effective are individual models at predicting change maintenance effort?

SVR was the best individual model for predicting software maintainability. In most cases, neither homogeneous ensemble models (i.e., bagging and additive regression) or heterogeneous ensemble models (i.e., stacking and APE) improved the prediction accuracy of

SVR. According to the results obtained from statistical test, the differences between SVR and other prediction models were not significant. However, there was significantly different between SVR and stacking in Eclipse PDE UI dataset (see Figure 5.10.E). The present findings seem to be consistent with other research, which found that SVR achieved the highest performance for predicting software maintainability across 26 datasets [83]. The findings of this research indicates that SVR can increase prediction accuracy and perform excellent generalisation [117].

RQ5.A.6) How do homogenous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

The bagging ensemble model improved the performance over most of the individual models. The results of the statistical tests indicate that there were no significant differences among all the individual models and bagging ensemble models, and the effect sizes were small. This finding is in line with findings reported in a previous study [1], and leads to a similar conclusion, confirming the effectiveness of using bagging ensemble models for predicting software maintainability. Although SVR as the base model in bagging outperformed the prediction accuracy of other base models, the bagging ensemble model had a minor impact or no impact on SVR. This result may be explained by the fact that bagging requires an unstable base model, such as SVR. Also, this seems consistent with previous research indicating that SVM may be considered a strong model whose performance is not always improved with ensemble models, and that SVM as the base model in bagging provided the best prediction accuracy compared with other prediction models [179]. In contrast, the bagging ensemble model had a considerable impact on RT and a minor impact on SVR and KNN. The detailed analysis shows that the bagging ensemble model improves the prediction accuracy of unstable models, such as RT in all datasets, but lower improvements are observed with more stable models, such as SVR and KNN in some datasets. This finding supports that of Breiman [128], who recommended using bagging with unstable models and also confirms the results of previous research, showing that applying bagging on KNN is not recommended because the output has few changes in the training data via sampling [182]. However, KNN becomes unstable if the number of nearest neighbours (K) has a small value [183] higher than one, as stated by Caprile et al. [184].

The results obtained from applying the additive regression ensemble model are consistent with the bagging ensemble model, whereas the additive regression ensemble increased the prediction accuracy over most of the individual models and achieved the best result with the SVR base model. However, the positive impact of applying the bagging ensemble model to the individual models was better than that of the additive regression, and the opposite result was reached in the first empirical study. Again, there were no significant differences in terms of the residual values between additive regression and the individual models, and the effect sizes were small. Additionally, additive regression did not influence KNN, and the same was observed in the first empirical study. A possible explanation for this finding is that additive regression begins with an empty ensemble and inserts KNN models sequentially. However, KNN calculates the nearest neighbour in the training datasets, and the first result of KNN is equal to further results. The prediction of additive regression is performed by inserting the predictions of each KNN model. For this reason, additive regression at each iteration produces the same results as KNN, which is the same as KNN as an individual model. Therefore, additive regression does not build the ensemble model from the KNN base model because it is unable to find an instance that can serve as an accurate prediction of the error. According to multiple comparison results, there were no significant differences between individual models and homogeneous ensemble models (see Figure 5.10).

RQ5.A.7) How do heterogeneous ensemble models perform in the context of predicting change maintenance effort when compared to the individual models?

Stacking ensemble models increased the performance of RT and MLP, whereas they decreased the performance over the remaining individual models. The results obtained by the statistical tests indicated that there were no significant differences between the individual models and the stacking ensemble model, and the effect sizes were small, except in the case of SVR in the Eclipse PDE UI dataset, which performed better than stacking. This finding was consistent with that in Chapter 4, in which stacking only improved the predictive accuracy of the models that did not perform well individually. Although stacking was built from the diverse individual models and applied on larger datasets, it did not yield a significant improvement. Consequently, the experimental results indicate that in some cases the stacking ensemble models were not better than the individual models.

APE improved the prediction accuracy over RT and MLP, and it had a minor or negative impact in terms of the prediction accuracy in the remaining individual models. These results are in accordance with those observed in the stacking ensemble models; however, overall, APE outperformed stacking. Regarding the statistical tests and effect size results, in most cases, there were no significant differences between individual models and APE ensemble models. However, APE outperformed and differed significantly from RT in the Eclipse PDE UI and Mylyn datasets, along with the MLP in the Equinox Framework dataset. Furthermore, there were significant differences between stacking and APE in the Eclipse PDE UI dataset.

However, the results of stacking and APE are not very encouraging, and it is difficult to compare their results with previous studies because these ensemble models are less widely used than popular ensemble models, such as bagging and boosting [107]. A possible explanation for is that these heterogeneous ensemble models require more base models to perform effectively [201]. For example, previous studies revealed that the proposed heterogeneous ensemble models substantially improved the performance of individual models; however, they used six base models in stacking [202] and seven in APE [131]. Another possible explanation for this is that these previous studies used different combinations of base models. For instance, APE was integrated with RF, stochastic gradient descent, gradient boosting, logistic regression, W-SVMs, Bernoulli naive Bayes, and multinomial naive Bayes [131], whereas stacking was integrated with MLP, radial basis function, pruned model tree, M5Rules, linear regression model and SVM [202]. Additionally, prior studies indicated that RT and MLP produced poor results in predicting software maintainability [13, 48]; therefore, stacking and APE performed better than these models.

Nevertheless, conflicting results between the MMRE and MAE and Pred values were obtained, which is considered a common problem in the empirical studies of software engineering [99]. The observations of the prediction accuracy using Pred values showed that APE and KNN as individual models or as base models in bagging and additive regression in most cases achieved the highest Pred values (see Figure 5.8 and Figure 5.9). This observation is supported by the work of Laradji et al., which shows that APE achieved a high prediction accuracy [131]. The second observation from these figures seems to be consistent with other research that reported the success of KNN in prediction models [178]. However, none of the implemented models in the second empirical study met the model accuracy criteria mentioned

in Chapter 3 ($MMRE \leq 0.25$ and/or $Pred(.30) \geq 0.70$ or $Pred(.25) \geq 0.75$) [34, 67]. This limitation is in agreement with De Lucia et al. [20], who reported that constructing accurate effort prediction models to meet the accepted criteria is very challenging [20].

Second, I provide the appropriate answers and discussion to address the following RQ for the study 5.B:

RQ5.B.1) What is the best-selected model by Auto-WEKA to predict software maintainability in each dataset?

The best-selected model by Auto-WEKA was different in each dataset except RF, which was selected as the best model in the Lucene and Mylyn datasets. By comparing this result with prior studies of software maintainability prediction [47, 48], it can be concluded that RF was the best model in two datasets to predict software maintainability. Therefore, the use of RF to predict software maintainability will be investigated in the next empirical study. However, KStar, SVM, and RandomSubSpace achieved the best prediction accuracy in the Equinox Framework, Eclipse JDT Core and Eclipse PDE UI datasets, respectively. This is consistent with what was found in a previous study showing that the performance of the prediction models of software maintainability was different for different datasets [16].

RQ5.B.2) How many configurations are attempted to select the best model?

Several configurations were performed to select the best model, ranging from 134 to 428. This result highlights the effectiveness of Auto-WEKA in producing a desired model and saving time and effort.

RQ5.B.3) What are the parameter tuning settings in the selected model?

Auto-WEKA provided various settings for the tuning parameter in each dataset (see Table 5.34). This finding confirms the usefulness of Auto-WEKA in saving time and effort.

RQ5.B.4) What are the selected features in the selected model?

Two types of FS, namely the BestFirst filter for attribute search and the CfsSubsetEval filter for attribute evaluation, are performed on Equinox Framework and Lucene datasets. One of the limitations of Auto-WEKA package is that it does not provide the metrics that were chosen by selected feature methods. However, metrics can be selected using “Select attributes” tab in Weka tool; therefore, FS techniques will be explored in the next chapter using Weka tool.

RQ5.B.5) What are the MAE and MMRE values for the selected models?

The results of the MAE and MMRE values are provided in Table 5.35. The results of the experiment clearly indicate the high performance of the models selected by Auto-WEKA.

RQ5.B.6) What is the performance of the selected model by Auto-WEKA compared with the performance of the baseline (i.e., ZeroR)?

All the selected models increased the performance over the baseline, with the percentage of change between MMRE values ranging from 53.88% to 89.78%. This indicates the positive impact of using Auto-WEKA to improve prediction accuracy. Based on these findings, Auto-WEKA is recommended as a useful tool to determine the best model for software maintainability prediction.

RQ5.B.7) What is the performance of the selected model by Auto-WEKA compared with the performance of the best model in the study 5.A?

The models selected by the Auto-WEKA tool achieved better prediction accuracy than the investigated models in Study 5.A, except for KNN and SVR in Eclipse JDT Core and Lucene, respectively. These findings further support the strong effect of KNN and SVR, as these models sometimes achieved higher prediction accuracy than the ensemble model and the best selected models by Auto-WEKA. This thesis calculated the statistical test only for individual and ensemble models using their default parameters because this thesis focuses on the default parameter of machine learning models. In future work, a statistical test can be performed to explore the performance difference between the model selected by Auto-WEKA and the best model in Study 5.A

5.6. Threats to Validity

This section examines the different threats that may influence the results of this study. These threats commonly occur in any empirical study of software engineering that uses open-source software projects [186] and should be considered. The proposed empirical study may face the following threats:

- The bug prediction datasets were extracted from Java systems [57]. As each programming language has unique features and systems vary in their characteristics, these are not representative of all software systems. Further studies should be performed to investigate machine learning models with other programming languages and a more extensive range of systems;

- The datasets (i.e., bug prediction datasets [57]) used in this empirical study are publicly available. However, no previous studies used these datasets for software maintainability, which makes comparisons of this study with other studies impossible;
- In this work, the CHANGE metric is used as a dependent variable, as it is well-known and commonly used in prior studies [7, 11-13, 15-18, 88, 152, 175] to indicate the amount of change in a class during the maintenance process or, in other words, to indicate maintainability. A higher number of changes refers to higher maintenance effort or low maintainability. Maintainability implies the ease to make and accommodate maintenance changes, and the CHANGE metric is more related to the amount of change that is likely to be made to a class. The CHANGE metric is calculated by summing two metrics: lines added until and lines removed until, which refer to the lines added to or removed from the classes during the maintenance period, respectively [57]. The main advantage of this metric is that it has strong relationships with other metrics (independent variables); hence, it can be used as an indicator for predicting the maintenance effort [9]. Therefore, there is no threat in the dependent variable because the CHANGE metric is acceptable for predicting the maintenance effort;
- The performance of the ensemble models varies with different datasets. Therefore, it is not possible to validate the capability of these ensembles using only five datasets, and this might be a recognised threat;
- In this work, only 17 metrics proposed in the bug prediction were used, including 6 CK metrics [26] and 11 OO metrics. CK metrics are commonly used and widely accepted in software maintainability prediction [16, 23]. In contrast, OO metrics are limited in the software maintainability prediction, and this may be considered a threat. Additionally, these metrics are used together for the first time to predict software maintainability, and there are several metrics published in the literature, which might be better predictors for software maintainability;
- The Auto-WEKA tool is used to select the best prediction models by combining different FS and tuning parameters in WEKA [149, 150]. However, KNN and SVR in Eclipse JDT Core and Lucene, respectively, performed better than the best selected model by Auto-WEKA, and this is considered a threat in this tool;

- In this study, ANOVA test was performed to test hypotheses about significant differences between the means of more than two groups, which are five groups in this empirical study: individual model and this individual model as the base model in bagging, additive regression, stacking and APE. One advantage of a parametric statistical test (ANOVA) over non-parametric statistical test is that it produces more effective results with both continuous and nonnormally datasets. ANOVA test assumes that the data in the groups are the same standard deviations and normally distributed, along with independent samples. Despite the apparent assumption were accepted and ANOVA test is suitable for this empirical study, only ten runs were used, which may have impacted the results.

5.7. Conclusion of the second empirical study

In Study 5.A, preprocessing techniques were performed on new and large datasets (i.e., bug prediction datasets) collected from five real-world open-source software systems (Eclipse JDT Core, Eclipse PDE UI, Equinox framework, Mylyn and Lucene) with the objective of producing high-quality datasets that are appropriate for software maintainability prediction. This chapter also empirically evaluated and compared the application of homogeneous (bagging and additive regression) and heterogeneous ensemble models (stacking and APE) with five individual models (RT, MLP, KNN, M5Rules and SVR) to predict software maintainability in OO systems. A new version of the high-quality datasets (i.e., software maintainability prediction) suitable for software maintainability prediction was provided. The experimental results indicate the following:

- Most of the proposed machine learning models improved the accuracy predictions over the baseline (i.e., ZeroR model);
- SVR achieved the best prediction accuracy among individual models, followed by KNN. These findings are in agreement with previous studies showing the excellent performance of SVR and KNN [117, 178];
- Although homogeneous ensemble models improved the accuracy prediction over most individual models, there were no significant differences between the homogeneous ensemble models and individual models, and the effect sizes were small;

- In some cases, the bagging ensemble model had a minor or negative impact on the prediction accuracy over SVR and KNN. This result may be explained by the fact that bagging improved the performance of unstable base models (e.g., RT), whereas it decreased the performance of stable models (e.g., KNN);
- As in the previous chapter, applying an additive regression ensemble model to KNN produced the same results as that of the KNN of the individual models because it is an instance-based rather than model-based approach, and additive regression was unable to improve the initial predictions of KNN. In contrast, the prediction accuracy of bagging ensemble models was better than that of additive regression ensemble models in this chapter and in Chapter 4;
- Stacking ensemble models increased the prediction accuracy over RT and MLP, but there was no significant difference between stacking and these models, and the effect sizes were small. This observation further supports the finding in the previous chapter, in which stacking only increased the performance of the individual models that did not perform well;
- APE increased the prediction accuracy over RT, MLP, and M5Rules, but there were no significant differences between these models and APE. However, APE outperformed and differed significantly from RT in the Eclipse PDE UI and Mylyn datasets, along with the MLP in the Equinox Framework dataset. APE enhanced the performance of all individual models using the Pred value and achieved the best, second-best, or third-best performance compared to other investigated models;
- The prediction accuracy of APE ensemble models was better than that of stacking ensemble models, and there were significant differences between stacking and APE in the Eclipse PDE UI dataset.

The findings of the current study suggest that ensemble models can improve the prediction accuracy of some individual models (i.e., RT, MLP and M5Rules), but there were no significant differences between individual models and ensemble models, except for a few cases in heterogeneous ensemble models. SVR and KNN as individual models or as a base model in bagging and additive regression is a recommended technique for software maintainability prediction, followed by APE. Additionally, the prediction models applied on

large datasets (i.e., Eclipse PDE UI and Mylyn) have higher accuracy prediction compared with those applied on small datasets.

In Study 5.B, the recently developed Auto-WEKA tool was demonstrated to the problem of identifying the best prediction accuracy model for software maintainability prediction among various machine learning models with different configurations for tuning parameters and selected features.

The final result provided the best-selected models in each dataset, which are SMOreg in Eclipse JDT Core dataset, RandomSubSpace in Eclipse PDE UI dataset, KStar in Equinox Framework dataset, and RandomForest in Lucene and Mylyn datasets. The results obtained in the study indicate that using the Auto-WEKA tool can considerably influence the performance of software maintainability prediction models. The results indicate that all the selected models using the Auto-WEKA tool outperformed the best model prediction in Study 5.A, except KNN and SVR in Eclipse JDT Core and Lucene, respectively.

The next chapter will further investigate and analyse recent and large datasets for software maintainability. In addition, it will explore the extent to which FS and sampling techniques can assist in meeting accurate predictions. Moreover, other types of machine learning models will be constructed for the classification problem to predict change-proneness.

Chapter 6. Third Empirical Study: Ensemble Techniques to Predict Change-Proneness Using Newest and Largest Datasets

This chapter aims to investigate the performance of the ensemble model, FS and sampling techniques on prediction change-proneness using four scenarios: (a) datasets without FS and sampling techniques, (b) datasets with FS and without sampling, (c) datasets without FS and with sampling and (d) datasets with FS and sampling. For this purpose, two types of filter-based feature-ranking techniques, namely Relief and Pearson correlation coefficient, were combined using the ensemble concept to determine the best metrics by averaging their ranks and selecting the best ten metrics. The sampling techniques (i.e., SMOTE, SpreadSubsample and randomize) were also combined to solve the imbalance dataset problem. Moreover, three individual models (NB, SVM, KNN) and one ensemble model (RF) were employed to predict change-proneness. These models were applied on seven publicly available datasets (i.e., refactoring datasets) extracted from open-source software systems [58]. The performance of the predicted models was compared and evaluated using the AUC. In the previous empirical study, the impact of parameter tuning using Auto-weka was explored; however, the best prediction model in the second empirical study performed better than the selected model by Auto-weka in two datasets. Therefore, this chapter uses a new method to evaluate the impact of the tuning Mtry parameter, which is the number of variables randomly sampled for splitting in RF using the grid search.

6.1. Introduction

Change-proneness is a dependent variable that indicates whether a change was performed in a given class (e.g., inserting, removing or editing) and can capture the element of maintainability. This Boolean variable has the value TRUE if a change was made on the class (regardless of the type or number of changes) or FALSE if no change was made [5]. A lower number of TRUE values in a system, or a lower value of change-proneness, indicates better

maintainability, i.e., requires lower maintenance effort. Koru and Tian stated that change-proneness is an essential external quality attribute that can decrease the costs of maintenance and increase the quality of source code [203]. Metrics have a powerful correlation with change-proneness and can be utilised to measure the internal features of software systems as independent variables, such as cohesion, complexity, and inheritance [203]. However, the number of metrics used in the literature to predict change-proneness is considered limited [204].

Ensemble concepts, which combine several outputs instead of a single output, have been performed in the machine learning problem to improve the final results. This concept can be applied in the machine learning models [16, 23], FS [205] and sampling techniques [206]. However, there are few studies predicting change-proneness using ensemble models [5], and no other study that uses ensemble FS and sampling techniques.

The key contributions of this chapter are:

- The capability of the ensemble model (RF) in the prediction of change-proneness was evaluated using four different scenarios. To the best of the author's knowledge, this is the first study to investigate aspects like the ensemble model, ensemble FS and ensemble sampling techniques in predicting change-proneness. The most important finding was that RF provided a significant improvement over other prediction models and obtained the highest value of AUC in all scenarios;
- Study using recent and varied datasets, two of which are large and contain more than 1000 classes (i.e., mct and titan). No previous studies were found using these datasets to predict change-proneness;
- The impact of Mtry parameter tuning in RF was explored using grid search.

6.2. Motivation

Many research studies have explored the utilisation of prediction models in software maintainability. Most of these studies have predicted change maintenance effort using CHANGE metric [7, 10-18] and MI [32, 83], but little progress has been made in predicting change-proneness in software maintainability [16, 23].

FS techniques have received increased attention in recent years owing to their ability to improve prediction accuracy and decrease the time to create the model. Additionally, perhaps the main advantage of FS is to identify the best subset attributes (i.e., independent variables) to predict the target attribute (i.e., dependent variable). In the literature related to software quality prediction, a range of various FS and prediction models have been used to predict software quality (i.e., defect, change metric, MI and change-proneness). Table 6.1 shows a summary of the selected studies in the systematic review of FS techniques [4] along with the datasets, prediction models and type of prediction. The ensemble method, which combines the output of several FS methods, was used in four selected studies (S4, S8, S9 and S15). According to the systematic review of FS techniques [4], three selected studies (i.e., S8, S9 and S15) indicated that the ensemble method produced a better prediction accuracy compared to single FS.

Class imbalance occurs when one class of the dataset has a small number of instances compared to the others. This is another problem of the dataset because machine learning models which fail to account for this end up predicting only from the majority class and ignoring the minority class. Various sampling techniques may be used to resolve this problem by adjusting the class distribution. These techniques are generally classified into three types: oversampling to increase the observations of the minority class (e.g., SMOTE [53] and UPSAMPLE [206]); under sampling to decrease the observations of the majority class (e.g., SpreadSubsample [50] and Random under sampling [207]); and ensemble sampling to combine the results of over and under sampling (e.g., SMOTE and bootstrap sampling [208]). Even though ensemble sampling techniques have been proven to increase the prediction accuracy, their application in software quality prediction is also relatively rare [206]. These sampling techniques were applied in various fields (e.g., telecommunications management [49], emerging patterns [50], medical diagnosis [51] and text categorisation [52]). Moreover, several studies in defect prediction have used multiple techniques to resolve the class imbalance problem, such as SMOTE [209] data resampling with boosting [210], random under sampling [54], threshold moving [211], resampling with adapt online change classification [212], random under sampling [207] and a roughly balanced bagging model [213]. However, the application of these techniques in studies of software maintainability prediction is limited (e.g., nearer neighbour [55]).

RF achieved the highest prediction accuracy compared with other prediction models to predict software change [47] and fault [37]. Although several studies have used RF with default parameter settings [37, 47, 131, 214], there is no theoretical justification to apply these default values [138]. As the performance of RF depends on parameter values, it is necessary to investigate RF tuning based on the parameters [138]. Therefore, the impact of Mtry parameter tuning in RF using the grid search was investigated in this study.

Table 6.1: Summary of FS, datasets and prediction models in software quality prediction.

Study ID	Ref	FS method	Dataset	Prediction model	Type of prediction
S1	[215]	Filter and wrapper method inside two classifiers: NB and DT	Public dataset from PROMISE software project repository	NB and DT	Defect
S2	[216]	Filter and wrapper method inside two classifiers: NB and DT	Public dataset from PROMISE software project repository	IB1 and DT	Defect
S3	[37]	Correlation-based feature selection	Public dataset from PROMISE software project repository	AIRS, CLONALG, Immunos, RF, DT, NB	Defect
S4	[217]	Ensemble FS: automatic hybrid search, rough sets, Kolmogorov-Smirnov and probabilistic search	Public datasets extracted from telecommunications software systems	KNN, MLP, SVM, NB and LR	Defect
S5	[218]	Wrapper method inside support vector machine	Public datasets extracted from telecommunications software systems	SVM, NB, MLP, KNN and LR	Defect
S6	[214]	DT induction, Relief and SVM of FS	Public dataset from PROMISE software project repository	18 classifiers: NB, DT, KNN, SVM, MLP, LR, RF et al.	Defect
S7	[219]	information gain, Chi-square (χ^2), two types of Relief (RF and RFW), gain ratio and symmetrical uncertainty	Public dataset from PROMISE software project repository	KNN and SVM	Defect
S8	[220]	Ensemble FS: gain ratio, Kolmogorov-Smirnov statistic, chi-square, Relief algorithm, information gain, symmetrical uncertainty, exhaustive, heuristic and automatic hybrid searches	Public datasets extracted from telecommunications software systems	NB, MLP, SVM, LR and KNN	Defect
S9	[221]	Ensemble FS: Relief, information gain, gain ratio, chi-square and symmetrical uncertainty	Public dataset from PROMISE software project repository	NB, MLP, KNN and LR.	Defect
S10	[222]	Information gain, Relief, gain ratio, chi-square, and symmetrical uncertainty	Public datasets extracted from telecommunications software systems	NB, MLP, KNN, SVM and LR	Defect
S11	[223]	F-score	Public dataset from PROMISE software project repository	LSTSVM, DT, NN, SVM, KNN and NB	Defect
S12	[224]	Bayesian networks and K2 search algorithm, CfsSubsetEval and BestFirst search method, Relief and Ranker attribute evaluation method	Public dataset from PROMISE software project repository	NB	Defect
S13	[131]	Greedy forward selection, correlation-based method with its two variants: Pearson's correlation and Fisher's criterion	Public dataset from PROMISE software project repository	Gradient boosting, LR, APE, RF, stochastic gradient descent, multinomial NB, Bernoulli NB, regular and weighted support vector machines	Defect
S14	[18]	Rough set analysis with K-means clustering	Public datasets extracted from two commercial software	Hybrid neural network and fuzzy logic approach	Change maintenance effort

			products and published as an appendix [9]		
S15	[83]	Ensemble FS: BestFirst, linear forward selection, greedy stepwise, evolutionary search, genetic algorithm, PSO, tabu search	Partial datasets extracted from 20 systems available in sourceforge.net	MLR, MLP, SVR and M5P regression tree	Maintainability index
S16	[23]	Correlation-based FS	Partial dataset extracted from android application packages	MVEC, WVEC, HIEC, WVHIEC and seven individual particle swarm optimizations	Change-proneness
<p>** NB: Naive Bayes, DT: Decision Tree, IB: aninstance–base classifier, RF: Random Forests, AIRS: Artificial Immune Recognition Systems, KNN: K Nearest Neighbors, SVM: Support vector machine, SVR: Support vector regression, MLP: Multilayer perceptron, LR: Logistic Regression, MLR: Multi Linear Regression, LSTSVM: Linear Twin Support Vector Machine, NN: Neural Network, CLONALG : Clonal selection algorithm, MVEC: Majority Voting Ensemble Classifier, WVEC: Weighted Voting Ensemble Classifier, HIEC: Hard Instance Ensemble Classifier, WVHIEC: Weighted Voting Hard Instance Ensemble Classifier.</p>					

The lessons learned from previous studies discussed above and selected primary studies in Chapter 2 are as follows:

- Relatively few studies have been performed in prediction change-proneness compared with other software maintainability measurements such as change maintenance effort and MI;
- Several machine learning models have been employed in regression problems, whereas less work has been performed in classification problems;
- A limited number of metrics have been used as predictors of change-proneness;
- Although ensemble models have yielded improved prediction accuracy over individual models, limited studies used these models to predict change-proneness (e.g., bagging and boosting in [16] (S47 in selected primary studies in Chapter 2) and majority voting, weighted voting and hard instance ensemble [23] (S56 in selected primary studies in Chapter 2));
- There is clear evidence of the lack of adequate research on the use of FS and sampling in change-proneness, and no application of the ensemble concept in these techniques in change-proneness was found;
- Despite several uses of FS proposed in Table 6.1, only one study used FS to predict change-proneness. This study used partial datasets, which are not publicly available, but they were collected from open-source software systems;
- The systematic review [4] indicated that the FS techniques used in Table 6.1 achieved better prediction accuracy than using all features (i.e., without applying

FS). Moreover, the use of ensemble FS outperformed other FS methods in three studies (i.e., S8, S9 and S15);

- RF was the most frequently used in the systematic review of FS techniques [4] and achieved the best prediction accuracy in previous studies [37, 47], Thus, there is a real need to perform RF in this chapter and investigate the influence of Mtry parameter tuning using the grid search.

To fill the gaps in these previous studies, the empirical study was designed to predict change-proneness using three individual models and one ensemble model. These models are the most frequently applied in Table 6.1. Additionally, ensemble FS techniques were used, including Relief and Pearson’s correlation, which are the most popular FS techniques used in Table 6.1. To make the predictive models refutable, confirmable and repeatable, recent public datasets published in [58] and available on the PROMISE Repository [109] were used.

6.3. Research Method

The research method is defined by the research objectives, RQs and research framework to address the problem of predicting change-proneness accurately. The primary objective of this chapter is to evaluate the impact of ensemble models (RF), ensemble FS (Relief and Pearson’s correlation coefficient) and ensemble sampling techniques (SMOTE, SpreadSubsample and randomize) on the performance of the prediction of change-proneness. The interaction between feature selection and sampling is examined via the following four scenario:

First scenario: datasets without FS and sampling techniques.

Second scenario: datasets with FS and without sampling techniques.

Third scenario: datasets without FS and with sampling techniques.

Fourth scenario: datasets with FS and sampling techniques.

Table 6.2 illustrates the scenarios explored in the empirical study.

Table 6.2: Scenarios in the empirical study.

	FS techniques	Sampling techniques
First scenario	No	No
Second scenario	Yes	No
Third scenario	No	Yes
Fourth scenario	Yes	Yes

These scenarios were studied because the datasets present two problems: (I) high dimensionality, which includes irrelevant and redundant features, and (II) imbalanced classes. This poses a difficulty regarding which model to apply first: the FS for the high dimensionality or sampling techniques for the imbalanced classes. Furthermore, these scenarios help to evaluate and compare the impact of ensemble FS and sampling techniques separately in the second and third scenarios and together in the fourth scenario. Three data analysis steps are performed before conducting these scenarios: (I) removal of empty values attributes; (II) removal of redundant attributes, which are perfectly correlated with the dependent variable; (III) removal of attributes that have a strong correlation with other attributes. Moreover, there are three further steps after applying data analysis methods, namely normalisation, ensemble FS and ensemble sampling techniques (see Figure 6.2).

The fourth scenario is designed to resolve both problems (high dimensionality and class imbalance) and to avoid biased results from the sampling techniques. Therefore, the ensemble FS was applied using two filter-based feature ranking techniques, Relief and Pearson's correlation coefficient, to the fourth set of differently sampled data purely for the purpose of performing FS techniques. The sampled data includes datasets without sampling, datasets with SMOTE sampling, datasets with SMOTE and SpreadSubsample with the parameter set to 1, and datasets with SMOTE and SpreadSubsample with the parameter set to 2.1, and these parameters define the distribution spread ratio between the minority and majority classes as 50:50 and 35:65. Then, the average of the feature ranking techniques across above-mentioned sampled data was computed. Finally, the best ten features with the highest ranking from the

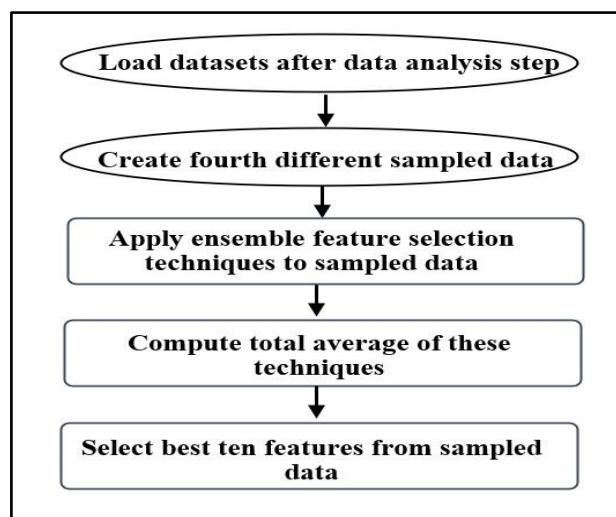


Figure 6.1: Framework of the fourth scenario.

datasets in the third scenario were selected (i.e., datasets without FS and with sampling techniques, namely SMOTE, SpreadSubsample with 2.1 ratio and randomize). Figure 6.1 illustrates the framework of the fourth scenario.

The five RQs were focused to accomplish the objectives of this chapter:

RQ6.1) What is the impact of ensemble FS techniques on the performance of prediction models?

RQ6.2) What is the impact of ensemble sampling techniques on the performance of prediction models?

RQ6.3) What is the impact of applying both ensemble FS and sampling techniques on the performance of prediction models?

RQ6.4) How effective are individual models and how do ensemble models perform when compared to the individual models in the context of predicting change-proneness?

RQ6.5) What is the impact of the Mtry parameter tuning in RF?

Figure 6.2 shows the framework of the research method, which contains several steps:

Step 1. Loading of seven datasets (i.e., oryx, junit, antlr4, mcMMO, MapDB, mct and titan) that were manually validated by Hegedús et al. [58] to analyse refactorings through several subsequent system releases [58]. These datasets were extracted from seven open-source Java systems in GitHub [27], and include 125 source code metrics that form the independent variables and one metric called Refact_Sum that is used as the dependent variable to reflect change-proneness and capture the element of maintainability. The datasets are described in Section 3.4.3;

Step 2. Analysis of the datasets to eliminate the metrics that contain empty values, direct relationships with the dependent variable, or are redundant metrics (strongly correlated with each other). This step is part of the feature selection process performed before all scenarios mentioned in Table 6.2 using manual analysis, descriptive statistics and Spearman correlation;

Step 3. Application of normalization to the set values of the dataset between 0 and 1;

Step 4. Execution of ensemble FS using the Relief and Pearson's correlation coefficients, calculation of the average of these techniques, and selection of the best ten metrics (features) from each dataset;

Step 5. Execution of ensemble sampling techniques, namely SMOTE, to perform oversampling by increasing the number of minority classes, and SpreadSubsample to perform

under sampling by decreasing the number of majority classes, and randomization to randomly rearrange the instances. The main reason for applying this randomization is to avoid overfitting in ten-fold cross-validation because the SMOTE technique inserts additional instances (True values) at the end of the dataset;

Step 6. Execution of four different scenarios across seven datasets: **First scenario** (i.e., Steps 1, 2 and 3), **Second scenario** (i.e., Steps 1, 2, 3 and 4), **Third scenario** (i.e., Steps 1, 2, 3 and 5) and **Fourth scenario** (i.e., Steps 1, 2, 3, 4 and 5) (see Figure 6.2).

Step 7. Division of all previous datasets into ten sets using ten-fold cross-validation. The datasets were divided into training sets to build prediction models and test sets to compare the performance of prediction models using AUC;

Step 8. Construction of prediction models, which encompass three individual models (NB, SVM and KNN) and one ensemble model (RF);

Step 9. Prediction of change-proneness by evaluation and comparison of the results of four prediction models across four scenarios to determine the most accurate prediction model using AUC as the measure of comparison.

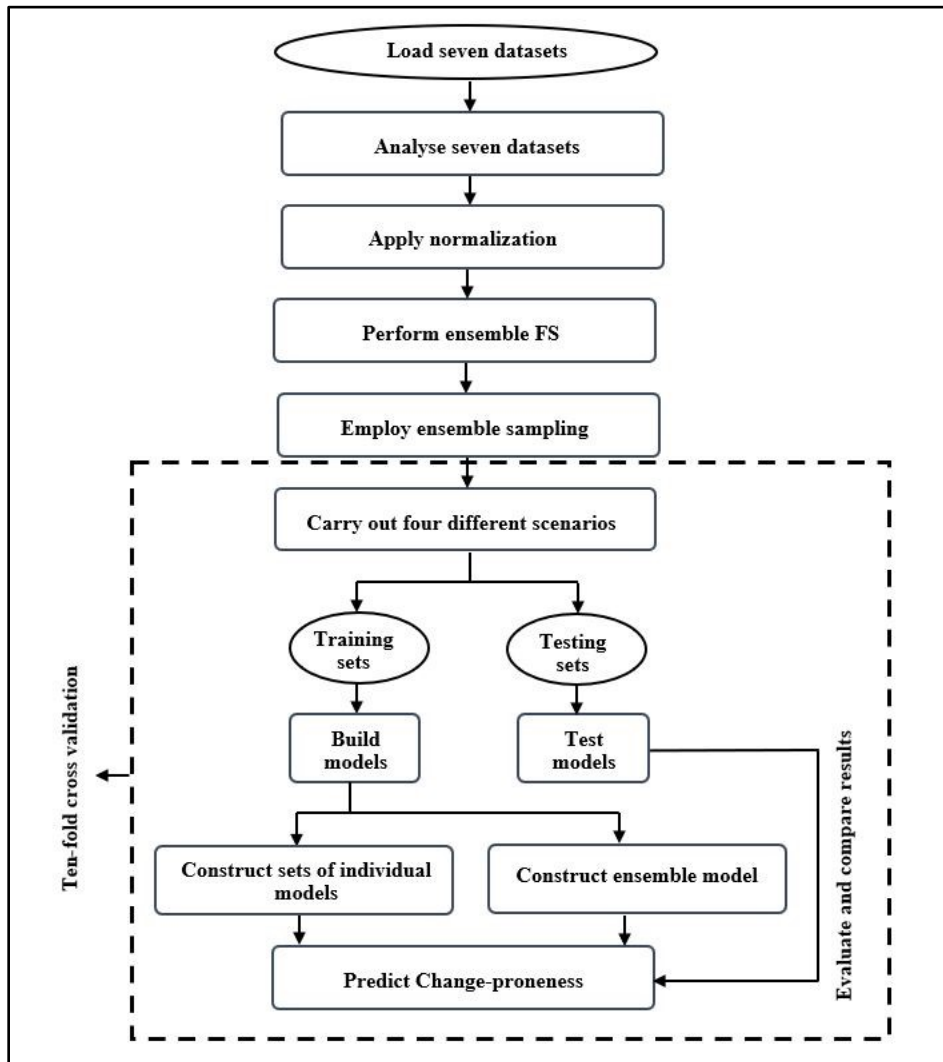


Figure 6.2: Framework of the research method.

6.4. Experimental Data Setup

The following subsections present the evaluation of the datasets performed in this empirical study, along with the explanation of the dependent and independent variables. They also provide details about the dataset analysis using descriptive statistics and Spearman correlation. Finally, they show the data pre-processing that includes normalization, FS and sampling.

6.4.1 Evaluation of refactoring datasets

In this chapter, seven publicly available datasets proposed in [44], called refactoring datasets, were performed. However, only the class metrics that contain 125 independent variables, and

one dependent variable (i.e., Refact_Sum), defined as the total number of refactoring changes that occurred, were used. Refact_Sum was used as an indicator for change-proneness and assumes the values True or False, making this a classification problem.

Additionally, the relative maintainability index attribute was removed from the datasets and not used as a dependent variable because it is a relative attribute derived from a set of source code metrics [58], some of which appear as independent variables. As this index is derived from metric values, rather than being directly based on observations of maintenance effort, it does not accurately reflect the maintenance effort; in addition, as a dependent variable, a machine learning model would just learn the function that defines the relative maintainability index [58]. The explanation of independent and dependent variables will be presented in the next two sections. Table 6.3 provides a summary of the datasets used in this chapter, including dataset name, number of classes, number of releases, time interval and URL [58]. The new version of these datasets after removing the refactoring variables and relative maintainability index and after converting Refact_Sum to Change_Prone is provided in the following link:

<https://zenodo.org/record/4266681#.X6rMvmgzY2w>

Table 6.3: Summary of the datasets.

Dataset name	# Classes	# Release	Time interval	URL
antlr4	436	5	21/01/2013–22/01/2015	https://github.com/antlr/antlr4
junit	657	8	13/04/2012–28/12/2014	https://github.com/junit-team/junit
MapDB	439	6	01/04/2013–20/06/2015	https://github.com/jankotek/MapDB
mcMMO	301	5	24/06/2012–29/03/2014	https://github.com/mcMMO-Dev/mcMMO
mct	2162	3	30/06/2012–27/09/2013	https://github.com/nasa/mct
oryx	536	4	11/11/2013–10/06/2015	https://github.com/cloudera/oryx
titan	1486	6	07/09/2012–13/02/2015	https://github.com/thinkaurelius/titan

6.4.2 Dependent variable: change-proneness

Change-proneness is a dependent variable that reflects changes performed in a given class (e.g., inserting, removing or editing) and can capture the element of maintainability. This Boolean variable has the values TRUE if the change was made on the class or FALSE if no change was made, regardless of the types and number of changes. The Refact_Sum attribute contains the total number of source code refactoring operations that have been applied in each class a certain observation period. This attribute was treated as Boolean by assigning TRUE if the sum of refactoring contained any number or FALSE if the sum of refactoring was zero. Table 6.4 lists the number of True and False values in the change-proneness attribute (i.e., dependent variable). This table indicates the differences between the number of True and False

values, which makes the class of dependent variable highly imbalanced. To the best of the author’s knowledge, there is no standard approach to classify the difference between the number of True and False values. Therefore, this difference was categorized into small, medium and large according to the percentage of True values in the datasets: values less than 1% correspond to a large difference, values less than 2% refer to a medium difference and values out of these ranges indicate a small difference. The percentage of True values in Table 6.4 is considered very small, ranging from 0.69% to 5.28%.

Table 6.4: Number of True and False values in the change-proneness attribute.

Dataset name	# Instances	# True value	% True value	# False value	% False value	Category of difference between True and False
antlr4	436	23	5.28%	413	94.72%	Small
junit	657	9	1.37%	648	98.63%	Medium
MapDB	439	4	0.91%	435	99.09%	Large
mcMMO	301	4	1.33%	297	98.67%	Medium
mct	2162	15	0.69%	2147	99.31%	Large
oryx	536	15	2.80%	521	97.20%	Small
titan	1486	13	0.87%	1473	99.13%	Large

6.4.3 Independent variables: source code metrics

The independent variables include 125 metrics, which can be grouped into ten categories as follows: cohesion, complexity, coupling, documentation, inheritance, size, code duplication, warning, rules and refactoring. All the independent variables are numeric and were collected using the SourceMeter static code analysis tool [156]. Hegedűs et al. describe how they extracted these metrics [58], and their explanation is also listed on the tool’s website [156]. The metrics used as independent variables in this chapter and their category are provided in Table 3.6 in Section 3.4.3.

6.4.4 Datasets analysis

The primary objective of the dataset analysis is to remove the metrics that are directly related to the dependent variable, correlated with each other, or with zero values. This stage is performed using the following techniques: manual evaluation, descriptive statistics and Spearman correlation. First, 23 refactoring metrics mentioned in Table 3.6 were removed from the independent variables. These are related to specific refactoring changes that have taken

place and are not relevant for the prediction of change proneness and would also prejudice the outcome of the study as they identify when a change has been made.

Second, metrics with zero values in the results of descriptive statistics were removed (see Table 6.5), because they cannot be used as predictors for target variable, as recommended by Briand et al. [198]. From Table 6.5, 21 metrics were removed from some of the datasets, and 7 metrics were removed from all datasets.

Table 6.5: Metrics with zero values that were removed using descriptive statistics

Metrics	Datasets						
	antl4	junit	MapDB	mcMMO	mct	oryx	titan
WarningBlocker	*	*	*	*		*	*
Android Rules	*	*	*	*	*	*	*
Brace Rules						*	
Clone Implementation Rules		*	*	*			
Code Size Rules	*	*	*	*	*	*	*
Comment Rules	*	*	*	*	*	*	*
Complexity Metric Rules		*					
Coupling Rules	*	*	*	*	*	*	*
Empty Code Rules	*						
Finalizer Rules	*	*	*	*		*	*
Import Statement Rules				*		*	
J2EE Rules			*	*		*	
Jakarta Commons Logging Rules		*	*				
JavaBean Rules		*		*	*		
MigratingToJUnit4 Rules	*	*	*	*	*	*	*
Migration Rules	*			*		*	*
Migration13 Rules	*	*	*	*	*	*	*
Migration14 Rules	*	*	*	*	*	*	*
Migration15 Rules	*	*	*	*	*	*	*
Security Code Guideline Rules				*			
Vulnerability Rules	*	*	*	*		*	*

Third, a Spearman correlation, which is a well-known statistical measurement to compute the strength (i.e., strong or weak) and direction (i.e., positive or negative) of the relationship between two variables, was performed [225]. This correlation was used to remove redundant variables and prevent multicollinearity that occurs when one independent variable in a prediction model can be predicted from other independent variables with a high accuracy [226]. This FS technique was applied before building any prediction model to avoid skewed or misleading results. Variables with a correlation coefficient of +1.0, +0.9, -1.0 or -0.9 were eliminated; however, no negative correlations were found in these datasets. Table 6.6 presents the results of the strong correlations and shows the attributes that were removed using Spearman correlation. The results of this table indicate that 35, 32, 31, 37, 33, 32 and 38 metrics were removed from antl4, junit, MapDB, mcMMO, mct, oryx and titan, respectively. Furthermore, the CC metric had a strong correlation with CCL, CCO, CI, CLC, CLLC, LDC,

LLDC and Clone Metric Rules in all datasets and NL and NLE were also strongly correlated in all datasets.

Table 6.6: Strong correlation metrics using Spearman correlation.

Strong correlation metrics with 1 or 0.90 values		Datasets						
Metrics retained	Metrics removed	antl4	junit	MapDB	mcMMO	mct	oryx	titan
CC	CCL	✓	✓	✓	✓	✓	✓	✓
	CCO	✓	✓	✓	✓	✓	✓	✓
	CI	✓	✓	✓	✓	✓	✓	✓
	CLC	✓	✓	✓	✓	✓	✓	✓
	CLLC	✓	✓	✓	✓	✓	✓	✓
	LDC	✓	✓	✓	✓	✓	✓	✓
	LLDC	✓	✓	✓	✓	✓	✓	✓
	Clone Metric Rules	✓	✓	✓	✓	✓	✓	✓
NL	NLE	✓	✓	✓	✓	✓	✓	✓
WMC	TNOS	✓						
	TNLM	✓						
	TLLOC	✓					✓	
	TLOC	✓					✓	
	NOS	✓						✓
	RFC	✓						
	LLOC	✓					✓	✓
	LOC	✓					✓	✓
	NLM	✓	✓				✓	✓
	NOD			✓				
	NLS			✓				
	TNPM				✓			
	Complexity Metric Rules				✓			
	String and StringBuffer Rules				✓			
RFC	AD							✓
	PDA							✓
AD	DLOC		✓	✓		✓	✓	
	PDA		✓	✓		✓	✓	
	TNOS				✓			
	TNPA				✓			
CD	TCD	✓	✓	✓		✓	✓	✓
	CLOC		✓	✓		✓	✓	✓
	TCLOC		✓	✓		✓	✓	
	TNLS				✓			
	TNS				✓			
DLOC	PDA				✓			
	TCLOC							✓
TCD	TCLOC				✓			
CLOC	DLOC	✓						
	TCLOC	✓						
PUA	NLPM	✓						
	TNLPM	✓						
	Documentation Metric Rules	✓						
DIT	NOA	✓	✓	✓	✓	✓	✓	✓
	NOP		✓	✓	✓	✓	✓	✓
NOC	CBO	✓						
	NOD		✓		✓	✓	✓	✓
LLOC	LOC		✓	✓				
	NOS		✓	✓	✓		✓	
	TLLOC		✓	✓				✓
	TLOC		✓	✓				✓
	TNOS			✓			✓	✓
	NPA				✓			
	TNLPA				✓			
	NLPA				✓			
LOC	TLLOC				✓			
	TLOC				✓			
NA	TNA		✓		✓	✓	✓	
NG	TNG	✓		✓	✓		✓	
	NLG				✓			

NLA	TNLA	✓	✓	✓	✓	✓	✓	✓
NLG	TNLG		✓				✓	✓
NLM	NLPM		✓	✓		✓		✓
	NM				✓			
	TNLM			✓	✓	✓	✓	✓
NLPA	TNLPA	✓					✓	✓
	NPA		✓					
NLPM	NII	✓						
	NLS	✓						
	TNLPM		✓		✓	✓	✓	
NLS	TNLS		✓			✓	✓	✓
	NS			✓	✓			
NPM	TNPM		✓			✓		
TNLM	TNM				✓			
NM	NPM	✓		✓		✓		✓
	TNM	✓		✓		✓	✓	✓
	TNPM	✓						✓
TNLS	TNS			✓				
NPA	TNPA							✓
NOS	TLLOC					✓		
	TLOC					✓		
	TNOS					✓		
NS	TNS		✓					✓
WarningInfo	TNS	✓						
	Documentation Metric Rules		✓					
	Design Rules				✓			
	Vulnerability Rules					✓		
	Brace Rules							✓
WarningMajor	Unnecessary and Unused Code Rules'							✓

Table 6.7 shows the number of metrics removed at each data analysis stage and reveals that the number of independent variables remaining was 54, 55, 57, 48, 60, 55 and 52 in antl4, junit, MapDB, mcMMO, mct, oryx and titan datasets, respectively.

Table 6.7: Number of metrics removed in each data analysis technique.

Analysis technique	Description of metrics removed	Datasets						
		antl4	junit	MapDB	mcMMO	mct	oryx	titan
		Number of metrics removed						
Manual analysis	Remove refactoring metrics that are directly related to the dependent variable	23	23	23	23	23	23	23
Descriptive statistics	Remove metrics with zero values	13	15	14	17	9	15	12
Spearman correlation	Remove metrics with a strong correlation with others	35	32	31	37	33	32	38

The descriptive static of these metrics after applying data analysis are presented in Table C.2, Table C.3, Table C.4, Table C.5, Table C.6, Table C.7 and Table C.8 in Appendix C. The minimum values are zero or one in all datasets. In contrast, the maximum values are 575 in CLOC metric in antlr4 dataset, 662 in TNM metric in junit dataset, 11272 in LLOC metric in MapDB dataset, 1160 in LOC metric in mcMMO dataset, 1390 in WarningInfo metric in mct dataset, 179 in NII metric in oryx dataset and 1104 in WarningMinor metric in titan dataset. Thus, there was a considerable difference between the maximum and minimum

values in all datasets. For this reason, normalization was applied, which will be described in the next section. Another important finding was that the average values of WarningInfo metrics were high in all datasets, ranging from 8 to 15. The remaining metrics have different values of descriptive static, which suggests that the datasets have varying characteristics.

6.4.5 Data pre-processing

Data pre-processing is a fundamental data mining technique performed to resolve issues related to the datasets, such as incorrect, missing, imbalanced and inconsistent data [125]. The datasets used in this chapter had several problems that required the application of some of the data pre-processing techniques. Initially, as the values of the independent variables had a different range, a normalization was applied to set their range from 0 to 1. Second, as the datasets are considered high dimensional because they contain several independent variables (i.e., 125 metrics), sets of FS were implemented to determine relevant features. Finally, as the classes in the dependent variable (i.e., change-proneness) were clearly imbalanced, as presented in Table 6.4; sampling techniques were applied to balance datasets. These techniques are described in the next sections.

A. Normalization

Normalization is a common technique employed when the values of numeric data have very different scales. It is also essential for the application of some machine learning models that use scale-sensitive distance metrics, such as KNN which uses Euclidean distance to identify the nearest neighbours. In this study, the datasets were linearly rescaled using the Min-Max normalization to normalize all numeric values in the datasets to the interval $[0, 1]$ [125].

B. Feature selection

Spearman correlation was applied in the data analysis step to avoid multicollinearity, and this was necessary for all scenarios. In this section, FS was performed in the second and fourth scenarios as a further step to improve the quality of the dataset as it still contained numerous features. Additionally, FS is considered one of the critical steps in data pre-processing, and it helps to address two general types of problems in the dataset [125]: irrelevant, which refers to features that do not have any effect on the target features, and redundant, which refers to two or more independent variables with the same role [227]. In addition, these techniques have been used to increase prediction accuracy, reduce the model building time, and identify the

most vital features that affect the target attribute (e.g., change-proneness or fault-proneness, etc.). There are four basic methods currently adopted in research on FS: (1) the filter method, which determines the features without building machine learning models using heuristically identified relevant knowledge [228]; (2) the wrapper method, which combines features into a prediction model to choose relevant features [229]; (3) the embedded method, which applies FS as a part of the modelling process and does not divide the model from the FS part [230]; and (4) the ensemble method, which integrates the output of several FS techniques (e.g., Pearson's correlation coefficient and best first techniques) based on a defined combination (e.g., the highest ranking and the best subsets features) [205]. Although the ensemble FS method has proven to be an excellent method to improve the prediction accuracy compared with other individual methods, a limited number of studies have applied this method in software quality prediction [4].

In this chapter, the ensemble method was used to determine the best features using two types of filter-based feature ranking techniques, namely Relief and Pearson's correlation coefficient. These techniques were chosen mainly because they are the most frequently performed in the selected primary studies mentioned in Table 6.1. First, these techniques were applied on the datasets that assign a score to each feature; then, the average score of two filter-based feature ranking techniques was calculated. Second, the best ten features that record the highest scores were selected. There is no clear evidence in the literature of a suitable number of features to select [207]; the number of features chosen was determined by previous studies either by identifying the number of features [222] or employing a cut-off value [118]. In most cases, these features had the same cut-off value (0.1). These techniques were applied using WEKA and the parameters were set as the default values. A brief description of the FS techniques used in this study is provided below.

1. Relief

Relief is one of the filter-based feature ranking techniques provided by Kira and Rendell [231]. The rank of features is computed in Relief by specifying a zero value to all feature weights. Then, Relief assigns a score of feature value by determining differences between the nearest pairs of instances. Relief compares a randomly selected instance with another instance in the same class (nearest hit), where the feature score decreases, and with another instance from a different class (nearest miss), where the feature score increases. Relief is performed by

searching nearest misses and hits and calculating their average to assign the weights of each feature. Finally, all the feature weights are modified to have a specific score. The Relief extends from Relief algorithm, which resolves multiclass and noise problems in the datasets [231].

2. Pearson's Correlation Coefficient

The Pearson's correlation coefficient technique chooses metrics that have a high correlation with the target attribute [232]. This technique is implemented by calculating the correlation coefficient between independent and dependent variables. Pearson's correlation is based on the covariance of two variables divided by the output of their standard deviation. According to the value obtained, the correlation can be classified into strong positive, with a value near to +1, strong negative, with a value near to -1, and uncorrelated, with a value equal to zero [177]. The relationship between variables can also be classified into positive (+), in which the variables are directly proportional, and negative (-), in which the variables are inversely proportional [48].

C. Sampling

The class imbalance occurs in the classification problem as the classes (True and False values) in the dependent variable (change-proneness) are not approximately similar in their distribution and one class has a very small minority. In this study, all the datasets have a very small minority of True values (see Table 6.4). Classes imbalance is considered a serious problem that leads to bias prediction model towards the majority class because a prediction model tends to increase prediction accuracy by disregarding the minority class and learning from the majority class [50].

To resolve the class imbalance problem, sampling techniques are introduced, including oversampling to increase the observations of the minority class and under sampling to decrease the observations of the majority class. The main advantage of oversampling is to maintain all the observations of both the majority and minority classes, but this may result in overfitting. In contrast, as under sampling eliminates some observations, essential information may be removed. However, there is no clear indication of the best technique [233].

Therefore, ensemble sampling techniques were performed in this study to resolve the class imbalance problem and improve the overall performance. Sets of three sampling techniques that involve SMOTE for oversampling, SpreadSubsample for under sampling, and

randomize for mixing the order of the instances were applied. Ensemble sampling integrates the results of over and under sampling, in which SMOTE inserts more True values and SpreadSubsample removes some False values. A brief explanation of the sampling techniques used in this study is presented in the following sections.

1. SMOTE

SMOTE is an oversampling technique that increases the number of observations in the minority class. SMOTE generates synthetic objects based on the nearest neighbour of each sample in the minority class. This synthetic object is computed using the variation between samples of the feature space under consideration for each dependent variable and its nearest neighbour. After that, this variation is multiplied by a random number between 0 and 1. Therefore, the new observations are produced by integrating both features of the dependent variable and its neighbours and the observations are not duplicated from the existing observations in the minority class. This procedure effectively increases the observations of the minority class and creates comprehensive samples [50, 234]. The number of observations is based on the SMOTE percentage, which is a multiple of 100 and should not exceed 300, as recommended in [235]. Although previous studies used different percentages for SMOTE [234, 236], to the best of the author's knowledge, there is no standard approach for defining the SMOTE percentage in the literature. Therefore, this parameter was adjusted to 100%, 200% or 300% based on the category of difference between True and False mentioned in Table 6.4, in which 100% was used for small, 200% for medium and 300% for large. However, the same ratio of SpreadSubsample was used, which creates datasets with the same percentage of True and False values. Table 6.10 in Section 6.5.2 includes results of SMOTE and more details about the selection of the percentage.

2. SpreadSubsample

The SpreadSubsample is an under sampling technique that decreases the number of observations in the majority class by generating a random subsample from the dataset. The SpreadSubsample identifies the class distribution by randomly eliminating observations from the majority class. This distribution is calculated using a Spread value, which is a parameter for the maximum class distribution spread between the minority and majority classes [50]. This parameter was set to 2.1, which indicates that the distribution spread ratio between the

minority and majority classes is 35:65. In the fourth scenario, this parameter was set to 1, which refers to a uniform distribution, thus the classes were balanced for FS only.

3. Randomize

The SMOTE technique inserts more instances (True values) at the end of the dataset, and this leads to overfitting in ten-fold cross-validation. For this reason, the randomization was applied to change the order of the instances [237].

Table 6.8 illustrates the parameters used in Weka for sampling techniques. As shown in this table, all the parameters are default values in Weka except the percentage in SMOTE, which is modified to 200 or 300 according to the difference between True and False, and the default value is 100. Additionally, the distribution spread in SpreadSubsample is changed to 1.0 or 2.1 (the default value is 0.0).

Table 6.8: Parameters used in Weka for sampling techniques.

Sampling techniques	Parameters
SMOTE	-C 0 -K 5 -P 100.0 -S 1 -C 0 -K 5 -P 200.0 -S 1 -C 0 -K 5 -P 300.0 -S 1
SpreadSubsample	-M 1.0 -X 0.0 -S 1 -M 2.1 -X 0.0 -S 1
Randomize	-S42

6.5. Results and Analyses

This section discusses the results obtained. Initially, the results of the application of ensemble FS and ensemble sampling are presented. In sequence, the prediction accuracy of four prediction models in terms of AUC across four different scenarios are compared and evaluated, and the best prediction models are determined. Finally, the results are validated using tests of significance.

6.5.1 Results of feature selection

Relief and Pearson's correlation coefficient techniques in the ensemble FS evaluate each feature or metric and assign a rank to them. Thus, the average of these techniques was calculated and the best ten metrics across seven datasets that impact on change-proneness were selected, which are provided in Table 6.9. The overall results demonstrate that different metrics subsets were obtained from each dataset, and that the maximum number of metrics selected in

each dataset was three, namely PUA, WarningInfo, WarningMajor, JUnit Rules, String and StringBuffer Rules, RFC, Cohesion Metric Rules and WarningMinor.

Table 6.9: Best ten metrics using ensemble FS method.

Metrics	Datasets						
	antl4	junit	MapDB	mcMMO	mct	oryx	titan
CC	✓	✓					
LCOM5	✓					✓	
PUA	✓	✓				✓	
NM	✓						
WarningInfo	✓	✓					✓
WarningMajor	✓		✓		✓		
Clone Metric Rules	✓						
JUnit Rules	✓		✓				✓
String and StringBuffer Rules	✓				✓		✓
Type Resolution Rules	✓						
NOI		✓			✓		
RFC		✓			✓		✓
LLOC		✓	✓				
TNM		✓					
TNOS		✓					
WarningCritical							✓
Cohesion Metric Rules		✓		✓		✓	
Type Resolution Rules		✓					
WarningMinor			✓	✓			✓
Basic Rules			✓	✓			
Complexity Metric Rules			✓		✓		
Controversial Rules			✓				
Migration Rules			✓				
Naming Rules			✓		✓		
Strict Exception Rules			✓				
NL				✓	✓		
CBOI				✓			
NII				✓			
NA				✓			
NLA				✓			
Empty Code Rules				✓			✓
Strict Exception Rules				✓			
CBO					✓		
Coupling Metric Rules					✓		
Optimization Rules					✓		
AD						✓	
CD						✓	
NLG						✓	
NLM						✓	
NLPM						✓	
NPM						✓	
Documentation Metric Rules						✓	✓
TNLPM							✓
Size Metric Rules							✓

6.5.2 Results of sampling

Sets of three sampling techniques were employed, namely SMOTE, SpreadSubsample and randomize, and their results were integrated using an ensemble concept. The main reason to use these techniques is to resolve the class imbalance problem in the dependent variable (change-proneness). First, SMOTE was applied to increase the number of instances in the minority class. Different percentages were set for SMOTE to present the amount of

oversampling, and these percentages depend on the category of difference between True and False values mentioned in Table 6.4. Table 6.10 provides the number of True and False values before and after applying SMOTE, along with their percentages.

Table 6.10: Results before and after applying SMOTE.

Dataset name	Before applying SMOTE					% of SMOTE	After applying SMOTE				
	# Classes	# True value	% True value	# False value	% False value		# Classes	# True value	% True value	# False value	% False value
antlr4	436	23	5.28%	413	94.72%	%100	459	46	10.02	413	89.98
junit	657	9	1.37%	648	98.63%	%200	675	27	4	648	96
MapDB	439	4	0.91%	435	99.09%	%300	451	16	3.55	435	96.45
mcMMO	301	4	1.33%	297	98.67%	%200	309	12	3.88	297	96.12
mct	2162	15	0.69%	2147	99.31%	%300	2207	60	2.72	2147	97.28
oryx	536	15	2.80%	521	97.20%	%100	551	30	5.44	521	94.56
titan	1486	13	0.87%	1473	99.13%	%300	1525	52	3.41	1473	96.59

The values in Table 6.10 indicate that after performing SMOTE, the classes in the dependent variable were still imbalanced. Although the classes in Table 6.10 were still imbalanced, the differences between the number of True and False values decreased compared with the differences in Table 6.4. In sequence, the SpreadSubsample technique was applied to decrease the number of instances in the majority class. The Spread value (default parameter) was changed from zero to 2.1, which indicates that the maximum ratio between the majority and minority classes is 35:65. The results of this technique are presented in Table 6.11.

Table 6.11: Results before and after applying SpreadSubsample.

Dataset name	Before applying SpreadSubsample					After applying SpreadSubsample				
	# Classes	# True value	% True value	# False value	% False value	# Classes	# True value	% True value	# False value	% False value
antlr4	459	46	10.02	413	89.98	142	46	32.39	96	67.61
junit	675	27	4.00	648	96.00	83	27	32.53	56	67.47
MapDB	451	16	3.55	435	96.45	49	16	32.65	33	67.35
mcMMO	309	12	3.88	297	96.12	37	12	32.43	25	67.57
mct	2207	60	2.72	2147	97.28	186	60	32.26	126	67.74
oryx	551	30	5.44	521	94.56	93	30	32.26	63	67.74
titan	1525	52	3.41	1473	96.59	161	52	32.30	109	67.70

Finally, applying SMOTE inserts additional classes (rows) at the end of each dataset, which leads to overfitting in ten-fold cross-validation. Thus, randomization was performed to rearrange the rows and avoid the overfitting problem.

6.5.3 Results of prediction models

In this section, the results of the empirical study are presented and analysed. Four prediction models involving three individual models (i.e., NB, SVM and KNN) and one ensemble model (RF) were employed on seven datasets. Each prediction model was constructed using four different datasets extracted from the four scenarios analysed (see Table 6.2). Therefore, the total number of prediction models was 112: 7 datasets \times 4 scenarios \times 4 prediction models.

Figure 6.3 shows the box plots of the AUC of each prediction model across the seven datasets for the four scenarios (see Table 6.2). The mean value of the AUC values is indicated by an “X”, the upper and lower lines of the box represent the first and third quartiles, and the middle horizontal line across the box represents the middle quartile. The prediction model which has the highest “X” values and the spread of the box is considered to be preferable. It is important to mention that because of the way that the scenarios were constructed, the test sets used for evaluating sampling methods in the third and fourth scenarios were different from those used for the non-sampling methods in the first and second scenarios. For this reason, the comparison between these two distinct groups of scenarios (first and second, versus third and fourth) has not been carried out as it would be invalid. It is observed in Figure 6.3 that RF attained the highest prediction accuracy in all four scenarios. Additionally, RF improved the prediction accuracy in the mct dataset over other datasets, reaching 0.86 and 0.82 values in the third and fourth scenarios, respectively. This finding further supports the concept that RF provides the best prediction accuracy for large datasets, as the mct dataset has a higher number of instances compared to other datasets [37] (2162 in the first and second scenarios and 186 in the third and fourth scenarios). Indeed, RF reaches good prediction accuracy with respect to the individual models.

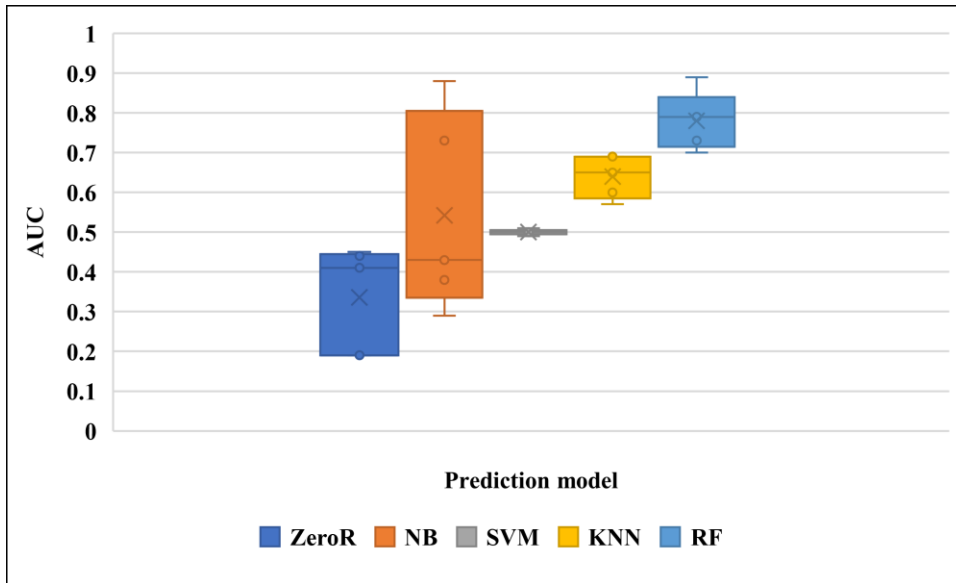


Figure 6.3.A: Box plot of the AUC values for prediction models on the first scenario.

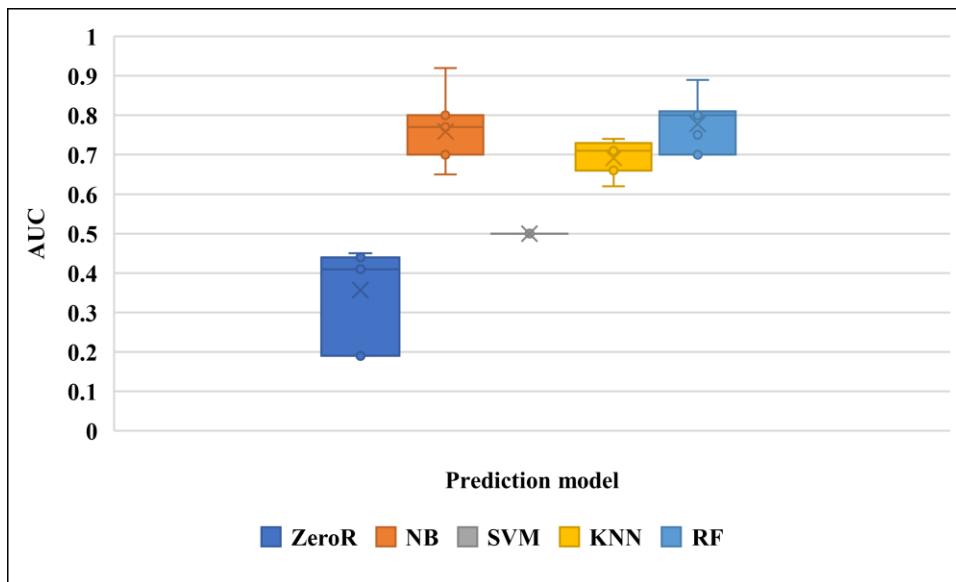


Figure 6.3.B: Box plot of the AUC values for prediction models on the second scenario.

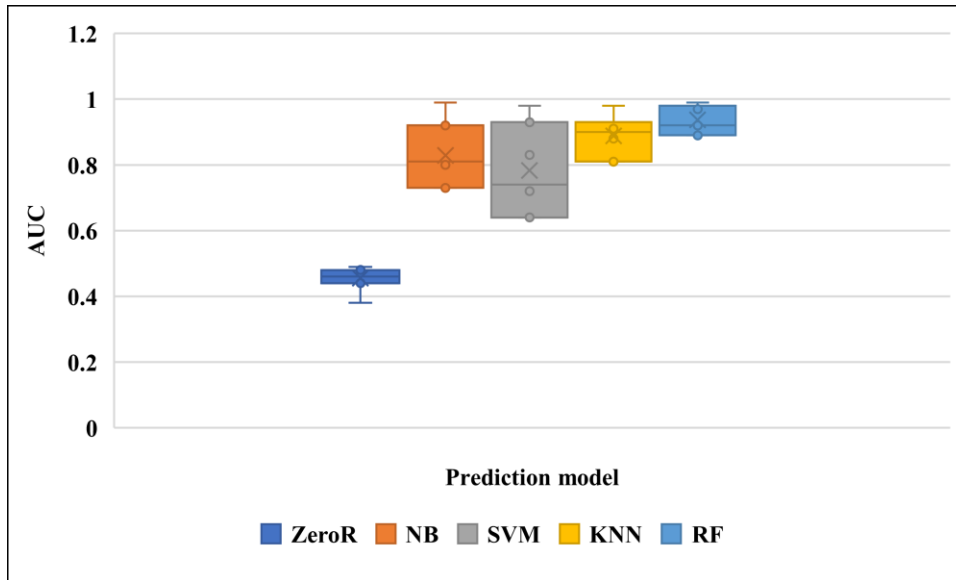


Figure 6.3.C: Box plot of the AUC values for prediction models on the third scenario.

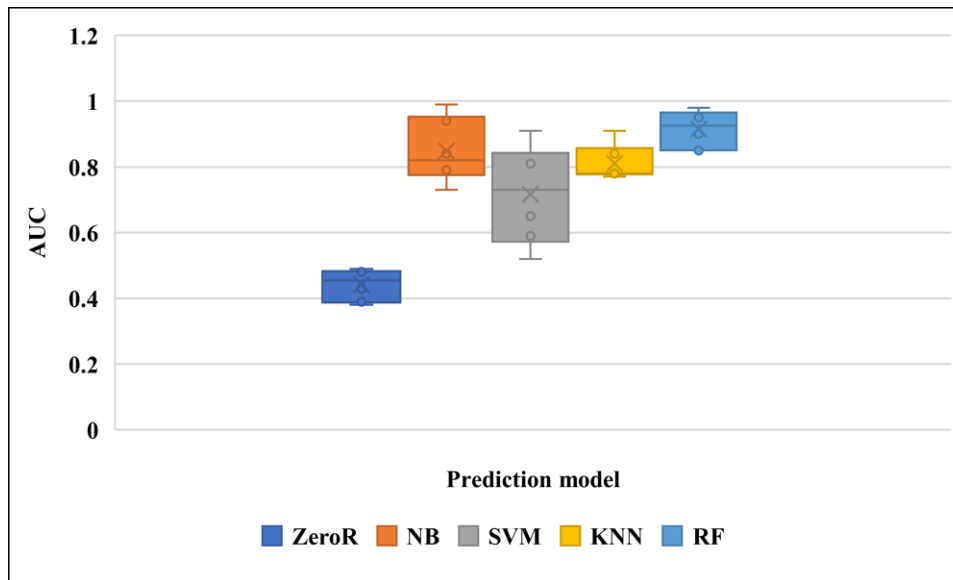


Figure 6.3.D: Box plot of the AUC values for prediction models on the fourth scenario.

Figure 6.3: Box plot of the AUC values for prediction models on the scenarios analysed.

Table 6.12, Table 6.13, Table 6.14 and Table 6.15 present the results of AUC for prediction models across seven datasets in the first, second, third and fourth scenarios, respectively. The prediction accuracy in terms of AUC measurement was evaluated, and the results of each scenario were presented separately considering the following aspects. **First**, the performance of the investigated prediction model was compared with the baseline, which is based on the dependent variable only (i.e., change-proneness) and predicts the mode value of this variable. **Second**, the best model in each dataset was identified (the highest AUC), and indicated by Boldface values (highlighted in light green) in the tables. **Third**, the best model in all datasets was identified and using Boldface with underlined values (highlighted in dark green in this table). **Finally**, the best model to predict change-proneness was determined.

A. Baseline

A baseline is provided in Table 6.12, Table 6.13, Table 6.14 and Table 6.15 for all scenarios. All the investigated models except NB in junit dataset in the first scenario (Table 6.12) achieved better prediction accuracy than the baseline. Consequently, these models have higher AUC values than those in the baseline model (i.e., ZeroR).

B. First scenario: datasets without FS or sampling

Table 6.12 provides the results of AUC values for prediction models in seven datasets in the first scenario. These results indicate that RF outperformed all other prediction models, as RF provided higher AUC values in all datasets except antlr4, in which NB achieved a slightly better prediction accuracy (only 0.03 higher). KNN achieved the best performance among individual models and was the second-best prediction model. From Table 6.12 the highest value of AUC in the first scenario was 0.89, obtained by the RF in the mct dataset.

Table 6.12: AUC values for performance evaluation of prediction models across seven datasets in the first scenario.

Models	Change-proneness dataset						
	antlr4	junit	MapDB	mcMMO	mct	oryx	titan
ZeroR	0.45	0.44	0.19	0.19	0.41	0.41	0.41
NB	0.73	0.43	0.38	0.29	0.88	0.58	0.74
SVM	0.51	0.5	0.5	0.49	0.5	0.5	0.5
KNN	0.57	0.6	0.69	0.65	0.69	0.65	0.7
RF	0.7	0.73	0.79	0.79	0.89	0.81	0.81
<p>Dark green: represents the best results in all datasets. Light green: represents the best results for each dataset.</p>							

C. Second scenario: datasets with FS and without sampling

Table 6.13 presents the AUC values for prediction models across seven datasets in the second scenario. This scenario presents the prediction accuracy using ensemble FS mentioned in

Section 6.4.5. Comparing the performance of the second scenario in Table 6.13 with that of the first scenario (without FS and sampling techniques) in Table 6.12, it is clear that FS improved the prediction accuracy in NB and KNN models except for one case (KNN in MapDB). In contrast, no impact was observed on SVM and RF compared to other prediction models. Additionally, FS produced either the same or an inferior performance to RF compared to the scenario without applying FS, except in antlr4 and junit. Although FS had no effect on RF, there was a clear competition between RF and NB to obtain the best prediction model in each dataset. Therefore, NB performed better than other individual models in terms of prediction accuracy, and achieved the best AUC value (0.92) in the mct dataset, which is considered outstanding according to the published criteria [169].

Table 6.13: AUC values for performance evaluation of prediction models across seven datasets in the second scenario.

Models	Change-proneness dataset						
	antlr4	junit	MapDB	mcMMO	mct	oryx	titan
ZeroR	0.45	0.44	0.19	0.19	0.41	0.41	0.41
NB	0.77	0.77	0.65	0.7	0.92	0.7	0.8
SVM	0.5	0.5	0.5	0.5	0.5	0.5	0.5
KNN	0.66	0.73	0.62	0.66	0.72	0.74	0.71
RF	0.81	0.75	0.7	0.7	0.89	0.8	0.8
<p>Dark green: represents the best results in all datasets. Light green: represents the best results for each dataset.</p>							

D. Third scenario: datasets without FS and with sampling

Table 6.14 shows the AUC values for prediction models across seven datasets in the third scenario. This scenario provides the prediction accuracy using ensemble sampling techniques mentioned in Section 6.4.5. From Table 6.14, it is evident that the AUC values were extremely high. This good performance was achieved because the datasets were modified with sampling techniques proposed in Section 6.5.2 without applying FS. However, some features were excluded after applying data analysis in Section 6.4.4. The most interesting finding from this scenario was that applying ensemble sampling techniques on the datasets that exclude improper features (i.e., features that have zero values and correlated with each other) in Section 6.4.4 is enough to reach a high prediction accuracy. The results of this scenario provide valuable insights into the positive influence of sampling techniques to improve the prediction accuracy of prediction models. However, the most evident result is that sampling techniques in the third scenario had a considerable impact on SVM, as seen in Table 6.14 had a great impact on SVM. The result of SVM in the first and second scenarios was 0.5, which indicates no discrimination according to the published criteria [169]. Furthermore, RF achieved the best

prediction accuracy in all datasets except in mcMMO, and the optimal AUC value (0.99) was obtained for the mct dataset. KNN was the second-best prediction model and outperformed other individual models.

Table 6.14: AUC values for performance evaluation of prediction models across seven datasets in the third scenario.

Models	Change-proneness dataset						
	antlr4	junit	MapDB	mcMMO	mct	oryx	titan
ZeroR	0.46	0.44	0.46	0.38	0.49	0.48	0.48
NB	0.73	0.80	0.73	0.99	0.92	0.82	0.81
SVM	0.64	0.74	0.64	0.98	0.93	0.83	0.72
KNN	0.81	0.88	0.81	0.91	0.98	0.9	0.93
RF	0.89	0.92	0.89	0.98	0.99	0.92	0.97
<p>Dark green: represents the best results in all datasets. Light green: represents the best results for each dataset.</p>							

E. Fourth scenario: datasets with both FS and sampling

Table 6.15 provides the results of AUC values for prediction models across seven datasets in the fourth scenario. This scenario lists the prediction accuracy using both the FS and sampling techniques mentioned in Section 6.4.5, and the method used in this scenario is mentioned in section 6.3. The main difference between the fourth and third scenarios is the use of different metrics, but they used the same sampling method. Overall, the results of the fourth scenario indicate a good prediction accuracy in most cases. However, the results in this scenario are worse than previous scenario. This suggests that applying both ensemble FS and sampling techniques decreased the prediction accuracy and using only sampling techniques was adequate to achieve high prediction accuracy. Again, RF outperformed the other prediction models in all datasets (except NB in mcMMO dataset) with AUC values ranging from 0.85 to 0.98, which is recognised as a good result. As in the first and second scenarios, KNN also was the second-best prediction model and performed better than other individual models.

Table 6.15: Performance of AUC for prediction models across seven datasets in the fourth scenario.

Models	Change-proneness dataset						
	antlr4	junit	MapDB	mcMMO	mct	oryx	titan
ZeroR	0.46	0.43	0.39	0.38	0.49	0.48	0.48
NB	0.51	0.79	0.84	0.99	0.94	0.73	0.8
SVM	0.51	0.65	0.82	0.91	0.81	0.52	0.59
KNN	0.71	0.78	0.78	0.84	0.91	0.78	0.77
RF	0.85	0.85	0.95	0.96	0.98	0.85	0.9
<p>Dark green: represents the best results in all datasets. Light green: represents the best results for each dataset.</p>							

Figure 6.4 illustrates the AUC results obtained from each prediction model across seven datasets in four scenarios, in which the higher AUC value indicates the better result. The comparison of the models indicates that the results of the third scenario provide valuable

insights into the positive influence of ensemble sampling techniques to improve the prediction accuracy of prediction models. The basic findings are consistent with research showing that the sampling techniques improved the overall performance [53, 206]. Regarding the overall results of the datasets, mcMMO and mct datasets achieved the highest prediction accuracy in both third and fourth scenarios.

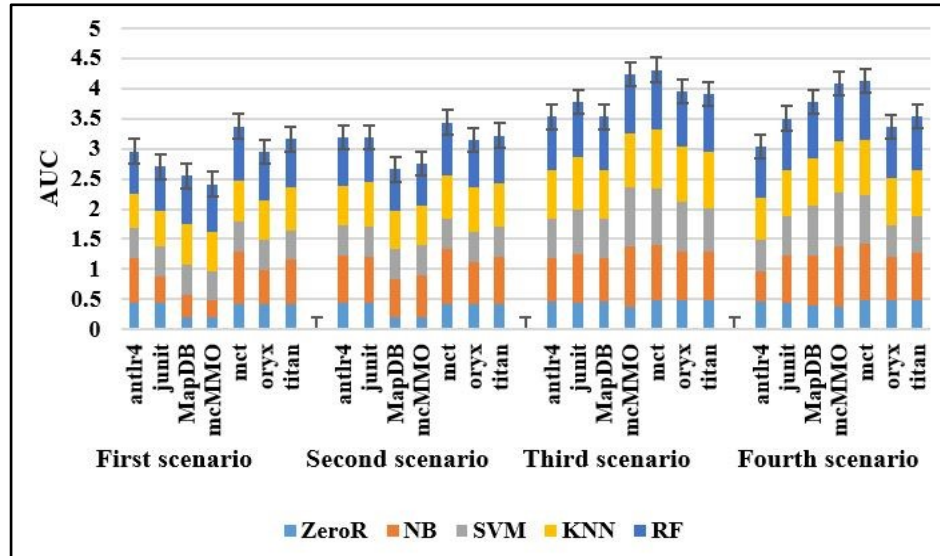


Figure 6.4: Ranking of the AUC values for prediction models on the scenarios analysed.

These findings are further supported by Figure C.1, Figure C.2, Figure C.3 and Figure C.4 in Appendix C, which provide graphs for multiple ROC curves for prediction models in the first, second, third and fourth scenarios, respectively. The highest curve, which is very close to 1, refers to the best results (e.g., RF in mct dataset in Figure C.3), whereas the lowest curve, which is very close to 0, refers to the worst results (e.g., ZeroR in mct dataset in Figure C.1).

6.5.4 Statistical tests of the third empirical study

ANOVA was applied to address RQ6.4 by comparing all prediction models across four scenarios using AUC. Factor A in ANOVA experiment is the prediction model (NB, SVM, KNN and RF). Table 6.16 shows one-way ANOVA results for prediction models using AUC. The significance level was defined as $\alpha = 0.05$ and the p-values in the tables were evaluated;

therefore, H_0 was rejected because all p-values in the tables were lower than 0.05. According to the standard classifications of Cohen proposed in Section 3.5.5, the results of eta-squared reveal that the effect size was large [180].

Table 6.16: One-way ANOVA results for prediction models using AUC.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-Value	Eta-Squared
Factor A	0.78	3.00	0.26	13.95	0.00	0.28
Error	2.02	108.00	0.02			
Total	2.80	111.00				

Additionally, multiple comparisons were performed using Tukey's confidence intervals [170] (see Figure 6.5). In the chart, it is possible to identify which pairs of Factor A (prediction models) significantly differ across scenarios. If a confidence interval does not include 0, then the pair is significantly different. The results obtained from Figure 6.5 indicates that there were significant differences between ensemble models (RF) and all individual models (NB, SVM and KNN). Similarly, there were significant differences between NB-SVM and SVM-KNN.

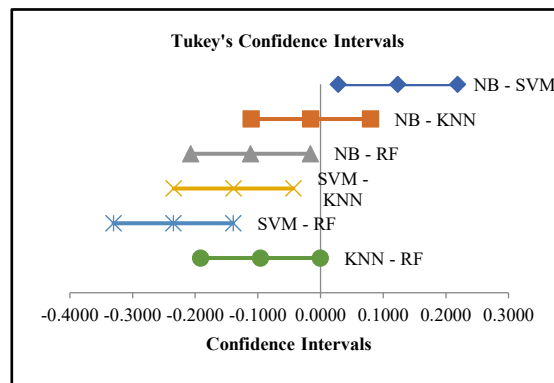


Figure 6.5: Multiple comparisons for prediction models using AUC.

6.5.5 Impact of parameter tuning for random forests.

The results of the parameters tuning of the individual models in Chapter 4 indicated that the default parameters outperformed the tuning parameters. Therefore, only the impact of the Mtry parameter tuning in RF using the grid search method proposed in Section 3.3.3 was explored. In Table 6.17, the performance of AUC for RF with default parameters was compared with that of Mtry parameter tuning. Boldface values in the table highlight the best results among each dataset in each scenario, whereas AUC-T refers to the AUC for parameter tuning and AUC-D refers to the AUC for default parameters. The comparison of the results indicated that

AUC-T outperformed AUC-D across all datasets in all scenarios, except mct dataset in the third scenario, which achieved the same result of AUC-D (0.99). AUC-T reached the optimal result (1.00) in the mcMMO dataset in the third scenario, and the average of AUC-T in all datasets in this scenario provided the highest prediction accuracy. Additionally, the grid search method provided different Mtry values for different datasets, which indicates that this method is an alternative to save time and effort instead of trying different parameters manually. The percentage of change between the average AUC-T and AUC-D across all datasets was 10.13%, 8.97%, 3.19% and 2.20% in the first, second, third and fourth scenarios, respectively. This indicates that tuning Mtry parameter in RF had a positive influence in each scenario. However, this influence was higher in the original datasets (e.g., without FS or sampling in the first scenario) and were lower in the edited datasets (e.g., with FS and sampling in the fourth scenario). Additionally, there is a good agreement between the findings in this section and those in the previous section, in which the third scenario achieved considerable performance. This scenario used metrics proposed in Table C.2, Table C.3, Table C.4, Table C.5, Table C.6, Table C.7 and Table C.8 in Appendix C without applying FS. The present findings seem to be consistent with prior research that used the same datasets to predict refactoring and found that using sampled instead of unsampled datasets improves the prediction accuracy [53, 206].

Table 6.17: AUC values for performance evaluation of RF with default and Mtry parameter tuning.

Scenario	First			Second			Third			Fourth		
	Mtry	AUC-T	AUC-D	Mtry	AUC-T	AUC-D	Mtry	AUC-T	AUC-D	Mtry	AUC-T	AUC-D
antlr4	3	0.81	0.7	2	0.84	0.81	6	0.94	0.89	1	0.87	0.85
junit	15	0.84	0.73	3	0.81	0.75	5	0.95	0.92	5	0.89	0.85
MapDB	2	0.90	0.79	10	0.72	0.7	8	0.96	0.89	2	0.98	0.95
mcMMO	11	0.89	0.79	2	0.89	0.7	6	1.00	0.98	5	0.98	0.96
mct	2	0.96	0.89	4	0.96	0.89	1	0.99	0.99	14	0.99	0.98
oryx	8	0.82	0.81	1	0.85	0.8	6	0.94	0.92	14	0.90	0.85
titan	8	0.90	0.81	6	0.85	0.8	15	0.99	0.97	2	0.93	0.9
Average	NA	0.87	0.79	NA	0.85	0.78	NA	0.97	0.94	NA	0.93	0.91
% of change	NA	10.13%		NA	8.97%		NA	3.19%		NA	2.20%	

6.5.6 Discussion and answers to research questions for the third empirical study

This section provides the discussion of the results presented above; and answers the RQs for the third empirical study.

RQ6.1) What is the impact of ensemble FS techniques on the performance of prediction models?

Ensemble FS techniques in the second scenario improved the prediction accuracy of NB and KNN models. Analysing the likely reasons behind this finding, firstly, NB is called naïve because it creates conditional assumption, which are independent from the features [238]. Second, using the Euclidean measure, KNN determines the closest neighbours, which are also independent from the features [114]. Therefore, these models may have performed well with FS techniques because they do not perform attribute selection [220].

However, ensemble FS techniques had no clear impact on the overall performance of SVM and RF. This result may be described by the fact that SVM algorithm includes the C parameter that selects the number of features, and the kernel function creates a suitable feature space [125]. Regarding the RF result, this occurs because RF has multiple decision trees that apply the same concept of FS using a top-down greedy search algorithm to choose the best feature at each step [239]. Therefore, SVM and RF algorithms already perform the FS concept during their creation. Additionally, RF includes many decision trees, which perform adequately with imbalanced dataset because they tend to build several tests to recognise the difference between the minority and majority classes [240]. For this reason, RF achieved the best prediction accuracy in the second scenario.

RQ6.2) What is the impact of ensemble sampling techniques on the performance of prediction models?

The machine learning models with ensemble sampling techniques achieved good performance. Again, RF achieved the best results of AUC values across all the datasets except mcMMO. These findings may help us to conclude that the performing ensemble sampling techniques and removing inappropriate features in the data analysis step without applying ensemble FS is sufficient to predict change proneness accurately. This is consistent with what has been found in a series of recent studies [53, 206] that used several types of sampling techniques: SMOTE [53], UPSAMPLE, SMOTE and RUSBoost [206], and emphasises the effect of these techniques to increase the performance of the prediction models.

Interestingly, ensemble sampling techniques in this particular case increased the prediction accuracy of SVM and RF. This supports the findings in the previous question indicating that SVM and RF algorithms perform FS during their creation. Consequently, these algorithms have a better response to sampling techniques compared to FS techniques.

RQ6.3) What is the impact of applying both ensemble FS and sampling techniques on the performance of prediction models?

Applying both ensemble FS and sampling techniques improved the performance of the prediction models, and RF achieved the best performance across all datasets except mcMMO. These findings corroborate the ideas of Kumar and Sureka, who used the same datasets to predict refactoring and performed principal component analysis and SMOTE techniques to extract the best features and resolve the imbalanced data problem [53]. Their results indicated that the prediction accuracy with the SMOTE technique was better than that without SMOTE, and the prediction accuracy of all metrics was better than that with FS. A possible explanation for these results may be the lack of adequate datasets, and the rank of the best ten features selected is considered low (the average ranking of best ten metrics ranges from 0.1 to 0.6), whereas the difference before and after applying sampling techniques is high (see Table 6.10 and Table 6.11).

RQ6.4) How effective are individual models and how do ensemble models perform when compared to the individual models in the context of predicting change-proneness?

KNN achieved the best prediction accuracy among most cases. RF outperformed other individual models and achieved the best result in terms of average AUC value (see Figure 6.4) across all scenarios. The results of the ANOVA test reveal that there were significant differences between RF and all individual models (see Figure 6.5). In addition, the results of the effect size were large (see Table 6.16). Therefore, applying ensemble sampling techniques on RF produced the highest accuracy to predict change-proneness. This finding is in accordance with previous studies, in which RF provided the best performance to predict change-proneness [47], software fault [37] and CHANGE metric [48].

RQ6.5) What is the impact of the Mtry parameter tuning in RF?

Mtry parameter tuning in RF using grid search method, along with RF, mlbench and caret packages improved prediction accuracy. This improvement increased in the original datasets (e.g., without FS or sampling) and decreased in the edited datasets (e.g., with FS and sampling in the fourth scenario). The findings of this RQ are consistent with those of Fernández-Delgado et al. who stated that RF created using caret package in R was the best model among 179 models applied on 121 original datasets [166]. Furthermore, the Mtry parameter differed from various datasets, and this difference was behind the study showing that there are no suggestions

to choose the specific number of the Mtry parameter [138]. In addition, this supports the use of automatic parameters tuning to simply and effectively improve performance [141, 142]. Based on these findings, it is recommended to tune Mtry parameter automatically to save time and efforts and improve the results. In future work, a statistical test will be used to investigate the performance difference between RF without and with parameter tuning.

6.6. Threats to Validity

The threats to validity usually are present in any experimental software engineering study that utilises open-source software projects [186]. This section describes the threats to validity that include four different types: external, conclusion, internal and constructed. Additionally, this section provides explanations on how they were solved.

6.6.1 External validity

Publicly available datasets extracted from open source software systems were used to enable reproducibility and comparison with other empirical studies that used the same datasets. Therefore, there is no threat in the datasets. However, these datasets were collected from Java systems, which restricts the generalisation of the findings to all programming languages (e.g., C++ and C#).

6.6.2 Conclusion validity

Conclusion validity relates to the statistical relationship between the results and the output of the experiment, which impacts on the capability to reach the right conclusion [187]. To avoid the threat of conclusion validity, ten-fold cross-validation was performed to reduce potentially biased results by selecting tests from the entire dataset. This validation was repeated ten times to generate statistically reliable results and avoid the conclusion threat. Finally, the conclusions were based on parametric statistical tests (i.e., ANOVA test), which is suitable for three or more groups (i.e., NB, KNN, SVM and RF). ANOVA requires some assumptions, such as normal distribution for the datasets and independent observations. However, these assumptions were met and one of the main advantages of the parametric statistical test over non-parametric statistical test is to provide more reliable results with both nonnormally and

continuous datasets. Hence, the threat to conclusion validity maybe exists due to using a parametric statistical test. This test also includes ten runs, which may have affected the results.

6.6.3 Internal validity

To prevent the threat of internal validity, the effectiveness of ensemble FS and sampling techniques employed to improve the accuracy of prediction models was explored. Furthermore, four scenarios were constructed to evaluate the performance of these techniques. Additionally, the four most frequently used models in Table 6.1 that were appropriate for classification problems were built. Weka, which is a well-known and common tool, was used to select features and build prediction models [120]. Therefore, there are no threats to internal validity.

6.6.4 Construct validity

In this study, 125 metrics manually validated and extracted from class-level were employed to capture several features of the software product [58]. Some of these metrics were eliminated before conducting empirical study (see Section 6.4.4) and some of them were removed by applying ensemble FS in Section 6.4.5. However, several of these metrics were used for the first time to predict change-proneness. Therefore, they present a threat to construct validity of these metrics. Furthermore, other validity concerns related to the dependent variable (i.e., change-proneness), which is a Boolean variable that reflects changes of refactorings (i.e., changes of the structure of the internal source code without affecting the functionality of source code [53]). As a result, the changes made in the systems may not be representative of all maintenance changes that could be made. The prediction of change-proneness variable has been investigated in several studies and it is considered as good indicator [16, 23, 47, 204, 241-247]. This study did not recognise the types of changes (i.e., adaptive, corrective, preventive or perfective). Therefore, this is also a threat to construct validity of the dependent variable. However, change-proneness is used as recommended by [203] because limited studies considered the types of the change proneness [47]. Regarding parameters tuning, only the grid search was applied to tune the Mtry parameter in RF. However, the prediction accuracy of RF may increase by tuning other parameters, such as the number of trees to grow. Additional

studies to more completely investigate the key tenets of other parameters in RF are required, along with tuning of parameters on other models (i.e., SVM, KNN and NB).

6.7. Conclusion of the third empirical study

Ensemble FS and sampling techniques can improve the prediction accuracy of machine learning models. However, the application of these techniques on software maintainability is limited. In this chapter, three individual models (NB, SVM and KNN) and one ensemble model (RF) were applied on seven publicly available datasets. The effectiveness of ensemble FS (i.e., Relief and Pearson's correlation coefficient) and ensemble sampling techniques (i.e., SMOTE, SpreadSubsample and randomize) on the performance of prediction change-proneness was evaluated and compared.

This chapter presents several insights based on comprehensive experimentation and analyses:

- The results obtained from this chapter provide evidence of the positive impact of ensemble FS in improving the performance of the prediction models (KNN and NB) that are FS method independent. Nevertheless, ensemble FS techniques had no clear effect on the overall performance of SVM and RF because these models have FS techniques as a part of the model's creation;
- A considerable improvement in performance was achieved by applying ensemble sampling methods on all prediction models, and there was a clear improvement in SVM and RF;
- Across all scenarios, the ensemble model (RF) achieved the best performance in predicting change-proneness compared to other models and there were significant differences between RF and all individual models. In addition, the effect size was large. A possible explanation of the good performance of RF in both high dimensional and imbalanced datasets is that RF performs FS through the creation of several decision trees;
- The experimental results in this empirical study presented that the performance of the ensemble models for predicting change-proneness was significantly improved and the effect size was large in all prediction models. In contrast to earlier findings in the first

and second empirical studies, where there were no significant differences between individual and ensemble models except few cases in heterogeneous ensemble models in the second empirical. A possible explanation is that this empirical study used larger datasets than those used in the first and second empirical studies;

- The Mtry parameter tuning in RF improved the performance compared to the use of the RF default parameters. The observed increase in the prediction accuracy of RF after applying parameter tuning is because the grid search selects the best value of the Mtry parameter (which determines the number of features randomly sampled at each split) in each dataset that provides the highest prediction accuracy. The results observed in this study reflect those of a previous study that examined the effect of caret package and found that RF was the best prediction model among 179 models analysed [166].

Chapter 7. Conclusion and Contributions

This chapter concludes this thesis and summarises the research results of the previous chapters. This chapter provides the answers to the RQs proposed in Chapter 1, describes the key contributions and recommendations for practitioners, and presents limitations and some directions and opportunities for future work.

7.1. Conclusion of the thesis

Chapter 2 systematically reviews 56 studies in 35 journals and 21 conference proceedings related to the prediction of maintainability of OO software systems using machine learning techniques. The review uses the standard SLR method applied to the most common computer science digital database libraries from January 1991 to July 2018. In the process of reviewing these studies, the fundamental research gaps and directions of research were determined in the methodology section to formalise three empirical studies in Chapters 4, 5, and 6. Thus, this thesis compared and evaluated the effectiveness of homogeneous (i.e., bagging, additive regression and RF) and heterogeneous (i.e., stacking and APE) ensemble models and sets of individual models (i.e., RT, MLP, M5Rules, KNN, SVR, SVM and NB) for predicting software maintainability of OO systems. These models were applied in various public datasets extracted from class level and collected after several years of OO system maintenance. To validate the investigated models in these empirical studies, statistical tests and effect size measurements were performed. In addition, the impact of parameter tuning was explored using caret package, Auto-WEKA and grid search in Chapters 4, 5 and 6, respectively.

The main aim of all empirical studies conducted was to improve the prediction accuracy and achieve more consistent results in the prediction of software maintainability in OO systems by applying ensemble models on different datasets and using several base models, as the core idea of the ensemble models is to improve the prediction accuracy over individual models. The creation of a highly accurate prediction of software maintainability was challenging because the relationships between software quality attributes and their metrics are often complicated, nonlinear and lead to a reduction in the accuracy of prediction models [10]. In contrast, the

key to predict software maintainability is the determination of software maintenance measurements, which are notoriously difficult to capture because of the problems in estimating the effort associated with the maintenance task.

The overall empirical results in this thesis indicate that ensemble models produced better prediction accuracy than most of the individual models; however, the results are different in various datasets and the base model. In most cases, KNN or SVR recorded the highest prediction accuracy compared to other individual models; moreover, these models as base models in bagging and additive regression achieved the best prediction accuracy, along with RF.

The following are the most important findings from each empirical study:

In the first empirical study:

- There were no significant differences between the ensemble and individual models and the effect sizes were small;
- KNN as the individual model or as the base model in additive regression ensemble model attained the best performance across all investigated models;
- The parameter tuning increased the prediction accuracy of ensemble models and did not increase the prediction accuracy of the individual models in most cases.

In the second empirical study:

- No significant differences have been found between the ensemble and individual models except few cases in the heterogeneous ensemble models and the effect sizes were small;
- SVR and KNN as an individual model, or sometimes as a base model in bagging and additive regression achieved the best accuracy to predict software maintainability, followed by APE;
- The selected models by Auto-WEKA tool performed better than the best model prediction in study 5.A except KNN and SVR in Eclipse JDT Core and Lucene, respectively.

In the third empirical study:

- RF performed significantly better than other individual models as well as the effect size was large;
- RF achieved the best prediction accuracy across all scenarios;

- Tuning the Mtry parameter in RF outperformed using the default parameters of RF.

7.2. Answers to Research Questions

This section answers RQs predetermined in the first chapter using evidence collected from the empirical studies.

RQ1) How effective are individual models at predicting software maintainability?

Answer: The performance of the individual models differed for each dataset. In most cases, KNN achieved the best prediction accuracy among individual models, followed by SVR. Also, SVR and M5Rules in some cases achieved the best performance in addition to KNN, whereas KNN and NB recorded the second-best prediction accuracy in some cases. Therefore, the overall results of individual models indicate that KNN was better than other individual models in most cases, followed by SVR.

RQ2) How do ensemble models perform in the context of predicting change maintenance efforts using well-established datasets when compared to the individual models?

Answer: Although ensemble models improved the prediction accuracy over all individual models except for a few cases, the differences were not statistically significant, and the effect sizes were small. KNN as the individual model or as the base model in additive regression attained the best prediction accuracy compared to all investigated models. Regarding parameter tuning, these parameters increased the prediction accuracy of the ensemble models and did not increase the prediction accuracy of the individual models in most cases. KNN as the base model in additive regression in the parameter tuning provided the best prediction accuracy.

RQ3) How do ensemble models perform in the context of predicting change maintenance efforts using more recent and larger datasets when compared to the individual models?

Answer: There were no significant differences between individual and ensemble models, except for a few cases in the heterogeneous ensemble models, and the effect sizes were small. The homogeneous ensemble models increased the prediction accuracy over most of the individual models, in which SVR as an individual model or a base model in bagging or additive regression outperformed all other prediction models. However, the heterogeneous ensemble models had a considerable impact on RT and a minor or no impact on KNN and SVR. Regarding parameter tuning, all the selected models using the Auto-WEKA tool performed

better than the best model prediction in the empirical study except KNN and SVR in Eclipse JDT Core and Lucene, respectively.

RQ4) How do ensemble models perform in the context of predicting change-proneness using the newest and largest datasets when compared to the individual models?

Answer: The results indicate that RF outperformed other individual models and obtained the highest value of the average of AUC. In addition, there were significant differences between individual and ensemble models, and the effect size was large. With respect to parameter tuning, the Mtry parameter tuning in RF increased the performance when using the default parameters of RF.

7.3. Contribution

This thesis provides knowledge and empirical evidence in both the machine learning and software maintainability fields. The fundamental contributions are as follows:

- Based on the findings obtained from the SLR, there is relatively little activity in the area of software maintainability prediction compared with other software quality attributes. The CHANGE metric was the most commonly used software measurement (dependent variable) employed in the selected primary studies, and most of them used class-level product metrics as the independent variables. Several private datasets were used in the selected studies, and there is a considerable need to publicly publish datasets. Most studies focused on regression problems and performed k-fold cross-validation. Although ensemble models used in selected primary studies improved the prediction accuracy over individual prediction models, their application is relatively rare compared with the individual prediction models applied in the majority of studies;
- Among several types of software maintenance measurements, the CHANGE metric, MI and change proneness are considered indirect measures and can be used as dependent variables to capture the element of maintainability. This thesis used the most common measurement performed in the selected primary studies (i.e., CHANGE metric), along with the rarest measurement (i.e., change proneness);
- Class level product metrics (i.e., L&H, C&K and other OO source-code metrics) used in this thesis as independent variables emerged as reliable and powerful predictors for

software maintainability (dependent variable). Additionally, these metrics are directly calculated on different parts of the software systems and can be used as early predictors to reduce cost, utilise resources and control future maintenance efforts;

- Although there are few public datasets for software maintainability, preprocessing techniques were applied to public software quality datasets (i.e., bug prediction datasets [57] and refactoring dataset [58]) to produce new versions of these datasets appropriate for software maintainability prediction;
- The findings of ensemble models, in general, show that the proposed ensemble models yield improved prediction accuracy over most of the individual models; moreover, the improvement was significant and the effect size was large only in Chapter 6, in which a larger number and size of the datasets was used;
- The proposed ensemble models in Chapters 4 and 5 were also found to be useful in predicting software maintainability. Also, there were no significant differences between the individual and ensemble models, and the effect sizes were small except for a few cases in Chapter 5;
- Although the results of the investigation of ensemble models in the three empirical studies show that these models are effective in predicting software maintainability and increasing the prediction accuracy over most of the individual models, in some cases, neither homogeneous nor heterogeneous ensemble models improved the prediction accuracy over SVR and KNN;
- The prediction accuracy of homogeneous ensemble models outperformed the heterogeneous ensemble models in some datasets, and the opposite occurred in other datasets. Moreover, additive regression exhibited better prediction accuracy compared to the bagging ensemble model in Chapter 4 and the opposite occurred in Chapter 5. KNN and SVR as an individual model or a base model in bagging or additive regression achieved the best prediction accuracy in Chapters 4 and 5, respectively, whereas RF achieved the best prediction accuracy in Chapter 6. Therefore, the prediction accuracy of ensemble models is different in various datasets and varies with the base model;
- KNN as a base model in additive regression in Chapters 4 and 5 produced the same result as the individual KNN models. Furthermore, KNN produced the highest

prediction accuracy compared to individual models in this thesis in most cases, and other prediction models that were implemented by selected previous studies for the QUES dataset in Chapter 4. In addition, for the QUES dataset, this model is the only model that nearly fulfils the criteria of accurate prediction proposed in Chapter 3. Additionally, KNN as the base model in the additive regression with tuning parameters achieved the best prediction accuracy and reached the optimal result (0) in terms of MMRE and MAE values;

- In Chapter 4, the parameter tuning using the caret package in most cases had a positive impact on the ensemble models and a negative impact on the individual models. In addition, the performance of the ensemble models with respect to the parameter tuning outperformed that of the individual models. In Chapter 5, the models selected by the Auto-WEKA tool performed better than the best model prediction in the second empirical study except for KNN and SVR in the Eclipse JDT Core and Lucene datasets, respectively. In Chapter 6, the Mtry parameter tuning in RF using grid search improved the prediction accuracy in all scenarios. However, the positive impact of this tuning was higher in the original datasets (e.g., without FS or sampling in the first scenario) and lower in the edited datasets (e.g., with FS and sampling in the fourth scenario);
- The results of the method used in Chapter 6 reveal that the ensemble FS and sampling techniques yield improved prediction accuracy over most of the investigated models. To the best of the author's knowledge, the implementation of these techniques has never been documented before in the area of software maintainability.

7.4. Recommendations for Practitioners

This section presents different critical results and provides various recommendations on the adequate use of the ensemble models to predict software maintainability as follows:

Size of the datasets: The first empirical study in Chapter 4 used relatively limited datasets, whereas the second empirical study in Chapter 5 used more recent and larger datasets. The results of the statistical tests in these studies showed no significant differences between individual and ensemble models, and the effect size was small except for a few cases in the heterogeneous ensemble models in the second empirical study. In contrast, the third empirical study in Chapter 6 employed the largest datasets compared with the datasets used in the

previous empirical studies, and the results indicate that there were significant differences between individual and ensemble models, and the effect size was large. Therefore, the size of the datasets should be large enough to create adequate ensemble models.

Base model: Most ensemble models investigated did not improve strong models (e.g., KNN and SVR). Hence, the ensemble models are more useful in improving the performance of weaker base models.

Parameter tuning: The parameter tuning of the ensemble models using the caret package in Chapter 4, along with grid search in Chapter 6, improved the prediction accuracy of the ensemble models and achieved better performance than the individual models. Consequently, parameter tuning must be performed on the ensemble models to improve the prediction accuracy.

One of the primary concerns of software practitioners is to improve software maintainability [248]. Predicting software maintainability accurately is a fundamental requirement for practitioners to save time, cost and effort of the maintenance of software. Therefore, the previous findings have produced several important recommendations for practitioners as follow:

1. The selection of the prediction model plays an important role to produce high prediction accuracy of software maintainability. It is recommended to select strong individual models (e.g., KNN or SVM) for regression problem and RF for classification problem.
2. Data pre-processing techniques are essential in improving prediction accuracy. However, the choice of these techniques depends on the nature of the datasets. Due to this fact, I recommend that datasets be investigated carefully and then suitable pre-processing techniques selected depending on the nature and characteristics of the dataset, such as resampling for imbalanced datasets or FS for high dimensionality datasets.
3. Consideration of parameter tuning is recommended to increase the performance of machine learning models.

7.5. Limitations and Possible Future Work

Several future work directions can be implemented to overcome the limitations in this thesis, as follows:

More software maintainability measurements: A fundamental ingredient of software maintainability prediction is the identification of the dependent variable to be predicted. A limitation of this thesis is that it only predicted the CHANGE metric and change proneness. Although the prediction of the number of changes in a certain class (CHANGE metric) is harder than predicting whether or not a change has been made in the class (change proneness), it provides more precise information [249]. To strike a balance between ease of prediction and more precise information, possible future research could be to predict a ranking or categorisation of software maintainability (change proneness) into small, medium, and high.

Exhaustive exploration of the parameter space: A limitation of this thesis is that it only evaluated a limited proportion of the parameter space. Researchers can address this limitation by doing a systematic extensive exploration of parameter tuning. This exploration will help to figure out to what extent the parameter space affects the prediction accuracy of machine learning models in software maintainability prediction. Also, it will help to identify which machine learning models are more sensitive to parameter tuning.

Additional datasets: The most important limitation of this thesis is that it used a limited set of public datasets. This limitation can be addressed by creating and publishing additional datasets that can be used by researchers. For example, datasets can be extracted from open-source software systems using various tools such as Analyst4j, CCCC and C&K Java Metrics as proposed in [250]. Moreover, other datasets can be created from real-world systems in industry.

Different OO programming languages: Another limitation related to the datasets is that the datasets used in this thesis were extracted from systems written in Ada (i.e., QUES and UIMS datasets [9]), or Java (i.e., bug prediction datasets [57] and refactoring dataset [58]). Therefore, further research needs to be done using datasets extracted from different OO programming languages, such as C++, JavaScript, C#, PHP and Python (even though some of these are not “pure” OO languages). This will enable the exploration of additional features and characteristics of software systems.

Additional ensemble models: A weakness of this thesis is that it only evaluated and compared the application of three homogenous (i.e., bagging, additive regression and RF), and two heterogeneous ensemble models (i.e., stacking and APE). Additional research is required to investigate other homogeneous and heterogeneous ensemble models (e.g., boosting, voting and linear ensemble) with additional base models (e.g., radial basis function network, linear regression and logistic regression).

Improved prediction accuracy: A major limitation of this thesis is that only a few models meet the prediction accuracy criteria proposed in Section 3.5.4. More work needs to be done to improve prediction accuracy using resampling or FS techniques and also looking at the application of other homogenous or heterogeneous ensemble models are mentioned above.

References

- [1] H. Alsolai, "Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques," in *International Conference on Software Maintenance and Evolution*, 2018, pp. 716-721.
- [2] H. Alsolai and M. Roper, "Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability," in *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019, pp. 370-373.
- [3] H. Alsolai and M. Roper, "Determining the Best Prediction Accuracy of Software Maintainability Models Using Auto-WEKA," in *International Conference on Computing*, 2019, pp. 60-70.
- [4] H. Alsolai and M. Roper, "A Systematic Review of Feature Selection Techniques in Software Quality Prediction," in *International Conference on Electrical and Computing Technologies and Applications*, 2019, pp. 1-5.
- [5] H. Alsolai and M. Roper, "A Systematic Literature Review of Machine Learning Techniques for Software Maintainability Prediction," *Information and Software Technology*, vol. 119, p. 106214, 2020.
- [6] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675-689, 1999.
- [7] M. A. Ahmed and H. A. Al-Jamimi, "Machine Learning Approaches for Predicting Software Maintainability: A Fuzzy-Based Transparent Model," *IET Software*, vol. 7, no. 6, pp. 317-326, 2013.
- [8] *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12, 1990, p. 84.
- [9] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *The Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, 1993.
- [10] M. M. T. Thwin and T.-S. Quah, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics," *Journal of Systems and Software*, vol. 76, no. 2, pp. 147-156, 2005.
- [11] C. van Koten and A. R. Gray, "An Application of Bayesian Network for Predicting Object-Oriented Software Maintainability," *Information and Software Technology*, vol. 48, no. 1, pp. 59-67, 2006.
- [12] Y. Zhou and H. Leung, "Predicting Object-Oriented Software Maintainability Using Multivariate Adaptive Regression Splines," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1349-1361, 2007.
- [13] M. O. Elish and K. O. Elish, "Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study," in *European Conference on Software Maintenance and Reengineering*, 2009, pp. 69-78.
- [14] S. K. Dubey, A. Rana, and Y. Dash, "Maintainability Prediction of Object-Oriented Software System by Multilayer Perceptron Model," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 5, pp. 1-4, 2012.

- [15] R. Malhotra and A. Chug, "Application of Group Method of Data Handling Model for Software Maintainability Prediction Using Object Oriented Systems," *International Journal of System Assurance Engineering and Management*, vol. 5, no. 2, pp. 165-173, 2014.
- [16] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three Empirical Studies on Predicting Software Maintainability Using Ensemble Methods," *Soft Computing*, vol. 19, no. 9, pp. 2511-2524, 2015.
- [17] L. Kumar and S. K. Rath, "Hybrid Functional Link Artificial Neural Network Approach for Predicting Maintainability of Object-Oriented Software," *Journal of Systems and Software*, vol. 121, pp. 170-190, 2016.
- [18] L. Kumar and S. K. Rath, "Software Maintainability Prediction Using Hybrid Neural Network and Fuzzy Logic Approach with Parallel Computing Concept," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 1487-1502, 2017.
- [19] F. Fioravanti and P. Nesi, "Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1062-1084, 2001.
- [20] A. De Lucia, E. Pompella, and S. Stefanucci, "Assessing Effort Estimation Models for Corrective Maintenance Through Empirical Studies," *Information and Software Technology*, vol. 47, no. 1, pp. 3-15, 2005.
- [21] S. C. Misra, "Modeling Design/Coding Factors That Drive Maintainability of Software Systems," *Software Quality Journal*, vol. 13, no. 3, pp. 297-320, 2005.
- [22] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics," *IEEE Transactions on Software Engineering*, vol. 29, no. 1, pp. 77-87, 2003.
- [23] R. Malhotra and M. Khanna, "Particle Swarm Optimization-Based Ensemble Learning for Software Change Prediction," *Information and Software Technology*, vol. 102, pp. 65-84, 2018.
- [24] C. Jones, "The Economics of Software Maintenance in the Twenty First Century," *Performance Engineering to Enhance the Maintenance*, pp. 1-19, 2006.
- [25] T. DeMarco, *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall, 1986, p. 304.
- [26] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.
- [27] "Github - The Largest Open Source Community in The World." <https://github.com/> (accessed 2017).
- [28] "SourceForge -The Complete Open-Source Software Platform." <http://www.sourceforge.net> (accessed 2017).
- [29] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution-The Nineties View," in *International Software Metrics Symposium*, 5-7 Nov. 1997 1997, pp. 20-32.
- [30] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33-53, 2007.
- [31] M. Riaz, E. Mendes, and E. Tempero, "A Systematic Review of Software Maintainability Prediction and Metrics," in *International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 367-377.

- [32] Y. Zhou and B. Xu, "Predicting The Maintainability of Open Source Software Using Design Metrics," *Wuhan University Journal of Natural Sciences*, vol. 13, no. 1, pp. 14-20, 2008.
- [33] J.-C. Chen and S.-J. Huang, "An Empirical Analysis of The Impact of Software Development Problem Factors on Software Maintainability," *Journal of Systems and Software*, vol. 82, no. 6, pp. 981-992, 2009.
- [34] S. G. MacDonell, "Establishing Relationships Between Specification Size and Software Process Effort in CASE Environments," *Information and Software Technology*, vol. 39, no. 1, pp. 35-45, 1997.
- [35] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, "What Accuracy Statistics Really Measure," *IEE Proceedings - Software*, vol. 148, no. 3, pp. 81-85, 2001.
- [36] N. Ueda and R. Nakano, "Generalization Error of Ensemble Estimators," in *Proceedings of International Conference on Neural Networks 1996*, vol. 1, pp. 90-95
- [37] C. Catal and B. Diri, "Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem," *Information Sciences*, vol. 179, no. 8, pp. 1040-1058, 2009.
- [38] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *Journal of Information & Computational Science*, vol. 8, no. 16, pp. 4241-4254, 2011.
- [39] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, 2008.
- [40] A. T. Mısırlı, A. B. Bener, and B. Turhan, "An Industrial Case Study of Classifier Ensembles for Locating Software Defects," *Software Quality Journal*, vol. 19, no. 3, pp. 515-536, 2011.
- [41] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-Project Defect Prediction Models: L'union fait la force," in *Software Maintenance, Reengineering and Reverse Engineering*, 2014, pp. 164-173.
- [42] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An Empirical Study of Classifier Combination for Cross-Project Defect Prediction," in *Annual Computer Software and Applications Conference*, 2015, pp. 264-269.
- [43] J. Petri, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "Building an Ensemble for Software Defect Prediction Based on Diversity Selection," in *International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1-10.
- [44] D. D. Nucci, F. Palomba, R. Oliveto, and A. D. Lucia, "Dynamic Selection of Classifiers in Bug Prediction: An Adaptive Method," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 202-212, 2017.
- [45] L. L. Minku and X. Yao, "Ensembles and Locality: Insight on Improving Software Effort Estimation," *Information and Software Technology*, vol. 55, no. 8, pp. 1512-1528, 2013.
- [46] M. Azzeh, A. B. Nassif, and L. L. Minku, "An Empirical Evaluation of Ensemble Adjustment Methods for Analogy-Based Effort Estimation," *Journal of Systems and Software*, vol. 103, pp. 36-52, 2015.
- [47] G. Catolino and F. Ferrucci, "An extensive evaluation of ensemble techniques for software change prediction," *Journal of Software: Evolution and Process*, vol. 31, no. 9, pp. 1-15, 2019.

- [48] A. Kaur, K. Kaur, and K. Pathak, "Software Maintainability Prediction by Data Mining of Software Code Metrics," in *International Conference on Data Mining and Intelligent Computing*, 2014, pp. 1-6.
- [49] K. J. Ezawa, M. Singh, and S. W. Norton, "Learning Goal Oriented Bayesian Networks for Telecommunications Risk Management," in *International Conference on Machine Learning*, 1996, pp. 139-147.
- [50] O. Loyola-González, M. García-Borroto, M. A. Medina-Pérez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and G. De Ita, "An Empirical Study of Oversampling and Undersampling Methods for Lcmine An Emerging Pattern Based Classifier," in *Mexican Conference on Pattern Recognition*, 2013, pp. 264-273.
- [51] K.-J. Wang, B. Makond, K.-H. Chen, and K.-M. Wang, "A Hybrid Classifier Combining SMOTE with PSO to Estimate 5-Year Survivability of Breast Cancer Patients," *Applied Soft Computing*, vol. 20, pp. 15-24, 2014.
- [52] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive Learning Algorithms and Representations for Text Categorization," in *International Conference on Information and knowledge management*, 1998, pp. 148-155.
- [53] L. Kumar and A. Sureka, "Application of LSSVM and SMOTE on Seven Open Source Projects for Predicting Refactoring at Class Level," in *Asia-Pacific Software Engineering Conference*, 2017, pp. 90-99.
- [54] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction," in *International Conference on Tools with Artificial Intelligence*, 2010, vol. 1, pp. 137-144.
- [55] R. Malhotra and K. Lata, "An empirical study on predictability of software maintainability using imbalanced data," *Software Quality Journal*, 2020, doi: 10.1007/s11219-020-09525-y.
- [56] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, *The promise repository of empirical software engineering data*. 2012.
- [57] M. D. Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *IEEE Working Conference on Mining Software Repositories*, 2-3 May 2010 2010, pp. 31-41.
- [58] P. Hegedűs, I. Kádár, R. Ferenc, and T. Gyimóthy, "Empirical Evaluation of Software Maintainability Based on a Manually Validated Refactoring Dataset," *Information and Software Technology*, vol. 95, pp. 313-327, 2018.
- [59] I. Iso, "Iec25010: 2011 Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (Square)—System and Software Quality Models," *International Organization for Standardization*, vol. 34, p. 2910, 2011.
- [60] M. Dagpinar and J. H. Jahnke, "Predicting Maintainability with Object-Oriented Metrics -An Empirical Comparison," in *Working Conference on Reverse Engineering*, 2003, pp. 155-164.
- [61] P. Oman and J. Hagemester, "Metrics for Assessing A Software System's Maintainability," in *Proceedings Conference on Software Maintenance*, 1992, pp. 337-344.
- [62] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using Metrics to Evaluate Software System Maintainability," *Computer*, vol. 27, no. 8, pp. 44-49, 1994.
- [63] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and Application of an Automated Source Code Maintainability Index," *Journal of Software Maintenance: Research and Practice*, vol. 9, no. 3, pp. 127-159, 1997.

- [64] M. Genero, M. Piattini, E. Manso, and G. Cantone, "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics," in *International Workshop on Enterprise Networking and Computing in Healthcare Industry*, 2004, pp. 263-275.
- [65] C. Sammut and G. Webb, *Encyclopedia of Machine Learning*. Springer Science & Business Media, 2011, p. 1031.
- [66] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *International Joint Conference on Artificial Intelligence*, 1995, vol. 14, no. 2, pp. 1137-1143.
- [67] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., 1986, p. 396.
- [68] F. Mosteller and J. W. Tukey, *Data analysis and regression: a second course in statistics*. Pearson, 1977, p. 588.
- [69] T. Fawcett, "An Introduction to ROC Analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861-874, 2006.
- [70] A. P. Bradley, "The Use of The Area Under The ROC Curve in The Evaluation of Machine Learning Algorithms," *Pattern recognition*, vol. 30, no. 7, pp. 1145-1159, 1997.
- [71] M. Jorgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 674-681, 1995.
- [72] E. Mendes and B. Kitchenham, "Further Comparison of Cross-Company and Within-Company Effort Estimation Models for Web Applications," in *International Symposium on Software Metrics*, 2004, pp. 348-357.
- [73] C. Jin and J.-A. Liu, "Applications of Support Vector Machine and Unsupervised Learning for Predicting Maintainability Using Object-Oriented Metrics," in *International Conference on Multimedia and Information Technology*, 2010, vol. 1, pp. 24-27.
- [74] F. Ye, X. Zhu, and Y. Wang, "A New Software Maintainability Evaluation Model Based on Multiple Classifiers Combination," in *International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, 2013, pp. 1588-1591.
- [75] A. Shafiabady, M. N. r. Mahrin, and M. Samadi, "Investigation of Software Maintainability Prediction Models," in *International Conference on Advanced Communication Technology*, 2016, pp. 783-786.
- [76] R. Kumar and N. Dhanda, "Maintainability Quantification of Object Oriented Design: A Revisit," *International Journal*, vol. 4, no. 12, pp. 461-466, 2014.
- [77] G. Tiwari and A. Sharma, "Maintainability Techniques for Software Development Approaches—A Systematic Survey," *IJCA Special Issue on Issues and Challenges in Networking, Intelligence and Computing Technologies*, vol. ICNICT, no. 4, pp. 28-31, 2012.
- [78] R. Burrows, A. Garcia, and F. Taïani, "Coupling Metrics for Aspect-Oriented Programming: A Systematic Review of Maintainability Studies," in *International Conference on Evaluation of Novel Approaches to Software Engineering*, 2008, pp. 277-290.
- [79] M. Riaz, "Maintainability Prediction of Relational Database-Driven Applications: A Systematic Review," in *International Conference on Evaluation & Assessment in Software Engineering*, 2012, pp. 263-272.

- [80] B. Kitchenham, "Procedures for Performing Systematic Reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1-26, 2004.
- [81] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571-583, 2007.
- [82] R. Malhotra, "A Systematic Review of Machine Learning Techniques for Software Fault Prediction," *Applied Soft Computing*, vol. 27, pp. 504-518, 2015.
- [83] B. R. Reddy and A. Ojha, "Performance of Maintainability Index Prediction Models: A Feature Selection Based Study," *Evolving Systems*, vol. 10, no. 2, pp. 179-204, 2019.
- [84] A. Liberati *et al.*, "The PRISMA Statement for Reporting Systematic Reviews and Meta-Analyses of Studies that Evaluate Healthcare Interventions: Explanation and Elaboration," *Annals of internal medicine*, vol. 151, no. 4, pp. 65-94, 2009.
- [85] D. Zhang and J. J. Tsai, "Machine Learning and Software Engineering," *Software Quality Journal*, vol. 11, no. 2, pp. 87-119, 2003.
- [86] I. Kádár, P. Hegedus, R. Ferenc, and T. Gyimóthy, "A Code Refactoring Dataset and Its Assessment Regarding Software Maintainability," in *International Conference on Software Analysis, Evolution, and Reengineering* 2016, vol. 1, pp. 599-603.
- [87] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," in *Conference proceedings on Object-Oriented Programming Systems, Languages, and Applications*, 1991, pp. 197-211.
- [88] H. Aljamaan, M. O. Elish, and I. Ahmad, "An Ensemble of Computational Intelligence Models for Software Maintenance Effort Prediction," in *International Work-Conference on Artificial Neural Networks*, 2013, pp. 592-603.
- [89] R. Land, "Measurements of Software Maintainability," in *Proceedings of ARTES Graduate Student Conference*, 2002, pp. 1-7.
- [90] P. Oman and J. Hagemester, "Construction and Testing of Polynomials Predicting Software Maintainability," *Journal of Systems and Software*, vol. 24, no. 3, pp. 251-266, 1994.
- [91] K. K. Aggarwal, Y. Singh, and J. K. Chhabra, "An Integrated Measure of Software Maintainability," in *Annual Reliability and Maintainability Symposium*, 2002, pp. 235-241.
- [92] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier, 1977, p. 127.
- [93] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976.
- [94] L. C. Briand, S. Morasca, and V. R. Basili, "Measuring and Assessing Maintainability at the end of High Level Design," in *Conference on Software Maintenance*, 1993, pp. 88-87.
- [95] M. Polo, M. Piattini, and F. Ruiz, "Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study," in *International Conference on Software Maintenance*, 2001, pp. 202-208.
- [96] M. I. Sarwar, W. Tanveer, I. Sarwar, and W. Mahmood, "A Comparative Study of MI tools: Defining the Roadmap to MI Tools Standardization," in *International Multitopic Conference*, 2008, pp. 379-385.
- [97] M. Genero, J. Olivas, M. Piattini, and F. Romero, "Using Metrics to Predict OO Information Systems Maintainability," in *International Conference on Advanced Information Systems Engineering*, 2001, pp. 388-401.

- [98] V. Nguyen, B. Boehm, and P. Danphitsanuphan, "A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks," *Information and Software Technology*, vol. 53, no. 6, pp. 682-691, 2011.
- [99] M. Shepperd and S. MacDonell, "Evaluating Prediction Systems in Software Project Estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820-827, 2012.
- [100] T. Menzies and M. Shepperd, "Special Issue on Repeatable Results in Software Engineering Prediction," *Empirical Software Engineering*, vol. 17, pp. 1-17, 2012.
- [101] "Coefficient of Determination." https://en.wikipedia.org/wiki/Coefficient_of_determination (accessed 2019).
- [102] S. Basri, N. Kama, H. M. Sarkan, S. Adli, and F. Haneem, "An Algorithmic-Based Change Effort Estimation Model for Software Development," in *Asia-Pacific Software Engineering Conference*, 2016, pp. 177-184.
- [103] S. K. Sehra, Y. S. Brar, N. Kaur, and G. Kaur, "Optimization of COCOMO Parameters Using TLBO Algorithm," *International Journal of Computational Intelligence Research*, vol. 13, no. 4, pp. 525-535, 2017.
- [104] A. Srivastava, S. Singh, and S. Q. Abbas, "Performance Measure of the Proposed Cost Estimation Model: Advance Use Case Point Method," in *Soft Computing: Theories and Applications*, 2019, pp. 223-233.
- [105] M. O. Elish, T. Helmy, and M. I. Hussain, "Empirical Study of Homogeneous and Heterogeneous Ensemble Models for Software Development Effort Estimation," *Mathematical Problems in Engineering*, vol. 2013, pp. 1-21, 2013.
- [106] N. Raj Kiran and V. Ravi, "Software Reliability Prediction by Soft Computing Techniques," *Journal of Systems and Software*, vol. 81, no. 4, pp. 576-583, 2008.
- [107] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169-198, 1999.
- [108] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Empirical Strategies," in *Experimentation in Software Engineering*, 2012, pp. 9-36.
- [109] B. Cukic, "Guest Editor's Introduction: The Promise of Public Software Engineering Data Repositories," *IEEE Software*, vol. 22, no. 6, pp. 20-22, 2005.
- [110] X. Wu *et al.*, "Top 10 Algorithms in Data Mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1-37, 2008.
- [111] R.-H. Li and G. G. Belford, "Instability of Decision Tree Classification Algorithms," in *International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 570-575.
- [112] W.-Y. Loh, "Classification and Regression Trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14-23, 2011.
- [113] J. Tang, C. Deng, and G. B. Huang, "Extreme Learning Machine for Multilayer Perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809-821, 2016.
- [114] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37-66, 1991.
- [115] G. Holmes, M. Hall, and E. Prank, "Generating Rule Sets from Model Trees," in *Australasian Joint Conference on Artificial Intelligence*, 1999, pp. 1-12.
- [116] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to the SMO algorithm for SVM regression," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1188-1193, 2000.

- [117] M. Awad and R. Khanna, "Support Vector Regression," in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, 2015, pp. 67-80.
- [118] J. Brownlee, *Machine Learning Mastery with Weka*. Ebook, 2019, p. 248.
- [119] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in *Proceedings of the conference on Uncertainty in artificial intelligence*, 1995, pp. 338-345.
- [120] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham, "Weka: Practical Machine Learning Tools and Techniques with Java Implementations," *Hamilton, New Zealand: University of Waikato, Department of Computer Science.*, 1999.
- [121] M. P. Basgalupp, R. C. Barros, and D. D. Ruiz, "Predicting Software Maintenance Effort Through Evolutionary-Based Decision Trees," in *Annual ACM Symposium on Applied Computing*, 2012, pp. 1209-1214.
- [122] S. Džeroski and B. Ženko, "Is Combining Classifiers with Stacking Better than Selecting the Best One?," *Machine Learning*, vol. 54, no. 3, pp. 255-273, 2004.
- [123] J. B. Bradley, *Neural networks: A comprehensive foundation*. Prentice Hall, 1994, p. 842.
- [124] J. R. Quinlan, "Learning with continuous classes," in *Australian joint conference on artificial intelligence*, 1992, vol. 92, pp. 343-348.
- [125] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2011, p. 744.
- [126] Z. Zhi-Qiang, Y. Hong-Bin, X. Hua-Rong, X. Yan-Qi, and G. Ji, "Fast Training Support Vector Machines Using Parallel Sequential Minimal Optimization," in *International Conference on Intelligent System and Knowledge Engineering*, 2008, vol. 1, pp. 997-1001.
- [127] B. Seijo-Pardo, I. Porto-Díaz, V. Bolón-Canedo, and A. Alonso-Betanzos, "Ensemble Feature Selection: Homogeneous and Heterogeneous Approaches," *Knowledge-Based Systems*, vol. 118, pp. 124-139, 2017.
- [128] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [129] J. H. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367-378, 2002.
- [130] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241-259, 1992.
- [131] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software Defect Prediction Using Ensemble Learning on Selected Features," *Information and Software Technology*, vol. 58, pp. 388-402, 2015.
- [132] L. Breiman, "Random Forests," *Machine learning*, vol. 45, no. 1, pp. 5-32, 2001.
- [133] M. Skurichina and R. P. Duin, "Bagging for Linear Classifiers," *Pattern Recognition*, vol. 31, no. 7, pp. 909-930, 1998.
- [134] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of statistics*, pp. 1189-1232, 2001.
- [135] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: A Statistical View of Boosting (With Discussion and A Rejoinder by the Authors)," *The Annals of Statistics*, vol. 28, no. 2, pp. 337-407, 2000.

- [136] Y. Freund and R. E. Schapire, "Experiments With A New Boosting Algorithm," in *International Conference on International Conference on Machine Learning*, 1996, pp. 148–156.
- [137] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10-18, 2009.
- [138] E. Scornet, "Tuning Parameters in Random Forests," *ESAIM: Proceedings and Surveys*, vol. 60, pp. 144-162, 2017.
- [139] K. M. Ting and I. H. Witten, "Issues in Stacked Generalization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 271-289, 1999.
- [140] S. Amari, *The Handbook of Brain Theory and Neural Networks*. MIT press, 2003.
- [141] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated Parameter Optimization of Classification Techniques for Defect Prediction Models," in *International Conference on Software Engineering 2016*, pp. 321-332.
- [142] W. Fu, T. Menzies, and X. Shen, "Tuning for Software Analytics: Is it Really Necessary?," *Information and Software Technology*, vol. 76, pp. 135-146, 2016.
- [143] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, "Exploiting the Essential Assumptions of Analogy-Based Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 425-438, 2011.
- [144] M. Kuhn, "Caret: Classification and Regression Training," *Astrophysics Source Code Library*, p. ascl: 1505.003, 2015.
- [145] J. Brownlee. "Caret R Package for Applied Predictive Modeling." <https://machinelearningmastery.com/caret-r-package-for-applied-predictive-modeling/> (accessed 2020).
- [146] "Model Training and Tuning." <https://topepo.github.io/caret/model-training-and-tuning.html> (accessed 2020).
- [147] M. Kuhn, "Building Predictive Models in R Using the Caret Package," *Journal of statistical software*, vol. 28, no. 5, pp. 1-26, 2008.
- [148] K. Hornik, C. Buchta, and A. Zeileis, "Open-Source Machine Learning: R Meets Weka," *Computational Statistics*, vol. 24, no. 2, pp. 225-232, 2009.
- [149] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826-830, 2017.
- [150] L. Kotthoff, C. Thornton, and F. Hutter, "User Guide for Auto-WEKA Version 2.3," *Dept. Comput. Sci., Univ. British Columbia, BETA lab, Vancouver, BC, Canada, Tech. Rep*, vol. 2, pp. 1-15, 2017.
- [151] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter Optimization to Improve Bug Prediction Accuracy," in *Workshop on Machine Learning Techniques for Software Quality Evaluation 2017*, pp. 33-38.
- [152] L. Kumar and S. Rath, "Predicting Object-Oriented Software Maintainability using Hybrid Neural Network with Parallel Computing Concept," in *India Software Engineering Conference, 2015 2015*, pp. 100-109.
- [153] M. D'Ambros, M. Lanza, and R. Robbes. "Bug Prediction Dataset." <http://bug.inf.usi.ch/index.php> (accessed 2018).
- [154] J. Yang and H. Qian, "Defect Prediction on Unlabeled Datasets by Using Unsupervised Clustering," in *International Conference on High Performance*

- Computing and Communications; International Conference on Smart City; International Conference on Data Science and Systems* 2016, pp. 465-472.
- [155] A. Boucher and M. Badri, "Using Software Metrics Thresholds to Predict Fault-Prone Classes in Object-Oriented Software," in *International Conference on Applied Computing and Information Technology, International Conference on Computational Science, Intelligence and Applied Informatics, International Conference on Big Data, Cloud Computing, Data Science and Engineering*, 2016, pp. 169-176.
- [156] "SourceMeter Static Code Analysis Tool." <https://www.sourcemeter.com/resources/java/> (accessed 2019).
- [157] B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan, "An Empirical Study of Maintenance and Development Estimation Accuracy," *Journal of Systems and Software*, vol. 64, no. 1, pp. 57-77, 2002.
- [158] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985-995, 2003.
- [159] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380-391, 2005.
- [160] M. Korte and D. Port, "Confidence in Software Cost Estimation Results Based on MMRE and PRED," in *International Workshop on Predictor Models in Software Engineering*, 2008, pp. 63-70.
- [161] L. Pickard, B. Kitchenham, and S. Linkman, "An Investigation of Analysis Techniques for Software Datasets," in *International Software Metrics Symposium*, 1999, pp. 130-142.
- [162] J. Al Dallal, "Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes," *Information and Software Technology*, vol. 55, no. 11, pp. 2028-2048, 2013.
- [163] "Simple guide to confusion matrix terminology." <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/> (accessed 2020).
- [164] S. B. Aher and L. Lobo, "Data Mining in Educational System Using Weka," in *International Conference on Emerging Technology Trends*, 2011, vol. 3, pp. 20-25.
- [165] A. Venkatesh and S. G. Jacob, "Prediction of Credit-Card Defaulters: A Comparative Study on Performance of Classifiers," *International Journal of Computer Applications*, vol. 145, no. 7, pp. 36-41, 2016.
- [166] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133-3181, 2014.
- [167] J. Bi and K. P. Bennett, "Regression Error Characteristic Curves," in *International Conference on Machine Learning*, 2003, pp. 43-50.
- [168] I. Myrtveit and E. Stensrud, "Validity and Reliability of Evaluation Procedures in Comparative Studies of Effort Prediction Models," *Empirical Software Engineering*, vol. 17, pp. 23-33, 2012.
- [169] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. John Wiley & Sons, 2013, p. 528.
- [170] M. L. Berenson, D. M. Levine, and M. Goldstein, *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice-Hall, Inc., 1983, p. 579.

- [171] D. Lakens, "Calculating and Reporting Effect Sizes to Facilitate Cumulative Science: A Practical Primer for T-Tests and ANOVAs," (in English), *Frontiers in Psychology*, vol. 4, no. 863, 2013.
- [172] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. 2013, p. 599.
- [173] N. Zumel and J. Mount, *Practical Data Science with R*. Manning Publications Co., 2014, p. 416.
- [174] R. Ihaka and R. Gentleman, "R: A Language for Data Analysis and Graphics," *Journal of Computational and Graphical Statistics*, vol. 5, no. 3, pp. 299-314, 1996.
- [175] L. Kumar, D. K. Naik, and S. K. Rath, "Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability," *Procedia Computer Science*, vol. 57, pp. 798-806, 2015.
- [176] C. W. Yohannese, T. Li, M. Simfukwe, and F. Khurshid, "Ensembles Based Combined Learning for Improved Software Fault Prediction: A Comparative Study," in *International Conference on Intelligent Systems and Knowledge Engineering 2017*, pp. 1-6.
- [177] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson Correlation Coefficient," in *Noise Reduction in Speech Processing*: Springer, 2009, pp. 1-4.
- [178] G. H. Chen and D. Shah, "Explaining the Success of Nearest Neighbor Methods in Prediction," *Foundations and Trends® in Machine Learning*, vol. 10, no. 5-6, pp. 337-588, 2018.
- [179] S.-j. Wang, A. Mathew, Y. Chen, L.-f. Xi, L. Ma, and J. Lee, "Empirical Analysis of Support Vector Machine Ensemble Classifiers," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6466-6476, 2009.
- [180] J. Cohen, "A Power Primer," *Psychological Bulletin*, vol. 112, no. 1, p. 155, 1992.
- [181] D. Soni. "Introduction to k-Nearest-Neighbors." <https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26> (accessed 2020).
- [182] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier 2016, p. 664.
- [183] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?," in *International conference on database theory*, 1999, pp. 217-235.
- [184] B. Caprile, S. Merler, C. Furlanello, and G. Jurman, "Exact Bagging with K-Nearest Neighbour Classifiers," in *International Workshop on Multiple Classifier Systems*, 2004, pp. 72-81.
- [185] S. I. Zahara, M. Ilyas, and T. Zia, "A Study of Comparative Analysis of Regression Algorithms for Reusability Evaluation of Object Oriented Based Software Components," in *International Conference on Open Source Systems and Technologies*, 2013, pp. 75-80, doi: 10.1109/ICOSST.2013.6720609.
- [186] H. K. Wright, M. Kim, and D. E. Perry, "Validity Concerns in Software Engineering Research," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, 2010, pp. 411-414.
- [187] T. M. Khoshgoftaar, N. Seliya, and N. Sundares, "An Empirical Study of Predicting Software Faults with Case-Based Reasoning," *Software Quality Journal*, vol. 14, no. 2, pp. 85-111, 2006.
- [188] G. J. Pai and J. B. Dugan, "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 675-686, 2007.

- [189] D. Pyle, *Data Preparation for Data Mining*. Morgan Kaufmann, 1999, p. 560.
- [190] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the NASA Software Defect Data Sets," in *International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1-5.
- [191] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction," in *Annual Conference on Evaluation & Assessment in Software Engineering*, 2011, pp. 96-103.
- [192] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA MDP Data Sets," *IET software*, vol. 6, no. 6, pp. 549-558, 2012.
- [193] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the Nasa Software Defect Datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208-1215, 2013.
- [194] B. Ghotra, S. McIntosh, and A. E. Hassan, "A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models," in *International Conference on Mining Software Repositories*, 2017, pp. 146-157.
- [195] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with Noise in Defect Prediction," in *International Conference on Software Engineering*, 2011, pp. 481-490.
- [196] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "Use of Genetic Algorithm for Software Maintainability Metrics' Conditioning," in *International Conference on Advanced Computing and Communications*, 2007, pp. 87-92.
- [197] "Quartile." <https://en.wikipedia.org/wiki/Quartile> (accessed 2020).
- [198] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the Relationships Between Design Measures and Software Quality in Object-Oriented Systems," *Journal of Systems and Software*, vol. 51, no. 3, pp. 245-273, 2000.
- [199] A. Boucher and M. Badri, "Predicting Fault-Prone Classes in Object-Oriented Software: An Adaptation of an Unsupervised Hybrid SOM Algorithm," in *International Conference on Software Quality, Reliability and Security*, 2017, pp. 306-317.
- [200] J. Brownlee. "How To Get Baseline Results And Why They Matter." <https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter/> (accessed 2017).
- [201] E. LeDell, "Scalable Ensemble Learning and Computationally Efficient Variance Estimation," UC Berkeley, 2015.
- [202] M. Graczyk, T. Lasota, B. Trawiński, and K. Trawiński, "Comparison of Bagging, Boosting and Stacking Ensembles Applied to Real Estate Appraisal," in *Asian Conference on Intelligent Information and Database Systems*, 2010, pp. 340-350.
- [203] A. G. Koru and J. Tian, "Comparing High-Change Modules and Modules with the Highest Measurement Values in Two Large-Scale Open-Source Products," *IEEE Transactions on Software Engineering*, vol. 31, no. 8, pp. 625-642, 2005.
- [204] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The Ability of Object-Oriented Metrics to Predict Change-Proneness: A Meta-Analysis," *Empirical software engineering*, vol. 17, no. 3, pp. 200-242, 2012.
- [205] V. Bolón-Canedo and A. Alonso-Betanzos, "Ensembles for Feature Selection: A Review and Future Trends," *Information Fusion*, vol. 52, pp. 1-12, 2019.
- [206] L. Kumar, S. M. Satapathy, and L. B. Murthy, "Method Level Refactoring Prediction on Five Open Source Java Projects using Machine Learning Techniques," in *India Software Engineering Conference*, 2019, pp. 1-10.

- [207] T. M. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A Comparative Study of Iterative and Non-Iterative Feature Selection Techniques for Software Defect Prediction," *Information Systems Frontiers*, vol. 16, no. 5, pp. 801-822, 2014.
- [208] Y. Liu, A. An, and X. Huang, "Boosting Prediction Accuracy on Imbalanced Datasets with SVM Ensembles," 2006, pp. 107-118.
- [209] L. Pelayo and S. Dick, "Applying Novel Resampling Strategies To Software Defect Prediction," in *Annual Meeting of the North American Fuzzy Information Processing Society*, 2007, pp. 69-72.
- [210] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to 'Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors''", *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637-640, 2007.
- [211] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434-443, 2013.
- [212] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online Defect Prediction for Imbalanced Data," in *International Conference on Software Engineering*, 2015, vol. 2, pp. 99-108.
- [213] N. Seliya, T. M. Khoshgoftaar, and J. V. Hulse, "Predicting Faults in High Assurance Software," in *International Symposium on High Assurance Systems Engineering*, 2010, pp. 26-34.
- [214] N. Gayatri, S. Nickolas, A. Reddy, S. Reddy, and A. Nickolas, "Feature Selection Using Decision Tree Induction in Class Level Metrics Dataset for Software Defect Predictions," in *Proceedings of the World Congress on Engineering and Computer Science*, 2010, vol. 1, pp. 124-129.
- [215] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting Fault Modules Applying Feature Selection to Classifiers," in *International Conference on Information Reuse and Integration*, 2007, pp. 667-672.
- [216] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, J. Aguilar-Ruiz, and M. Garre, "Attribute Selection in Software Engineering Datasets for Detecting Fault Modules," in *Conference on Software Engineering and Advanced Applications*, 2007, pp. 418-423.
- [217] K. Gao, T. M. Khoshgoftaar, and H. Wang, "An Empirical Investigation of Filter Attribute Selection Techniques for Software Quality Classification," in *International Conference on Information Reuse & Integration*, 2009, pp. 272-277.
- [218] T. M. Khoshgoftaar and K. Gao, "Feature Selection With Imbalanced Data for Software Defect Prediction," in *International Conference on Machine Learning and Applications*, 2009, pp. 235-240.
- [219] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction," in *International Conference on Tools with Artificial Intelligence*, 2010, vol. 1, pp. 137-144.
- [220] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing Software Metrics for Defect Prediction: An Investigation on Feature Selection Techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579-606, 2011.
- [221] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "Software Measurement Data Reduction Using Ensemble Techniques," *Neurocomputing*, vol. 92, pp. 124-132, 2012.
- [222] K. Gao, T. M. Khoshgoftaar, and R. Wald, "Combining Feature Selection and Ensemble Learning for Software Quality Estimation," in *International Florida Artificial Intelligence Research Society Conference*, 2014, pp. 47-52.

- [223] S. Agarwal and D. Tomar, "A Feature Selection Based Model for Software Defect Prediction," *International Journal of Advanced Science and Technology*, vol. 65, pp. 39-58, 2014.
- [224] A. Okutan and O. T. Yıldız, "Software Defect Prediction Using Bayesian Networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154-181, 2014.
- [225] G. W. Corder and D. I. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2014, p. 288.
- [226] W. Badr. "Why Feature Correlation Matters A Lot!" <https://towardsdatascience.com/why-feature-correlation-matters-a-lot-847e8ba439c4> (accessed 2020).
- [227] D. A. Bell and H. Wang, "A Formalism for Relevance and its Application in Feature Subset Selection," *Machine Learning*, vol. 41, no. 2, pp. 175-195, 2000.
- [228] C. H. Ooi, M. Chetty, and S. W. Teng, "Differential Prioritization in Feature Selection and Classifier Aggregation for Multiclass Microarray Datasets," *Data Mining and Knowledge Discovery*, vol. 14, no. 3, pp. 329-366, 2007.
- [229] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," in *Machine Learning Proceedings*: Elsevier, 1994, pp. 121-129.
- [230] T. N. Lal, O. Chapelle, J. Weston, and A. Elisseeff, "Embedded Methods," in *Feature Extraction*: Springer, 2006, pp. 137-165.
- [231] K. Kira and L. A. Rendell, "A Practical Approach to Feature Selection," in *Machine Learning Proceedings*: Morgan Kaufmann, 1992, pp. 249-256.
- [232] M. A. Hall, "Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning," in *International Conference on Machine Learning*, 2000, pp. 359-366.
- [233] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in *Data Mining and Knowledge Discovery Handbook*: Springer, 2009, pp. 875-886.
- [234] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.
- [235] "SMOTE." <https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/sMOTE> (accessed 14-03-2020).
- [236] M. Alghamdi, M. Al-Mallah, S. Keteyian, C. Brawner, J. Ehrman, and S. Sakr, "Predicting Diabetes Mellitus Using SMOTE and Ensemble Machine Learning Approach: The Henry Ford Exercise Testing (FIT) Project," *PloS one*, vol. 12, no. 7, 2017.
- [237] L. Trigg. "Class Randomize." <http://weka.sourceforge.net/doc.stable-3-8/weka/filters/unsupervised/instance/Randomize.html> (accessed 01/12/2019).
- [238] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in *Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 338-345.
- [239] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263-1284, 2009.
- [240] G. E. Batista, R. C. Prati, and M. C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20-29, 2004.

- [241] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 407-437, 2013.
- [242] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 566-571.
- [243] A. Peer and R. Malhotra, "Application of adaptive neuro-fuzzy inference system for predicting software change proneness," in *International Conference on Advances in Computing, Communications and Informatics 2013: IEEE*, pp. 2026-2031.
- [244] A. Peer and R. Malhotra, "Application of adaptive neuro-fuzzy inference system for predicting software change proneness," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2013: IEEE, pp. 2026-2031.
- [245] R. Malhotra and M. Khanna, "Inter Project Validation for Change Proneness Prediction using Object Oriented Metrics," *Software engineering: An International Journal*, vol. 3, no. 1, pp. 21-31, 2013.
- [246] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 4, pp. 273-286, 2013.
- [247] L. Kumar, S. K. Rath, and A. Sureka, "Empirical analysis on effectiveness of source code metrics for predicting change-proneness," in *Proceedings of the 10th Innovations in Software Engineering Conference*, 2017, pp. 4-14.
- [248] R. Malhotra and K. Lata, "An Empirical Study to Investigate The Impact of Data Resampling Techniques on The Performance of Class Maintainability Prediction Models," *Neurocomputing*, 2020, doi: <https://doi.org/10.1016/j.neucom.2020.01.120>.
- [249] L. L. Minku, "Which Machine Learning Method Do You Need?," in *Perspectives on Data Science for Software Engineering*: Elsevier, 2016, pp. 155-159.
- [250] L. R, L. J, and L. W, "Comparing Software Metrics Tools," in *International Symposium on Software Testing and Analysis*, 2008, pp. 131-142.
- [251] G. K. Gill and C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Transactions on Software Engineering*, vol. 17, no. 12, pp. 1284-1288, 1991.
- [252] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, "Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software," *Empirical Software Engineering*, vol. 1, no. 2, pp. 109-132, 1996.
- [253] J. C. Granja-Alvarez and M. J. Barranco-García, "A Method for Estimating Maintenance Cost in a Software Project: A Case Study," *Journal of Software Maintenance: Research and Practice*, vol. 9, no. 3, pp. 161-175, 1997.
- [254] Y. Lee and K. H. Chang, "Reusability and Maintainability Metrics for Object-Oriented Software," in *Proceedings of the Annual on Southeast Regional Conference*, 2000, pp. 88-94.
- [255] F. T. Sheldon, K. Jerath, and H. Chung, "Metrics for Maintainability of Class Inheritance Hierarchies," *Journal of Software Maintenance*, vol. 14, no. 3, pp. 147-160, 2002.
- [256] J. H. Hayes, S. C. Patel, and L. Zhao, "A Metrics-Based Software Maintenance Effort Model," in *European Conference on Software Maintenance and Reengineering*, 2004, pp. 254-258.

- [257] J. S. Lim, S. R. Jeong, and S. R. Schach, "An Empirical Investigation of the Impact of the Object-Oriented Paradigm on the Maintainability of Real-World Mission-Critical Software," *Journal of Systems and Software*, vol. 77, no. 2, pp. 131-138, 2005.
- [258] J. H. Hayes and L. Zhao, "Maintainability Prediction: a Regression Analysis of Measures of Evolving Systems," in *International Conference on Software Maintenance*, 2005, pp. 601-604.
- [259] K. Kaur and H. Singh, "Determination of Maintainability Index for Object Oriented Systems," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 2, pp. 1-6, 2011.
- [260] P. Bhattacharya and I. Neamtiu, "Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++," in *International Conference on Software Engineering*, 2011, pp. 171-180.
- [261] P. Singh, S. Singh, and J. Kaur, "Tool for Generating Code Metrics for C# Source Code Using Abstract Syntax Tree Technique," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 1-6, 2013.
- [262] R. Malhotra and A. Chug, "A Metric Suite for Predicting Software Maintainability in Data Intensive Applications," 2014, in *Transactions on Engineering Technologies*, pp. 161-175.
- [263] W. Zhang, L. Huang, V. Ng, and J. Ge, "SMPLearner: Learning to Predict Software Maintainability," *Automated Software Engineering*, vol. 22, no. 1, pp. 111-141, 2015.
- [264] A. Kaur, K. Kaur, and K. Pathak, "A Proposed New Model for Maintainability Index of Open Source Software," in *International Conference on Reliability, Infocom Technologies and Optimization*, 2014, pp. 1-6.
- [265] A. Jain, S. Tarwani, and A. Chug, "An Empirical Investigation of Evolutionary Algorithm for Software Maintainability Prediction," in *Students' Conference on Electrical, Electronics and Computer Science*, 2016, pp. 1-6.
- [266] S. Almgrin, W. Albattah, and A. Melton, "Using Indirect Coupling Metrics to Predict Package Maintainability and Testability," *Journal of Systems and Software*, vol. 121, pp. 298-310, 2016.
- [267] S. Levin and A. Yehudai, "Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes," in *International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 97-106.

Appendix A

This Appendix presents Table A.1, Table A.2 and Figure A.1, which belong to Chapter 2.

Table A.1: Selected primary studies.

Study ID	Ref	Topic	Author	Year	Place published	Publication name	Type
S1	[251]	Cyclomatic Complexity Density and Software Maintenance Productivity	Gill and Kemerer	1991	IEEE	IEEE Transactions on Software Engineering	Journal
S2	[9]	Object-Oriented Metrics that Predict Maintainability	Li and Henry	1993	Elsevier	Journal of Systems and Software	Journal
S3	[94]	Measuring and Assessing Maintainability at the End of High Level Design	Briand et al.	1993	IEEE	Conference on Software Maintenance	Conference
S4	[26]	A Metrics Suite for Object Oriented Design	Chidamber and Kemerer	1994	IEEE	IEEE Transactions on Software Engineering	Journal
S5	[90]	Construction and Testing of Polynomials Predicting Software Maintainability	Oman and Hagemester	1994	Elsevier	Journal of Systems and Software	Journal
S6	[62]	Using Metrics to Evaluate Software System Maintainability	Coleman et al.	1994	ACM DL	Computer	Journal
S7	[252]	Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software	Daly et al.	1996	Springer link	Empirical Software Engineering	Journal
S8	[63]	Development and Application of an Automated Source Code Maintainability Index	Welker et al.	1997	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S9	[253]	A Method for Estimating Maintenance Cost in a Software Project: A Case Study	Granja-Alvarez and Barranco-García	1997	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S10	[254]	Reusability and Maintainability Metrics for Object-Oriented Software	Lee and Chang	2000	ACM DL	38th Annual on Southeast Regional Conference	Conference
S11	[97]	Using Metrics to Predict Object-Oriented Information Systems Maintainability	Genero et al.	2001	ACM DL	13th International Conference on Advanced Information Systems Engineering	Conference
S12	[19]	Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems	Fioravanti and Nesi	2001	IEEE	IEEE Transactions on Software Engineering	Journal
S13	[95]	Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study	Polo et al.	2001	IEEE	Proceedings IEEE International Conference on Software Maintenance.	Conference
S14	[255]	Metrics for Maintainability of Class Inheritance Hierarchies	Sheldon et al.	2002	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S15	[91]	An Integrated Measure of Software Maintainability	Aggarwal et al.	2002	IEEE	Annual Reliability and Maintainability Symposium.	Conference
S16	[60]	Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison	Dagginar and Jahnke	2003	IEEE	10th Working Conference on Reverse Engineering.	Conference
S17	[22]	Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics	Bandi et al.	2003	IEEE	IEEE Transactions on Software Engineering	Journal
S18	[20]	Assessing Effort Estimation Models for Corrective Maintenance Through Empirical Studies	De Luciaa et al.	2004	Elsevier	Information and Software Technology	Journal
S19	[256]	A Metrics-Based Software Maintenance Effort Model	Hayes et al.	2004	IEEE	Eighth European Conference on Software Maintenance and Reengineering	Conference
S20	[10]	Application of Neural Networks for Software Quality Prediction using Object-Oriented Metrics	Thwin and Quah	2005	Elsevier	Journal of Systems and Software	Journal
S21	[21]	Modeling Design/Coding Factors That Drive Maintainability of Software Systems	Misra	2005	Springer link	Software Quality Journal	Journal
S22	[257]	An Empirical Investigation of the Impact of the Object-Oriented Paradigm on the	Lim et al.	2005	Elsevier	Journal of Systems and Software	Journal

		Maintainability of Real-World Mission-critical software					
S23	[109]	The Promise of Public Software Engineering Data Repositories	Cukic	2005	IEEE	IEEE Software	Journal
S24	[258]	Maintainability Prediction: A Regression Analysis of Measures of Evolving Systems	Hayes and Zhao	2005	IEEE	IEEE International Conference on Software Maintenance	Conference
S25	[11]	An Application of Bayesian Network for Predicting Object-Oriented Software Maintainability	Koten and Gray	2006	Elsevier	Information and Software Technology	Journal
S26	[12]	Predicting Object-Oriented Software Maintainability Using Multivariate Adaptive Regression Splines	Yuming Zhou and Hareton Leung	2007	Elsevier	Journal of Systems and Software	Journal
S27	[250]	Comparing Software Metrics Tools	Lincke et al.	2008	ACM DL	International Symposium on Software Testing and Analysis	Conference
S28	[96]	A Comparative Study of MI Tools: Defining the Roadmap to MI Tools Standardization	Sarwar et al.	2008	IEEE	IEEE International Multitopic Conference	Conference
S29	[32]	Predicting the Maintainability of Open Source Software Using Design Metrics	Yuming and Baowen	2008	Springer link	Wuhan University Journal of Natural Sciences	Journal
S30	[13]	Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study	Elish and Elish	2009	IEEE	Software Maintenance and Reengineering	Conference
S31	[33]	An Empirical Analysis of the Impact of Software Development Problem Factors on Software Maintainability	Chen and Huang	2009	Elsevier	Journal of Systems and Software	Journal
S32	[73]	Applications of Support Vector Machine and Unsupervised Learning for Predicting Maintainability Using Object-Oriented Metrics	Jin and Liu	2010	IEEE	Second International Conference on MultiMedia and Information Technology	Conference
S33	[259]	Determination of Maintainability Index for Object-Oriented Systems	Kaur and Singh	2011	ACM DL	ACM SIGSOFT Software Engineering Notes	Journal
S34	[98]	A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks	Nguyen et al.	2011	Elsevier	Information and Software Technology	Journal
S35	[260]	Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++	Bhattacharya and Neamtii	2011	ACM DL	33rd International Conference on Software Engineering	Conference
S36	[14]	Maintainability Prediction of Object-Oriented Software System by Multilayer Perceptron model	Dubey et al.	2012	ACM DL	ACM SIGSOFT Software Engineering Notes	Journal
S37	[121]	Predicting Software Maintenance Effort through Evolutionary-Based Decision Trees	Basgalupp et al.	2012	ACM DL	27th Annual ACM Symposium on Applied Computing	Conference
S38	[7]	Machine Learning Approaches for Predicting Software Maintainability: A Fuzzy-Based Transparent Model	Ahmed and Al-Jamimi	2013	IEEE	IET Software	Journal
S39	[162]	Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes	Al Dallal	2013	Elsevier	Information and Software Technology	Journal
S40	[74]	A New Software Maintainability Evaluation Model Based on Multiple Classifiers Combination	Ye et al.	2013	IEEE	International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering	Conference
S41	[261]	Tool for Generating Code Metrics for C# Source Code using Abstract Syntax Tree Technique	Singh et al.	2013	ACM DL	ACM SIGSOFT Software Engineering Notes	Journal
S42	[15]	Application of Group Method of Data Handling Model for Software Maintainability Prediction Using Object-Oriented Systems	Malhotra and Chug	2014	Springer link	International Journal of System Assurance Engineering and Management	Journal
S43	[48]	Software Maintainability Prediction by Data Mining of Software Code Metrics	Kaur et al.	2014	IEEE	International Conference on Data Mining and Intelligent Computing	Conference
S44	[262]	A Metric Suite for Predicting Software Maintainability in Data Intensive Applications	Malhotra and Chug	2014	Springer link	Transactions on Engineering Technologies	Conference
S45	[263]	SMPLeaRner: Learning to Predict Software Maintainability	Zhang et al.	2014	Springer link	Automated Software Engineering	Journal
S46	[175]	Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability	Kumara et al.	2015	Elsevier	Procedia Computer Science	Conference
S47	[16]	Three Empirical Studies on Predicting Software Maintainability Using Ensemble Methods	Elish et al.	2015	Springer link	Soft Computing	Journal

S48	[264]	A Proposed New Model for Maintainability Index of Open Source Software	Kaur et al.	2015	IEEE	International Conference on Reliability, Infocom Technologies and Optimization	Conference
S49	[17]	Hybrid Functional Link Artificial Neural Network Approach for Predicting Maintainability of Object-Oriented Software	Kumar and Rath	2016	Elsevier	Journal of Systems and Software	Journal
S50	[265]	An Empirical Investigation of Evolutionary Algorithm for Software Maintainability Prediction	Jain et al.	2016	IEEE	IEEE Students Conference on Electrical, Electronics and Computer Science	Conference
S51	[266]	Using Indirect Coupling Metrics to Predict Package Maintainability and Testability	Almugrin et al.	2016	Elsevier	Journal of Systems and Software	Journal
S52	[18]	Software Maintainability Prediction Using Hybrid Neural Network and Fuzzy Logic Approach with Parallel Computing Concept	Kumar and Rath	2017	Springer link	International Journal of System Assurance Engineering and Management	Journal
S53	[267]	Boosting Automatic Commit Classification into Maintenance Activities by Utilizing Source Code Changes	Levin and Yehudai	2017	ACM DL	13th International Conference on Predictive Models and Data Analytics in Software Engineering	Conference
S54	[83]	Performance of Maintainability Index Prediction Models: A Feature Selection Based Study	Reddy and Ojha	2017	Springer link	Evolving Systems	Journal
S55	[58]	Empirical Evaluation of Software Maintainability Based on a Manually Validated Refactoring Dataset	Péter Hegedus et al.	2018	Elsevier	Information and Software Technology	Journal
S56	[23]	Particle Swarm Optimization-Based Ensemble Learning for Software Change Prediction	Malhotraa and Khanna	2018	Elsevier	Information and Software Technology	Journal

Table A.2: Quality assessment result.

Study ID	QA 1	QA 2	QA 3	QA 4	QA 5	QA 6	QA 7	QA 8	QA 9	QA 10	QA 11	QA 12	QA 13	QA 14	QA 15	Total score	Rating
S1	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	P	N	P	Y	9	Fair
S2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	N	13.5	Excellent
S3	Y	Y	Y	Y	Y	Y	P	N	N	N	N	N	N	Y	P	8	Fair
S4	Y	N	N	Y	Y	Y	Y	Y	N	N	N	Y	N	Y	Y	9	Fair
S5	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	P	Y	Y	N	10	Good
S6	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	Y	Y	10	Good
S7	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	P	N	Y	Y	11.5	Good
S8	Y	Y	Y	Y	Y	Y	N	P	N	Y	N	P	P	Y	Y	10.5	Good
S9	Y	Y	Y	Y	P	N	Y	P	N	N	N	P	N	Y	P	8	Fair
S10	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	Y	Y	9	Fair
S11	Y	Y	Y	Y	Y	Y	Y	P	Y	N	N	Y	Y	Y	Y	12.5	Good
S12	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	P	Y	Y	12.5	Good
S13	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	P	Y	Y	13	Good
S14	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	Y	Y	6	Fair
S15	Y	Y	Y	Y	N	Y	N	N	N	N	N	P	P	Y	P	7.5	Fair
S16	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	P	P	Y	P	11.5	Good
S17	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	N	Y	Y	12	Good
S18	Y	Y	Y	Y	Y	Y	P	P	N	Y	Y	Y	P	Y	Y	12.5	Good
S19	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	P	P	Y	Y	10	Good
S20	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	N	12.5	Good
S21	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	N	P	P	Y	P	13	Good
S22	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N	N	Y	Y	8	Fair
S23	Y	N	N	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	N	8	Fair
S24	P	Y	Y	Y	Y	N	N	N	N	Y	N	P	P	Y	P	7.5	Fair
S25	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	13.5	Excellent
S26	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	14.5	Excellent
S27	Y	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	N	Y	Y	10.5	Good
S28	Y	Y	Y	Y	Y	N	Y	N	Y	P	N	N	N	Y	Y	8.5	Fair
S29	Y	Y	Y	Y	Y	N	Y	Y	P	Y	Y	Y	P	Y	N	12	Good
S30	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	14	Excellent
S31	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	Y	Y	Y	Y	12	Good
S32	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	P	P	N	11	Good
S33	Y	Y	Y	Y	Y	N	Y	Y	P	Y	N	Y	P	Y	N	11	Good
S34	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	14	Excellent
S35	Y	N	P	Y	Y	Y	Y	Y	P	Y	N	N	N	Y	Y	10	Good
S36	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	P	13	Good
S37	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	Y	N	12	Good
S38	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	12.5	Good
S39	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	P	P	Y	Y	13.5	Excellent
S40	N	Y	Y	Y	P	N	Y	Y	P	Y	Y	Y	Y	P	N	10.5	Good
S41	P	N	N	Y	Y	Y	Y	Y	P	N	N	N	N	N	P	6	Fair
S42	Y	N	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y	Y	Y	12	Good
S43	Y	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	Y	Y	P	11.5	Good
S44	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	14.5	Excellent
S45	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	14.5	Excellent
S46	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	P	P	12.5	Good
S47	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15	Good
S48	P	Y	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y	P	Y	12.5	Good
S49	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15	Excellent
S50	N	Y	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y	P	N	11	Good
S51	Y	Y	Y	Y	Y	Y	Y	N	P	Y	N	N	N	Y	Y	10.5	Good
S52	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	P	14	Excellent
S53	P	Y	Y	N	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	12	Good
S54	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	13.5	Excellent
S55	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	N	Y	Y	12.5	Good
S56	Y	Y	Y	Y	Y	Y	Y	N	P	Y	Y	Y	Y	Y	Y	13.5	Excellent

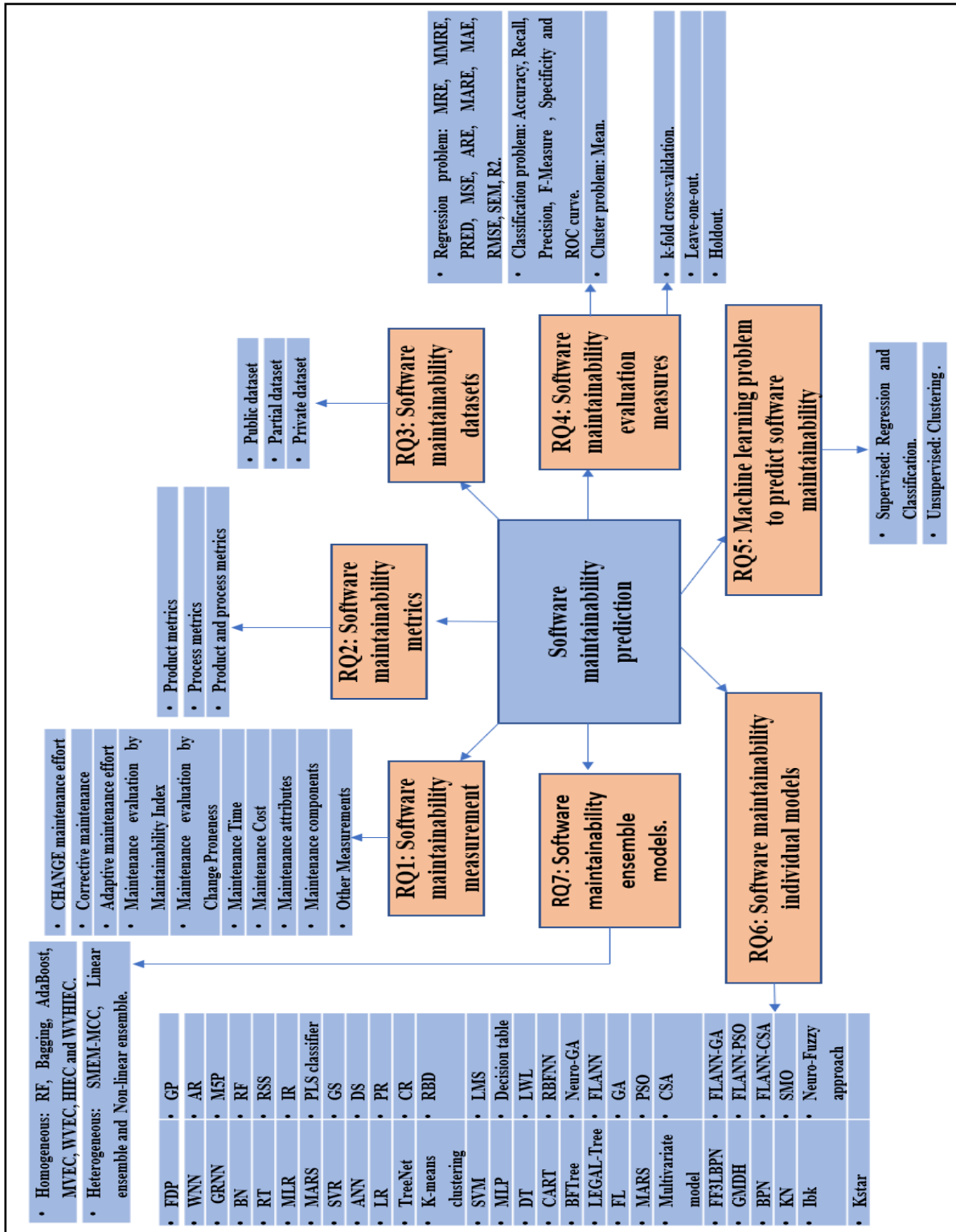


Figure A.1: The mind map of software maintainability prediction.

Appendix B

This Appendix provides Table B.1, which refers to Chapter 5.

Table B.1: Descriptive static of software maintainability prediction datasets.

Metrics	CBO	DIT	FanIn	FanOut	LCOM	NOC	NOA	NOAI	LOC	NOM	NOMI	NOPRA	NOPRM	NOPA	NOPM	RFC	WMC
Eclipse JDT Core																	
Min	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max	114	8	102	61	7381	26	313	388	2475	122	255	35	38	312	74	1026	489
Median	5	2	1	3	21	0	3	12	56	7	31	0	0	0	4	21	15
Mean	7.42	2.59	2.86	4.8	56.7	0.489	4.93	74.4	89.2	8.31	44.3 2	1.2 06	0.46 8	1.65	5.33	34.4	23.6
Stdev	8.35	1.58	5.84	5.43	287. 7	1.889	18.88	121. 3	130	7.26	42.6 2	2.4 98	1.80 7	12.0 4	4.98	50.8	30.8
Eclipse PDE UI																	
Min	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max	153	9	146	30	630	46	2169	158	724	36	523	15	21	2168	23	352	172
Median	5	2	1	3	15	0	1	0	41	6	8	1	0	0	3	20	10
Mean	7.42	2.12 2	2.55	4.97	33.7	0.556	4.9	3.12	62.47	7.03	24.6	1.8 68	1.11 3	2.3	4.51	29.9 7	15.0 4
Stdev	8.26	1.42 1	6.68	4.83	54.6	2.471	63.4	14	61.74	5.01	54.6	2.2 67	2.11 8	63	3.65	29.6 7	14.7 2
Equinox Framework																	
Min	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max	29	3	27	24	703	6	138	74	456	38	111	14	14	137	38	196	126
Median	5	1	1	3	6	0	2	0	30	4	9	1	0	0	3	16	9
Mean	6.53	1.22 5	2.1	4.65	36.4	0.163	4.61	1.36	61.6	6.34	14.0 8	2.1 2	0.74	1.46	4.38	27.9	16.6
Stdev	6.49	0.46 7	3.3	5.22	72.4	0.702	11.49	6.05	78.7	6.24	13.9 8	3.1 2	1.84	10.2 3	4.58	35	20.7
Lucene																	
Min	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max	179	5	174	16	1225	42	56	56	686	50	80	11	11	27	39	124	61
Median	5	2	1	3	6	0	2	0	31.5	4	16	0	0	0	3	13	7
Mean	7.3	1.82 3	3.82	3.52	17.9	0.67	3	1.09	45.8	5.05	20.1 8	1.4 34	0.31 8	0.76 5	3.85	19.2 8	10.3 5
Stdev	13.7	0.89 3	13.2	3.21	57.3	3	4.02	4.15	53.2	3.94	13.4 6	2.0 99	0.91	2.25 1	3.48	18.2 6	10.0 1
Mylyn																	
Min	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max	229	5	223	58	2485	49	128	18	1144	71	226	68	23	127	52	470	199
Median	4	1	1	2	6	0	2	0	31	4	9	1	0	0	3	13	7
Mean	5.52	1.44 1	2.19	3.4	20.9	0.319	3.53	0.55 1	46.1	5.21	13.2 1	1.9 76	0.49 5	0.91	4.17 5	19.8 4	10
Stdev	9.68	0.72 4	8.61	4	88.2	1.708	8.49	1.83 6	60	4.45	15.9 6	2.8 99	1.21 8	7.3	3.82 2	24.0 2	11.2 8

Appendix C

Appendix C presents the abbreviation of metrics used in chapter 6. Also, appendix C shows Table C.2, Table C.3, Table C.4, Table C.5, Table C.6, Table C.7 and Table C.8, which are descriptive static of the datasets used in Chapter 6. Further, this Appendix provides multiple ROC curves for datasets applied in Chapter 6.

Table C. 1: The abbreviation of metrics.

Metrics name	Abbreviation	Metrics name	Abbreviation
Lack of Cohesion in Methods 5	LCOM5	Number of Local Attributes	NLA
Nesting Level	NL	Number of Local Getters	NLG
Nesting Level Else-If	NLE	Number of Local Methods	NLM
Weighted Methods per Class	WMC	Number of Local Public Attributes	NLPA
Coupling Between Object classes	CBO	Number of Local Public Methods	NLPM
Coupling Between Object classes Inverse	CBOI	Number of Local Setters	NLS
Number of Incoming Invocations	NII	Number of Methods	NM
Number of Outgoing Invocations	NOI	Number of Public Attributes	NPA
Response set For Class	RFC	Number of Public Methods	NPM
API Documentation	AD	Number of Setters	NS
Comment Density	CD	Number of Statements	NOS
Comment Lines of Code	CLOC	Total Lines of Code	TLOC
Documentation Lines of Code	DLOC	Total Logical Lines of Code	TLLOC
Public Documented API	PDA	Total Number of Attributes	TNA
Public Undocumented API	PUA	Total Number of Getters	TNG
Total Comment Density	TCD	Total Number of Local Attributes	TNLA
Total Comment Lines of Code	TCLOC	Total Number of Local Getters	TNLG
Depth of Inheritance Tree	DIT	Total Number of Local Methods	TNLM
Number of Ancestors	NOA	Total Number of Local Public Attributes	TNLPA
Number of Children	NOC	Total Number of Local Public Methods	TNLPM
Number of Descendants	NOD	Total Number of Local Setters	TNLS
Number of Parents	NOP	Total Number of Methods	TNM
Lines of Code	LOC	Total Number of Public Attributes	TNPA
Logical Lines of Code	LLOC	Total Number of Public Methods	TNPM
Number of Attributes	NA	Total Number of Setters	TNS
Number of Getters	NG	Total Number of Statements	TNOS
Clone Classes	CCL	Clone Line Coverage	CLC
Clone Complexity	CCO	Clone Logical Line Coverage	CLLC
Clone Coverage	CC	Lines of Duplicated Code	LDC
Clone Instances	CI	Logical Lines of Duplicated Code	LLDC

Table C.2: Descriptive static of antl4 dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.05	0.16	0.00
LCOM5	0.00	78.00	1.57	4.86	1.00
NL	0.00	10.00	1.07	1.60	0.00
WMC	0.00	216.00	14.74	26.47	5.00
CBOI	0.00	79.00	4.61	9.07	1.00
NOI	0.00	78.00	4.78	8.68	1.00
AD	0.00	1.00	0.21	0.25	0.12
CD	0.00	0.75	0.13	0.15	0.09
CLOC	0.00	575.00	16.74	50.23	2.00
PDA	0.00	25.00	1.43	2.71	1.00
PUA	0.00	91.00	5.91	8.25	3.00
DIT	0.00	6.00	1.35	1.38	1.00
NOC	0.00	33.00	0.57	2.18	0.00
NOD	0.00	104.00	1.10	6.32	0.00
NOP	0.00	3.00	0.69	0.55	1.00
NA	0.00	67.00	7.09	7.14	5.00
NG	0.00	28.00	3.44	4.92	2.00
NLA	0.00	59.00	2.45	4.88	1.00
NLG	0.00	28.00	1.22	3.01	0.00
NLPA	0.00	34.00	1.50	3.45	0.00
NM	0.00	139.00	17.36	19.43	9.50
NPA	0.00	37.00	5.31	6.14	4.00
NS	0.00	11.00	0.77	1.78	0.00
TNA	0.00	152.00	7.77	9.94	6.00
TNLG	0.00	28.00	1.33	3.28	0.00
TNLS	0.00	11.00	0.26	1.02	0.00
TNPA	0.00	53.00	5.61	6.67	4.00
WarningCritical	0.00	11.00	0.23	0.95	0.00
WarningInfo	0.00	371.00	28.52	47.24	12.00
WarningMajor	0.00	246.00	3.11	14.58	0.00
WarningMinor	0.00	212.00	12.00	25.35	2.00
Basic Rules	0.00	2.00	0.04	0.23	0.00
Brace Rules	0.00	56.00	2.51	7.06	0.00
Clone Implementation Rules	0.00	1.00	0.00	0.05	0.00
Cohesion Metric Rules	0.00	3.00	0.29	0.53	0.00
Complexity Metric Rules	0.00	13.00	0.38	1.52	0.00
Controversial Rules	0.00	14.00	0.35	1.36	0.00
Coupling Metric Rules	0.00	7.00	0.28	0.76	0.00
Design Rules	0.00	109.00	5.93	14.16	1.00
Import Statement Rules	0.00	3.00	0.02	0.20	0.00
Inheritance Metric Rules	0.00	2.00	0.05	0.24	0.00

J2EE Rules	0.00	1.00	0.00	0.05	0.00
JUnit Rules	0.00	92.00	2.32	10.03	0.00
Jakarta Commons Logging Rules	0.00	4.00	0.01	0.19	0.00
Java Logging Rules	0.00	38.00	0.39	2.85	0.00
JavaBean Rules	0.00	1.00	0.00	0.05	0.00
Naming Rules	0.00	91.00	0.99	5.24	0.00
Optimization Rules	0.00	17.00	0.21	1.06	0.00
Security Code Guideline Rules	0.00	3.00	0.04	0.26	0.00
Size Metric Rules	0.00	71.00	3.94	5.44	4.00
Strict Exception Rules	0.00	10.00	0.11	0.62	0.00
String and StringBuffer Rules	0.00	26.00	0.78	2.96	0.00
Type Resolution Rules	0.00	92.00	1.41	6.78	0.00
Unnecessary and Unused Code Rules	0.00	55.00	0.22	2.77	0.00

Table C.3: Descriptive static of junit dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.02	0.14	0.00
LCOM5	0.00	63.00	1.74	3.15	1.00
NL	0.00	5.00	0.30	0.73	0.00
WMC	0.00	87.00	4.36	7.95	2.00
CBO	0.00	68.00	3.80	4.50	3.00
CBOI	0.00	124.00	2.74	8.33	1.00
NII	0.00	392.00	3.03	19.35	0.00
NOI	0.00	34.00	2.31	4.34	1.00
RFC	0.00	87.00	5.71	8.68	3.00
AD	0.00	1.00	0.11	0.27	0.00
CD	0.00	0.81	0.07	0.17	0.00
PUA	0.00	62.00	3.12	3.83	2.00
DIT	0.00	5.00	0.64	0.97	0.00
NOC	0.00	71.00	0.34	2.99	0.00
LLOC	1.00	371.00	19.01	29.86	9.00
NA	0.00	9.00	0.94	1.37	0.00
NG	0.00	9.00	0.56	1.43	0.00
NLA	0.00	8.00	0.63	1.14	0.00
NLG	0.00	9.00	0.25	0.95	0.00
NLPA	0.00	6.00	0.16	0.53	0.00
NLS	0.00	3.00	0.04	0.24	0.00
NM	0.00	80.00	11.60	18.71	3.00
NPM	0.00	75.00	9.44	16.26	2.00
NS	0.00	4.00	0.31	0.75	0.00
TNG	0.00	17.00	0.76	1.92	0.00

TNLM	0.00	99.00	4.60	7.85	2.00
TNLPA	0.00	17.00	0.27	1.09	0.00
TNM	0.00	662.00	17.35	42.10	3.00
TNOS	0.00	181.00	8.42	17.53	2.00
TNPA	0.00	17.00	0.30	1.11	0.00
WarningCritical	0.00	11.00	0.11	0.72	0.00
WarningInfo	0.00	325.00	15.28	24.32	8.00
WarningMajor	0.00	85.00	1.61	4.26	1.00
WarningMinor	0.00	173.00	4.67	11.04	1.00
Basic Rules	0.00	5.00	0.03	0.29	0.00
Brace Rules	0.00	28.00	0.81	2.96	0.00
Cohesion Metric Rules	0.00	23.00	0.54	1.29	0.00
Controversial Rules	0.00	5.00	0.13	0.57	0.00
Coupling Metric Rules	0.00	2.00	0.05	0.23	0.00
Design Rules	0.00	108.00	1.95	5.54	1.00
Empty Code Rules	0.00	5.00	0.04	0.30	0.00
Import Statement Rules	0.00	7.00	0.02	0.28	0.00
Inheritance Metric Rules	0.00	3.00	0.05	0.25	0.00
J2EE Rules	0.00	4.00	0.01	0.17	0.00
JUnit Rules	0.00	89.00	1.75	6.58	0.00
Java Logging Rules	0.00	3.00	0.01	0.15	0.00
Migration Rules	0.00	8.00	0.04	0.46	0.00
Naming Rules	0.00	18.00	0.50	1.80	0.00
Optimization Rules	0.00	30.00	0.39	1.63	0.00
Security Code Guideline Rules	0.00	1.00	0.00	0.07	0.00
Size Metric Rules	0.00	48.00	1.28	3.71	0.00
Strict Exception Rules	0.00	14.00	0.33	1.11	0.00
String and StringBuffer Rules	0.00	4.00	0.05	0.32	0.00
Type Resolution Rules	0.00	28.00	0.33	1.40	0.00
Unnecessary and Unused Code Rules	0.00	2.00	0.02	0.13	0.00

Table C.4: Descriptive static of MapDB dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.12	0.27	0.00
LCOM5	0.00	22.00	2.07	2.49	1.00
NL	0.00	29.00	1.33	2.07	1.00
WMC	0.00	1525.00	23.40	93.92	4.00
CBO	0.00	40.00	3.74	4.70	2.00
CBOI	0.00	111.00	2.94	9.79	0.00
NII	0.00	284.00	5.40	23.10	0.00
NOI	0.00	54.00	5.74	9.00	2.00

RFC	0.00	156.00	13.58	20.97	6.00
AD	0.00	1.00	0.15	0.27	0.00
CD	0.00	0.95	0.10	0.16	0.02
PUA	0.00	152.00	5.78	10.99	3.00
DIT	0.00	4.00	0.52	0.71	0.00
NOC	0.00	11.00	0.22	1.00	0.00
LLOC	2.00	11272.00	119.63	634.97	22.00
NA	0.00	77.00	5.97	10.64	2.00
NG	0.00	19.00	0.77	2.19	0.00
NLA	0.00	34.00	2.14	4.49	0.00
NLG	0.00	19.00	0.43	1.50	0.00
NLM	0.00	156.00	7.85	15.01	3.00
NLPA	0.00	18.00	0.21	1.35	0.00
NM	0.00	156.00	12.89	21.53	3.00
NPA	0.00	18.00	2.35	5.20	0.00
TNA	0.00	255.00	8.81	18.37	2.00
TNLG	0.00	31.00	0.62	2.29	0.00
TNLPA	0.00	235.00	0.92	11.56	0.00
TNLPM	0.00	208.00	8.90	19.59	3.00
TNLS	0.00	6.00	0.14	0.57	0.00
TNPA	0.00	252.00	3.73	13.55	0.00
TNPM	0.00	252.00	14.88	28.98	3.00
WarningCritical	0.00	92.00	1.17	7.10	0.00
WarningInfo	0.00	1516.00	40.84	109.93	13.00
WarningMajor	0.00	153.00	2.27	10.61	0.00
WarningMinor	0.00	4541.00	36.20	246.20	4.00
Basic Rules	0.00	925.00	3.43	48.22	0.00
Brace Rules	0.00	270.00	6.07	19.52	0.00
Cohesion Metric Rules	0.00	10.00	0.55	1.04	0.00
Complexity Metric Rules	0.00	45.00	0.81	3.57	0.00
Controversial Rules	0.00	123.00	1.93	7.63	0.00
Coupling Metric Rules	0.00	18.00	0.26	1.18	0.00
Design Rules	0.00	71.00	2.81	7.59	1.00
Documentation Metric Rules	0.00	587.00	22.70	45.75	9.00
Empty Code Rules	0.00	22.00	0.27	1.73	0.00
Import Statement Rules	0.00	9.00	0.04	0.48	0.00
Inheritance Metric Rules	0.00	2.00	0.02	0.16	0.00
JUnit Rules	0.00	2320.00	16.55	131.09	0.00
Java Logging Rules	0.00	10.00	0.17	0.86	0.00
JavaBean Rules	0.00	4.00	0.07	0.32	0.00
Migration Rules	0.00	980.00	3.71	51.19	0.00
Naming Rules	0.00	179.00	2.01	11.07	0.00
Optimization Rules	0.00	20.00	0.34	1.54	0.00

Security Code Guideline Rules	0.00	9.00	0.08	0.58	0.00
Size Metric Rules	0.00	202.00	3.19	12.12	0.00
Strict Exception Rules	0.00	176.00	1.64	12.21	0.00
String and StringBuffer Rules	0.00	22.00	0.28	1.54	0.00
Type Resolution Rules	0.00	23.00	0.34	2.05	0.00
Unnecessary and Unused Code Rules	0.00	941.00	3.29	49.06	0.00

Table C.5: Descriptive static of mcMMO dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.12	0.27	0.00
LCOM5	0.00	138.00	2.03	8.25	1.00
NL	0.00	7.00	1.50	1.62	1.00
WMC	0.00	261.00	19.07	31.35	8.00
CBO	0.00	48.00	4.76	6.11	3.00
CBOI	0.00	101.00	4.00	11.84	1.00
NII	0.00	250.00	6.80	25.67	1.00
NOI	0.00	86.00	8.10	12.14	4.00
RFC	1.00	200.00	15.94	22.41	9.00
AD	0.00	0.97	0.12	0.24	0.00
CD	0.00	0.70	0.08	0.13	0.00
CLOC	0.00	355.00	10.97	30.89	0.00
DLOC	0.00	355.00	9.21	29.35	0.00
PUA	1.00	181.00	5.49	14.26	3.00
TCD	0.00	0.70	0.08	0.13	0.00
DIT	0.00	2.00	0.37	0.51	0.00
NOC	0.00	13.00	0.23	1.31	0.00
LLOC	3.00	917.00	64.69	95.16	32.00
LOC	3.00	1160.00	90.96	136.45	42.00
NA	0.00	36.00	4.55	6.07	3.00
NG	0.00	197.00	3.14	13.85	0.00
NLA	0.00	32.00	2.82	4.88	1.00
NLM	0.00	199.00	7.84	17.23	4.00
NLPM	0.00	197.00	6.08	16.92	2.00
NLS	0.00	15.00	0.47	1.60	0.00
NPM	0.00	201.00	6.87	17.33	2.00
TNLG	0.00	197.00	3.30	14.74	0.00
WarningCritical	0.00	15.00	0.42	1.44	0.00
WarningInfo	2.00	381.00	25.71	42.10	15.00
WarningMajor	0.00	39.00	0.77	3.13	0.00
WarningMinor	0.00	62.00	5.66	9.15	2.00
Basic Rules	0.00	3.00	0.07	0.36	0.00

Brace Rules	0.00	2.00	0.02	0.17	0.00
Cohesion Metric Rules	0.00	1.00	0.27	0.45	0.00
Controversial Rules	0.00	4.00	0.07	0.41	0.00
Coupling Metric Rules	0.00	8.00	0.35	1.09	0.00
Documentation Metric Rules	1.00	367.00	17.35	32.35	9.00
Empty Code Rules	0.00	2.00	0.03	0.18	0.00
Inheritance Metric Rules	0.00	1.00	0.01	0.08	0.00
JUnit Rules	0.00	6.00	0.02	0.35	0.00
Jakarta Commons Logging Rules	0.00	4.00	0.05	0.43	0.00
Java Logging Rules	0.00	8.00	0.11	0.72	0.00
Naming Rules	0.00	55.00	0.50	3.41	0.00
Optimization Rules	0.00	18.00	0.17	1.14	0.00
Size Metric Rules	0.00	33.00	1.88	3.70	0.00
Strict Exception Rules	0.00	10.00	0.14	0.79	0.00
Type Resolution Rules	0.00	14.00	0.19	1.00	0.00
Unnecessary and Unused Code Rules	0.00	2.00	0.02	0.19	0.00

Table C.6: Descriptive static of mct dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.19	0.36	0.00
LCOM5	0.00	60.00	1.37	2.42	1.00
NL	0.00	18.00	0.95	1.47	0.00
WMC	0.00	413.00	9.31	19.98	3.00
CBO	0.00	56.00	3.86	5.07	2.00
CBOI	0.00	409.00	2.92	13.75	1.00
NII	0.00	503.00	3.89	18.84	0.00
NOI	0.00	199.00	4.92	10.50	1.00
RFC	0.00	246.00	10.02	17.18	4.00
AD	0.00	1.00	0.16	0.32	0.00
CD	0.00	0.82	0.10	0.17	0.00
PUA	0.00	100.00	3.88	6.64	2.00
DIT	0.00	5.00	0.60	0.91	0.00
NOC	0.00	65.00	0.27	2.63	0.00
NA	0.00	124.00	5.07	8.75	2.00
NG	0.00	60.00	3.58	7.68	0.00
NLA	0.00	124.00	2.51	6.27	1.00
NLG	0.00	45.00	1.18	3.50	0.00
NLPA	0.00	122.00	0.31	3.95	0.00
NLS	0.00	51.00	0.53	2.71	0.00
NM	0.00	175.00	12.78	22.32	4.00
NOS	0.00	896.00	26.93	65.45	6.50

NPA	0.00	122.00	0.78	4.15	0.00
NS	0.00	51.00	1.39	3.82	0.00
TNG	0.00	1099.00	5.64	26.62	1.00
TNLG	0.00	50.00	1.50	4.25	0.00
TNLPA	0.00	163.00	0.47	5.52	0.00
TNPA	0.00	163.00	1.04	5.69	0.00
TNS	0.00	409.00	2.12	10.09	0.00
WarningBlocker	0.00	1.00	0.00	0.02	0.00
WarningCritical	0.00	22.00	0.24	1.37	0.00
WarningInfo	0.00	1390.00	24.99	52.73	12.00
WarningMajor	0.00	37.00	0.71	2.42	0.00
WarningMinor	0.00	318.00	4.59	15.05	1.00
Basic Rules	0.00	22.00	0.14	1.11	0.00
Brace Rules	0.00	42.00	0.65	2.83	0.00
Clone Implementation Rules	0.00	2.00	0.01	0.10	0.00
Cohesion Metric Rules	0.00	8.00	0.33	0.67	0.00
Complexity Metric Rules	0.00	14.00	0.23	1.05	0.00
Controversial Rules	0.00	58.00	0.39	2.01	0.00
Coupling Metric Rules	0.00	16.00	0.24	0.94	0.00
Design Rules	0.00	138.00	1.90	5.90	0.00
Documentation Metric Rules	0.00	311.00	13.96	22.15	7.00
Empty Code Rules	0.00	6.00	0.03	0.23	0.00
Finalizer Rules	0.00	1.00	0.00	0.03	0.00
Import Statement Rules	0.00	64.00	0.06	1.48	0.00
Inheritance Metric Rules	0.00	49.00	0.09	1.09	0.00
J2EE Rules	0.00	2.00	0.00	0.08	0.00
JUnit Rules	0.00	286.00	1.27	10.54	0.00
Jakarta Commons Logging Rules	0.00	12.00	0.05	0.52	0.00
Java Logging Rules	0.00	17.00	0.06	0.57	0.00
Migration Rules	0.00	4.00	0.01	0.14	0.00
Naming Rules	0.00	133.00	0.36	3.13	0.00
Optimization Rules	0.00	20.00	0.21	1.09	0.00
Security Code Guideline Rules	0.00	4.00	0.02	0.20	0.00
Size Metric Rules	0.00	364.00	2.26	9.10	0.00
Strict Exception Rules	0.00	11.00	0.09	0.56	0.00
String and StringBuffer Rules	0.00	35.00	0.14	1.03	0.00
Type Resolution Rules	0.00	12.00	0.13	0.75	0.00
Unnecessary and Unused Code Rules	0.00	14.00	0.06	0.43	0.00

Table C.7: Descriptive static of oryx dataset.

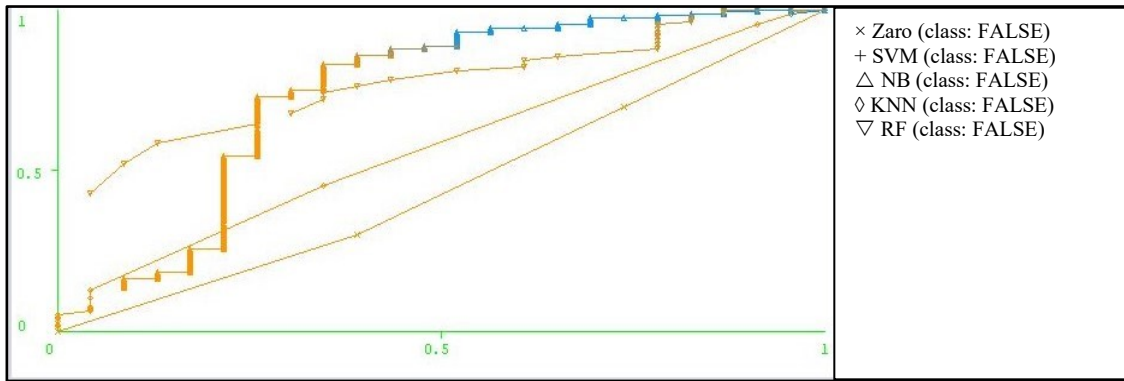
Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.07	0.19	0.00
LCOM5	0.00	10.00	1.37	1.16	1.00
NL	0.00	13.00	1.33	1.48	1.00
WMC	1.00	121.00	8.93	12.02	5.00
CBO	0.00	38.00	4.04	4.12	3.00
CBOI	0.00	77.00	3.47	7.78	1.00
NII	0.00	179.00	4.33	12.94	1.00
NOI	0.00	66.00	5.25	6.73	3.00
RFC	1.00	104.00	9.75	9.14	7.00
AD	0.00	1.00	0.27	0.34	0.11
CD	0.00	0.60	0.12	0.14	0.07
PUA	0.00	26.00	3.26	3.51	2.00
DIT	0.00	5.00	0.75	0.95	1.00
NOC	0.00	64.00	0.41	3.13	0.00
NA	0.00	17.00	3.74	3.51	3.00
NG	0.00	14.00	2.04	2.59	1.00
NLA	0.00	15.00	1.73	2.28	1.00
NLG	0.00	14.00	0.88	1.95	0.00
NLM	1.00	50.00	4.50	4.83	3.00
NLPA	0.00	10.00	0.07	0.54	0.00
NLPM	0.00	46.00	3.43	4.28	2.00
NLS	0.00	4.00	0.05	0.31	0.00
NM	1.00	60.00	10.14	9.43	7.00
NPA	0.00	10.00	0.29	0.86	0.00
NPM	0.00	54.00	5.13	4.81	4.00
NS	0.00	4.00	0.37	0.84	0.00
TNPA	0.00	10.00	0.30	0.88	0.00
TNPM	0.00	63.00	5.57	5.82	4.00
TNS	0.00	4.00	0.38	0.85	0.00
WarningCritical	0.00	1.00	0.01	0.07	0.00
WarningInfo	0.00	160.00	15.70	18.21	10.00
WarningMajor	0.00	25.00	0.56	1.77	0.00
WarningMinor	0.00	90.00	3.02	7.40	1.00
Basic Rules	0.00	1.00	0.01	0.11	0.00
Clone Implementation Rules	0.00	1.00	0.00	0.04	0.00
Cohesion Metric Rules	0.00	3.00	0.24	0.47	0.00
Complexity Metric Rules	0.00	9.00	0.25	0.96	0.00
Controversial Rules	0.00	7.00	0.19	0.67	0.00
Coupling Metric Rules	0.00	10.00	0.18	0.67	0.00
Design Rules	0.00	12.00	0.79	1.40	0.00
Documentation Metric Rules	0.00	115.00	11.62	11.69	8.00

Empty Code Rules	0.00	1.00	0.00	0.06	0.00
Inheritance Metric Rules	0.00	2.00	0.01	0.14	0.00
JUnit Rules	0.00	89.00	1.64	7.21	0.00
Jakarta Commons Logging Rules	0.00	2.00	0.02	0.15	0.00
Java Logging Rules	0.00	1.00	0.01	0.09	0.00
JavaBean Rules	0.00	1.00	0.07	0.25	0.00
Naming Rules	0.00	21.00	0.42	1.44	0.00
Optimization Rules	0.00	2.00	0.02	0.18	0.00
Security Code Guideline Rules	0.00	7.00	0.08	0.46	0.00
Size Metric Rules	0.00	12.00	0.69	1.42	0.00
Strict Exception Rules	0.00	5.00	0.03	0.26	0.00
String and StringBuffer Rules	0.00	4.00	0.05	0.30	0.00
Type Resolution Rules	0.00	23.00	0.26	1.52	0.00
Unnecessary and Unused Code Rules	0.00	1.00	0.00	0.04	0.00

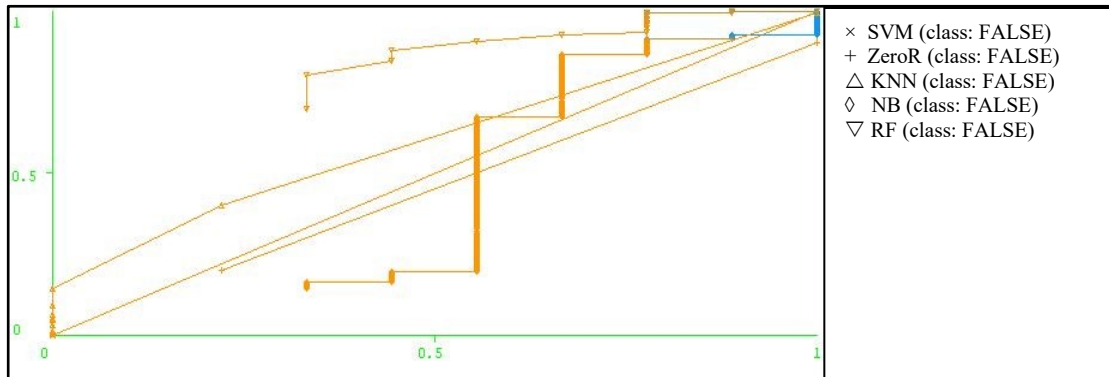
Table C.8: Descriptive static of titan dataset.

Metrics	Minimum	Maximum	Mean	Stdev	Median
CC	0.00	1.00	0.05	0.18	0.00
LCOM5	0.00	60.00	2.65	3.97	1.00
NL	0.00	10.00	1.15	1.59	1.00
WMC	0.00	323.00	10.11	19.73	4.00
CBO	0.00	124.00	3.15	6.85	1.00
CBOI	0.00	160.00	1.52	6.03	0.00
NII	0.00	47.00	0.95	3.66	0.00
NOI	0.00	223.00	1.66	10.77	0.00
RFC	0.00	299.00	6.69	15.96	3.00
CD	0.00	0.96	0.10	0.14	0.04
DLOC	0.00	539.00	5.64	21.58	0.00
PUA	0.00	80.00	4.45	5.66	3.00
DIT	0.00	9.00	0.73	1.23	0.00
NOC	0.00	36.00	0.23	1.29	0.00
NA	0.00	160.00	3.76	7.16	2.00
NG	0.00	44.00	2.37	5.40	0.00
NLA	0.00	160.00	2.12	5.78	1.00
NLG	0.00	35.00	1.13	2.78	0.00
NLPA	0.00	139.00	0.47	4.16	0.00
NLS	0.00	15.00	0.18	0.80	0.00
NM	0.00	123.00	10.74	17.98	4.00
NPA	0.00	139.00	1.31	5.08	0.00
NS	0.00	15.00	0.37	1.21	0.00
TNG	0.00	76.00	2.61	5.96	0.00

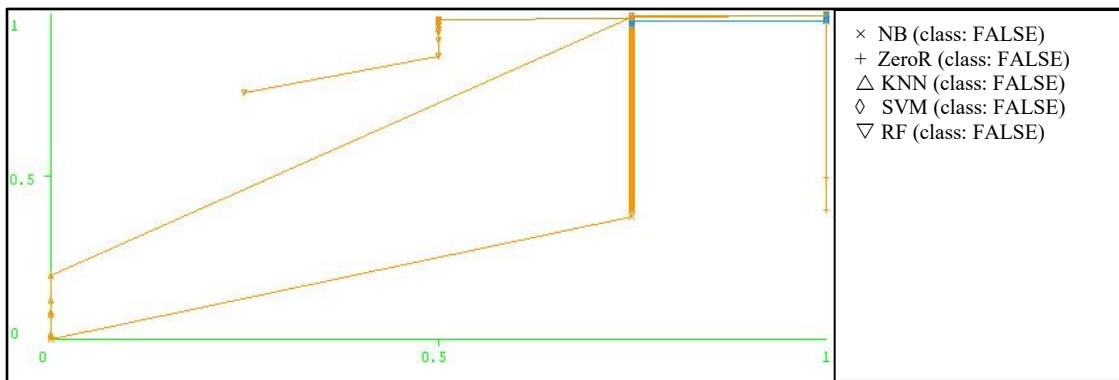
TNLPM	0.00	93.00	5.20	8.28	3.00
WarningCritical	0.00	75.00	0.17	2.08	0.00
WarningInfo	0.00	428.00	20.35	30.89	10.00
WarningMajor	0.00	391.00	6.61	18.08	1.00
WarningMinor	0.00	1104.00	7.15	32.65	1.00
Basic Rules'	0.00	71.00	0.11	1.88	0.00
Clone Implementation Rules	0.00	2.00	0.00	0.07	0.00
Cohesion Metric Rules	0.00	19.00	0.60	0.87	0.00
Complexity Metric Rules	0.00	17.00	0.32	1.28	0.00
Controversial Rules	0.00	27.00	0.31	1.34	0.00
Coupling Metric Rules	0.00	24.00	0.14	0.96	0.00
Design Rules	0.00	40.00	1.26	3.10	0.00
Documentation Metric Rules	0.00	265.00	15.46	22.13	9.00
Empty Code Rules	0.00	71.00	0.08	1.87	0.00
Import Statement Rules	0.00	92.00	0.17	2.66	0.00
Inheritance Metric Rules	0.00	3.00	0.10	0.43	0.00
J2EE Rules	0.00	3.00	0.01	0.17	0.00
JUnit Rules	0.00	945.00	2.73	27.04	0.00
Jakarta Commons Logging Rules	0.00	11.00	0.06	0.53	0.00
Java Logging Rules	0.00	6.00	0.03	0.28	0.00
JavaBean Rules	0.00	1.00	0.00	0.03	0.00
Naming Rules	0.00	14.00	0.40	1.12	0.00
Optimization Rules	0.00	11.00	0.12	0.57	0.00
Security Code Guideline Rules	0.00	4.00	0.03	0.23	0.00
Size Metric Rules	0.00	67.00	1.64	3.92	0.00
Strict Exception Rules	0.00	10.00	0.12	0.61	0.00
String and StringBuffer Rules	0.00	67.00	0.21	2.13	0.00
Type Resolution Rules	0.00	17.00	0.23	1.16	0.00



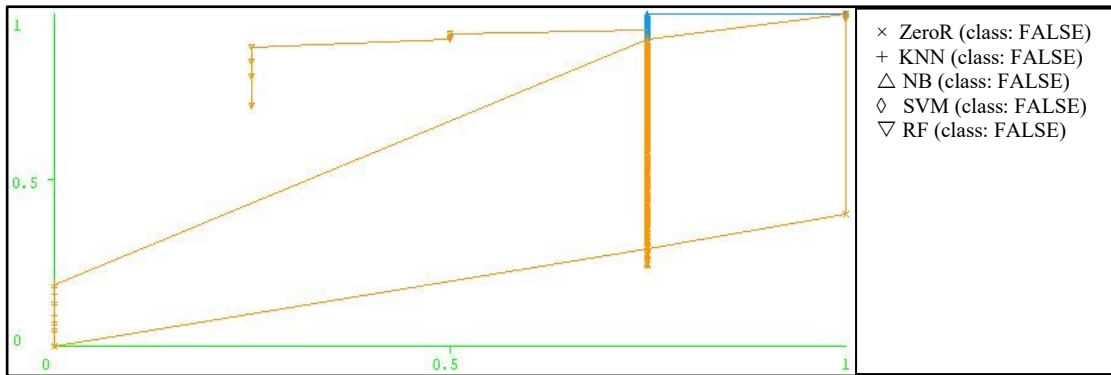
(antl4 dataset)



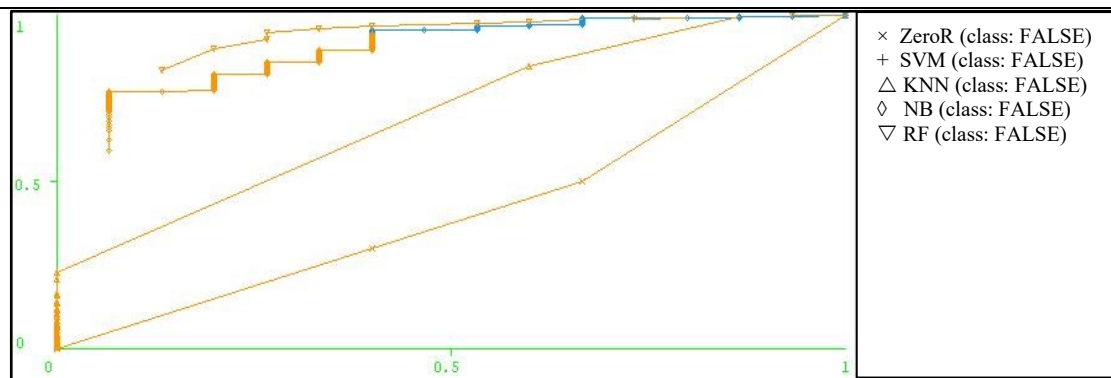
(junit dataset)



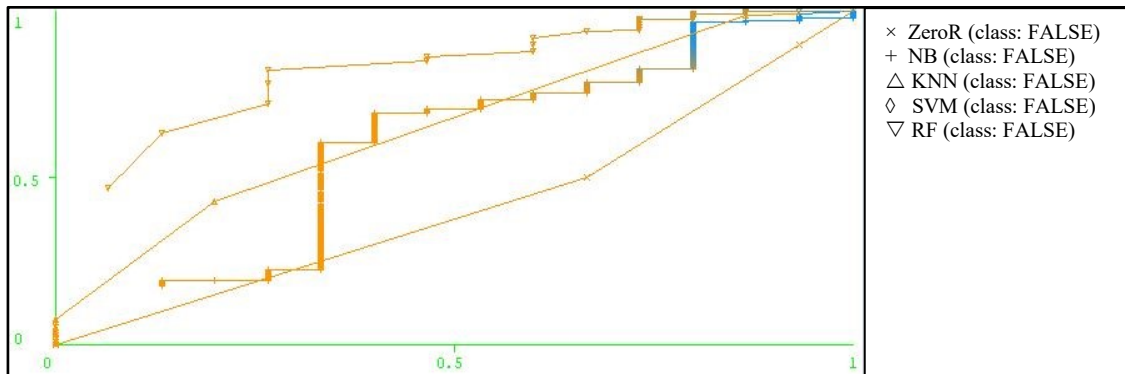
(MapDB dataset)



(mcMMO dataset)



(mct dataset)



(oryx dataset)

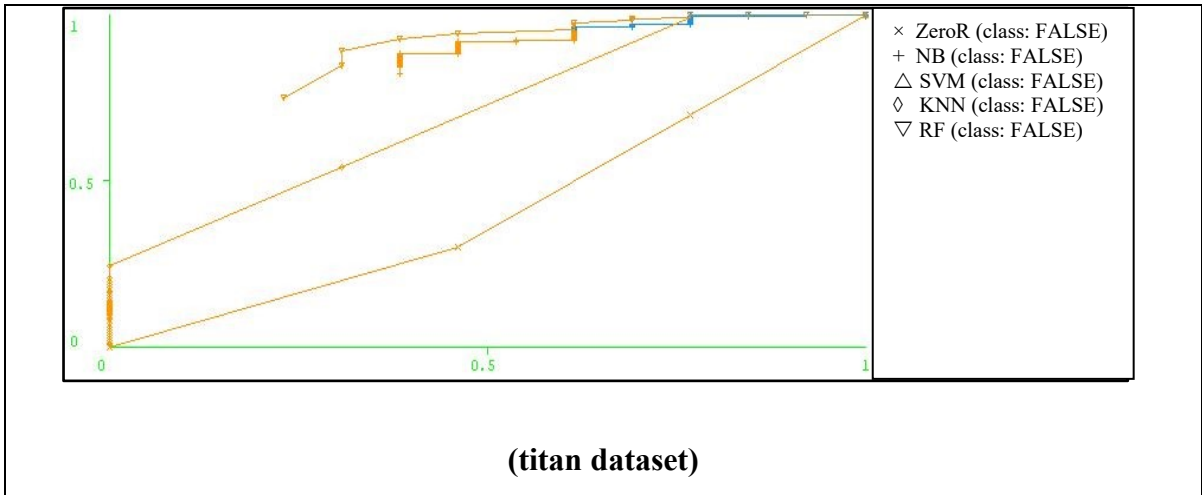
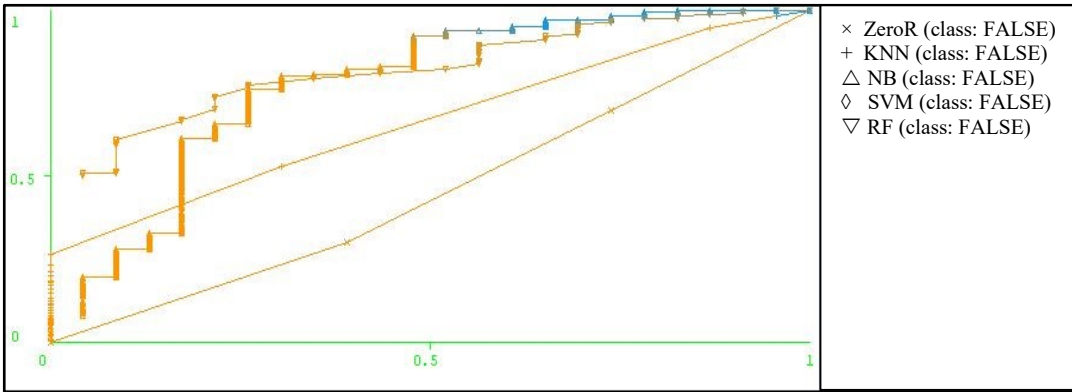
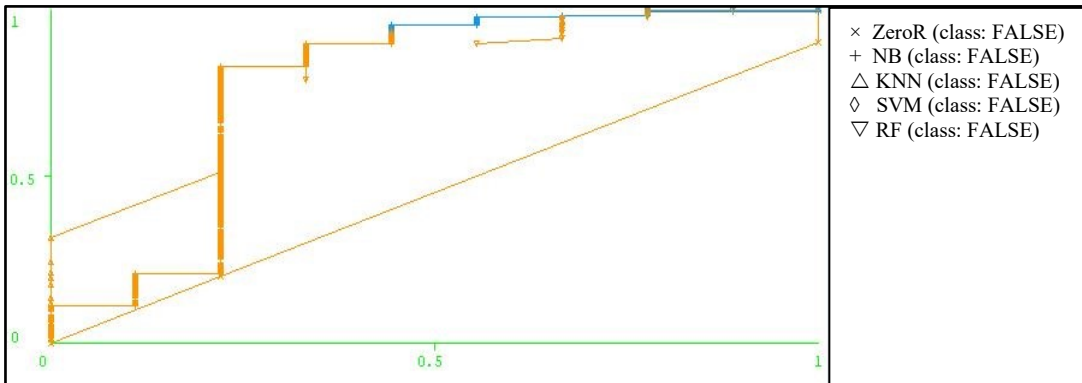


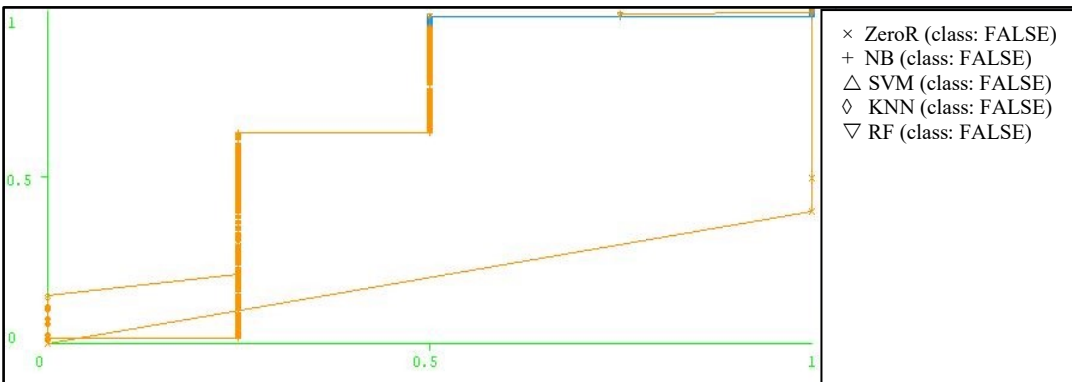
Figure C.1: Multiple ROC curves for prediction models in the first scenario.



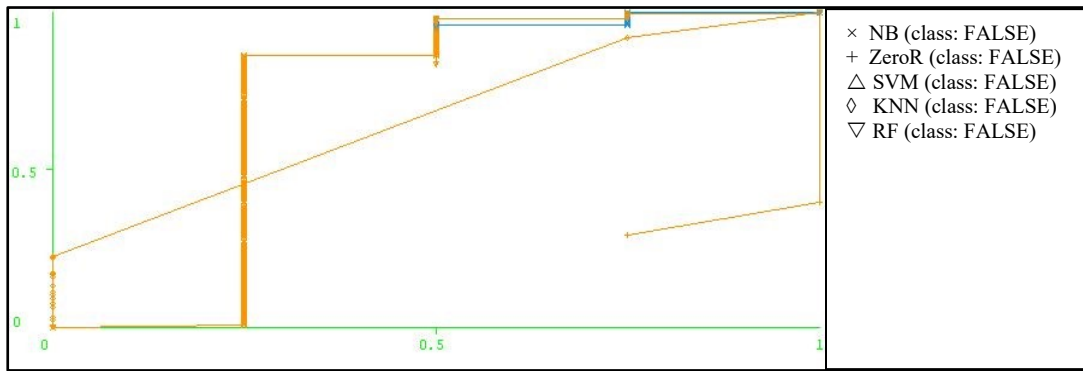
(antl4 dataset)



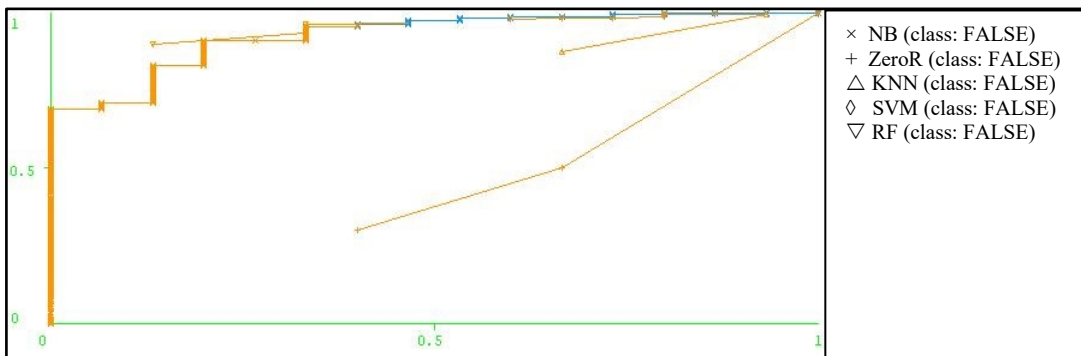
(junit dataset)



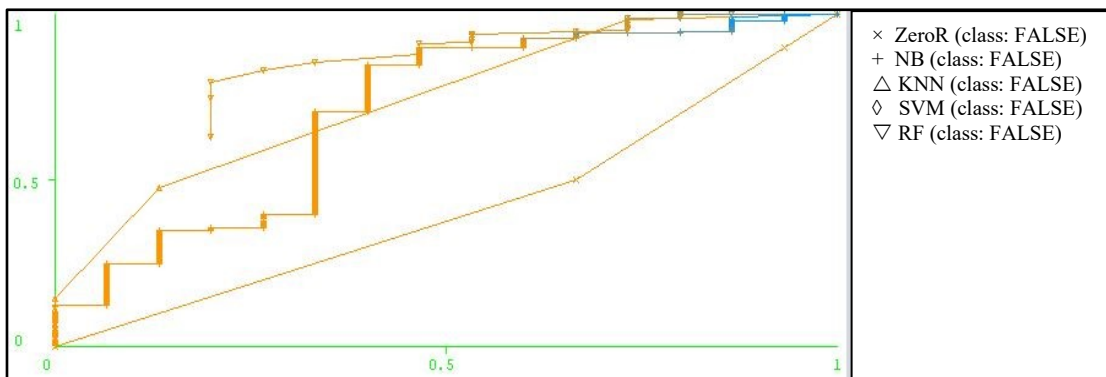
(MapDB dataset)



(mcMMO dataset)



(mct dataset)



(oryx dataset)

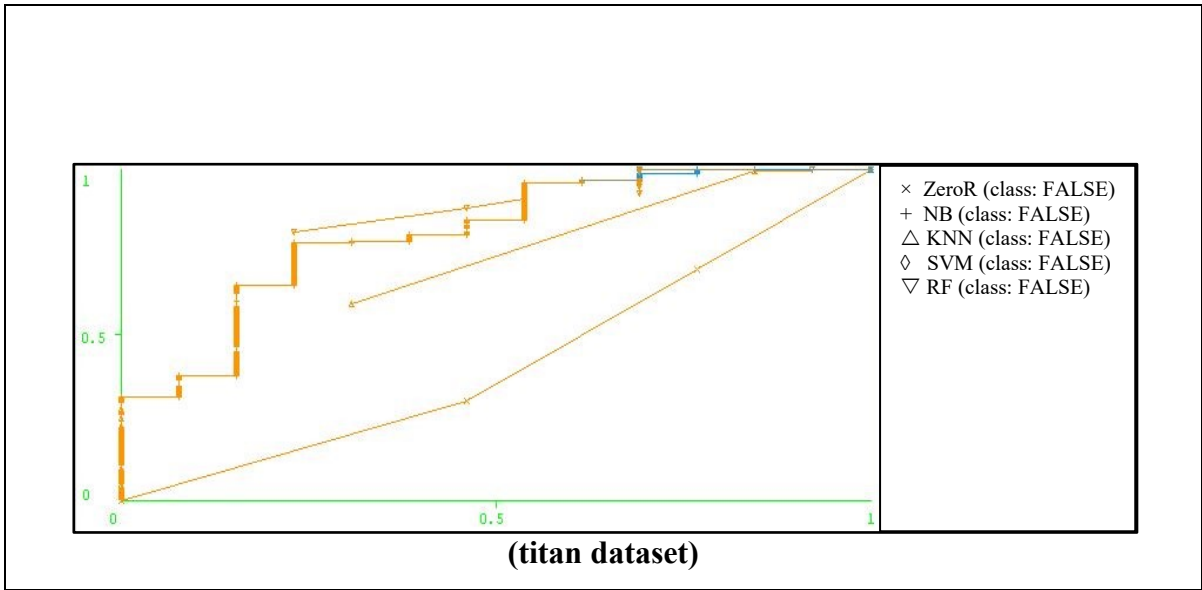
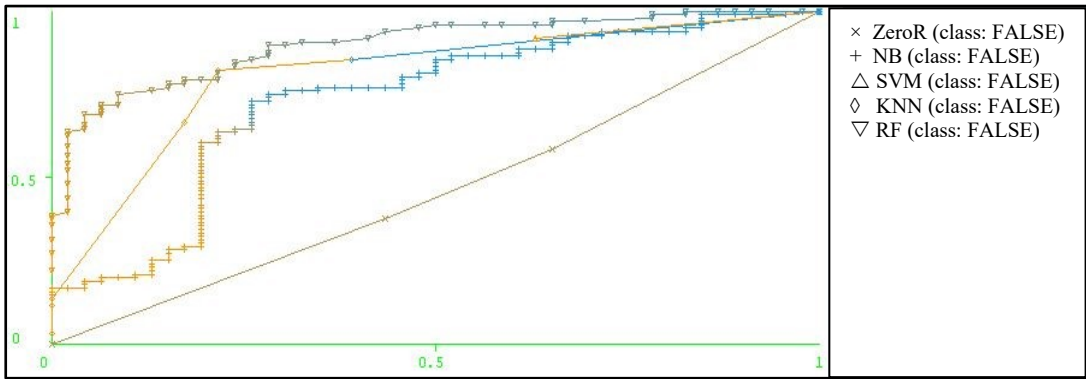
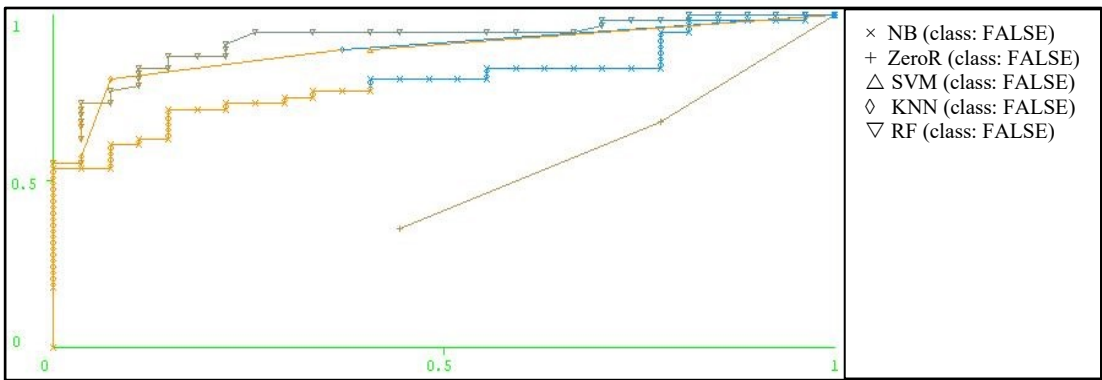


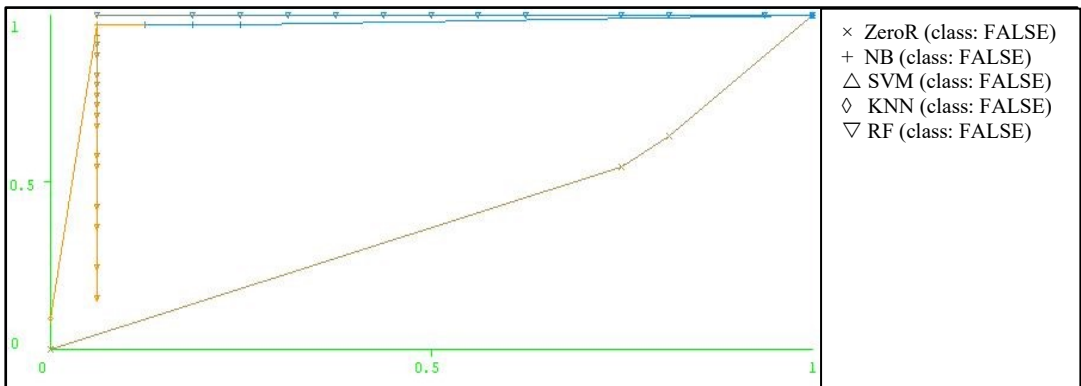
Figure C.2: Multiple ROC curves for prediction models in the second scenario.



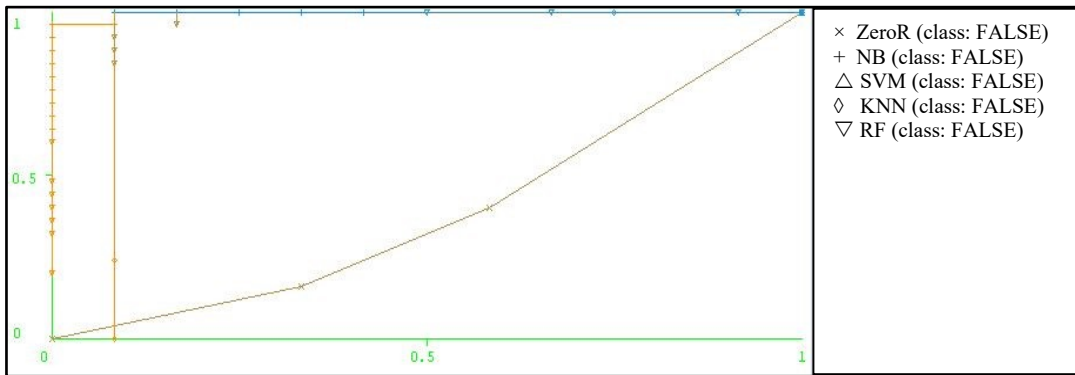
(antl4 dataset)



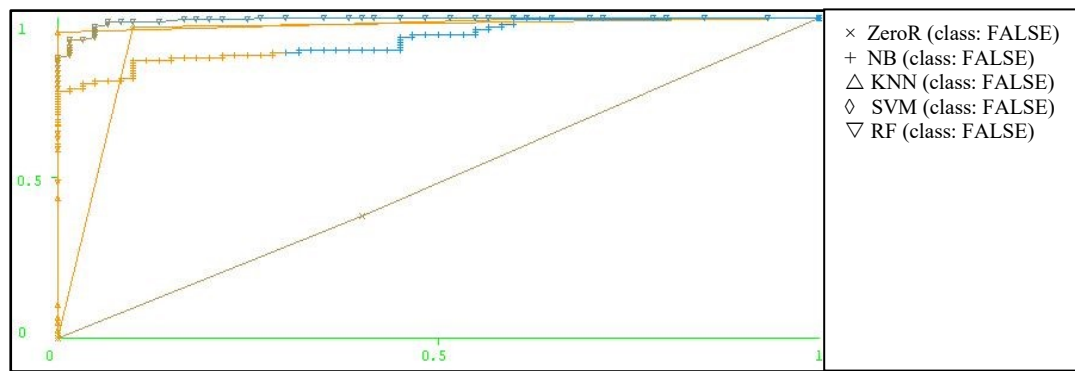
(junit dataset)



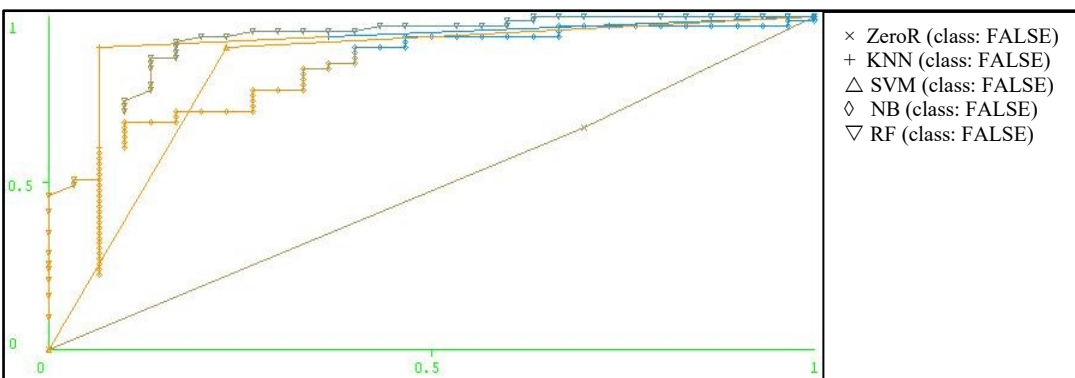
(MapDB dataset)



(mcMMO dataset)



(mct dataset)



(oryx dataset)

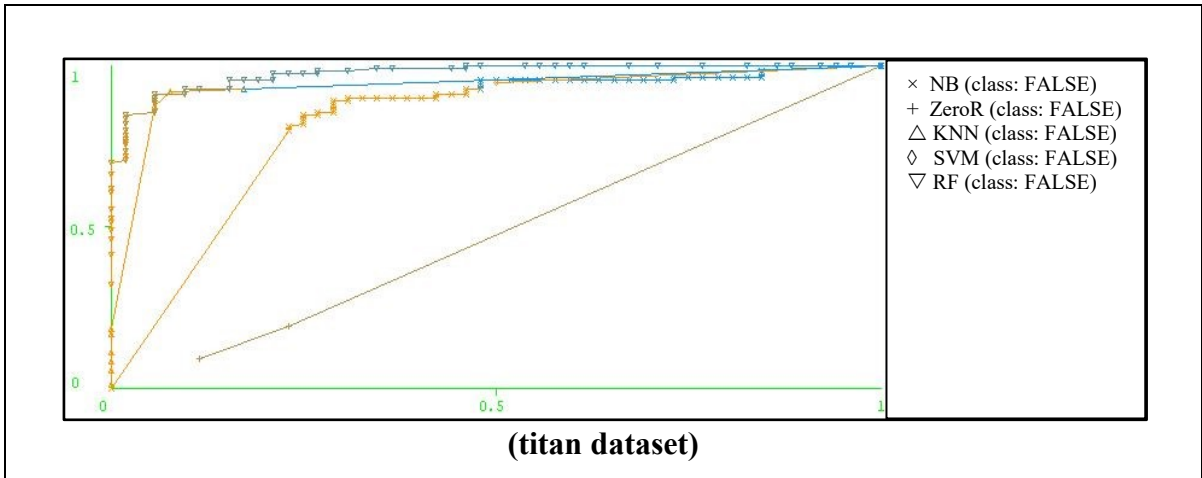
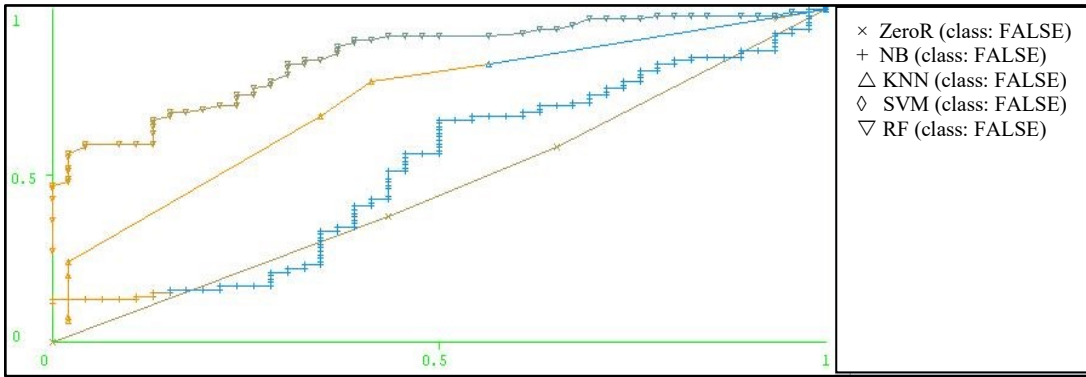
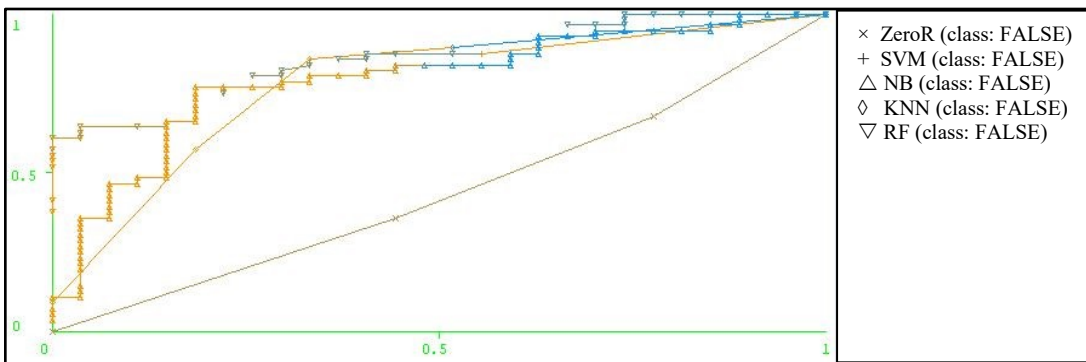


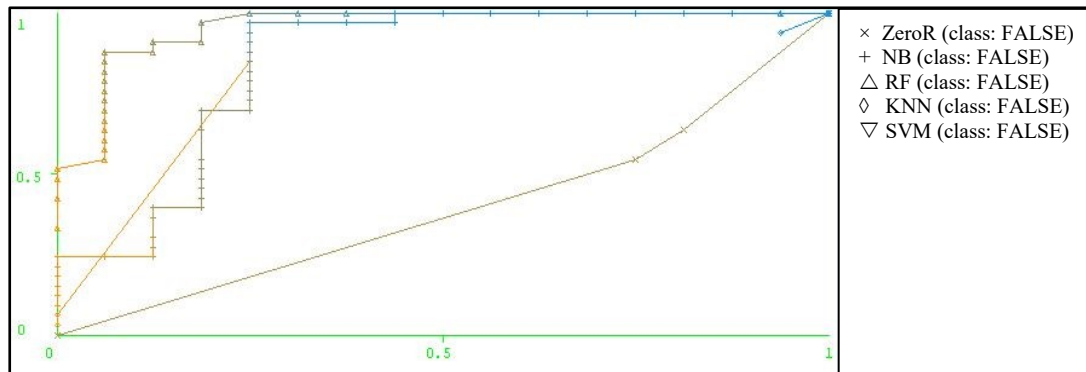
Figure C.3: Multiple ROC curves for prediction models in the third scenario.



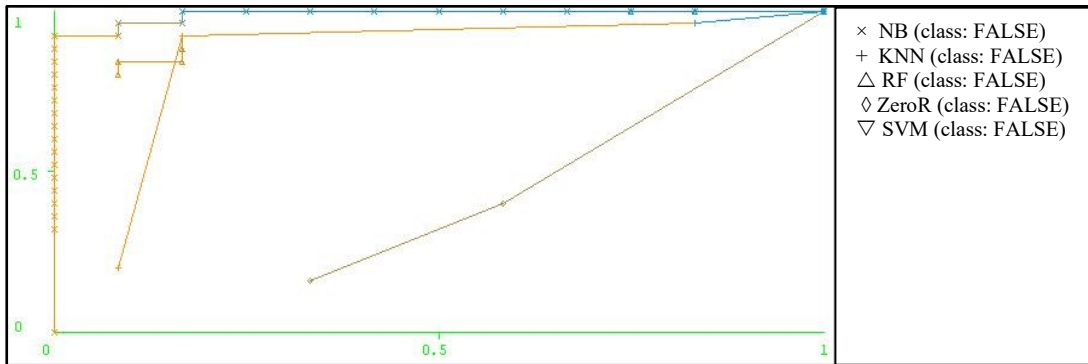
(antl4 dataset)



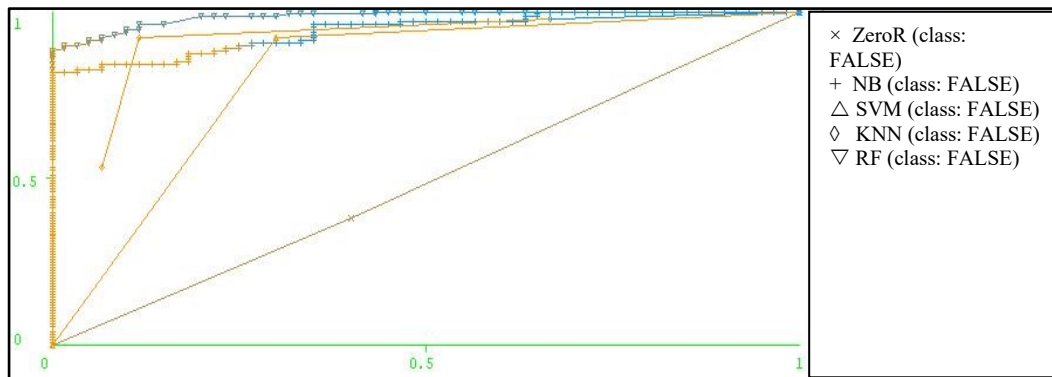
(junit dataset)



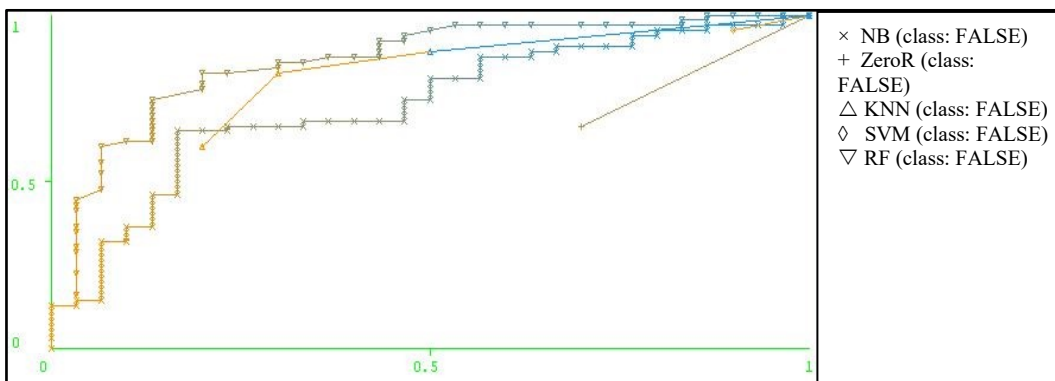
(MapDB dataset)



(mcMMO dataset)



(mct dataset)



(oryx dataset)

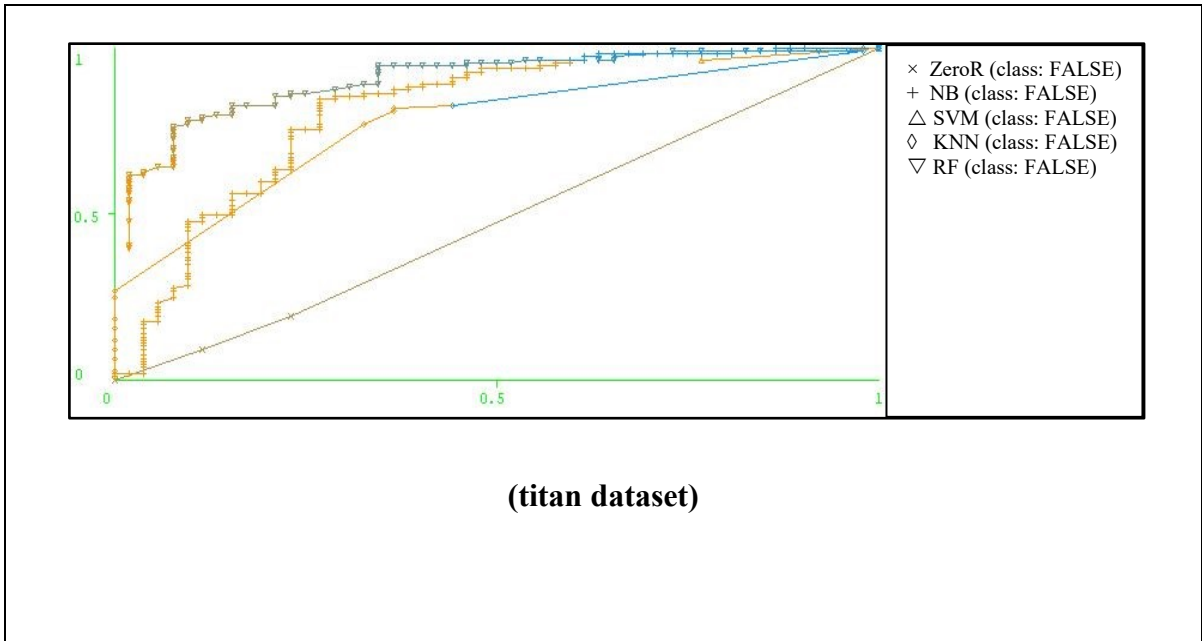


Figure C.4: Multiple ROC curves for prediction models in the fourth scenario.