

UNIVERSITY OF BAHRAIN

**College of Information
Technology**



**Department of
Computer Science**

Predicting Successful Software Reuse Using Machine Learning Algorithms

A Thesis Submitted in Partial Fulfilment of the Requirements for the
M.Sc.Degree in Software Engineering

Submitted by

Student Name: Mariam Khalifa Amin

ID Number: 19980608

Supervised by

Dr. Mustafa Hammad

أستاذ مشارك

جامعة البحرين

December 29, 2021



Dissertation Deposit Form

Author Name:	Mariam Khalifa Rashid Amin		
Email:	19980608@stu.uob.edu.bh	Contact No:	66665252
College:	Information Technology		
Department:	Computer Science		
Thesis Title:	Predicting Succesful Software Reuse Using Machine Learning		
Year:	2021		

Author's Declaration: I agree to the following conditions:

The above thesis will be available (in the University of Bahrain library and externally) and reproduced as necessary at the discretion of the University of Bahrain. It may also be digitized by the University of Bahrain and made available via the university's repository or via other parties on the Internet.

This condition shall apply to ALL copies including electronic copies.

The above thesis has been provided on the understanding that it is copyright protected material and that no quotation from this thesis may be published without proper acknowledgement.

Signature of the Author:		Date:	28-Feb-22
--------------------------	---	-------	-----------

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: -----

Maryam Khalifa Rashed Mohamed (Candidate)

Date: 29/12/2021

The Thesis defense committee considers the thesis titled:

Predicting Successful Software Reuse Using Machine Learning Algorithms

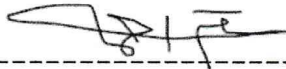
Submitted by **Maryam Khalifa Rashed Mohamed** to the College of Information Technology is satisfactory and acceptable for the Master of Science in Software Engineering.

The Defense Committee:



Dr. Mustafa Hammad
(Supervisor)

Department of Computer Science
University of Bahrain



Dr. Ahmad Fawzi Ootom
(Member)

The Hashemite University
Jordan



Dr. Fawzi AlBlooshi
(Member)

Department of Computer Science
University of Bahrain

Dedication

This research is lovingly dedicated to my parents, Fatima Mahmood and Khalifa Amin. To my siblings Rashid Amin, Ahmed Amin and Abdulrahman Amin. And lastly, to my precious children Salman Tariq and Jawan Tariq.

Acknowledgment

"He who is not grateful to people is not grateful to Allah" - Prophet Mohammad PBUH.

It was a challenging journey that prospered my research skills. I have learned so much in this journey. Therefore, I owe thanks to all who stood by me and allowed me to go through this journey. Firstly, I want to express my sincere thank and gratitude to Maj. Gen. (Prof.) Khalid bin Ali Al Khalifa the Director of The Royal Medical Services at Bahrain Defense Force, for the sponsorship. Also, I want to express my thank to the head of my department, Dr. Muzammil Abdul Rahman and Maj. Khalifa Ahmed Al Omari for believing in my skills and experience and recommending me as a candidate for this sponsorship, and following up on the sponsorship approval process. I want to thank my supervisor Maj. Nayef Yousif Bu Ali for his understanding of the pressure I was going through the master's courses.

I want to thank my advisor Dr. Nabil Hewahi and my master's instructors; I learned so much from their experience. Also, I would like to give my special thanks to my thesis supervisor and mentor, Dr. Mustafa Hammad, for invaluable advice and supervision. His insight, dynamism, sincerity has inspired me. He taught me the alphabet of research and always encouraged me to continue pursuing my academic study. It was a privilege and honor to study under his supervision. Any Attempt at any level can not be satisfactorily completed without his guidance and support.

I want to thank my parents, family, and friends for their continuous support and understanding. Finally, I would like to thank Allah for his countless blessings and for giving me the willingness, courage, and strength to pass through all difficulties.

Abstract

Software engineering is witnessing a market struggle in delivering functional products due to time constraints. Nowadays, software developers are reusing already existing software components instead of developing the whole software from scratch. Therefore, software reuse is considered a state of practice in the software development industry. The primary goal of software reuse is to reduce the development effort and project budget and increase productivity. However, some factors may lead to a software reuse failure. Wherefore, software development companies have to consider these factors to prevent project failure due to software reuse. Also, it has been seen clearly in the past the disastrous consequences of wrong software reuse. Thus, predicting a successful software reuse experience can help prior any software reuse attempt to avoid any loss caused by a wrong software reuse decision. However, until this day, there is no silver bullet solution that could predict software reuse experience nor estimate software components' reusability.

This thesis tackles two scenarios that help during software reuse decision-making. These scenarios do not focus on factors related to the technical aspects of the project, but they also cover the companies' managerial decisions. The first scenario is predicting the success of software reuse experience in a project using machine learning. In this thesis, the potential of six machine learning algorithms to predict successful software reuse experiences were evaluated. Naive Bayes, Artificial Neural Network, K-Star, Hoeffding Tree, Support Vector Machine, and K-Nearest Neighbor were used to evaluate the prediction on a public dataset. Results showed that machine learning performed very well in predicting software reuse, with accuracy reaching 100%. Also, feature selection techniques were introduced to extract the most relevant attributes from the dataset. The empirical results showed that feature selection improved software reuse experience prediction. Finally, we tried to handle the dataset's weakness by incorporating ensemble machine learning. To the best of our knowledge, this is the first attempt at using ensemble learning in predicting successful software reuse experiences. The empirical study showed remarkable results that scored an accuracy of 100%.

The second scenario proposes a software reusability estimation model. The model analyzes the static metric of java classes using data clustering and regression to estimate its reusability. The model was built based on a public dataset that used import line code as a reuse rate. Also, the goal, question, metric paradigm was used as a feature selection technique to extract the most relevant metrics in the used dataset. The evaluation results of the proposed model have successfully evaluated java classes reusability with very minimal error rates.

List of Publications

- Amin, M. and Hammad, M. (2021). Predicting successful software reuse using machine learning. In *3rd Smart Cities Symposium (SCS 2020)*, pages 1–5. Institution of Engineering and Technology (IET).
- Amin, M. and Hammad, M. (2020). Improving software reuse prediction using feature selection algorithms. In *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, pages 1–6. IEEE.
- Hammad, M. and Amin, M. (2021). Assuring software reuse Success using ensemble machine learning algorithms. *International Journal of Computing and Digital Systems*. (Submitted).
- Amin, M. and Hammad, M. (2021). Estimating java classes reusability using data clustering and static metrics analysis. *International Journal of Computing and Digital Systems*. (Submitted).

Contents

DEDICATION	i
ACKNOWLEDGMENT	ii
ABSTRACT	iii
LIST OF PUBLICATIONS	iv
TABLE OF CONTENTS	v
LIST OF TABLES	vii
LIST OF FIGURES	viii
LIST OF ACRONYMS	ix
1 Introduction	1
1.1 Overview	1
1.2 Research Background	2
1.3 Research Problem	4
1.4 Research Questions	5
1.5 Research Objectives	7
1.6 Research Significance	8
1.7 Research Setting	9
1.8 Summary	10
2 Literature Review	11
2.1 Overview	11
2.2 Effect of Software Reuse	11
2.3 Software Reuse Success and Failure Factors	14
2.4 Software Reuse Approaches	15
2.5 Machine Learning Utilization	17
2.6 Software Reusability Assessment and Estimation	19
2.7 Software Metrics and The Goal, Question, Metrics (GQM) Model	20
2.8 Summary	21

3	Methods and Procedures	23
3.1	Overview	23
3.2	Prediction of Software Reuse Experience	23
3.2.1	Used Dataset	24
3.2.2	Machine Learning Algorithms	28
3.2.3	Evaluation Criteria	33
3.3	Software Reusability Estimation Model	36
3.3.1	Used Dataset	36
3.3.2	Used Evaluation Criteria	36
3.3.3	Reusability Estimation Modeling	38
3.4	Summary	46
4	Empirical Results and Discussion	47
4.1	Overview	47
4.2	Prediction of Software Reuse Experience Experimental Results	47
4.2.1	Machine Learning Assessment	48
4.2.2	Feature Selection Experimental Results	49
4.2.3	Prediction Model Establishment Experimental Results	59
4.3	Reusability Estimation Model Evaluation	67
4.3.1	Effectiveness Evaluation	68
4.3.2	Accuracy Evaluation	68
4.4	Discussion	72
4.4.1	Prediction of Software Reuse Experience	72
4.4.2	Software Reusability Estimation Model	74
4.5	Threats to Validity	76
4.6	Summary	78
5	Conclusion and Recommendations	79
5.1	Overview	79
5.2	Conclusion	79
5.3	Recommendations and Future Work	81
	REFERENCES	81
	ملخص	92

List of Tables

3.1	Dataset Statistical Details	25
3.2	Description of Dataset State Variables	27
3.3	Description of Dataset Control Variables	28
3.4	Attributes of The Dataset Used in Java Classes' Reusability Estimation . . .	37
3.5	Reuse Rate Statistical Information	41
3.6	Comparison of Two Dataset Instances	42
3.7	Dataset Instances Distributed Among Clusters	42
3.8	SVR Results of Cluster 1, Cluster 2 & 5, and Cluster 3	44
3.9	SVR Results of Cluster 4,6,7	45
3.10	SVR Results of Cluster 8-10	45
4.1	Evaluation Criteria Related to Instances Classification	48
4.2	Evaluation Criteria Related to Error Rates	49
4.3	Evaluation Criteria Related to Performance	49
4.4	Performance Results Using Classifier Attribute Evaluation	51
4.5	Results of Feature Selection Using Correlation Attribute Evaluation	53
4.6	Result Before and After Applying InfoGrain Subset Evaluation	54
4.7	Performance Results Using Wrapper subset Evaluation	56
4.8	Readings Before and After Applying Classifier Subset Evaluation	58
4.9	Results of Evaluation Criteria Using Correlation-based Feature Selection . .	60
4.10	Readings Before and After Applying Classifier Subset Evaluation	61
4.11	Results of applying Stacking Algorithm on Selected Features	62
4.12	Results of Prediction Software Reuse Using Bagging	63
4.13	Evaluation Criteria Results of Using Bagging with Feature Selection	64
4.14	Performance Results of applying Voting Learning on The Full Dataset	66
4.15	Evaluation Criteria of Voting Learning on Selected Features	66
4.16	Overview of Java Classes Reusability Scores Computed Using the Proposed Model	70

List of Figures

1.1	3Cs Model of Software Component	3
1.2	Publications Statistics which included "machine learning" as a keyword from 2011 to 2021	6
1.3	Publications Statistics which included "machine learning" and "software engineering" as keywords from 2011 to 2021	7
3.1	Prediction Model Development Stages	24
3.2	Proposed Ensemble Prediction Models Overview	25
3.3	Ensemble Learning Algorithms	32
3.4	Model Design Flow Chart	39
3.5	Selected Features Using GQM Model	40
3.6	Reusability Score Distribution of LCOM5 in Cluster 8	43
4.1	MAE Variance of Classifier Attribute Evaluation	50
4.2	Change of MAE using Correlation Attribute Evaluation	52
4.3	MAE Changes InfoGrain Subset Evaluation	55
4.4	Variances of MAE Value Using Wrapper Subset Evaluation	57
4.5	MAE Values Before and After Classifier Subset Evaluation	57
4.6	Variance of MAE After Applying Correlation-based Feature Selection	59
4.7	Progressions of The Error Rates Before and After Feature	62
4.8	Effect of Feature Selection on MAE rates	65
4.9	Progression of MAE Before and After Using Feature Selection	67
4.10	Comparison of Error Rates	69
4.11	Performances of Stacking Prediction Models Before and After Feature Selection	74
4.12	Performances of Bagging Prediction Models Before and After Feature Selection	75
4.13	Performances of Voting Prediction Models Before and After Feature Selection	76
4.14	Feature Extracted using SelectKBest in Python	77

List of Acronyms

OO	Object-Oriented
OSS	Open Source Software
COTS	Commercial Off-The-Shelf
IRS	Inertial Reference System
ML	Machine Learning
GQM	Goal, Question, Metrics
SPL	Software Product Line
NLP	Natural Language Processing
IR	Information Retrieval
CK	Chidamber and Kemerer
AOS	Aspect-Oriented Systems
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
KNN	K-Nearest Neighbour
CBSD	Component-Based Software Development
LMNN	Large Margin Nearest Neighbor
AOSD	Aspect-Oriented Software Development
SoC	Separation of Concern
IoT	Internet of Things
CART	Classification and Regression Tree
REI	Reusability Index

ANN Artificial Neural Network

AGA Adaptive Genetic Algorithm

ELM Extreme learning machines

LM Levenberg Marquardt

WoS Web of Service

IBK Instance Based K-Nearest Neighbour

OOM Object Oriented Metrics

OOP Object Oriented Programming

NB Naive Bayes

HT Hoeffding Tree

SVM Support Vector Machine

SMO Sequential Minimal Optimization

SGD Stochastic Gradient Descent

MLP Multilayer Perceptron

CCI Correctly Classified Instances

ICI Incorrectly Classified Instances

TPR True Positive Rate

FPR False Positive Rate

KS Kappa-statistics

MAE Mean Absolute Error

RMSE Root Mean Square Error

RAE Relative Absolute Error

RRSE Root Relative Squared Error

LWL Locally Weighted Learning

ROC Receiver Operating Characteristic

SSE Sum of Squared Error

SVR Support Vector Regression

PCC Pearson Correlation Coefficient

DIT Depth of Inheritance Tree

FRS Final Reuse Score

Chapter 1

Introduction

1.1 Overview

In 1968, the NATO science committee sponsored a conference that discussed all software development and software engineering discipline concerns. Some of these concerns tackled the massive gap between the users' expectations and the developed software. Also, the developed software lack meeting the requirements' specifications and suffered from budget overrun. Furthermore, it took more time than scheduled to complete the software projects (Naur and Randell, 1968). It has been over a half-century since the NATO conference, but still, these concerns persist.

Moreover, they became more complicated nowadays due to the growth of software project complexity. Typically, developing software takes a lot of time and effort. Recruiting more workforces may significantly increase the project cost. Large-scale software projects or critical software such as space navigation systems require even more time and budget since any software defect could be catastrophic.

Therefore, the software development industry faces a huge demand to develop fully competent software in minimal time. The massive growth of the software industry and the increase of systems complexity obstruct fulfilling the market needs. Furthermore, the rapid growth of software complexity and technology hinder delivering efficient and reliable products. Moreover, software engineering standards are neglected during development due to market pressure and delivery schedules. As a result, the developed software lacks quality and functionality. Therefore, there is a demand to adopt practices and techniques to enhance software development productivity while minimizing the software project constraints, cost, budget, and schedule.

In response to this demand and to improve the deliverables of the software projects, software reuse became a prevalent practice during development, especially after the introduction of Object-Oriented (OO) programming (Jalender et al., 2010). Nowadays, software is developed from existing software components and artifacts rather than implementing from scratch. Software reuse cuts down the development time and increases productivity. Also, software reuse reduces effort and work repetition since many software components have already been developed. Moreover, software reuse has a positive influence on the project’s budget. Also, it uplifts the software reliability since the behavior of the reused component is already known. Nevertheless, reuse assets are error-prone since they are tested repeatedly. Moreover, reused assets can be isolated instantly when any fault occurs.

According to Lehman’s laws of software evolution (Lehman, 1996), software changes are inevitable. The software’s complexity will continue to grow until a software replacement is more cost-effective than maintaining the existing one. Furthermore, it is trivial to implement complex software from scratch. Therefore, reusing well tested software assets would increase software development productivity and reduce the project cost. In addition to that, the surging growth Open Source Software (OSS) has offered a wide selection of reusable software components (Lampropoulos et al., 2018). Although software reuse increases the software dependability and reduces the error margins, selecting reusable software components can be critical. In 1996, during flight 501 of Ariane 5, the rocket veered the flight path at an altitude of 3700 miles and exploded. The investigation concluded that the cause of failure was a bug in a reused source code in the flight control system (Lions et al., 1996). Therefore, assessing software assets is for reuse is very crucial and must be systematic.

1.2 Research Background

Software reuse is a practice that identifies and utilizes pre-existing software artifacts that have similarities to software project requirements or design—these artifacts could be reused instead of developing the required components from scratch. Reuse software components help in improving software projects’ productivity and performance while elevating the software quality attributes. Software reuse as a practice requires several vital features to accomplish its purpose. First, it has to be systematic to exploit the similarities application and the requirements. Also, knowledge competency is required so software reuse can efficiently contribute to achieving project goals. Moreover, the software reuse mechanism needs to be well defined to get the full advantage of software reuse (Ezran et al., 2002).

Software Reuse was first proposed by (McIlroy et al., 1968). The authors suggested that examining the software components' variability could help to develop software components inventories. Software reuse was formally acknowledged as a discipline in the eighties after a conference sponsored by ITT (Biggerstaff and Perlis, 1989), and the publication of the software reusability book (Programming, 1983). In the eighties, five major reusability topics were considered state of the art. These topics are reusing modules, reusing common systems programs, reusing architecture, reusing design, and reusing data (Jones, 1984). In 1991, (Latour et al., 1991) proposed a reference model called "3Cs". The proposed model promotes the separation between the implementations and specifications to emphasize the software component's reusability. The software component consists of three aspects. Figure 1.1 presents the model's aspects which are:

- Context describes how the component is used; it is represented as the parameters.
- Concept defines the component; it corresponds to the abstraction and formal methods of the component.
- Content specifies how the component works, representing the implementation of the component.

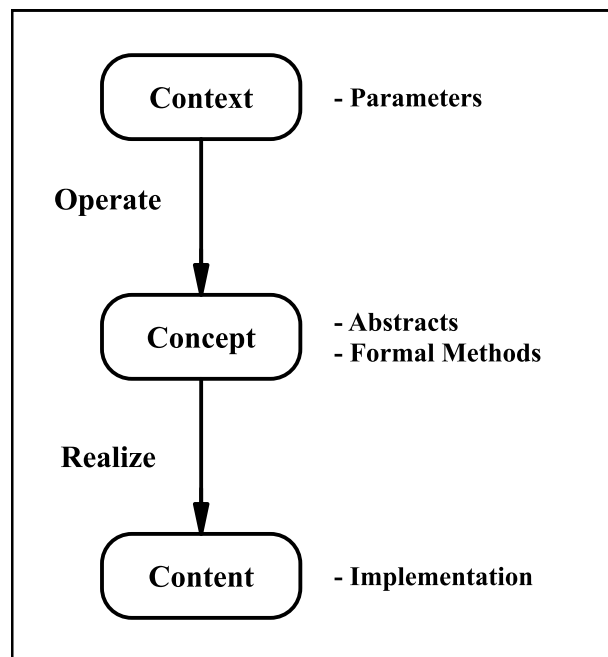


Figure 1.1: 3Cs Model of Software Component

To achieve the most significant benefit from reusable components, they must be designed to work in a specific domain. Also, when designing a reusable component, the changed

aspects must be identified and the method that will demonstrate the unchanged elements. The 3Cs model became a blueprint of software reuse researches in the past decades.

Mainly, there are two types of software reuse: horizontal reuse and vertical reuse. Horizontal reuse focuses on reusing software components in a broad spectrum of application domains. The usage of the third-party application also the Commercial Off-The-Shelf (COTS) within the large-scale applications are examples of horizontal reuse. Also, the applicability of this type of reuse is high since the software components are generic. Meanwhile, vertical reuse aims to reuse within a specific domain or applications that are closely related. Therefore, it focuses more on developing product lines (Jalender et al., 2010),(Anasuodei et al., 2021).

Furthermore, the motivation and planning of reusing software components classify software reuse into two types. These types are systematic reuse and opportunistic reuse. The systematic reuse may influence the reuse-driven software development model. This model considers software reuse while building the software component. Meanwhile, opportunistic reuse opposes systematic reuse in terms that it is unplanned. Usually, it occurs when the software developers realize the similarity of an existing artifact with the pursued functionality (Barros-Justo et al., 2018).

1.3 Research Problem

Developing software is considered an intellectual task that requires analytical and critical thinking due to its complexity. Moreover, software development goes through various activities such as requirements analysis, software design, implementation, testing, and maintenance. With the rapid increase of technology complexity, these activities became more challenging and time-consuming. Furthermore, the high demand for business digitization has created a massive gap between the market demand and the software development progress. There is debate in the software engineering discipline about accelerating the software development progress effectively without compromising the software quality.

Software reuse was introduced as a practice that would help to revolutionize the software development process. However, the absence of a defined software reuse strategy in most software development organizations contributed to the failure of software reuse to become a standard practice in the software engineering discipline. Although the software development organization believes that software reuse would influence software development productivity, there is no standard practice that manages software reuse. Also, the non-existence of an assessment model that determines the reusable software artifact. This assessment model

could help the software professionals to select the reusable software assets without depending on the software developers' skills and experiences.

Reusing improper software assets may lead to software faults and failure. As mentioned earlier, the destruction of Ariane 5 in 1996 was due to faults in the spaceship Inertial Reference System (IRS). The cause of this accident was an integer overflow caused by converting the velocity data from a 64-bit floating-point to a 16-bit integer. Also, the flight system did not disclose a clause that handles this type of error value. This IRS was reused for its predecessor Ariane 4 (Lions et al., 1996). Another incident is the Therac-25 accidents; Therac-25 was a radiation therapy unit used to treat malignant tumors. Six patients received a massive overdose of radiation given by Therac-25 between 1985 and 1987. The radiation unit reused the same software control used in the radiation unit predecessors Therac-6 and Therac-20. The overdose accidents have caused death to four patients plus severe injuries to others. Failure to thoroughly test the reused software since its development teams consist of one programmer only prevented detecting software bugs (Leveson and Turner, 1993). A recent incident was the Equifax data breach which happened in 2017. Equifax, one of the largest credit reporting agencies in the United States, has announced a data breach of the personal information of 148 million of their consumers. The company gathers and analyzes consumers' information to create reports and credits scores; these reports are sold to third parties. The cause of the data breach was the failure to update a reused open-source Java library (U.S. House of Representatives Committee on Oversight and Government Reform, 2018).

Lesson learned from these examples; the importance of a software reuse assessment framework that helps identify the reusable assets and understand the impact of reusing software components. Also, some factors are related to the project settings, such as projects' staffing may lead to software reuse failure. Therefore, the assessment framework must consider technical and non-technical factors to ensure its success. Also, the framework would help to get full acknowledgment of reusable assets and ease the reuse decision-making.

1.4 Research Questions

Machine Learning (ML) has widely penetrated every aspect of our lives. ML algorithms are highly utilized in decision making, recommendation, data analysis, prediction, and much more. Therefore, ML and intelligent solutions are considered a hot topic nowadays. According to (Dimensions, 2021), the last decade witnessed a significant increase in ML publications. Nonetheless, the publications concerned ML and software engineering, as shown in Figure

1.2 and Figure 1.3.

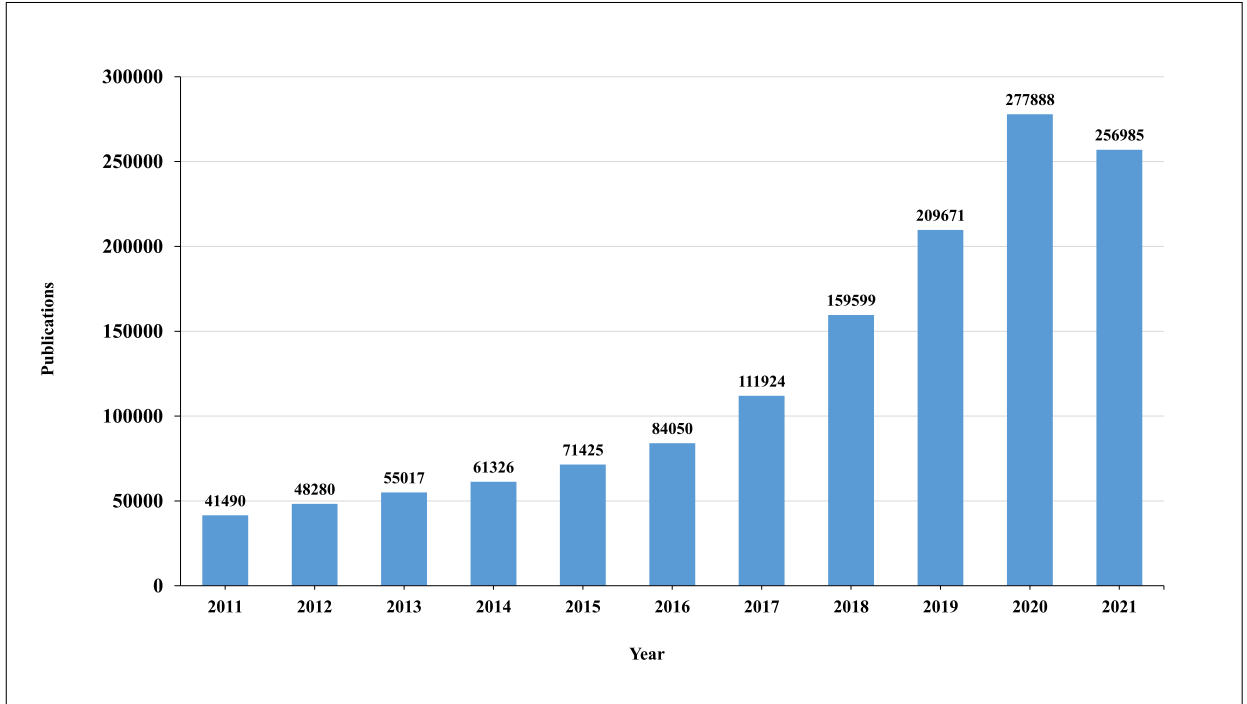


Figure 1.2: Publications Statistics which included "machine learning" as a keyword from 2011 to 2021

In this study, we are trying to find two software reuse assessment methods using ML. We structured the research questions to help achieve our research objectives. The research questions are as the following:

RQ1. What are the ML potential in distinguishing successful and failure software reuse experiences?

RQ2. Can feature selection algorithms improve the prediction of successful software reuse experiences?

RQ3. What is the most relevant feature subset that would improve the successful software reuse experience prediction?

RQ4. How does ensemble learning perform in predicting successful software reuse experiences?

RQ5. What type of analysis could be used to estimate software artifacts' reusability?

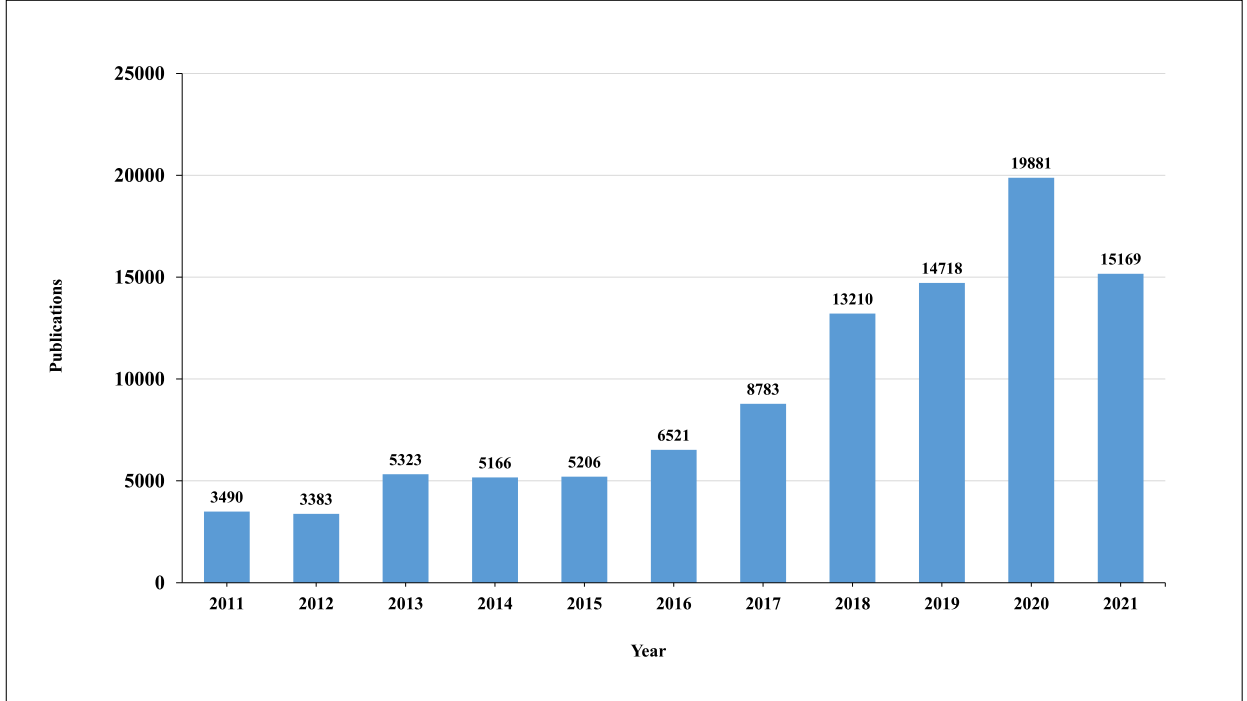


Figure 1.3: Publications Statistics which included "machine learning" and "software engineering" as keywords from 2011 to 2021

RQ6. Can the GQM paradigm be used as a feature selection technique?

RQ7. What are the metrics which can be used to model a software reusability estimator?

RQ8. How can data clustering improve the software reusability estimation?

1.5 Research Objectives

The primary aim of this research is to introduce ML models. These models can be utilized in the software artifact reuse assessment framework. The utilization would lead to a systematic reuse assessment. Nevertheless, the proposed models do not look at the technical aspect of the software asset but also highlight the importance of non-technical factors. Therefore, we need to investigate the non-technical factors which play a significant role in the software reuse experiences. In addition to that, find a type of analysis that could help to identify software artifact reusability.

The previous section listed the question which should be answered as a contribution to the software reuse literature. These questions have split our objectives into two groups. The

first group is the objectives related to predicting software reuse experience. These objectives are the following:

- O1.** Investigate the current state-of-art related software reuse, ML.
- O2.** Study ML algorithms and feature selection techniques.
- O3.** Determine the most appropriate ML algorithm that could be used as a classifier for the feature selection technique.
- O4.** Study the used dataset attributes and extract the subset of the most relevant attributes, which can improve software reuse experience prediction.
- O5.** Test the performance of the proposed prediction model.

Meanwhile, the second group of research objectives is related to the modeling of a software reusability estimator. These objectives are the following:

- O6.** Explore the state-of-art of estimation model, GQM, and regression algorithms.
- O7.** Investigate the effectiveness of static metric analysis in estimating software components' reusability.
- O8.** Study the used dataset and extract the used features using the GQM paradigm as a feature selection technique.
- O9.** Investigate the effectiveness of data clustering in estimating software components' reusability.

1.6 Research Significance

Reusing software assets became a prominent practice during software development. Software reuse has a positive influence on the project's budget and productivity. Also, it uplifts the software reliability since the behavior of the reused component is already known. Nevertheless, reuse assets are error-prone since they are tested repeatedly. Moreover, reused assets can be isolated instantly when any fault occurs.

The primary aim of reusing software assets is to increase software development productivity. Since software reuse saves the time, effort, and cost needed to develop the required components. When a software reuse decision is taken, the initial task is to search and assess appropriate reusable software components. However, this task can be challenging since the

assessment task directly depends on software developers' experience. The sole dependence on human knowledge and expertise could increase the failure of finding and assessing software components. Therefore, there is a constant demand for assessment models that assess software components without relying on software developers' expertise.

The acquisition and adaptation of reusable components help develop and deploy complex software rapidly with low engineering costs and effort. In the industrial domain, software reuse positively impacts software development productivity. Also, successful software reuse enhances the software quality and maintainability. Practitioners can use the proposed models to measure their software assets' reusability and assess the success of software reuse of their projects. Therefore, software reuse promotes effective resource management and improves project execution performance. These benefits are also applied while assessing software components' reusability. Also, many pieces of literature have proposed methodologies and approaches to assess software reusability. However, few of these pieces of literature promote the introduction and the importance of the reuse assessment framework in the software reuse cycle.

When assessing a software component, it is essential to establish an assessment framework. In this study, we propose two assessment models. These models focus on the non-technical and technical aspects during an assessment. Also, both models utilized ML. The first model predicts the success of the software reuse experience by assessing the project settings. Meanwhile, the second model estimates the software components' reusability by analyzing the component's static metrics. The proposed model can be introduced as a part of the assessment framework.

1.7 Research Setting

According to (Hasselbring and Giesecke, 2006), research methods in the software engineering discipline can be either quantitative or qualitative. The quantitative study aims to explore or investigate a specific scenario and evaluate the results. Experiment, survey, case study are examples of quantitative research methods. Meanwhile, the qualitative study reason the effect of a particular scenario, such as group discussion or interviews. In addition to that, (Basili, 1993) believes that software engineering pieces of literature fall into two research paradigms. The first paradigm is the empirical paradigm, while the second is the analytical paradigm. The researches in the empirical paradigm follow a scientific method. The scientific method validates a hypothesis by analyzing the observations of experimenting with a proposed model

or theory. On the other hand, the analytical paradigm includes the mathematical method, which focuses on developing theory and deriving results.

Our study aims to propose two models and evaluate the effectiveness and accuracy of these models. Therefore, this research is considered quantitative research that follows the empirical paradigm and the scientific method. Also, the thesis will be organized as follows: Chapter 2 reviews literature related to software reuse. The literature review focused on the effect of software reuse, software reuse success and failure factors, software reuse approaches, ML utilization, and software reusability assessment and estimation. Then, Chapter 3, which presents research methodologies for the proposed models and how the empirical studies were conducted. Also, the chapter depicted the used datasets, used algorithms, and defined the used evaluation criteria. The next chapter, Chapter 4, evaluates the proposed models by presenting the experimental results and details some threats that compromise the validity of this study. Moreover, the chapter elaborates a discussion that focuses on answering the research question. Finally, Chapter 5 summarizes the work done in this thesis and presents some recommendations and possible future work.

1.8 Summary

This chapter provided an overview of the thesis topic, which focuses on software reuse assessment. Also, it illustrates the background of software reuse and its standpoint in the software engineering discipline. This chapter also highlighted the importance of the assessment phase while finding reusable software artifacts. Further, the chapter presents the thesis question, which this thesis intends to answer to achieve the thesis objectives. Finally, the chapter summarizes the research method and paradigm which will be adopted in this thesis.

Chapter 2

Literature Review

2.1 Overview

The main objective of this chapter is to review pieces of literature concerned with software reuse, machine learning, and software reusability estimation. The literature review would help set a background on recent software reuse publications and identify the direction of this thesis. The chapter highlights the influence of software reuse on software quality. Furthermore, it depicts some factors that play a role in the success and failure of software reuse. Moreover, this review focuses on machine learning utilization in software reuse prediction and reusability estimation.

2.2 Effect of Software Reuse

Software reuse improves software quality. (Mateen et al., 2017) conducted a survey to find the impact of the software reuse approaches on software quality attributes. This approach mixes different software reuse, such as component-based reuse and ad hoc reuse approaches. The survey focused on the influence of software reuse on the quality of the Software Requirements Specifications (SRS) and the software design specification. (Mohagheghi and Conradi, 2007) assessed software reuse effects in the industrial context. The authors reviewed industrial pieces of literature, which were published between 1995 and 2005. The primary aim of this study is to evaluate and measure how software reuse influences software development productivity and the project deliverables quality. Moreover, the authors pursued identifying the metrics used to evaluate the software reuse approaches and their effect on the projects. (Mateen et al., 2017), and (Mohagheghi and Conradi, 2007) concluded that software reuse

increased software productivity and quality. In addition to that, the authors in (Mohagheghi and Conradi, 2007) believed that the wide variety of software metrics and the absence of well-defined metrics made the evaluation and comparison of any solution that could solve software issues more difficult. In general, reviewing the literature is considered an excellent method to find the effect of any used approach. However, the failure to detect literature or focus on a particular publication channel could lead to the failure of reviewing essential publications. Also, most of the researcher reports the successful experiences and when results are significant. Furthermore, the work in (Cau, 2016) examined and compared the quality of a real-time geometric measurement prototype before and after software reuse. The author used the standard model ISO 25000 to evaluate the prototype quality. The study findings support that software reuse has increased software quality. Moreover, it showed a marginal improvement with the prototype defect density after reusing source codes. Also, there was a significant improvement in the prototype’s functional completeness after software reuse. The author focused on the software reuse effect on a specific type of software. The effect may software reuse may differ in other types of software. (Mohagheghi et al., 2004) studied the impact of reused components on a large-scale telecom system by monitoring the modification rates. They found that reused components tend to have less defects-density by 51% than the non-reused components. Furthermore, the study showed that reused components have lower modification rates than non-reused components. The low modification rate indicated that reused components are more stable than non-reused components. However, the study did not consider the functionality and complexity differences between the reused and non-reused components. Similar work presented in (Imoize et al., 2019) explored the positive and negative impact of software reuse by reviewing previous works. The exploratory study concluded that reuse could boost the development process as well as reduce the project budget. However, a reused component that lacks documentation can be costlier during maintenance.

Software reuse reduces software development costs. (AL-BADAREEN et al., 2021) studied software reuse decision-making based on the cost of specific scenarios for software development and reuse purposes. Also, the authors depicted the types of software components that are applicable for reuse. Furthermore, the reuse cost was estimated by analyzing a dataset with the cost of developing software components for reuse seven purposes. The study showed that developing a new reusable software component from scratch might cost more than developing a standard software component. However, reusing an acquired reusable software asset might reduce the project cost by about 23%. The work in (Krüger and Berger, 2020)

compares the cost of two software reuse strategies. These strategies are platform-oriented and clone and own by interviewing 28 practitioners from a large-scale organization (Axis) that integrates both reuse strategies. In addition to systematically reviewing 57 pieces of literature. The objective was to identify the activities and practices adopted when reusing a software component, the cost of adopting these activities and practices, cost factors that may influence the reuse strategy. The study compared the data collected for the interviews and literature. Also, it discussed the variance between the collected data. The authors concluded a similar result as (AL-BADAREEN et al., 2021) which is that developing reusable components is more expensive than developing a single-use software component. Also, in practice, reuse strategies are adopted in combination to a variant extent rather than adopting a single reuse strategy. (Patrick, 2020) hypothesized that software reuse cost is directly affected by the software understandability and indirectly by software complexity. Furthermore, the study associated problem reports with software complexity and support requests with software understandability. The author proposed Lexical ANalysis for LAbelling iNquiries (LANLAN) approach that uses ML and word embedding. The classifier’s training data was extracted from a Question and Answer (Q&S) forum. The classifier had to distinguish whether the software reuse cost was influenced by software understandability or software complexity. Also, the proposed approach would help explore the relationship between software reuse and reported issues. The proposed classifier performed very well with a ROC curve of about 0.9. In addition to that, the author mined the dataset’s features and found that half of the support request instances were initiated for complex software. This finding contradicts the author’s hypothesis.

Software reuse may introduce some security vulnerabilities. (Gkortzis et al., 2021) investigated the relationship between software reuse and security vulnerabilities. The authors conducted multi-case studies which analyzed the effect of reusing source codes in 1244 OSS projects. The results showed that, in general, large-scale projects are more likely to be exposed the security threats. However, these threats can be equally associated with reused source code or native source codes. Also, there is a proportional relationship between security vulnerabilities and dependencies numbers. Similar work in (Kula et al., 2018) statistically analyzed 4659 projects in GitHub to investigate the regularity of libraries dependencies updates and their influence on security vulnerabilities. The authors studied the projects’ tracking systems and libraries updates and interviewed the concerned developers. The analysis findings showed that although projects relied heavily on libraries dependencies, only 18.5% of these projects have updated dependencies. In addition to that, the study showed that 69% of the

interviewed developers were not aware of the security threats of reusing source codes. In (Ponta et al., 2018), an approach was proposed to exploit and mitigate reachable security vulnerabilities caused by reusing OSS libraries. The proposed approach combined dynamic and static assessment, which analyzed the call graph of an application. A call is not considered high risk if the native source code did not call a vulnerable component. The proposed approach was evaluated on 500 applications. The evaluation showed that the approach is viable and effective.

There have been security vulnerability incidents related to software reuse. For instance, the Heartbleed was announced independently by a Finnish software testing organization called Codenomicon and Google’s researchers. The incident occurred in 2014 and was caused by a defect in an OpenSSL extension called Heartbeat. This defect allowed to over-read the buffer of a heap memory; as a result, external users were able to exploit confidential data and encryption keys. About 500,000 active web servers were affected. Google, Facebook, and Amazon servers were among these servers (Harris, 2018).

2.3 Software Reuse Success and Failure Factors

Domain analysis is one of the factors that can lead to successful software reuse. (Poulin, 1999) depicted the impact of successful software reuse experiences and its lesson learned. Also, the author presented the effect of the domain analysis on software reuse success. The study found that projects could benefit up to 80% of reusing assets when domain analysis is performed. The work in (Sierra et al., 2019) performed domain analysis by comparing two video game engines, Unity and P5, to develop core assets for Software Product Line (SPL) for hearing rehabilitation video games. The study showed that P5 was a better option to develop these unique kinds of video games. Moreover, the domain analysis helped to identify the most appropriate video game engine SPL. The SPL can be reused to build various video games successfully. However, domain analysis can help to reuse specific purpose software.

There are different factors than domain analysis that could have a positive impact on software reuse. (Lucrédio et al., 2008) surveyed 57 software development organizations to identify the key factors that influence successful software reuse. The authors concluded that assigning a dedicated team for reuse development and developing family products play a significant role in software reuse success. This survey has evaluated the key factors of software reuse success in the Brazilian software development organization. A geographical factor influence could affect the survey outcomes. In addition to that, some factors may contribute to soft-

ware reuse failure. (Frakes and Fox, 1995) surveyed software development practitioners and researchers from the USA and Europe. They indicated that software reuse might fail in some domains, such as aerospace. Moreover, the participants believe that developers' education, quality of software assets, application domain, economic understanding of reusing software, and the organization's development process could affect the systemic software reuse. The work in (Doublait, 1997) reviewed software reuse practices in a software development company called Sodalia. This company considers software reuse as a strategic practice that could help to develop high-quality telecommunications software. The company's strategic objective is to lead the software reuse production market in Europe. According to (Doublait, 1997), lacking management commitment and adhering to systematic software reuse contributes to the failure of the software reuse experience. Since the management commitment helps allocate qualified human reuses and adequate funding to achieve the organization's strategic objectives. Furthermore, the wrong assignment of the software reuse team could cause the failure or limiting the pursued objectives. Missing practical software component assessment framework plays a role in software reuse failure.

2.4 Software Reuse Approaches

There are two approaches to software reuse. The first approach is the preplanned approach. The second approach is the pragmatic approach. The preplanned approach considers reusing during software development. (Gaffney et al., 1992) presented an economic model for pre-planning software reuse. The authors concluded that designing with reuse consideration is not cost-effective. As well as, the preplanned artifacts might be feasible for a specific domain. On the other hand, the pragmatic approach does not consider reuse during software development. According to (Walker and Cottrell, 2016), pragmatic reuse is a frequent practice in development activities and found it as a valuable practice even though many academics are dismissing it. The survey conducted in (Walker and Cottrell, 2016) showed that software developers lean towards pragmatic reuse, but they investigated the original system carefully before attempting to reuse it. The work in (Holmes and Walker, 2012) proposed a model that uses pragmatic reuse plan to transform the source code from the originating software and integrated it into the developer's software semi-automatically. Their proposed model made pragmatic reuse systematic, less risky, time-efficient, and more disciplined.

Software reuse is not exclusive to source code. (Morisio et al., 2002) conducted structured interviews with 19 software development companies. The authors concluded that source code,

design, and requirements were reused in 24 industrial projects. The work in (Pacheco et al., 2015) proposed a model that utilizes the requirements catalog to reuse these requirements. The proposed model, Requirements Reuse for Software Requirements Catalog (RRSRC), has successfully obtained reusable requirements in small-scale software development organizations. However, the model requires further improvement as well as testing in large-scale development organizations. (Biddle et al., 2002) proposed a web-based tool that supports reusing essential use cases. They concluded that the reusable source code underlies a reusable use case.

Software reuse is considered a goal in modern development methodologies. For instance, in SPLs, (Abbas et al., 2020) proposed an approach that recommended reusing software assets based on customer requirements. The proposed approach links the new requirements with existed assets in the requirements repository using Natural Language Processing (NLP) and ML algorithms. The evaluation showed that the proposed approach could match new requirements with existing assets with about 74% accuracy. In (Ali et al., 2020), The authors proposed a reuse system. This system was integrated into the DevOps process to manage and maintain the reuse repository. Also, the proposed system would ease the process of reviewing, identifying, and retrieving the reusable assets. A controlled experiment was conducted to evaluate the effectiveness of the proposed system. The results showed that the proposed system performed very well but needed further evaluation in more complex situations. (Lahouij et al., 2020) proposed an approach that helps preserve the reliability of the composite cloud services. The proposed approach focuses on finding and reusing the appropriate substitution if any composite service component is unavailable. The authors' evaluation concluded that the proposed approach had reconfigured the composite cloud services successfully. The work in (Alves et al., 2008) proposed a framework that used the vector space model to investigate the capability of Information Retrieval (IR) in identifying the commonalities in requirements specifications. The evaluation of the proposed framework was promising. The authors suggested that using a mining structure may improve the results.

Software reuse strategies may vary between embedded and non-embedded software. (Varnell-Sarjeant and Andrews, 2015) compared the outcomes of reusing five development approaches in 43 works of literature. The authors concluded that the outcomes of non-embedded software were more positive than embedded software. Also, their review showed that reuse in embedded systems could not be extended to non-embedded systems. The work in (Parkhomenko et al., 2016) investigated software reuse in remote laboratory development. The exploratory study sums up that reused components in embedded systems are costlier than creating new

components.

2.5 Machine Learning Utilization

ML was leveraged in various ways in software reuse. (Negi and Tiwari, 2020) assessed the reusability of open source OO source code using ML. The authors generated a dataset using Chidamber and Kemerer (CK) metrics from selected source codes. Random Forest algorithm was trained on CK metrics to assess reusability software components. The experimental results concluded that the Random Forest algorithm has a high potential in assessing components' reusability with accuracy reaching about 99% and a correlation coefficient of about 0.85. Similar work in (Sandhu et al., 2020) predicted software component reusability by proposing an algorithm that integrates gradient boosting and random forest. The results conclude that the proposed algorithm outperformed eight supervised ML algorithms by 23%. In (Prakash et al., 2012), data mining was used to evaluate OO software components for reuse. The experiments showed that data mining could be useful in assessing software reusability. A novel approach (Deepclone) was proposed by (Hammad et al., 2020). This approach uses deep learning to facilitate the reuse of code clones from a source code repository. Deepclone predicts the next required tokens based on what has been written. The evaluation of the proposed approach showed promising results. (Kaur and Kaushal, 2020) used fuzzy logic to evaluate the reusability of Aspect-Oriented Systems (AOS) at the package level. The authors validated the proposed model used by analyzing its predictions. The predictions analysis showed that the model performed correctly. The work in (Panigrahi et al., 2019) selected seven ML algorithms to estimate reusability utilizing data, which were extracted from social media repositories system. The experimental results showed the Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) had the highest precision results. A fuzzy logic approach was proposed in (Manju and Bhatia, 2021), which measures the reusability of java classes by assessing their static and dynamic metrics. The findings concluded that the dynamic metric is considered a better reusability prediction factor than the static metrics. (Mangayarkarasi, 2020) proposed an automated approach that could select reusable components by analyzing the software design pattern. The author used two ML algorithms. These algorithms are Decision Tree and Neural Network. The study's outcome the proposed approach is effective with an accuracy of about 93%.

In addition to that, reusability estimation models were proposed for all software reuse approaches. (Wangoo and Singh, 2018) proposed a model (CRePT) that uses K-Nearest

Neighbour (KNN) to predict the reuse cost and reuse level for Component-Based Software Development (CBSD). The author evaluated reuse level by the ratio of sizes of total reused components and the application components. The proposed model did not report any defects. However, the precision of the model had not been evaluated. The work in (Maggo and Gupta, 2014) proposed an automated model that assesses and measures the reusability of OO Software Components. The model uses Large Margin Nearest Neighbor (LMNN) to identify the correlation between software metrics and reusability. The evaluation of the proposed model showed that accuracy has reached about 94.2% and an error rate of 5.8%. In (Singh et al., 2015), an exploratory study was conducted to identify metrics, which could impact Aspect-Oriented Software Development (AOSD) by utilizing fuzzy logic. The study identified four primary metrics that could affect reusability. These metrics are coupling, size and complexity, Separation of Concern (SoC), and cohesion.

Ensemble ML improves the prediction model outcomes. (Zhang et al., 2020) used the Internet of Things (IoT) and ensemble learning to improve the accuracy of electricity transformers fault detection. The ensemble model evaluation showed a prediction improvement reached about 99%. The work in (Lahmiri et al., 2020) assesses the performance of five ensemble ML algorithms in classifying financial data. The experiment results showed that ensemble learning is effective and reliable in predicting corporate bankruptcy. In (Brunese et al., 2020), an ensemble learning model was proposed. This model, which utilizes voting, detects brain cancer. The results showed that the model was able to diagnose brain tumors with an accuracy of 99%. (Gao et al., 2019) proposed an adaptive voting algorithm to improve the ensemble prediction of four ML classifiers. The authors compared the performance of the proposed algorithm with the prediction performance in previous works. The results concluded that the proposed voting algorithm improved the prediction outcomes. The work in (Feng et al., 2018) studied the effect of proposed bagging-based ensemble learning on classifying imbalanced datasets. The evaluation showed that there was a significant improvement in the performance of the proposed algorithms.

Ensemble learning was utilized in the software engineering discipline. (Khuat and Le, 2019) proposed ensemble learning and sampling techniques to predict software faults. The authors split the dataset into bins; each bin had the same number of minority instances. Also, voting with majority voting as the combination rule was selected. The voting algorithms used five ML classifiers. The empirical results showed that ensemble learning improved the performance of the prediction model. In (Khan, 2020) a hybrid model was proposed to predict software defects. This model used the Adaboost and bagging algorithms as ensemble

classifiers. Results concluded that the Adaboost algorithm had outperformed the bagging algorithms. (Li et al., 2019) proposed a two stages ensemble learning approach to predict software defects. The first ensemble model was used as a data sampling technique; the second model was employed as a predictor. The proposed approach’s performance and accuracy were compared with similar approaches. The comparison results showed that the proposed approach had outperformed the other approaches by about 45%.

2.6 Software Reusability Assessment and Estimation

Many techniques have been used to assess software reusability. (Pahareeya et al., 2018) applied five ML algorithms on a public dataset to predict software reuse success. Also, the authors chose Classification and Regression Tree (CART) as a feature selection technique. The experimental results showed that ML algorithms had performed very well, especially after applying feature selection. Meanwhile, the work in (Ampatzoglou et al., 2018) used a metric-based approach. The authors introduced Reusability Index (REI), which measures seven quality factors. These factors helped in identifying the metrics used to assess the reusability frameworks and libraries. Moreover, the authors used the number of downloads as a reuse metric. Backward linear regression was utilized to aggregate the REI final value. Similar approach was used in (Kakarontzas et al., 2013) for evaluating white-box reuse in OO program objects. This approach used linear regression with CK metrics to measure the java classes’ reusability. The evaluation of proposed reusability assessment models in (Ampatzoglou et al., 2018), (Kakarontzas et al., 2013) provided an accurate assessment of the targeted software components. Both models could be used to assess software reuse during software development. (Sharma et al., 2009) used Artificial Neural Network (ANN) to measure software component reusability. The authors chose four metrics to assess software reusability. These metrics are portability, interface complexity, customizability, and understandability. The experimental results showed that the ANN-based model was able to predict software reusability with acceptable accuracy. However, the proposed model is applicable only for black-box reuse.

Various software reusability estimation techniques assessed CK metrics. For instance, (Padhy et al., 2019b) proposed a model that utilizes ANN, Adaptive Genetic Algorithm (AGA), Extreme learning machines (ELM), Levenberg Marquardt (LM), and CK metrics to estimate the reusability of Web of Service (WoS) software. The evaluation showed that the proposed techniques had outperformed other reusability prediction techniques with an

accuracy of 97.71%. The work in (Shri et al., 2010) used clustering and CK metrics to predict software reusability. The developed prediction model scored high precision results. A comparative study in (Zahara et al., 2013) evaluated the software reusability by assessing CK metrics using four regression algorithms. The experimental results should that a tuned Instance Based K-Nearest Neighbour (IBK) had the best error rate and correlation coefficient.

Several pieces of literature have tackled software reusability estimation by analyzing software metrics other than CK metrics. (Fazal-e Amin and Oxley, 2011) proposed a software reusability assessment model for SPLs. The authors used GQM approach to extract the input for the proposed reusability assessment model. The evaluation results showed that the proposed assessment had performed very well. Also, the evaluation summed up that the selected attributes have a significant impact on software reusability. The work in (Papamichail et al., 2019a) has estimated software components reusability for java classes and packages using binning and polynomial regression. The authors collected a dataset that has 28 attributes. These attributes were grouped into six primary groups. These groups are coupling, cohesion, documentation, size, complexity, and inheritance. The evaluation of the estimation model showed that the proposed model had outperformed the estimation model in (Fazal-e Amin and Oxley, 2011).

2.7 Software Metrics and The GQM Model

Object Oriented Metrics (OOM) are quantitative source code measuring technique. (Deshpande et al., 2021) proposed a ML classifier that utilizes seven ML algorithms and OOM to predict software reliability. The evaluation proposed classifier showed that Random Forest had scored the best performance results. Also, the work in (Padhy et al., 2019a) used OOM and proposed an estimation model that measures the complexity of C++ and Python software projects. The authors analyzed the OOM of forty C++ and Python programs. The findings showed that Python is more suitable for Object Oriented Programming (OOP), and it had a better OOM than C++. (Kaur and Sharma, 2019) experimented prediction of software faults with ANN. Experimental results showed that the proposed ANN model had scored an accuracy of about 92.5% and performance reached 0.92. (Malhotra and Khanna, 2018) proposed a model that recognizes classes prone to changes by analyzing the OOM and evolution-based metrics. The authors introduced the collected dataset from two Android packages changes logs to seven ML models. Also, the experiments evaluated the prediction of the class change prone in four scenarios. The scenarios evaluated the prediction using

evolution-based metrics solely, OOM solely, OOM and evolution-based metrics in conjunction, and the classes' internal probability of changes. The experimental results showed that combining OOM and evolution-based metrics had the best prediction results.

Many pieces of literature have proposed GQM models for software quality evaluation. The work in (Ergasheva et al., 2019) proposed a GQM model that identified the software metrics that could be used to evaluate adaptive systems quality. The goal of the proposed model focuses on the success of completing the project and its phases. Also, improve the development's processes quality, adapt effective testing methodologies, and improve the effort assessment process. (Ito et al., 2017) proposed a requirements analysis approach that combined the GQM model and requirements analysis techniques. The GQM was utilized to pinpoint current software issues and identify the metrics to solve these issues. The evaluation of the proposed approach proved the effectiveness of the GQM in identifying the problematic areas and developing appropriate metrics to resolve these problems. In (Song et al., 2020), GQM model and ML were used to explore the relationship between source metrics and software quality. Also, the authors proposed a quality assessment model based on the extracted metrics. The proposed model was about 75% accurate.

The software reuse-related incidents showed that these incidents were not only a result of unnoticed coding problems but also weaknesses in the software development and project management practices. The works of literature either measured the component's reusability or studied the factors of the success or failure of software reuse. Still, they have never been introduced or proposed as software reuse assessment frameworks that focus on software development practices. Machine learning was utilized in various software reuse models; however, to the best of our knowledge, ensemble learning algorithms were not used in distinguishing software reuse experiences. Also, we have noticed that ensemble learning is actively used in software defect prediction. In addition to that, the proposed reusability estimation used ML algorithms as a feature selection technique; meanwhile, we intend to test the effectiveness of the GQM model as a feature selection technique.

2.8 Summary

This chapter presented some previous work that is related to software reuse. The review of the literature helped in our investigation related to the state of the art of software reuse and how machine learning can be utilized in software reuse experience prediction and estimating software artifact reusability. In addition to that, we achieved our research objectives **O1** and

O6.

Chapter 3

Methods and Procedures

3.1 Overview

The primary objective of this study is to utilize ML algorithms to predict successful software reuse. Therefore, this study tackles ML utilization in two parts. The first part is predicting successful software reuse experiences. This prediction focuses on the non-technical elements that would influence successful software reuse, such as the organization size, employees training, project settings. Meanwhile, the second part of this study focuses on the technical aspects of software reusability. In this part, a software reusability estimation model was proposed. This model assesses the reusability of java classes by analyzing their static metrics.

3.2 Prediction of Software Reuse Experience

It is necessary to evaluate the ML potential in predicting successful software reuse. Also, eliminate any irrelevant features that might affect the ML model performance. Therefore, this first part of the study consists of three stages. Figure 3.1 presents the study stages. The first stage assesses the ML potentials in software reuse experience prediction. The second stage is extracting the most relevant attribute using feature selection algorithms. The extracted features would improve the prediction model performance. The final stage is developing the prediction model, which uses ensemble learning to predict successful software reuse.

Figure 3.2 depicts the proposed models. According to Figure 3.2, the first proposed ensemble prediction model utilizes the full dataset to predict software reuse experience. In the second model, features selection technique was introduced to the ensemble prediction model, as shown in 3.2. The purpose of introducing feature selection to the proposed ensemble pre-

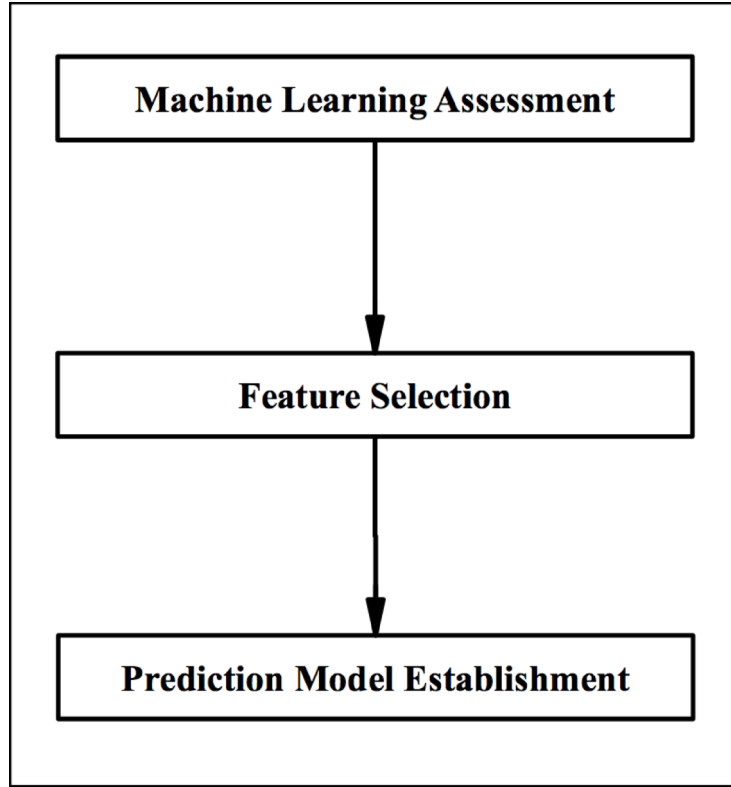


Figure 3.1: Prediction Model Development Stages

diction model is to extract the most relevant features from the dataset. Since, the redundant and irrelevant features increase the data dimensionality without adding new information to the dataset could negatively affect the performance of the prediction model.

3.2.1 Used Dataset

The used data was taken from a public repository (Promise Repository, 2004). This dataset contains data from 24 projects, constructed by interviewing 19 software development companies that introduced software reuse in their industrial projects. The interviews were guided by a questionnaire that was given to the interviewees. Furthermore, the interview questions focus on collecting information about the companies such as business domain and the number of staff. Also, information related to the reuse at the organization level includes management commitment to software reuse and the employees' training and awareness about software reuse. In addition to that, reuse processes and assets. The features of predicting software reuse fall into three main categories. The first category is features related to the development process, such as the software staff, type of software. The second category is features related

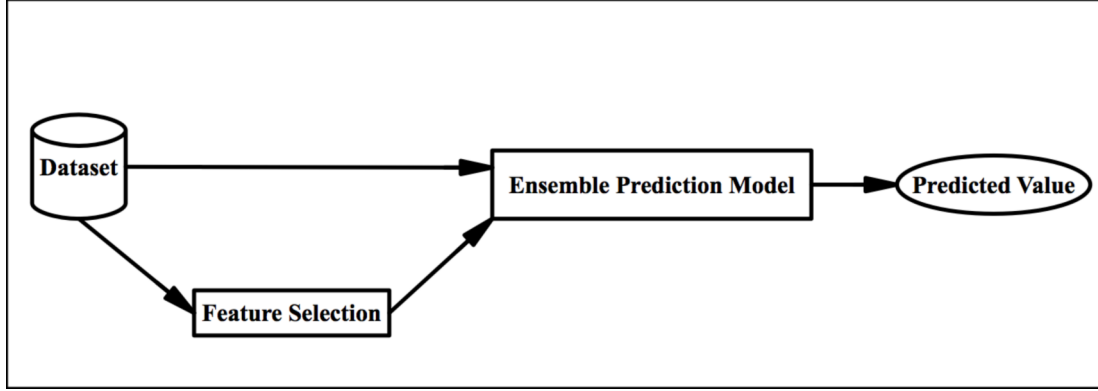


Figure 3.2: Proposed Ensemble Prediction Models Overview

to the organization, such as the overall staff and application domain. The last category is features related to the reuse process, such as reuse role, human factors, and reuse introduction (Morisio et al., 2002).

Table 3.1 presents statistical information about the used dataset. The dataset is classified into two classes, successful software reuse and failure software reuse. The number of instances with successful software reuses is 18 instances, about 65% of the dataset instances. Meanwhile, the total number of failure instances is eight, around 35% of the dataset instances. The reason for selecting this dataset is its uniqueness. This dataset is the only dataset we could find that considers the key factors which contribute to the success of the software reuse experience.

Table 3.1: Dataset Statistical Details

Statistical Component	Statistical Information
Number of the instances	23
Number of the attributes	24
Number of successful software reuse instances	15
Number of failure reuse instances	8
Percentage of successful software reuse	65.22%
Percentage of failure software reuse	34.78%

Furthermore, the dataset attributes were divided into groups based on the company's control over these attributes. These groups are state variables and control variables. Any attribute that requires weeks to change was considered as a state variable. Table 3.2 presents the state variables of the used dataset, the value of the attributes, and the attributes' descrip-

tion. The software development companies have no control over them. As shown in Table 3.2, the attribute type of software production can be either product-family, which refers to, related projects or isolated that are unrelated projects. The dataset authors defined a product-family company that develops software that may evolve with time. Meanwhile, the product company develops software that shares anything in common. According to (Morisio et al., 2002), fifteen projects from the 20 projects were developed by product-family companies have succeeded in reusing software assets. Meanwhile, none of the product companies have succeeded in reusing software.

Another attribute that is the application domain holds various domains. The dataset covered 13 application domains. These domains are Air Traffic Control (ATC), bank, book-keeping, engine_control, Flight Management System (FMS), manufacturing, measurement, Software Tools (SE-Tools), Space, telecommunication (TLC), Train Simulation (TS), and Train Traffic Control (TTC). TLC and manufacturing application projects are more likely to succeed in reusing software (Morisio et al., 2002). Moreover, the attribute type of software can be technical, which means control software that is non-embedded, non-real-time with limited database, or business that is database software non-embedded, non-real-time database software. Embedded-RT means embedded, real-time software type; meanwhile, non-embedded-RT describes non-embedded, real-time software, as shown in Table 3.2.

The second attribute group is the control variables, which are influenced the company's process of decisions. These variables are divided into high-level control and low-level control. Table 3.3 presents the control variable, possible attribute values, and the attributes' description. According to (Morisio et al., 2002), most of the projects that introduced the high control variables had a successful software reuse experience. Also, projects that have a loose reuse approach were successful. In addition to that, companies with higher-grained assets are more likely to succeed in their projects. Projects 51-100 or 100+ assets have a higher number of successful software reuse experiences.

The low-level control group consists of 10 attributes with different field ranges. Table 3.3 shows that the reuse approach, which describes the coupling of reused assets that can be either tight or loose. Another attribute that is work products define the reused assets. These assets can be a combination of Code (C), Requirements (R), or Design (D). The attribute origin defines whether the reused components were developed from scratch (Ex-novo), reused as it is (As-is), Reengineered (Reeng). Meanwhile, the high-level control in management decisions for software reuse. The value of the eight high-level control attributes shown in Table 3.3 can be either yes or no.

Table 3.2: Description of Dataset State Variables

Attribute	Attribute Values	Attribute Description
Software staff	S: (Small)	Software development team is small which about 1- 50 team members
	M: (Medium)	Software development team is medium which can between 51- 200 team members
	L: (Large)	Software development team is large and ranges from 201 and more
Overall staff	S: (Small)	Company's employee is small which ranges from 1 to 50 employees
	M: (Medium)	Hired employees is considered medium which can be between 51 and 200
	L: (Large)	Overall staff is large and ranges from 201and 500
	X: (Extra-large)	Company's staff is extra-large which is more the 501 employees
Type of software production	Isolated	There is no commonality between the company's project
	Product Family	There is commonality between the company's project
Software and product	Product	The developed software is embedded inside a product
	Alone	The developed software is a standalone product
	Process	The developed software embedded inside a process
SP maturity	Low	The Capability Maturity Model (CMM) level of the software process is 1 or it is not ISO9001 certified
	Middle	The CMM level of the software process is 2 or it is ISO9001 certified
	High	The CMM level of the software process is 3 or more
Application domain	TLC	Telecommunications
	SE-Tools	Software tools
	Bank	Bank application
	Engine Controller	Engine controller application
	FMS	Fire management application
	TS	Train simulation application
	ATC	Air traffic control application
	Manufacturing ⁷	Manufacturing application
	Measurement	Measurement environment control and management application
	Finance	Finance application
	TTC	Train Traffic Application
	Space	Aerospace application
	Book-keeping	Book keeping application
Type of software	Technical	The develop software neither embedded nor real-time with limit or no database
	Business	The develop software neither embedded nor real-time but database intensive
	Embedded-RT	The developed software is embedded and real-time
	Non-Embedded-RT	The develop software is not embedded but real-time
Size of baseline	S: (Small)	The size of reused assets in a projects is less than 10KLOC and take 10 person-months of effort
	M: (Medium)	The size of reused assets in a projects is between 10KLOC and 100KLOC and take effort of 10 to 100 person-months
	L: (Large)	The size of reused assets in a projects ranges between 100 to 500KLOC and take more than 100 person-months of effort
Development approach	OO	Object oriented development approach
	Proc	Procedural development approach
Staff experience	Low	The experience of project members is one year or less
	Middle	The experience of project members between 2 and 4 years
	High	The experience of project members is more than five years

Table 3.3: Description of Dataset Control Variables

Control Level	Attribute	Attribute Values	Attribute Description
High	Top management commitment	yes	Whether the top management are committed to adopt software reuse as a prominent practice
		no	The reuse initiative came from the middle management or the individuals
	Key reuse roles introduced	yes	Indicator whether any reuse role was given to a project team member
		no	
	Reuse processes introduced	yes	At least one software reuse process was introduced
		no	
	Non-reuse processes modified	yes	The development process which are not related to reuse were modified
		no	
Low	Repository	yes	Whether the software assets is established are assets are stored in repository
		no	
	Human factors	yes	Whether the staff were aware and trained about software reuse
		no	
	Reuse approach	Tight	Reusable product the tightly coupled cannot be isolated
		Loose	Reusable product the loosely coupled can be isolated
	Work products	D+C	Type of the reusable assets Design D, Requirement R, Code C
		C	
		R+D+C	
	Domain analysis	yes	Whether is the baseline of the product was established by domain analysis or not
		no	
	Origin	ex-novo	Software assets were developed from scratch
		as-is	Software assets were reused as it is
		reeng	Software assets were reengineered from existing product
	Independent team	yes	Indicator that there is a dedicated reuse team with the project to develop the reusable assets
		no	
	When assets developed	before	The reusable assets were developed before the project's need
		Just in time	The reusable assets were developed during the project
	Qualification	yes	There is a quality process that defined the reusable assets
		no	
	Configuration management	yes	The reusable assets are maintained by a configuration managements
		no	
	Rewards policy	yes	The existence of a reward policy that promotes software reuse
		no	
	# Assets	1 - 20	Total number of the reusable assets which are stored in the repository
		21 - 50	
		51 - 100	
		100+	

3.2.2 Machine Learning Algorithms

ML algorithms were leveraged in all prediction model stages, as shown in Figure 3.1. The following subsections describe the algorithms used in each stage.

3.2.2.1 Machine Learning Assessment

In this stage, six ML algorithms were used on the chosen dataset. These algorithms are:

- *Naive Bayes (NB)*: NB is a classification algorithm based on Bayes Theorem. The algorithm assumes that features occurrences are independent of each other. This algorithm is simple and relatively fast in real-time and optimal for text classification (Murphy and others, 2006).
- *ANN*: ANN classification uses a Multilayer Perceptron (MLP) classifier, multilayer feed-forward neural network learning. The neural network consists of an input layer, one or more hidden layers, and an output layer. The learning process starts with the input layer and ends with the output layer. Dataset attributes are weighted and fed simultaneously from a layer to the next layer (kumar and Sahoo, 2012).
- *K-Star*: K-star is an instance-based learner, which classifies the instances according to similarity. The closest the instances, the more similar they are. The algorithms use entropy to measure the distance between the cases (Painuli et al., 2014).
- *Hoeffding Tree (HT)*: HT constructs an incremental decision tree that splits attributes mathematically using the Hoeffding bound. Hoeffding bound attains prescribed confidence level by running a certain number of instances. This algorithm can handle massive data learning very well (Ghogare, 2019).
- *Support Vector Machine (SVM)*: SVM uses Sequential Minimal Optimization (SMO) algorithm, which transfers all nominal attributes and null values into binary ones. Then, the algorithms try to identify a margin that divides the data into different classes (Luts et al., 2010).
- *KNN*: KNN is a simple instance-based learning classifier. The algorithm calculates the distance between the test set and the training sets. Then, the test set gathers with the nearest neighbour to create a class (Cover and Hart, 1967).

3.2.2.2 Feature Selection

An optimal subset of valuable attributes helps in improving the performance and the results of any ML model. Feature selection is a technique that chooses a subset of the most ef-

fective attributes from the original dataset. This subset can increase the efficiency and the effectiveness of the prediction model. Also, it prevents data overfitting.

There are two main methods for selecting features. The first method, which is the wrapper method, evaluates the dataset's features by inducing them to an algorithm. The algorithm's role is to assess the features and produce a subset of valuable features (Kohavi and John, 1997). Meanwhile, the second method that is the filter method, does not require an induction algorithm. It depends on the general characteristics of the training set to extract the relevant features (Langley, 1994). In this stage, three filter method algorithms were tested. These algorithms are the Classifier Attribute Evaluation, Correlation Attribute Evaluation, and InfoGain Subset Evaluation. Also, three wrapper method algorithms were evaluated, which are Wrapper Subset Evaluation, Classifier Subset Evaluation, and CfsSubset Evaluation.

3.2.2.3 Prediction Model Establishment

Ensemble learning is a ML methodology that combines multiple learning classifiers. The primary purpose of combining ML classifiers is to eliminate the errors and risks when using a single classifier (Sagi and Rokach, 2018). Moreover, ensemble learning improves the performance of the prediction by counterbalance the weakness of a single classifier. Furthermore, the volume of the dataset may cause problems while training the classifier. For instance, it is impractical to introduce a single classifier on a large dataset. Therefore, the optimal solution is to divide the dataset into subsets where each subset is used to train multiple classifiers. Also, ensemble learning may resolve problems that are caused when using a small dataset (Polikar, 2006). In this work, three ensemble algorithms were used to improve the prediction models' performance. These algorithms are Stacking, Bagging, and Voting. The selected ensemble algorithms are described as following:

- Stacking: is a type of ensemble learning that combines multiple classification algorithms. Basically, the learning process of stacking learning consists of two stages. The first stage combines different ML classifiers $P = \{P_i(s, f), i=1, \dots, I\}$, which are called base classifiers. Each base classifier utilizes the training set f . The second stage is the meta-learning stage, where a single ML classifier M uses the outputs on the base classifiers as an input to generate the ensemble learning final outcome (Sagi and Rokach, 2018), as shown in Figure 3.3a. Therefore:

$$j_{stack}(s) = M(s, P) \quad (3.1)$$

where $j_{\text{stack}}(s)$ is the final stacking prediction of the input s . In this study, six supervised ML classifiers were chosen for the first stage of the stacking. These classifiers are HT, Random Forest, NB, Stochastic Gradient Descent (SGD), J48, and Multilayer Perceptron (MLP). Also, three ML algorithms were used as meta-learners in the second stage of the stacking. These algorithms are SGD, SMO, MLP.

- Bagging: this type of ensemble algorithms uses a single ML classifier. However, it generates different replicates from the classifier. These replicates use different subsets of instances from the dataset (Breiman, 1996). Bagging algorithms improve the prediction outcome by reducing the classifier variance and the mean square error, which will create a more stable classifier (Bühlmann and Yu, 2002). The predictor $P(s, f)$ that utilizes the training set of $f\{(j_1, \dots, j_m, s_n), m = 1, \dots, M, n = 1, \dots, N\}$ and input j to predict the value j can predict a better results if replicates of the predictor were trained on different subsets of the training set $f_k \{f_k \subset f, k=1, \dots, K\}$, Therefore:

$$j_{\text{bag}}(s) = \underset{k=1}{\operatorname{argmax}}^K (P(s, f_k)) \quad (3.2)$$

where $j_{\text{bag}}(s)$ is the maximum aggregated prediction, K is the number of training subsets. The bagging algorithm's final result is generated by plurality voting (Bühlmann and Yu, 2002), shown in Figure 3.3b. In the bagging experiments, three ML algorithms were used as base classifiers. These algorithms are SGD, SMO, and MLP.

- Voting: the vote combiner utilizes different robust ML classifiers on the same dataset f . The predictions' results are combined using a voting rule to obtain the final outcome, as shown in Figure 3.3c. Voting can provide a better output than using a single ML classifier. Therefore, it improves the prediction model reliability (Parhami, 1994). In this study, six ML classifiers were combined using voting ensemble learning to predict successful software reuse experience. These classifiers are Hoeffding Tree, Random Forest, NB, SGD, J48, and MLP. Moreover, the voting learning experiment was repeated five times for each combination rules. The selected combination rules are as following (Kittler et al., 1996):

- Average of Probabilities (AP): this voting rule calculates the mean value of each predicted class. The final predict would the class that has the maximum mean value such that

$$j_{AP}(s) = \max_{k=1}^K \sum_{m=1}^M P_m(s|f)/m \quad (3.3)$$

- Product of Probabilities (PP): the final class is class that has the maximum product value of among the products values of predicted classes. Therefore final classification is computed by the following formula:

$$j_{PP}(s) = \max_{k=1}^K \prod_{m=1}^M P_m(s|f) \quad (3.4)$$

- Majority Voting (MV): when assigning the final class using the MV the class the has the highest votes is considered the final results. Therefore,

$$j_{MV}(s) = \max_{k=1}^K P_k(s|f) \quad (3.5)$$

- MINimum Probability (MinP): in this combination rule, the minimum posterior probabilities of each class computed. Then, the final classification is the maximum value among these minimum values

$$j_{MinP}(s) = \max_{k=1}^K \min_{m=1}^M P_m(s|f) \quad (3.6)$$

- MAXimum Probability (MaxP): this combination rule assign the final classification to the maximum value among the maximum posterior probabilities for each individual. Therefore, the final prediction is calculated by the following equation:

$$j_{MaxP}(s) = \max_{k=1}^K \max_{m=1}^M P_m(s|f) \quad (3.7)$$

Where k is the number of voting classifiers, m number of predictions of each classifier, P is the posterior probabilities of the classifier, and s is the true class of the features f.

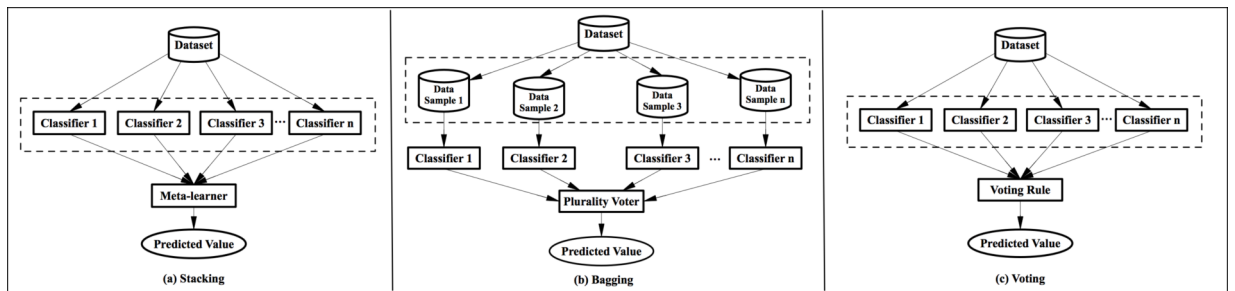


Figure 3.3: Ensemble Learning Algorithms

3.2.3 Evaluation Criteria

There are various statistical metrics to assess the performance of machine learning models. Also, evaluation criteria help to select the most appropriate machine learning algorithm for the chosen dataset. Thirteen evaluation metrics were selected to evaluate the chosen ML algorithms. These metrics are the following:

- Time Taken to Build the Model:

This evaluation criterion presents the execution time required to build the model and classify the dataset instances (Alabdulwahab and Moon, 2020). The time taken to build the model is valuable when assessing the model's effectiveness (Özdemir et al., 2019).

- Correctly Classified Instances (CCI) and Incorrectly Classified Instances (ICI):

These criteria present the number of instances that the machine learning classifier has classified correctly and incorrectly.

- True Positive Rate (TPR):

TPR also referred to as recall or sensitivity, is one of the evaluation criteria derived from the confusion matrix. TPR is the ratio of the number of correctly classified instances and the instance of actual positive instances (Sammur and Webb, 2011). TPR can be calculated using the following formula

$$TPR = TP / (TP + FN) \quad (3.8)$$

where TP is the number of true positive instances, FN the number of false negative instances.

- False Positive Rate (FPR):

False Positive Rate (FPR) is the rate at which the classifier has incorrectly classified the instances as false positives. The FPR value is the fraction of the number of false positive instances by the number of actual negative instances (Sammur and Webb, 2011). The FPR is evaluated as the following

$$FPR = FP / (FP + TN) \quad (3.9)$$

where FP is the number of false positive instances, TN the number of true negative

instances.

- Precision:

This criterion presents the ratio between the true positive instances and positive classified instances (Sammut and Webb, 2011). The following formula can derive the precision value

$$Precision = TP / (TP + FP) \quad (3.10)$$

where TP is the number of true positive instances, and FP is the total number of false positives instances.

- Kappa-statistics (KS):

KS is statistical measure the degree agreement of categorizing items in a dataset. The following equation is used to calculate KS:

$$KS = O(A) - C(A) / (1 - C(A)) \quad (3.11)$$

where O(A) is the observed agreement, C(A) is the agreement by chance. The value of KS is between 0 and 1, the higher KS value the better the classifier is performing.

- Mean Absolute Error (MAE):

MAE is an evaluation metric that measures the error of predicting the overall instances in the dataset (Sammut and Webb, 2011). This error rate can be derived from the following formula

$$MAE = \sum_{i=1}^n |y_i - x_i| / n \quad (3.12)$$

where y_i is the real target value of an instance, x_i is the predicted value of an instance, n is the total number of dataset instances.

- Root Mean Square Error (RMSE):

RMSE is a quadratic mean that measures the difference between predicted and observed data. There is an inverse relationship between the RMSE and the model's accuracy. A low value of RMSE indicated high accuracy. The following formula can estimate the following error

$$RMSE = \sqrt{\sum_{i=1}^m (P_i - \hat{P}_i)^2 / m} \quad (3.13)$$

where $\hat{P}_i$ the predicted values, P_i is the observed values, and m is the number of observations.

- Relative Absolute Error (RAE):

RAE normalizes the total absolute error by dividing it by the total absolute error of the trivial model. The following equation obtains RAE

$$RAE = \frac{\sum_{i=1}^m |\hat{P}_i - P_i|}{\sum_{i=1}^m |P_i - \bar{P}|} \quad (3.14)$$

where $\hat{P}_i$ is the predicted value, P_i is the target value, $\bar{P}$ is the mean of P_i , m is the number of instances, and $i = 1$ through m .

- Root Relative Squared Error (RRSE):

RRSE is the square root of the total of the relative error divided by the total relative error of the naive model. RRSE is calculated using the following equation

$$RRSE = \sqrt{\frac{\sum_{i=1}^m (\hat{P}_i - P_i)^2}{\sum_{i=1}^m (P_i - \bar{P})^2}} \quad (3.15)$$

where $\hat{P}_i$ is the predicted value, P_i is the target value, $\bar{P}$ is the mean of P_i , m is the number of instances, and $i = \{1, 2, 3, \dots, m\}$.

- F-measure:

The f-measure helps in assessing the performance of the prediction model. F-measure creates a harmonic mean that combines the values of the recall and precision. This combined value is used to judge whether the prediction model has a good performance or not (Sammut and Webb, 2011). The following formula calculates the f-measure value

$$f - measure = \frac{2(Precision * Recall)}{(Precision + Recall)} \quad (3.16)$$

- Receiver Operating Characteristic (ROC) area

The ROC area value shows the tradeoff between the sensitivity and FPR. It represents the threshold value, which the classifier distinguishes, the positive from negative classification. ROC area is useful to identify how to minimize the error rates. Also, it is

helpful to compare the performance between multiple classifiers (Sammut and Webb, 2011).

- Accuracy

Accuracy is the percentage in which the algorithm classified the dataset into the correct categorical labels. Accuracy is calculate using the following formula

$$Accuracy = (TS + TF)/(TS + FS + TF + FF) \quad (3.17)$$

where TS is true success reuse, TF is true failure reuse, FS is false success reuse and FF is false failure reuse.

3.3 Software Reusability Estimation Model

The subsection depicts in detail used dataset and the workflow of the proposed model for estimating java classes reusability.

3.3.1 Used Dataset

The used dataset was collected by (Papamichail et al., 2019b). This dataset was generated by analyzing the most popular project in the Maven Repository. The dataset consists of static metrics of java classes and packages in addition to the reuse rate of these components. However, this study will focus on the reusability of java classes only. Therefore, the packages' dataset will be neglected. The primary aim of this study is to explore the correlation between the classes' static metrics and their reuse rate. Also, study the effect of data clustering on the reusability estimation. Table 3.4 presents the attributes of the used dataset. According to dataset has 28 attributes. All these attributes are numeric. The attribute reuse rate is the primary attribute, which the proposed estimation reusability model will based on. Also, the dataset attributes were classified into six properties. These properties are complexity, size, cohesion, inheritance, coupling, and documentation. Moreover, the dataset has 24794 instances.

3.3.2 Used Evaluation Criteria

In this part, we have utilized machine learning algorithms. Therefore, it is required to use some statistical evaluation criteria to evaluate the performance of the used machine learning

Table 3.4: Attributes of The Dataset Used in Java Classes' Reusability Estimation

Property	Attribute Name and Description	Attribute Data Type
Size	TLOC: total number of line of codes	Numeric
	TLLOC: total number of logical lines	Numeric
	LOC: number of line codes	Numeric
	LLOC: number of logical lines	Numeric
	NG: number of getters	Numeric
	TNA: number of attributes	Numeric
	TNG: total number of getters	Numeric
	TNM: number of methods	Numeric
	TNOS: number of statements	Numeric
	TNPM: number of public methods	Numeric
Cohesion	LCOM5: lack of cohesion in methods	Numeric
Documentation	AD: documentation of the API	Numeric
	CD: the density of comments	Numeric
	TCD: the total of comments density	Numeric
	CLOC: number of comments lines	Numeric
	TCLOC: total number of comments lines	Numeric
	DLOC: number of documentation lines	Numeric
	PDA: total lines of documentation	Numeric
Coupling	CBO: coupling between the classes object	Numeric
	CBOI: inverse coupling between the classes object	Numeric
	NII: number of incoming invocations	Numeric
	NOI: number of outgoing invocations	Numeric
	RFC: response set for class	Numeric
	DIT: the inheritance tree depth	Numeric
Inheritance		
Complexity	NL: maximum level of nesting	Numeric
	NLE: maximum level of else- if nesting	Numeric
	WMC: weighted methods for each class	Numeric
Reuse Information	Reuse rate: extent which the developers select the component for reuse	Numeric

algorithms. The performance of the clustering algorithm was assessed using the Sum of Squared Error (SSE). SSE presents the summation of the squared distance between the observation and the cluster centroid (Liu, 2011). SSE can be obtained according to the following formula:

$$SSE = \sum_{s=1}^b \sum_{j \in C_s} (j - R_j)^2 \quad (3.18)$$

where b is the number of desired clusters, C_s is the cluster, R_j is the centroid of the cluster C_s and j is the observed data point.

Meanwhile, the Support Vector Regression (SVR) performance was evaluated using the three evaluation criteria. These criteria are the following:

- RMSE:

Advantages of the mean absolute error MAE over the root mean square error RMSE in assessing average model performance. However, the RMSE measures the square root of the squared difference total between the predicted and actual values (Willmott and Matsuura, 2005). The following formula calculates the RMSE

$$RMSE = \left[\sum_{s=1}^j (y_s - y'_s)^2 / j \right]^{-1/2} \quad (3.19)$$

where y_s is the actual value, y'_s is the predicted value, and j is the size of the sample.

- Pearson Correlation Coefficient (PCC):

This coefficient measures the linear correlation between pair variables of the same size. The value of the PCC could vary between -1 and 1. The sign of the PCC represents the direction of the correlation. Negative PCC indicates a negative correlation, while the positive indicates a positive correlation. However, the PCC value describes the correlation strength. The closer the PCC value to 1 or -1, the stronger the correlation. Meanwhile, the correlation gets weaker when the PCC value is closer to 0 (Benesty et al., 2008),(Pearson, 1896).

3.3.3 Reusability Estimation Modeling

This section discusses the design of the proposed reusability estimation model. This model has two fundamental elements which are the static metrics and reuse rate. Also, the model consists of four steps. These steps are feature selection, clusters generation, reusability score

generation, reusability estimation model construction, as shown in Figure 3.4. The following subsection presents the details of each step.

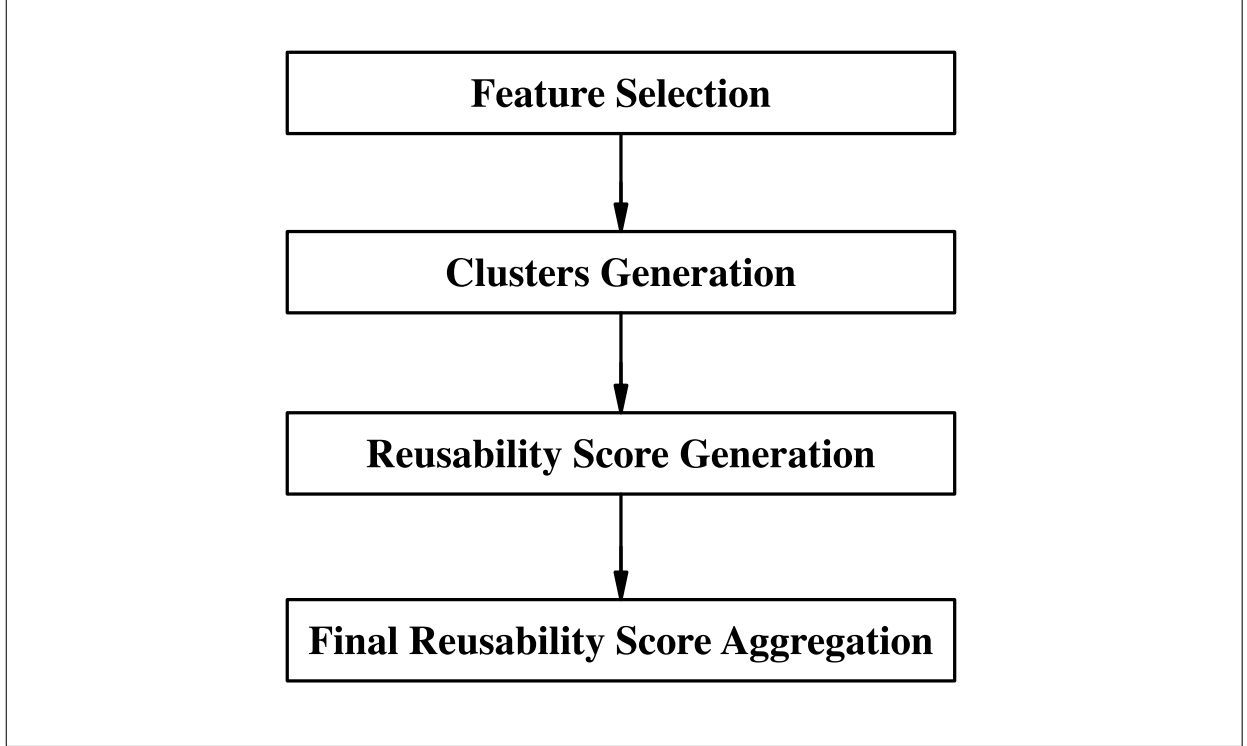


Figure 3.4: Model Design Flow Chart

3.3.3.1 Feature Selection

In this study, we selected the software metrics using the GQM paradigm. Figure 3.5 presents our GQM model when determining the metrics. The goal was to assess the reusability of Java classes by analyzing the classes' static metrics. These questions focus on the components' variability, size, dependability, and complexity, as shown in Figure 3.5. A similar model was used in (Fazal-e Amin and Oxley, 2011); however, we could not use the same model due to the difference in the dataset attributes. Software component documentation influences its reusability. Well-structured and clear documentation helps the software developers while analyzing and understanding the software component. As a result, proper component documentation could increase its reusability. Also, a high cohesive class indicates that it has good modularity. Hence cohesive promotes simplicity; therefore, it has a positive impact on the class reusability. Meanwhile, high coupling enables class complexity which has a negative influence on the components' reusability. As a result, complex classes are more likely to

be non-reusable (MohanGoel and Kumar Bhatia, 2012). Furthermore, software components with a deep inheritance tree are more complicated since the high involvement of methods and classes. However, a high Depth of Inheritance Tree (DIT) produces more reusable software components (Manoj and Nandakumar, 2016).

Goal: Measure java classes reusability from the viewpoint of the software developer. Question: How cohesive are the class elements? Metric: LCOM5 Question: How complex is the java class? Metric: NL, WMC Question: Is the component tightly or loosely coupled? Metric: CBO Question: Is the class well documented? Metric: TCLOC Question: How independent is the class? Metric: DIT Question: What is the size of the class? Metric: TLOC, TNOS Question: What is the variability of the class? Metric: TNM, TNA, TNPM Question: To what extent the class is imported? Metric: Reuse Rate

Figure 3.5: Selected Features Using GQM Model

3.3.3.2 Clusters Generation

The selected dataset consists of independent attributes. Therefore, an exploratory data analysis helps to understand how these attributes and instances are related. The primary aim of the exploratory data analysis is to identify how reuse rate is related to the remaining dataset's attributes. Furthermore, the dataset has a wide range of reuse rates that are between 0 and 3964. Table 3.5 depicts a statistical summary of the reuse rates. According to Table 3.5, about 72% of the dataset instances have a zero-reuse rate. Meanwhile, the number of instances with the highest reuse rate is 1, about 0.36% of the dataset.

Moreover, there are similarities between static metrics regardless the reuse rates are high or low. Table 3.6 illustrates the static metrics of two instances, one with 0 reuse rate while the other instance has the highest reuse rate of 3964. The two instances almost have similar static metrics. However, their reuse rate is different. Therefore, there is a need to group the

Table 3.5: Reuse Rate Statistical Information

Reuse Rate Interval	Number of instances	Percentage
0	17932	72.320%
[1,10)	5468	22.054%
[10,20)	546	2.202%
[20,30)	223	0.899%
[30,40)	145	0.585%
[40,50)	92	0.371%
[50,60)	60	0.242%
[60,70)	46	0.186%
[70,80)	37	0.149%
[80,90)	29	0.117%
[90,100)	21	0.085%
[100,1000)	186	0.750%
[1000,2000)	8	0.032%
≥ 2000	1	0.004%

dataset instances based on reuse rate to avoid the effect of similarity between the static metrics and the gap between the reuse rates while generating the reusability scores.

Data clustering was utilized to recognize the similarity patterns among the dataset instances. Also, data clustering would help to prevent any negative interference of unrelated instances. In this study, density-based clustering algorithm was used. Many clustering algorithms assume that a particular type of probability distribution was used to generate the data. As a result, these algorithms produce spherical shape clusters. Also, density-based clustering can identify and eliminate outliers and noises (Aggarwal and Reddy, 2013).

The used dataset was clustered using the WEKA Python. The algorithm was set to act on the reuse rate. The algorithm generated ten clusters with a SSE of 0.0433. Table 3.7 presents the dataset clusters and the number of instances their percentage. Cluster 1 has about 86% of the dataset instances; Cluster 1 has a high number of dataset instances since most of these instances have a reuse rate value of zero. As aforementioned, about 72% of the dataset instances hold a zero reuse rate shown in Table 3.5. Meanwhile, Cluster 5 has only one instance, as shown in Table 3.7. Therefore, Cluster 2 and Cluster 5 were merged into one cluster while establishing the reuse score since cluster 5 has one instance.

Table 3.6: Comparison of Two Dataset Instances

Metric	Metric Value	Metric Value
LCOM5	1	1
NL	3	3
WMC	62	68
CBO	9	5
TCLOC	104	93
DIT	0	1
TLOC	372	343
TNA	26	11
TNM	23	43
TNOS	129	132
TNPM	3	36
Reuse Rate	3964	0

Table 3.7: Dataset Instances Distributed Among Clusters

Cluster #	Number of Instances	Percentage
1	21244	85.682%
2	12	0.048%
3	278	1.121%
4	51	0.205%
5	1	0.004%
6	141	0.056%
7	2497	10.070%
8	532	2.145%
9	15	0.060%
10	23	0.092%

3.3.3.3 Reusability Score Generation

The second step is to translate the metric values of each cluster into a reuse score. The estimation of the reuse score is based on the distribution reuse rate over the full range of the static metrics. The following steps were applied for each metric in a cluster to generate the reuse score.

First, we assigned a reuse score which is the sum of the reuse rate of the java classes. Figure 3.6 depicts the reuse score of the LCOM5 metric in cluster 8. According to Figure 3.6, LCOM5 of value 1 has the highest reuse score while LCOM5 of 21 has a reuse score of 5. Theoretically, low LCOM5 indicates good cohesion, which promotes the component's reusability. Figure 3.6 showed that low LCOM5 values have higher reuse scores, which aligns with the theoretical interpretation of LCOM5. However, the reason that LCOM5 with 0 value has a lower reusability score is due to the number of dataset instances.

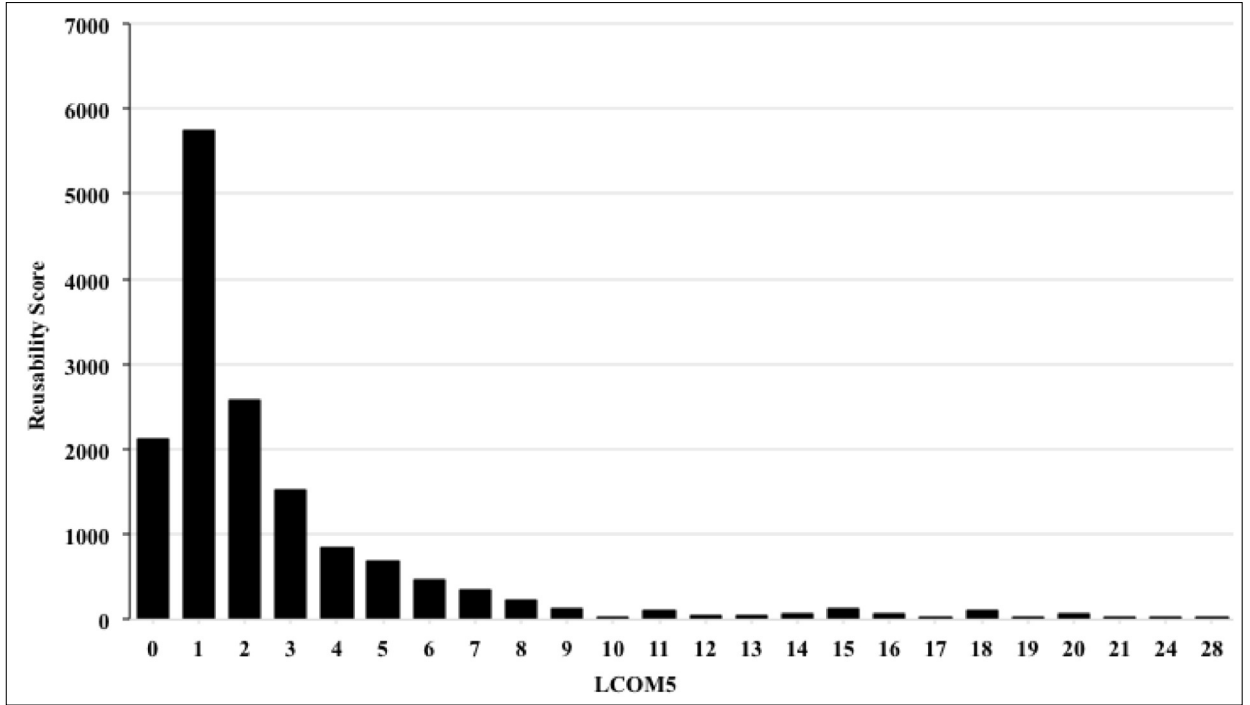


Figure 3.6: Reusability Score Distribution of LCOM5 in Cluster 8

Then, we rescaled the reuse scores after assigning the reuse score for the metric by min-max normalization. The values of the normalized reuse scores all fall within the interval $[0,1]$. The normalization would unify the value of the reuse scores for all metrics of all clusters. The normalized reuse score was calculated using the following equation

$$NRS = \frac{(RS - RS_{min})}{(RS_{max} - RS_{min})} \quad (3.20)$$

where RS is the reuse score, RS_{min} is the minimum reuse score, RS_{max} is the maximum reuse score.

Lastly, SVR was applied to the obtained dataset with the static metric value and the reuse score. The SVR utilizes the SVM to generalize the optimal tube shape training instances

with the best continuous values outputs while reducing the model’s error rates and complexity (Awad and Khanna, 2015). Tables 3.8-3.10 presents regression results of all static metrics in all generated clusters. Overall, error rates of the regression models are very minimal Cluster 9 scores the highest error rates where MAE rates vary from 0.195 to 0.104 and RMSE rates from 0.146 to 0.295. Meanwhile, Cluster 1 had the best error rates. Also, the PCC results show a wide range of correlations. The grey cells present the metrics with low correlation with reuse scores. Furthermore, DIT and reuse scores had high correlations in all clusters, as shown in Tables 3.8-3.10.

Table 3.8: SVR Results of Cluster 1, Cluster 2 & 5, and Cluster 3

Metric	Cluster 1			Cluster 2 & Cluster 5			Cluster 3		
	MAE	RMSE	PCC	MAE	RMSE	PCC	MAE	RMSE	PCC
LCOM5	0.043	0.140	0.79	0.219	0.313	0.56	0.044	0.141	0.62
NL	0.046	0.087	0.98	0.244	0.324	0.61	0.044	0.051	1.00
WMC	0.038	0.092	0.90	0.127	0.238	0.67	0.095	0.143	0.62
CBO	0.040	0.087	0.95	0.264	0.424	0.36	0.087	0.147	0.80
TCLOC	0.021	0.077	0.82	0.120	0.237	0.37	0.122	0.177	0.44
DIT	0.049	0.057	0.99	0.138	0.270	0.94	0.090	0.189	0.90
TLOC	0.038	0.079	0.89	0.121	0.253	0.01	0.078	0.129	0.42
TNA	0.039	0.090	0.93	0.182	0.249	0.82	0.050	0.102	0.81
TNM	0.034	0.092	0.87	0.131	0.248	0.29	0.082	0.148	0.59
TNOS	0.024	0.070	0.90	0.118	0.228	0.74	0.076	0.127	0.56
TNPM	0.035	0.101	0.88	0.122	0.265	0.09	0.079	0.131	0.71

3.3.3.4 Final Reusability Score Aggregation

The final process of the proposed reuse estimation model is to compute the java classes reuse score by aggregating the reuse score obtained by the previous steps. We chose to calculate the final score is using the weighted average. The reason for selecting the weighted average rather than the arithmetic average is that the static metrics have a different influence on the java class reusability. Therefore, we incorporated the PCC as the weight to ensure that the static metrics’ influence on reusability was considered. The Final Reuse Score (FRS) of the

Table 3.9: SVR Results of Cluster 4,6,7

Metric	Cluster 4			Cluster 6			Cluste 7		
	MAE	RMSE	PCC	MAE	RMSE	PCC	MAE	RMSE	PCC
LCOM5	0.091	0.221	0.50	0.072	0.169	0.81	0.045	0.138	0.81
NL	0.093	0.144	0.95	0.062	0.091	0.98	0.051	0.109	0.96
WMC	0.109	0.188	0.60	0.097	0.172	0.66	0.039	0.091	0.89
CBO	0.159	0.262	0.35	0.090	0.134	0.90	0.034	0.090	0.92
TCLOC	0.111	0.188	0.02	0.106	0.171	0.72	0.049	0.086	0.80
DIT	0.120	0.220	0.93	0.101	0.222	0.91	0.058	0.112	0.97
TLOC	0.118	0.194	0.11	0.130	0.189	0.51	0.021	0.057	0.80
TNA	0.146	0.224	0.61	0.073	0.132	0.78	0.047	0.115	0.91
TNM	0.126	0.201	0.21	0.098	0.144	0.57	0.032	0.075	0.88
TNOS	0.114	0.173	0.15	0.058	0.108	0.65	0.024	0.071	0.83
TNPM	0.153	0.238	0.09	0.115	0.162	0.58	0.034	0.085	0.90

Table 3.10: SVR Results of Cluster 8-10

Metric	Cluster 8			Cluster 9			Cluster10		
	MAE	RMSE	PCC	MAE	RMSE	PCC	MAE	RMSE	PCC
LCOM5	0.048	0.143	0.85	0.193	0.250	0.72	0.122	0.257	0.53
NL	0.037	0.048	0.99	0.195	0.246	0.82	0.113	0.181	0.92
WMC	0.053	0.101	0.79	0.189	0.266	0.56	0.133	0.249	0.32
CBO	0.066	0.136	0.89	0.122	0.193	0.72	0.158	0.223	0.63
TCLOC	0.083	0.133	0.51	0.141	0.192	0.80	0.132	0.250	0.36
DIT	0.092	0.178	0.92	0.116	0.146	0.97	0.142	0.201	0.97
TLOC	0.044	0.082	0.59	0.175	0.259	0.45	0.109	0.199	0.24
TNA	0.069	0.175	0.77	0.145	0.220	0.74	0.110	0.218	0.54
TNM	0.071	0.115	0.86	0.104	0.198	0.77	0.106	0.198	0.19
TNOS	0.054	0.117	0.67	0.111	0.200	0.74	0.111	0.194	0.25
TNPM	0.079	0.129	0.85	0.125	0.214	0.69	0.145	0.281	0.28

java class is calculated using the following formula

$$FRS = \sum_{s=1}^j (w(s) * ReuseScore_{metric(s)}) / \sum_{s=1}^j w(s) \quad (3.21)$$

where j is the number of the static metrics, $w(s)$ is the obtained cluster's PCC between the reuse score and the static metric.

3.4 Summary

This chapter presents the study methodology and the stages of the proposed model that would predict the software reuse experience. Three primary stages are machine learning assessment, feature selection, and prediction model establishment. The first stage assesses the machine learning potentials in recognizing successful software reuse experience. Meanwhile, the second stage extracts the most relevant features in the dataset. The final stage establishes prediction in the light of the findings of the aforementioned stages. Also, we demonstrate the proposed prediction model. In addition to that, the chapter illustrated the used dataset and described the dataset's attributes. The dataset attributes fall into categories which are state variables and control variables. Also, control variables may have a high or low influence on the software reuse experience. Finally, the chapter outlined the statistical evaluation criteria that were used to evaluate the performance of machine learning algorithms.

The second part of this chapter outlined the flowchart of the reusability estimation modeling. Also, it presented the GQM paradigm and how it was utilized as a feature selection technique. Moreover, the chapter showed how data clustering and regression were used to compute the static metrics' reusability scores which were aggregated to find the java classes' reusability score.

Chapter 4

Empirical Results and Discussion

4.1 Overview

This chapter aims to analyze the experimental results and presents whether the study's objectives are fulfilled. Also, evaluation assesses the effectiveness of the proposed models. In this study, we have hypothesized that machine learning has the potentials and the ability to predict successful software reuse experiences. Also, we assumed the feature selection could improve the performance of the proposed model. In addition to that, we introduced the used dataset to ensemble learning in an attempt to overcome some of the dataset weaknesses.

Furthermore, we stated that static metric analysis could be used to estimate java classes' reusability. Also, GQM model can effectively be used as a feature selection technique to identify the most relevant static metrics when assessing software components' reusability. In addition to that, we hypothesized the data clustering could improve the reusability estimation.

This chapter will present the experimental results of machine learning evaluation criteria and analyze these results. Also, the chapter evaluates the proposed estimation model by analyzing the static metrics reusability scores and comparing the proposed model with the estimation model presented in (Papamichail et al., 2019a).

4.2 Prediction of Software Reuse Experience Experimental Results

This section presents the experimental results of establishing the proposed software reuse prediction model. The following subsections illustrates assessing machine learning algorithms,

selecting feature selection algorithm, and establishing the ensemble learning model.

4.2.1 Machine Learning Assessment

In ML assessment, all experiments were conducted in WEKA 3.8.5 (Waikato, 2020). WEKA is a workbench that provides various ML algorithms and data mining. The experiments used ten folds cross-validation for performance evaluation. The experiment was performed on 23 instances. One instance (Project X) was excluded from the dataset since most of its attributes hold null values. Also, the instance was never used in the dataset analysis (Morisio et al., 2002).

Overall, the six algorithms have very close results. According to Table 4.1, NB, ANN, HT, and SVM have correctly classified the 23 instances. K-star and KNN have classified one instance incorrectly. ANN and SVM took more time while building the model that is 2.5 and 0.3 seconds, respectively. NB, ANN, HT, and SVM have given 100% accuracy. Meanwhile, K-star and KNN accuracy was 95.65%.

Table 4.1: Evaluation Criteria Related to Instances Classification

Prediction Algorithm	Time Taken (seconds)	CCI	ICI	Accuracy
NB	0.00	23	0	100.00%
ANN	2.25	23	0	100.00%
K-Star	0.00	22	1	95.65%
HT	0.00	23	0	100.00%
SVM	0.03	23	0	100.00%
KNN	0.00	22	1	95.65%

Table 4.2 shows that NB, ANN, HT, and SVM KS value is 1, which indicates that these classifiers have a perfect degree of agreement while categorizing the dataset. On the other hand, k-star and KNN KS value is 0.901, which show that they have almost perfect agreement degree. Looking at MAE and RAE values, SVM has the lowest MAE and RAE values that are 0 and 0%, respectively, which shows that SVM has the best predication accuracy while KNN has the highest MAE and RAE values that are 0.078 and 16.854% respectively.

The evaluation of prediction model shown in Table 4.3, NB, ANN, HT, and SVM have a higher precision value that is 1. On the other hand, K-star and KNN precision is 0.95. The six algorithms have the best recall and precision that is 1. NB, ANN, HT, and SVM scored the best f-measure that is 1 while k-star and KNN precision is 0.95 and f-measure is 0.97.

Table 4.2: Evaluation Criteria Related to Error Rates

Prediction Algorithm	KS	MAE	RAE
NB	1.000	0.031	6.754%
ANN	1.000	0.050	10.870%
K-Star	0.901	0.048	10.329%
HT	1.000	0.031	6.754%
SVM	1.000	0.000	0.000%
KNN	0.901	0.078	16.854%

Table 4.3: Evaluation Criteria Related to Performance

Prediction Algorithm	TPR	FPR	Precision	F-Measure
NB	1	0.00	1.00	1.00
ANN	1	0.00	1.00	1.00
K-Star	1	0.13	0.95	0.97
HT	1	0.00	1.00	1.00
SVM	1	0.00	1.00	1.00
KNN	1	0.13	0.95	0.97

4.2.2 Feature Selection Experimental Results

Feature selection experiments were executed using WEKA 3-8-5 (Waikato, 2020). WEKA is a toolkit that is used for machine learning and data mining. The used dataset has 24 attributes initially. The first attribute, which is project ID, was omitted since it used for identification purposes. Moreover, Project x was excluded since it was not considered while reviewing the dataset (Morisio et al., 2002).

In this study, six feature selection algorithms were performed in eight ML methods. These methods are JRip, Locally Weighted Learning (LWL), SGD, REPTree, Simple Logistic, OneR, Random Committee, and Decision Table. Cross-validation with ten folds was applied as a training technique. In this technique, training the ML model is repeated for ten iterations. Ninety percent of the dataset is used for training, while the rest is used for testing.

Six statistical metrics were chosen to evaluate the ML performance. These metrics are KS, MAE, RMSE, RAE, and RRSE. Furthermore, six feature selection algorithms were applied to select the most affecting features in the dataset. Three selection algorithms, Classifier

Attribute Evaluation, Classifier Subset Evaluation, and Wrapper Subset Evaluation required a classifier. Therefore, SMO was selected due to its performance in classifying the used dataset, which was tested in the previous section. Also, all experiments were conducted using default settings. The following subsections demonstrate the results of filter and wrapper algorithms.

4.2.2.1 Filter Method Algorithms

In this study, three filter method algorithms were used. For each algorithm, feature selection was repeated eight times for every ML model. Features were selected based on the best performance of the ML model with a minimal number of attributes. Table 4.4 shows the evaluation criteria before and after feature selection and the difference between them. These results were obtained using the Classifier Attribute Evaluation. Overall, KS has increased, which indicates an improvement in the degree of agreement between actual classification and the predicated one. Moreover, the average of MAE and RMSE are 0.0583 and 0.0647, respectively. JRip has experienced the most MAE and RMSE reduction by about 0.15 points. Both RAE and RRSE have also decreased by 12-13%, shown in Table 4.4. Figure 4.1 depicts the variance of MAE before and after feature selection. In general, all prediction models experienced an improvement except for Simple Logistic, where MAE has increased by 0.0072 points. Meanwhile, JRip error has reduced from 0.3168 to 0.1669.

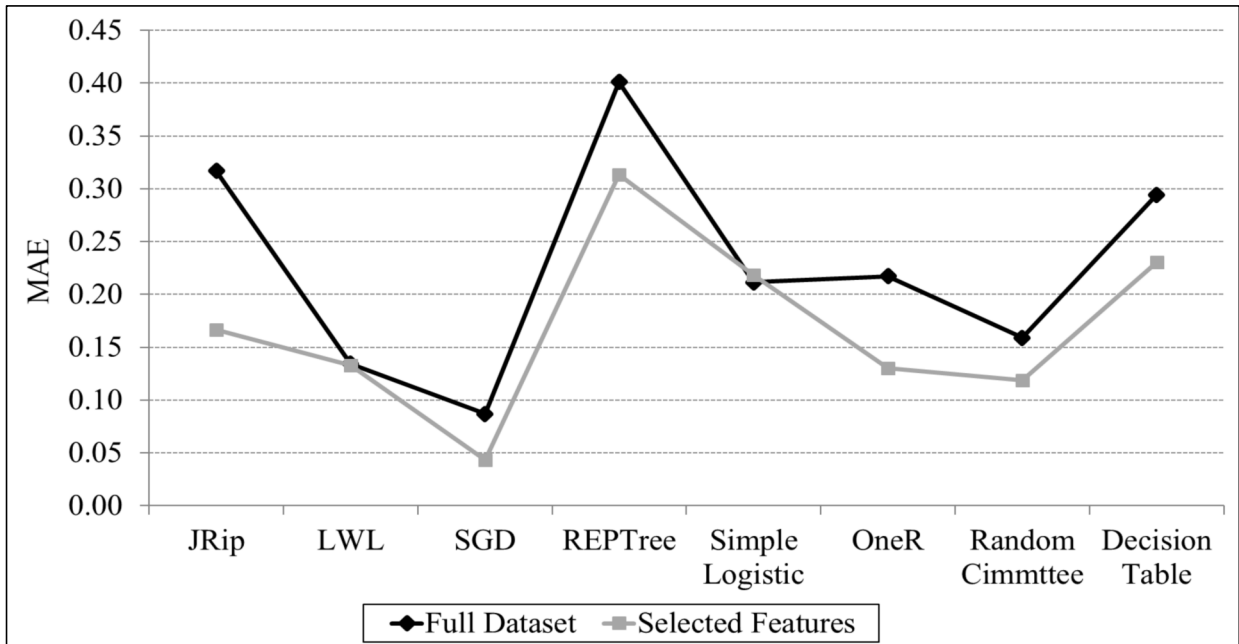


Figure 4.1: MAE Variance of Classifier Attribute Evaluation

Table 4.4: Performance Results Using Classifier Attribute Evaluation

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.7039	0.1669	0.3647	36.0177%	75.5945%
Difference	0.3949	0.1499	0.1741	32.3534%	36.0988%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.7039	0.1331	0.3612	28.7193%	74.8760%
Difference	0.0000	0.0014	0.0001	0.3166%	0.0164%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	0.9013	0.0435	0.2085	9.3843%	43.2244%
Difference	0.0930	0.0435	0.0864	9.3842%	17.9042%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.4749	0.3137	0.4764	67.7146%	98.7655%
Difference	0.2603	0.0876	0.0539	18.9026%	11.1700%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.6167	0.2187	0.3803	47.1986%	78.8343%
Difference	0.0000	-0.0072	-0.0127	-1.5463%	-2.6386%
OneR (Full Dataset)	0.5064	0.2174	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.7039	0.1304	0.3612	28.1528%	74.8669%
Difference	0.1975	0.0870	0.1051	18.7686%	21.7859%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	0.9069	0.1186	0.2083	25.5946%	43.1759%
Difference	0.0986	0.0404	0.0296	8.7300%	6.1490%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.7039	0.2306	0.3512	49.7702%	72.7964%
Difference	0.1975	0.0637	0.0812	13.7412%	16.8424%
Average	0.1552	0.0583	0.0647	12.5813%	13.4160%
Minimum	0.0000	-0.0072	-0.0127	-1.5463%	-2.6386%
Maximum	0.3949	0.1499	0.1741	32.3534%	36.0988%

Table 4.5 presents the result of feature selection using Correlation Attribute Evaluation. Looking at Table 5, all prediction models have experienced an improvement in their performance expect Simple Logistic, MAE negatively increased by 0.0183 points. Furthermore, the change in LWL was inconsiderable; meanwhile, JRip, OneR, and REPTree performances have elevated sharply. Figure 4.2 shows MAE progression before and after feature selection. There was a sharp change in MAE rate, especially JRip, OneR, and REPTree, where the error rate dropped significantly.

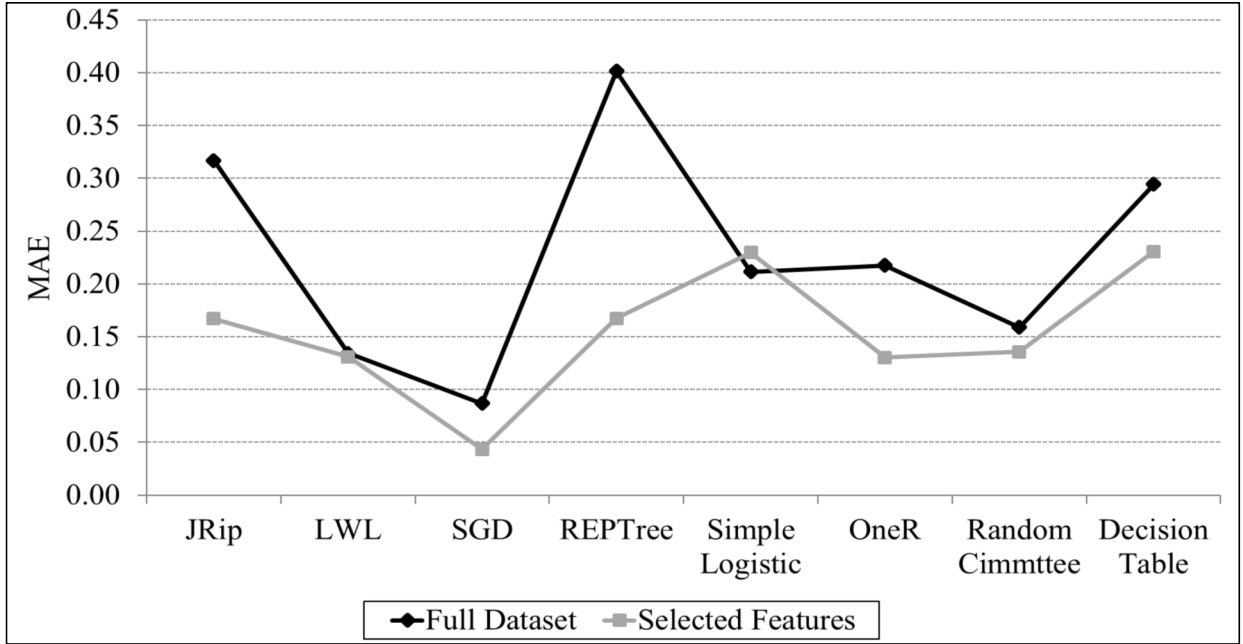


Figure 4.2: Change of MAE using Correlation Attribute Evaluation

The values are shown in Table 4.6 are the results of attribute selection using InfoGrain Subset Evaluation. Overall, there is noticeable progress in the evaluation criteria. SGD has scored the best results; KS and error rates have scored the best values. Meanwhile, Simple Logistic outcomes remain the same. For the LWL, results are slightly negative since RMSE has changed from 0.2379 to 0.2477 after applying feature selection. The progression of the algorithms can be seen clearly in Figure 4.3. As depicted in Figure 4.3, REPTree and JRip have experienced a significant decrease in MAE.

4.2.2.2 Wrapper Method Algorithms

Three wrapper methods are chosen in this empirical study. Like the filter method experiments, experiments were repeated eight times for each wrapper selection method. Table 4.7

Table 4.5: Results of Feature Selection Using Correlation Attribute Evaluation

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.7039	0.1669	0.3647	36.0177%	75.5945%
Difference	0.3949	0.1499	0.1741	32.3534%	36.0988%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.7039	0.1309	0.3612	28.2579%	74.8677%
Difference	0.0000	0.0036	0.0001	0.7780%	0.0247%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	0.9069	0.0435	0.2085	9.3843%	43.2244%
Difference	0.0986	0.0435	0.0864	9.3842%	17.9042%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.7039	0.1674	0.3648	36.1209%	75.6147%
Difference	0.4893	0.2339	0.1655	50.4963%	34.3208%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.7039	0.2298	0.3421	49.6025%	70.9241%
Difference	0.0872	-0.0183	0.0255	-3.9502%	5.2716%
OneR (Full Dataset)	0.5064	0.217	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.7039	0.1304	0.3612	28.1528%	74.8669%
Difference	0.1975	0.0870	0.1051	18.7686%	21.7859%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	0.9013	0.1357	0.2323	29.2789%	48.1538%
Difference	0.0930	0.0233	0.0056	5.0457%	1.1711%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.7039	0.2306	0.3512	49.7702%	72.7964%
Difference	0.1975	0.0637	0.0812	13.7412%	16.8424%
Average	0.1948	0.0733	0.0804	15.8272%	16.6774%
Minimum	0.0000	-0.0183	0.0001	-3.9502%	0.0247%
Maximum	0.4893	0.2339	0.1741	50.4963%	36.0988%

Table 4.6: Result Before and After Applying InfoGrain Subset Evaluation

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.7039	0.1669	0.3647	36.0177%	75.5945%
Difference	0.3949	0.1499	0.1741	32.3534%	36.0988%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.7039	0.1312	0.3612	28.3154%	74.8705%
Difference	0.0000	0.0033	0.0001	0.7205%	0.0219%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	1.0000	0.0000	0.0000	0.0000%	0.0000%
Difference	0.1917	0.0870	0.2949	18.7685%	61.1286%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.7039	0.1669	0.3647	36.0177%	75.5945%
Difference	0.4893	0.2344	0.1656	50.5995%	34.3410%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Difference	0.0000	0.0000	0.0000	0.0000%	0.0000%
OneR (Full Dataset)	0.5064	0.2174	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.7039	0.1304	0.3612	28.1528%	74.8669%
Difference	0.1975	0.0870	0.1051	18.7686%	21.7859%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	0.8083	0.1513	0.2477	33.0237%	54.2557%
Difference	0.0000	0.0077	-0.0098	1.3009%	-4.9308%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.7039	0.2306	0.3512	49.7702%	72.7964%
Difference	0.1975	0.0637	0.0812	13.7412%	16.8424%
Average	0.1839	0.0791	0.1014	17.0316%	20.6610%
Minimum	0.0000	0.0000	-0.0098	0.0000%	-4.9308%
Maximum	0.4893	0.2344	0.2949	50.5995%	61.1286%

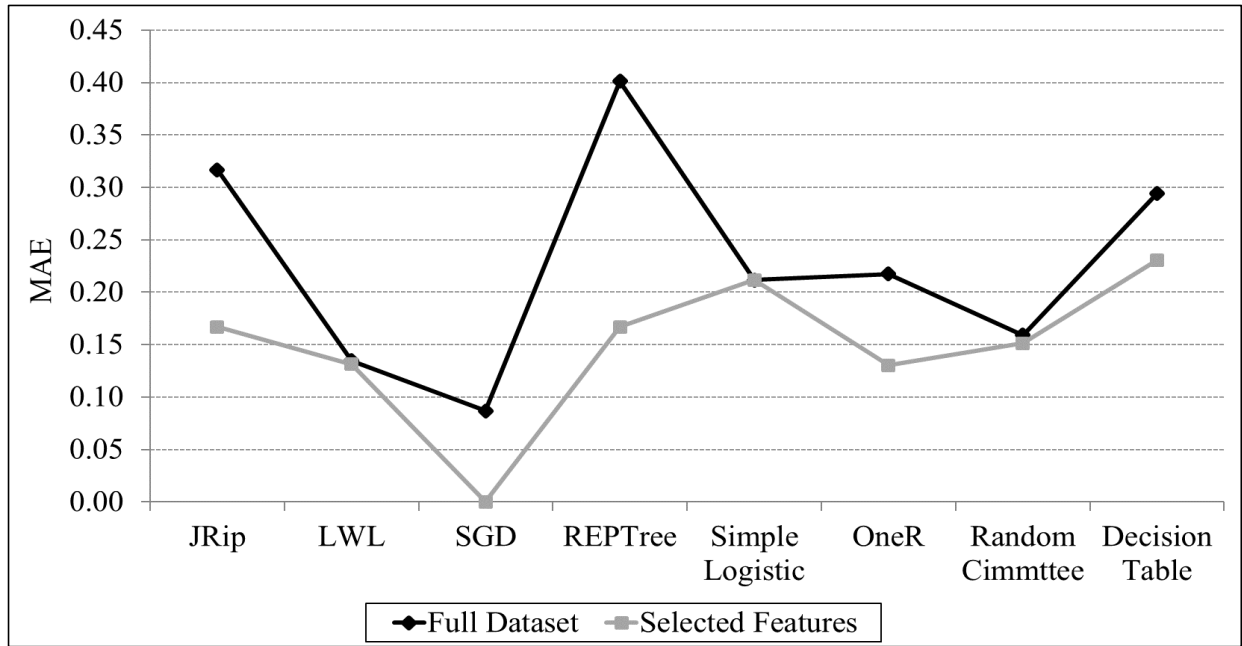


Figure 4.3: MAE Changes InfoGrain Subset Evaluation

presents the progression of the Wrapper subset Evaluation using the default settings. Overall, all ML models have shown improvement after feature selection. KS values of SGD and Random Committee algorithms have jumped to its maximum value that is 1. Also, JRip and REPTree measures have elevated by 0.5923 and 0.6867, respectively. Furthermore, MAE and RMSE decreased mainly for REPTree algorithms, where error rates have reduced by 0.3 points. Nevertheless, the error percentage dropped by nearly 67%. Figure 4.4 illustrates this improvement. The upper line shows MAE values for ML models on the full dataset; meanwhile, the lower line presents the MAE value after feature selection. The improvement is evident for all ML models.

Like Wrapper Subset Evaluation, Classifier Subset Evaluation helped in increasing the ML models' performance. According to Table 4.8, JRIP and REPTree error percentages have steeply declined by 50.4963% and 68.7424%, respectively. Also, the average of MAE and RMSE has slightly better by 0.1462 and 0.1765, sequentially. Although Simple Logistics and Random Committee scored the lowest reading yet, it considered being positive progress. Figure 4.5 provides MAE values before and after applying the Classifier Subset Evaluation. None of the ML models experience a rise of MAE value, which is considered substantial progress for all tested ML models.

Table 4.9 shows the result of the final experiment, which evaluates using Correlation-based Feature Selection. According to Table 4.9, Some predicting models' performance has

Table 4.7: Performance Results Using Wrapper subset Evaluation

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.5923	0.2340	0.3238	50.4963%	67.1273%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.1974	0.0517	0.1463	11.1611%	30.3264%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	1.0000	0.0000	0.0000	0.0000%	0.0000%
Difference	0.1917	0.0870	0.2949	18.7685%	61.1286%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.6867	0.3185	0.3153	68.7424%	65.3695%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.9013	0.1707	0.2287	36.8361%	47.4041%
Difference	0.2846	0.0408	0.1389	8.8162%	28.7916%
OneR (Full Dataset)	0.5064	0.2174	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.9013	0.0435	0.2085	9.3843%	43.2244%
Difference	0.3949	0.1739	0.2578	37.5371%	53.4284%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	1.0000	0.0188	0.0837	4.0637%	17.3486%
Difference	0.1917	0.1402	0.1542	30.2609%	31.9763%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.9013	0.1636	0.2318	35.3139%	48.0483%
Difference	0.3949	0.1307	0.2006	28.1975%	41.5905%
Average	0.3668	0.1471	0.2290	31.7475%	47.4673%
Minimum	0.1917	0.0408	0.1389	8.8162%	28.7916%
Maximum	0.6867	0.3185	0.3238	68.7424%	67.1273%

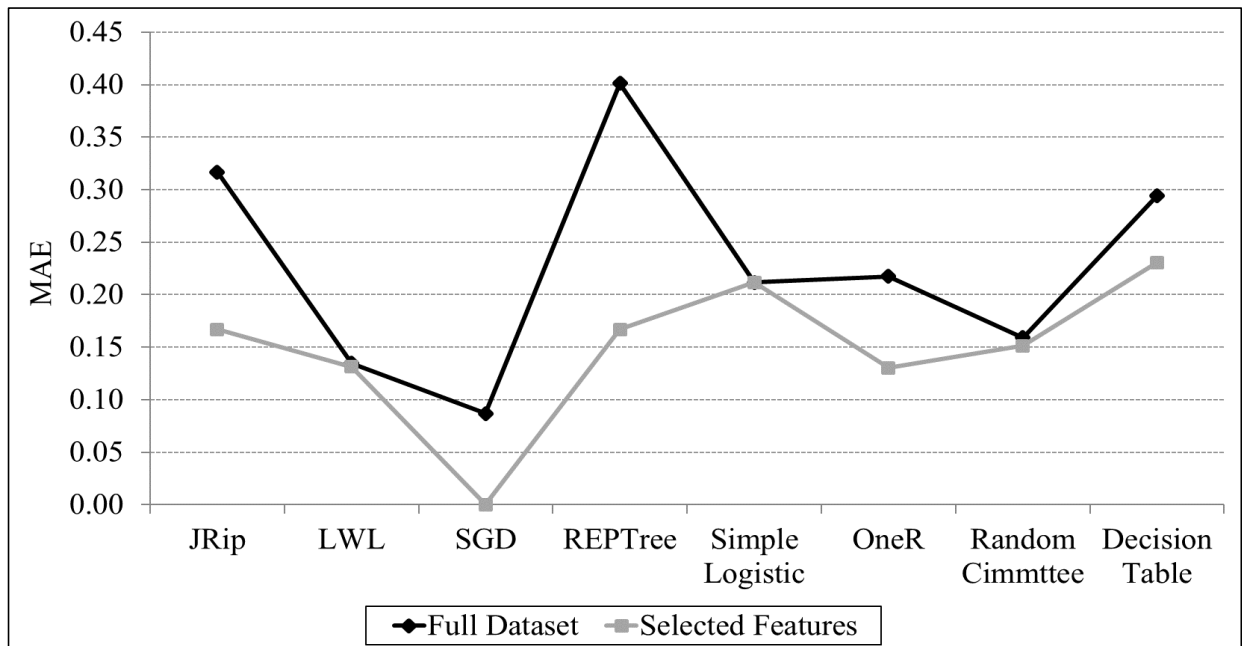


Figure 4.4: Variances of MAE Value Using Wrapper Subset Evaluation

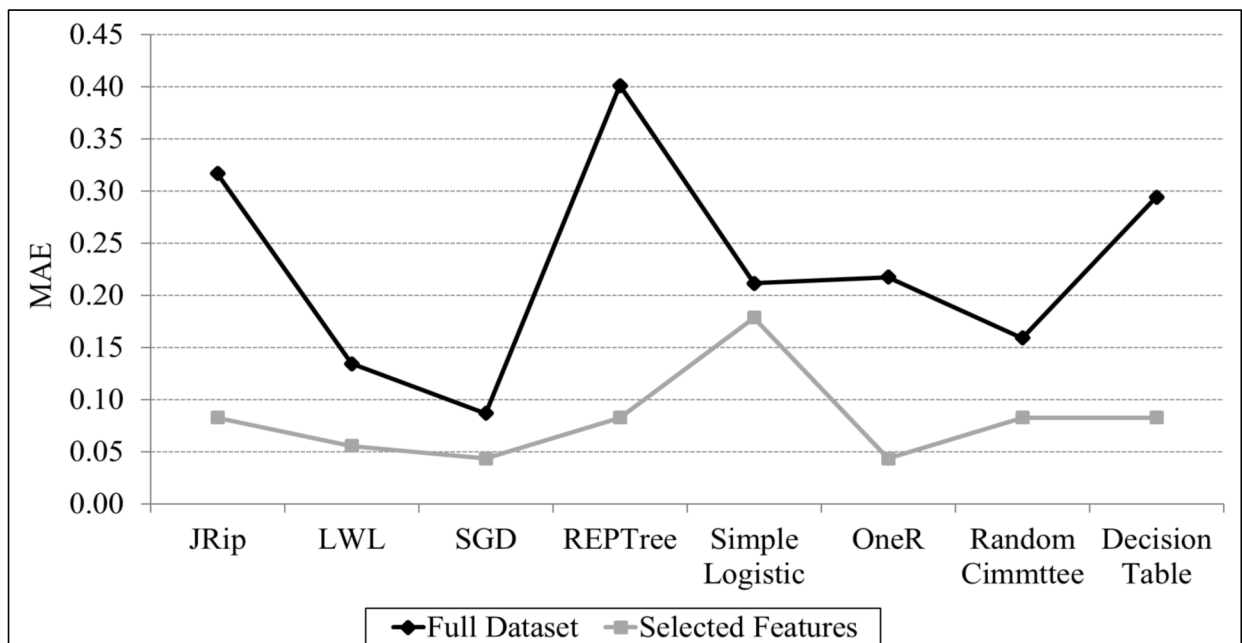


Figure 4.5: MAE Values Before and After Classifier Subset Evaluation

Table 4.8: Readings Before and After Applying Classifier Subset Evaluation

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.5923	0.2340	0.3238	50.4963%	67.1273%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.9013	0.0556	0.2091	12.0006%	43.3536%
Difference	0.1974	0.0789	0.1522	17.0353%	31.5388%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	0.9013	0.0435	0.2085	9.3843%	43.2244%
Difference	0.0930	0.0435	0.0864	9.3842%	17.9042%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.6867	0.3185	0.3153	68.7424%	65.3695%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.7039	0.1787	0.3312	38.5655%	68.6605%
Difference	0.0872	0.0328	0.0364	7.0868%	7.5352%
OneR (Full Dataset)	0.5064	0.2174	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.9013	0.0435	0.2085	9.3843%	43.2244%
Difference	0.3949	0.1739	0.2578	37.5371%	53.4284%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.093	0.0762	0.0229	16.4498%	4.7589%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.9013	0.0828	0.2150	17.8748%	44.5660%
Difference	0.3949	0.2115	0.2174	45.6366%	45.0728%
Average	0.3174	0.1462	0.1765	31.5461%	36.5919%
Minimum	0.0872	0.0328	0.0229	7.0868%	4.7589%
Maximum	0.6867	0.3185	0.3238	68.7424%	67.1273%

improved after applying feature selection except for Random Committee where got slightly worse. Meanwhile, OneR and Decision Table remains steady. Figure 4.6 illustrates the variance of MAE before and after feature selection. The change of MAE was moderate, comparing it to the previous selection algorithms. JRip, LWL, SGD, and REPTree have slightly improved, shown in Figure 4.6.

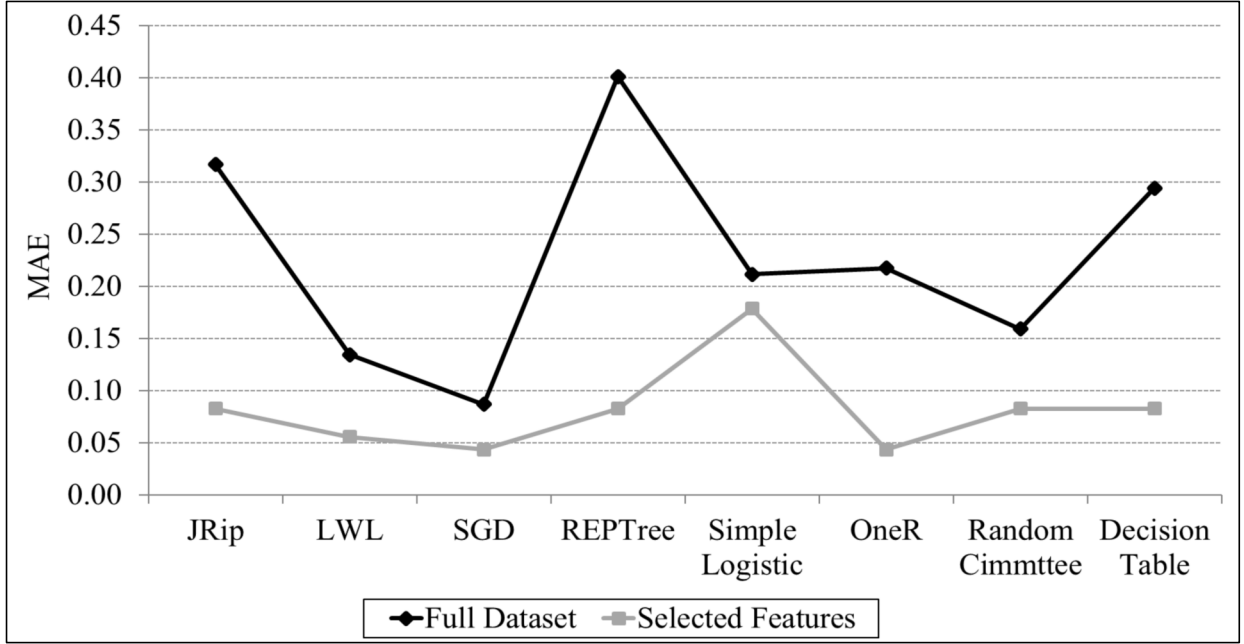


Figure 4.6: Variance of MAE After Applying Correlation-based Feature Selection

4.2.3 Prediction Model Establishment Experimental Results

The experiments were conducted using WEKA 3-8-5 (Waikato, 2020) with the default settings. The selected training technique was ten folds cross-validation. This technique is widely used to test the machine learning classifier due to its reliability. Training and testing the machine learning classifier is repeated in iterations. Also, the dataset is divided equally into ten subsets called folds. In each iteration, the classifier used nine folds for training and one-fold for testing. This process ended when the classifier tested all the data (Smith and Frank, 2016). The following subsections present the experiments' results in detail.

4.2.3.1 Stacking

This subsection presents the results of using stacking algorithm to predict successful software reuse experience. Also, the subsection shows the effect of incorporating feature selection with

Table 4.9: Results of Evaluation Criteria Using Correlation-based Feature Selection

Prediction Model	KS	MAE	RMSE	RAE	RRSE
JRip (Full Dataset)	0.3090	0.3168	0.5388	68.3711%	111.6933%
JRip (Attribute Selected)	0.7039	0.1781	0.3674	38.4446%	76.1612%
Difference	0.3949	0.1387	0.1714	29.9265%	35.5321%
LWL (Full Dataset)	0.7039	0.1345	0.3613	29.0359%	74.8924%
LWL (Attribute Selected)	0.9013	0.0435	0.2085	9.3843%	43.2244%
Difference	0.1974	0.0910	0.1528	19.6516%	31.6680%
SGD (Full Dataset)	0.8083	0.0870	0.2949	18.7685%	61.1286%
SGD (Attribute Selected)	1.0000	0.0000	0.0000	0.0000%	0.0000%
Difference	0.1917	0.0870	0.2949	18.7685%	61.1286%
REPTree (Full Dataset)	0.2146	0.4013	0.5303	86.6172%	109.9355%
REPTree (Attribute Selected)	0.3491	0.3123	0.4667	67.3962%	96.7490%
Difference	0.1345	0.089	0.0636	19.2210%	13.1865%
Simple Logistic (Full Dataset)	0.6167	0.2115	0.3676	45.6523%	76.1957%
Simple Logistic (Attribute Selected)	0.6167	0.1982	0.3633	42.7866%	75.3162%
Difference	0.0000	0.0133	0.0043	2.8657%	0.8795%
OneR (Full Dataset)	0.5064	0.2174	0.4663	46.9214%	96.6528%
OneR (Attribute Selected)	0.5064	0.2174	0.4663	46.9214%	96.6528%
Difference	0.0000	0.0000	0.0000	0.0000%	0.0000%
Random Committee (Full Dataset)	0.8083	0.1590	0.2379	34.3246%	49.3249%
Random Committee (Attribute Selected)	0.7039	0.1363	0.3357	29.4264%	69.5803%
Difference	-0.1044	0.0227	-0.0978	4.8982%	-20.2554%
Decision Table (Full Dataset)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Decision Table (Attribute Selected)	0.5064	0.2943	0.4324	63.5114%	89.6388%
Difference	0.0000	0.0000	0.0000	0.0000%	0.0000%
Average	0.1018	0.0552	0.0737	11.9164%	15.2674%
Minimum	-0.1044	0.0000	-0.0978	0.0000%	-20.2554%
Maximum	0.3949	0.1387	0.2949	29.9265%	61.1286%

stacking algorithm.

4.2.3.1.1 Prediction Using Full Dataset

This subsection analyzes the results obtained from applying stacking algorithms on the full dataset. Table 4.10 shows the performance results of the stacking prediction models. Overall, all prediction models took about eight seconds to build the model. The stacking experiments' TPRs vary between 0.826 and 0.913, while the FPRs were between 0.105 and 0.209. The MAE rates were relatively low. The error rate values vary between 0.0969 and 0.1739. Also, the ROC area results indicate the prediction models' good performance. In general, all prediction models have a high capability of distinguishing between success and the failure of the software reuse experience. However, applying stacking algorithms with MLP as a meta-classifier has outperformed the other models since its ROC area scored 0.973, which is the closest to 1.

Table 4.10: Readings Before and After Applying Classifier Subset Evaluation

Prediction Model (Full Dataset)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)
Stacking + SGD	0.870	0.186	0.869	0.842	0.1304	8.75
Stacking + SMO	0.826	0.209	0.826	0.808	0.1739	8.76
Stacking + MLP	0.913	0.105	0.913	0.973	0.0969	8.88

4.2.3.1.2 Prediction After Feature Selection

This subsection analyzes the results of using stacking algorithm after applying the feature selection technique. Table 4.11 stated the performance criteria results, shows that evaluation criteria scored the best results. All prediction models have successfully classified all the dataset instances shown since TPR and FPR are 1 and 0, respectively. Furthermore, the MAE of the Stacking with MLP was 0.0154, which is extremely low to negligible. Meanwhile, the MAE of stacking with SMO and SGD was 0. Also, the precision results of the prediction models scored were the best value that is 1. In addition to that, all models took about 0.5 seconds to be built. The ROC area results indicate that the stacking prediction models have an excellent performance. The models scored the best results, which is 1. A ROC area equals one indicates that the models have a good measure of separating the successful software reuse experience from failed experience.

Table 4.11: Results of applying Stacking Algorithm on Selected Features

Prediction Model (Selected Features)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)
Stacking + SGD	1.000	0.000	1.000	1.000	0.0000	0.53
Stacking + SMO	1.000	0.000	1.000	1.000	0.0000	0.54
Stacking + MLP	1.000	0.000	1.000	1.000	0.0154	0.54

4.2.3.1.3 Comparison Performance Results Before and After Feature Selection

This subsection presents the effect of applying feature selection techniques on stacking algorithm. According to Table 4.10 and Table 4.11, there is a significant improvement in instances classification. The prediction models failed to classify some of the dataset instances before applying the feature selection technique; meanwhile, the prediction models have all the instances successfully. Figure 4.7 shows the improvement of MAE rates after applying feature selection. The error rates of stacking using the SGD and SMO as meta-classifiers have sharply plunged to zero. Meanwhile, the error rate stacking with MLP has significantly declined from about 0.1 to 0.01.

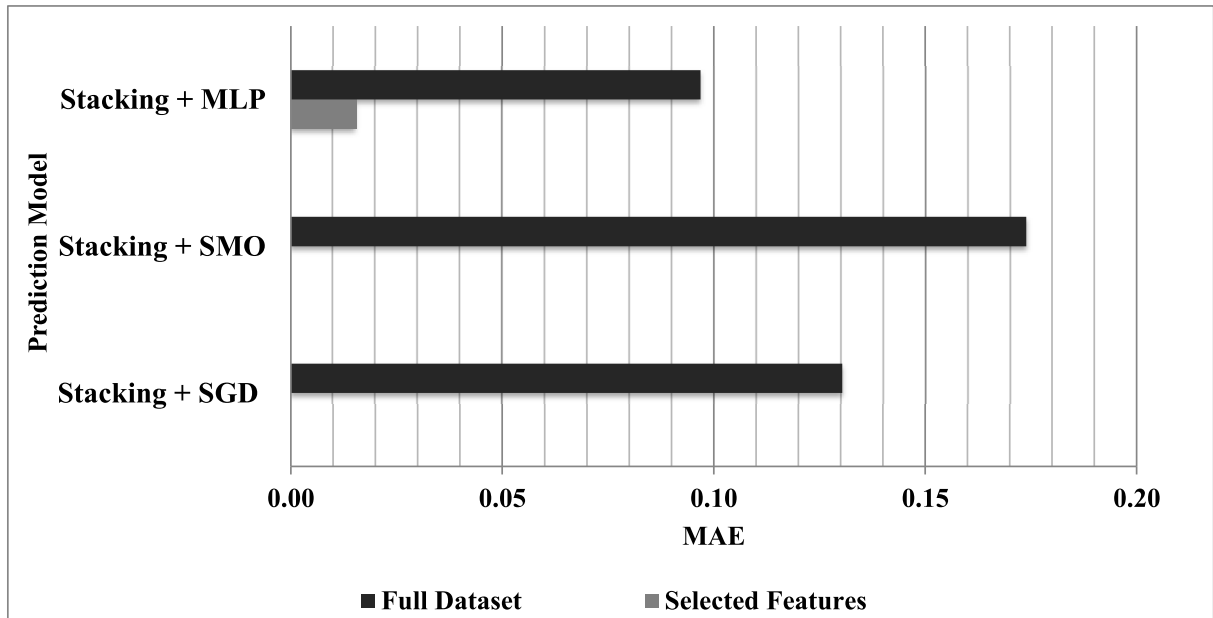


Figure 4.7: Progressions of The Error Rates Before and After Feature

4.2.3.2 Bagging

The subsection presents experimental results of applying bagging as an ensemble prediction model on the full dataset. Also, it discusses the effect of feature selection on bagging algorithms.

4.2.3.2.1 Prediction Using Full Dataset

Table 4.12 presents the evaluation criteria of applying bagging ensemble learning of the full dataset. The TPR and FPR of bagging with SMO have the best results that are 1 and 0, respectively. This points out that bagging with SMO had classified all the instances correctly. Meanwhile, bagging with SGD and MLP has failed in classifying all the instances correctly. The effectiveness of bagging with SGD and SMO is more than bagging with MLP since the time taken to build the model was 8.86 seconds. Moreover, the error rates were minimal. The bagging with MLP scored the highest MAE rate that is 0.0618, which indicates that this prediction model is less efficient than bagging with SMO and SGD. However, the ROC area of bagging MLP has the best value that is 1. The ROC area measures the performance of the prediction model. The closer the ROC area value is to 1, the better. Therefore, bagging with MLP and SMO outperformed bagging with SGD. Furthermore, the precision of bagging with SMO scored the best value; meanwhile, the precision of bagging with MLP and bagging with SGD was slightly lower by 0.087 and 0.0041, respectively.

Table 4.12: Results of Prediction Software Reuse Using Bagging

Prediction Model (Full Dataset)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)
Bagging + SGD	0.913	0.105	0.913	0.992	0.0478	0.04
Bagging + SMO	1.000	1.000	1.000	1.000	0.0304	0.49
Bagging + MLP	0.957	0.082	0.959	1.000	0.0618	8.86

4.2.3.2.2 Prediction After Feature Selection

To improve the bagging algorithms' performance feature selection technique was applied. Table 4.13 presents the results of the bagging prediction models after selecting the relevant attributes. According to Table 4.13, all bagging models have correctly classified the dataset instances. The TPR, FPR, and ROC area scored 1,0, and 1 respectively. The results are

considered the best outcome that any classifier can achieve. The ROC area, which evaluates the prediction model performance, showed that all bagging models have good performance since the ROC area has reached 1. The MAE rates range between 0.0087 and 0.0711, which are very superficial. Also, bagging with SGD and SMO were more effective than bagging with MLP since the time taken to build the models is better by 0.44 and 0.32 seconds. Also, all prediction models have the best precision value.

Table 4.13: Evaluation Criteria Results of Using Bagging with Feature Selection

Prediction Model (Selected Features)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)
Bagging + SGD	1.000	0.000	1.000	1.000	0.0087	0.01
Bagging + SMO	1.000	0.000	1.000	1.000	0.0087	0.13
Bagging + MLP	1.000	0.000	1.000	1.000	0.0711	0.45

4.2.3.2.3 Comparison Performance Results Before and After Feature Selection

Features selection technique helps to improve the prediction model's performance. The bagging models have benefited from feature selection. For instance, the time taken to build the model has significantly improved, especially for bagging with MLP, where the time taken has decreased from 8.86 to 0.54 seconds, as shown in Table 4.12 and Table 4.13. However, Figure 4.8 shows the error rate of bagging with MLP has slightly increased by 0.00093 points. On the other hand, the MAE of bagging with SGD and SMO have efficiently improved by 0.0391 and 0.0217 points, respectively.

4.2.3.3 Voting

This subsection of presents the voting results of the voting ensemble prediction models before and after feature selection.

4.2.3.3.1 Prediction Using Full Dataset

Table 4.14 shows that the best combination rule to classify the dataset was the majority voting. The majority voting got the best TPR and FPR results that is, 1 and 0 consecutively. Therefore, the classified majority voting has classified all instances correctly. Meanwhile, the worst combination rule was the maximum probability since it scored the lowest TPR and the

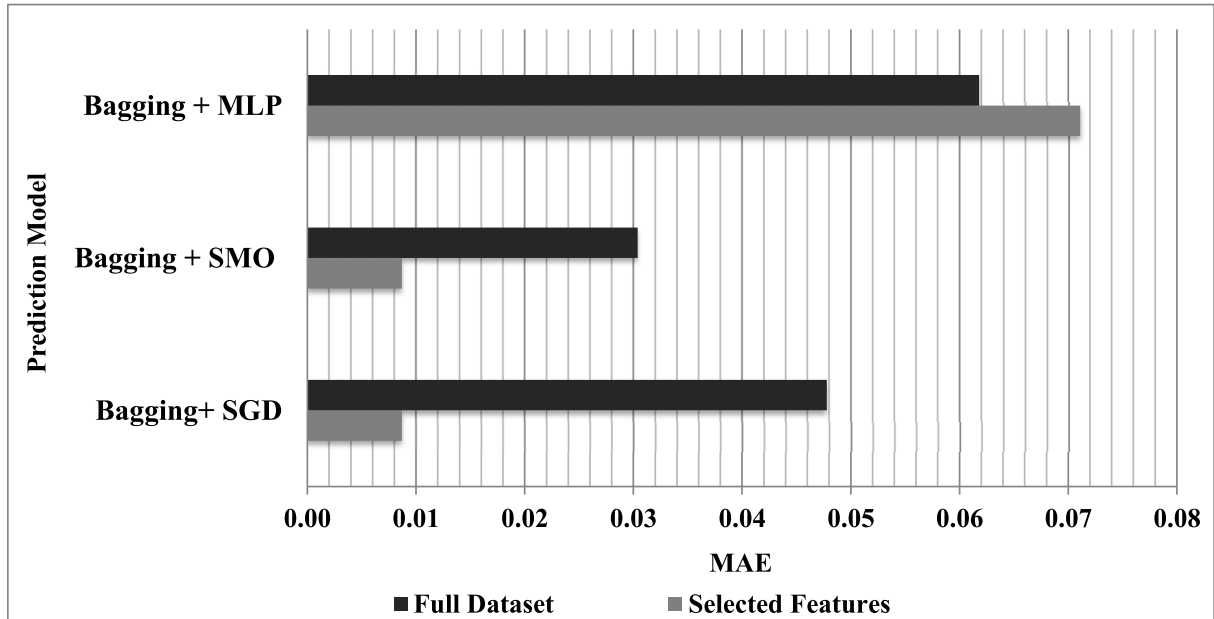


Figure 4.8: Effect of Feature Selection on MAE rates

highest FPR. Also, the error rates are considered low. The MAE results were between 0.0952 and 0.2126. Furthermore, the products of probabilities and minimum probability voting rules failed to classify two instances.

Although voting with majority voting has the best results of the evaluation criteria, the model is the least effective since it took 0.93 seconds to build it. Two voting models that use the product of probabilities and minimum probability could not classify two instances, while the rest of the voting models classified all the instances.

The ROC area results measure the prediction model's ability to distinguish between the dataset's classes. The closer the ROC area value to one, the better. According to Table 4.14, voting with majority voting has the best performance, followed by voting with average probabilities. Meanwhile, the least performance model was the voting using minimum possibility and product of probabilities. Also, voting with maximum probability has the least precision results, which 0.869 meanwhile, the most precise voting model use the majority voting.

4.2.3.3.2 Prediction After Feature Selection

Table 4.15 illustrates the results of the voting prediction models after applying the feature selection technique. The results showed that two of the voting models have correctly classified all the dataset instances. These voting models used the average of probabilities, majority

Table 4.14: Performance Results of applying Voting Learning on The Full Dataset

Prediction Model (Full Dataset)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)	Unclassified Instances
Voting + AP	0.913	0.105	0.913	0.992	0.1063	0.92	0
Voting + PP	0.905	0.119	0.905	0.861	0.0952	0.91	2
Voting + MV	1.000	0.000	1.000	1.000	0.0000	0.93	0
Voting + MinP	0.905	0.119	0.905	0.861	0.0952	0.88	2
Voting + MaxP	0.870	0.186	0.869	0.963	0.2126	0.89	0

voting. The TPRs and FPRs for the voting with products of possibilities and minimum probability were 1 and 0, respectively. However, these products of possibilities and minimum probability could not classify one instance. Moreover, the TPR and FPR of voting with maximum probability indicate that the model had misclassified some instances.

Furthermore, the voting models' error rates were minimal and varied between 0.140 and 0. In addition to that, the ROC area results show that all voting models have a good performance in predicting software reuse experience. Furthermore, voting using the maximum probability, as a voting rule, was the least precise model among the voting model, shown in Table 4.15.

Table 4.15: Evaluation Criteria of Voting Learning on Selected Features

Prediction Model (Selected Features)	TPR	FPR	Precision	ROC Area	MAE	Time taken to build model (seconds)	Unclassified Instances
Voting + AP	1.000	0.000	1.000	1.000	0.067	0.08	0
Voting + PP	1.000	0.000	1.000	0.980	0.000	0.06	1
Voting + MV	1.000	0.000	1.000	1.000	0.000	0.07	0
Voting + MinP	1.000	0.000	1.000	0.980	0.000	0.06	1
Voting + MaxP	0.957	0.082	0.959	1.000	0.140	0.07	0

4.2.3.3.3 Comparison Performance Results Before and After Feature Selection

This subsection highlights the features selection technique's effect on the prediction models'

performance. In general, there is a significant improvement. The time taken to build the model was reduced for all voting models in an average of 0.838 seconds. Also, the TPRs and FPRs of voting with average of probabilities, products of possibilities, and minimum probability improved to the best results, which is 1 and 0, respectively. Moreover, the number of unclassified instances for voting with products of probabilities and minimum probability has reduced to one instance, as shown in Table 4.14 and Table 4.15.

Figure 4.9 illustrates the error rate improvement before and after applying feature selection techniques. MAE result for voting with majority voting remains the same, which is 0. Voting with average of probabilities has improved by 0.0389 points. In addition to that, voting with maximum probability has also been enhanced by 0.0725 points. Meanwhile, error rates of voting minimum probability and products of probabilities experienced a reduction to the best error rate results that is 0.

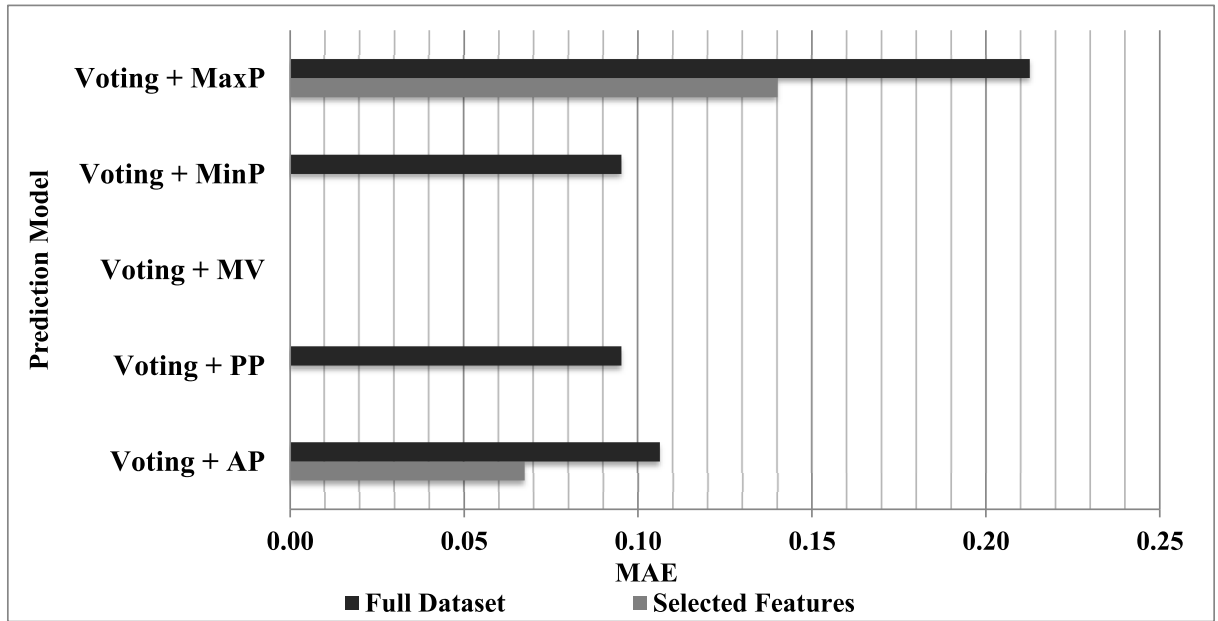


Figure 4.9: Progression of MAE Before and After Using Feature Selection

4.3 Reusability Estimation Model Evaluation

In this section, we assess the effectiveness and accuracy of our proposed model. The evaluation phase consists of two phases. In the first phase, we manually evaluate the reuse score using the proposed model on the static metrics of selected java classes. Meanwhile, the

second phase aims to assess the accuracy of our methodology but compare it with different reusability models. The following subsection depicts the evaluation phases in detail.

4.3.1 Effectiveness Evaluation

As mentioned earlier, the evaluation of the model’s effectiveness was performed by manual examination of a select java class static metric. This evaluation aims to confirm that the model’s obtained reusability scores are reasonable and interpretable to the java classes’ static metric. The first step of analyzing the static metrics is by identifying their cluster membership. The cluster membership identification would determine the statics metrics reuse scores and PCC that has to be used to aggregate the java classes’ reusability scores. Table 4.16 presents the static metrics of the selected java classes with their reusability scores. The cohesive java classes, which have low LCOM5, have scored higher reusability scores than non-cohesive classes. Moreover, complexity could negatively affect the component’s reusability. NL and WMC describe the class’s complexity. The obtained reusability score has shown the expected. The classes that have high NL and WMC have lower reusability scores lower.

From a coupling perspective, the components that appear to have low CBO values had higher reusability scores. In addition to documented components tend to be more reusable. However, the quality of the documentation depends on the TCLOC and the component’s size. Therefore, a high value of TCLOC does not necessarily mean that the component is well documented since the sufficiency of the documentation does not depend on the component’s size. Some classes have TCLOC values; however, their reusability score is relatively low due to their large size, as shown in Table 4.16.

4.3.2 Accuracy Evaluation

To evaluate the accuracy of our proposed estimation model. We compared the performance of our regression models with a different reusability estimation model proposed by (Papamichail et al., 2019a). The reason we choose this model is that we are using the dataset. However, we applied the GQM model as a feature selection strategy. Meanwhile, the work in (Papamichail et al., 2019a) used the entire dataset. In addition to that, we both used univariate analysis to predict the reusability scores. However, our regression model was trained on each static metric separately within a cluster.

Figure 4.10 presents a comparison of the MAE values between our model and the model in (Papamichail et al., 2019a). Since our regression model was trained on each cluster separately, we selected the best, average, and worst regression model results to compare them with

the MAE results of the regression model used in (Papamichail et al., 2019a). In general, our regression model has lower error rates than the (Papamichail et al., 2019a) approach. However, their regression model has lower error rates for some static metrics. Our proposed model provided more accurate results on most of the cases, as shown in Figure 4.10.

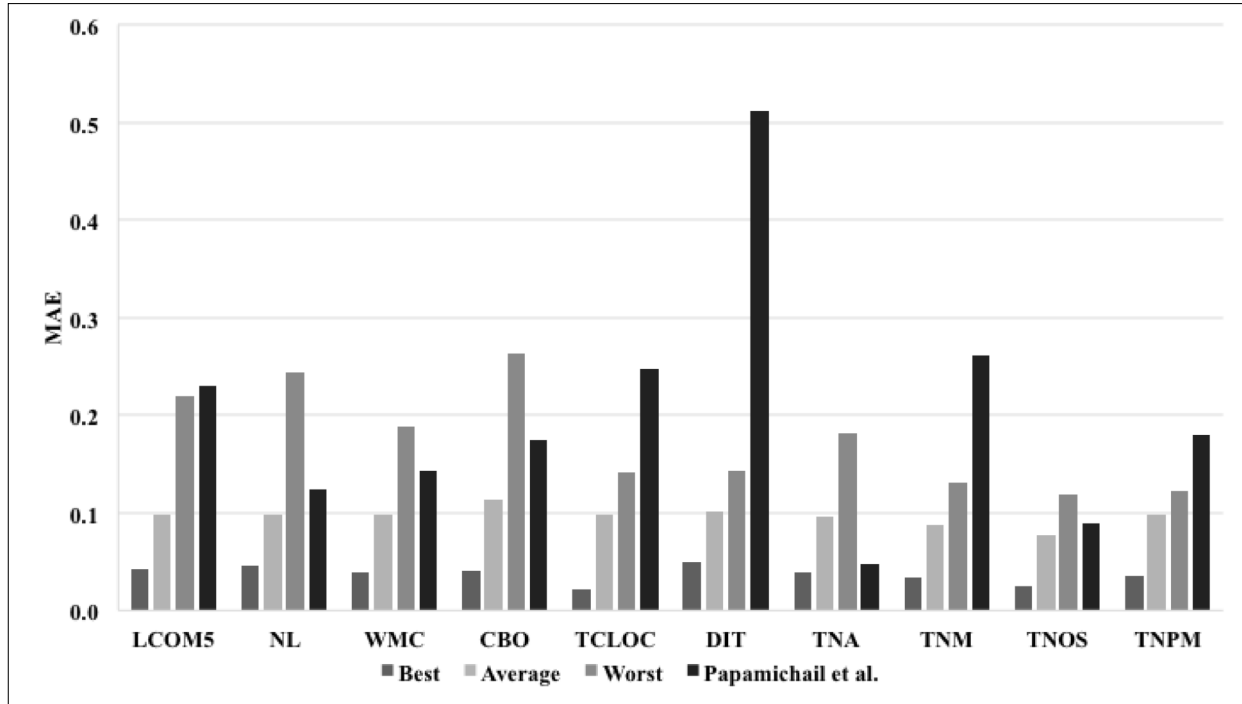


Figure 4.10: Comparison of Error Rates

Table 4.16: Overview of Java Classes Reusability Scores Computed Using the Proposed Model

Cluster 1											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
1	0	1	1	0	1	2	5	1	1	1	0.70 or 70%
9	1	17	11	35	8	120	2	43	28	34	0.20 or 20%
Cluster 2 & 5											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
4	1	23	2	271	0	373	12	22	14	13	0.44 or 44%
39	3	59	3	263	1	434	1	90	73	85	0.33 or 33%
Cluster 3											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
1	1	30	5	112	0	189	4	16	48	11	0.57 or 57%
2	5	104	12	40	0	785	13	440	393	426	0.24 or 24%
Cluster 4											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
0	0	4	0	21	0	37	2	4	4	4	0.62 or 62%
79	1	130	59	978	2	1526	4	170	116	159	0.25 or 25%
Cluster 6											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
1	0	11	2	69	0	139	5	10	32	8	0.61 or 61%
1	8	589	15	190	0	2936	17	117	1591	11	0.26 or 26%
Cluster 7											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
1	1	17	1	15	0	65	4	8	26	8	0.62 or 62%

Table 4.16 Continued: Overview of Java Classes Reusability Scores Computed Using the Proposed Model

13	2	99	16	13	1	560	18	141	146	133	0.16 or 16%
Cluster 8											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
0	0	4	1	256	0	334	0	35	1	35	0.63 or 63%
18	1	40	18	200	1	346	5	28	38	26	0.31 or 31%
Cluster 9											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
2	0	4	1	15	1	48	1	29	6	26	0.99 or 99%
21	3	107	31	657	1	1933	53	1742	386	1456	0.27 or 27%
Cluster 10											
LCOM5	NL	WMC	CBO	TCLOC	DIT	TLOC	TNA	TNM	TNOS	TNPM	Reusability Score
1	0	4	0	12	0	24	1	3	2	2	0.57 or 57%
14	3	161	20	1205	1	1796	12	107	221	101	0.12 or 12%

4.4 Discussion

After presenting the experimental results, in this section, we intend to answer the research questions. The following subsections answer the questions regarding the proposed software reuse experience prediction model and the software reusability estimation model.

4.4.1 Prediction of Software Reuse Experience

The subsection is concerned with answers related to the proposed ensemble model that predicts successful software reuse.

RQ1. *What is the machine learning potential in distinguishing successful and failure software reuse experiences?*

According to Table 4.1-4.3, the results of selected machine algorithms showed that most of the algorithms scored the best results. Most of the evaluation criteria had the highest value, which criteria can hold. For instance, the TPRs of prediction models were 1, which is the highest results a TPR can hold. Also, the KS results were 1, which is the best result a KS can score. Furthermore, the error rates are considered very minimal. Therefore, machine learning algorithms have an outstanding potential to distinguish between successful and failure software reuse experiences. In addition, results showed that the SMO or SVM algorithm is an excellent candidate for a feature selection classifier since it scored the lowest MAE that is 0. By answering this question, we have fulfilled achieving research objectives **O2** and **O3**.

RQ2. *Can feature selection algorithms improve the prediction of successful software reuse experiences?*

To answer this question, we looked at the progression of error rate presented in MAE before and after applying feature selection techniques. This progression is depicted in Figures 4.1-4.6. In general, the feature selection technique has improved the prediction of software reuse experiences. However, there are a few cases in which the prediction models' accuracy has decreased. Nevertheless, we can conclude that feature selection has a positive influence in predicting successful software reuse. By answering this question, we accomplished the research objective **O2**.

RQ3. *What is the most relevant feature subset that would improve the successful software reuse experience prediction?*

To select the most relevant features, we have looked at the progression of error rates in all tested machine learning algorithms. According to Figures 4.1-4.6, the wrapper subset evaluation has significantly improved the error rates of all selected machine learning algorithms. The subset extracted consists of three features: human factors, type of software production, and application domain. The answer to the question helps in achieving the research objective **O4**.

RQ4. *How does ensemble learning perform in predicting successful software reuse experiences?*

The f-measure was utilized to evaluate, compare the performance for experimented prediction models. Moreover, two evaluation criteria were used to identify the best and worst ensemble prediction models. These evaluation criteria are the f-measure and MAE. Figure 4.11 illustrates the stacking models' performance also depicts the effect of using the Wrapper Subset Evaluation as a feature selection technique. Overall, the performance of all stacking ensemble models had improved after applying feature selection. Stacking with SMO as a meta-learner witnessed the best progression. The f-measure has jumped from about 0.8 to 1. This shows the wrapper subset evaluation has a positive impact on the stacking ensemble learning performance. The best stacking ensemble model was the stacking with SMO since the MAE and f-measure have achieved the best results, which are 0 and 1, sequentially. Meanwhile, stacking with MLP is considered the worst since it scored the highest MAE even after applying feature selection, as shown in Figure 4.7.

Figure 4.12 depicts the bagging ensemble learning performance with and without feature selection. In general, all bagging prediction models have scored the best results for the f-measure after utilizing feature selection performance. The performance of bagging using SGD on the full dataset was the worst. However, bagging with SGD performance had sharply improved when feature selection was introduced to the model. Meanwhile, the feature selection had neither a positive nor negative impact on bagging with SMO. The f-measure of bagging with SMO was one, which is the highest result a prediction model could achieve. The performance of bagging with MLP has significantly improved; however, the error rate has slightly increased after applying feature selection, as shown in Figure 4.8. When comparing the error rates in Figure 4.8 and performance in Figure 4.12, it can be deduced that introducing Wrapper Subset Evaluation to bagging with MLP is best in terms of performance and efficiency. Meanwhile, bagging with MLP is the worst among the bagging models.

The impact of the feature selection on the voting ensemble prediction models was also positive. However, voting ensemble prediction with majority voting did not improve when

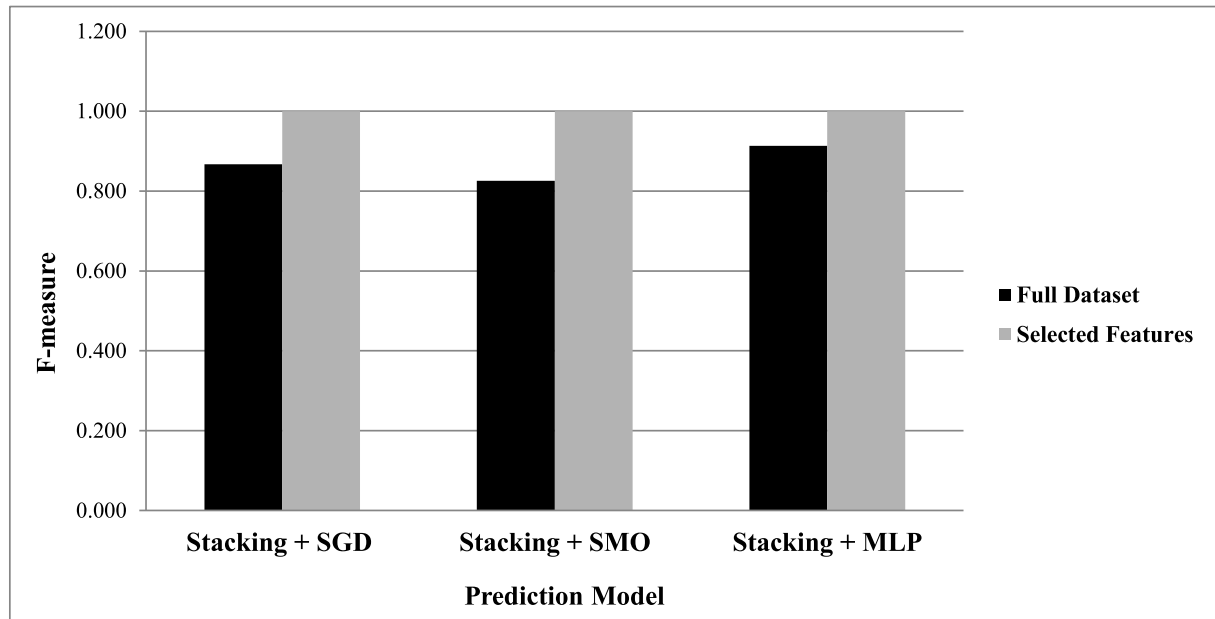


Figure 4.11: Performances of Stacking Prediction Models Before and After Feature Selection

applying feature selection since the majority voting outperformed all the other voting rules with or without feature selection. The model had scored the best performance results, which is one, as shown in Figure 4.13. Also, Figure 4.9 shows that majority voting has the lowest error rates. Meanwhile, the maximum probability has the most deficient performance and error rates among the voting prediction models. When comparing the empirical results of ensemble learning models. We concluded the voting with majority voting since the prediction model has a better effectiveness result presented in the time taken to build the model. By answering this question, we have accomplished the research objective **O5**.

4.4.2 Software Reusability Estimation Model

This subsection answers the research questions related to estimating software components' reusability.

RQ5. *What type of analysis could be used to estimate software artifacts' reusability?*

According to the literature review, static metric analysis is actively used to measure software quality. A software assessment model is required to be efficient and not time-consuming. The static metric analysis is a practical approach since it does not require the execution of the evaluated component. Also, statics metric analysis helps statistically identify the areas that

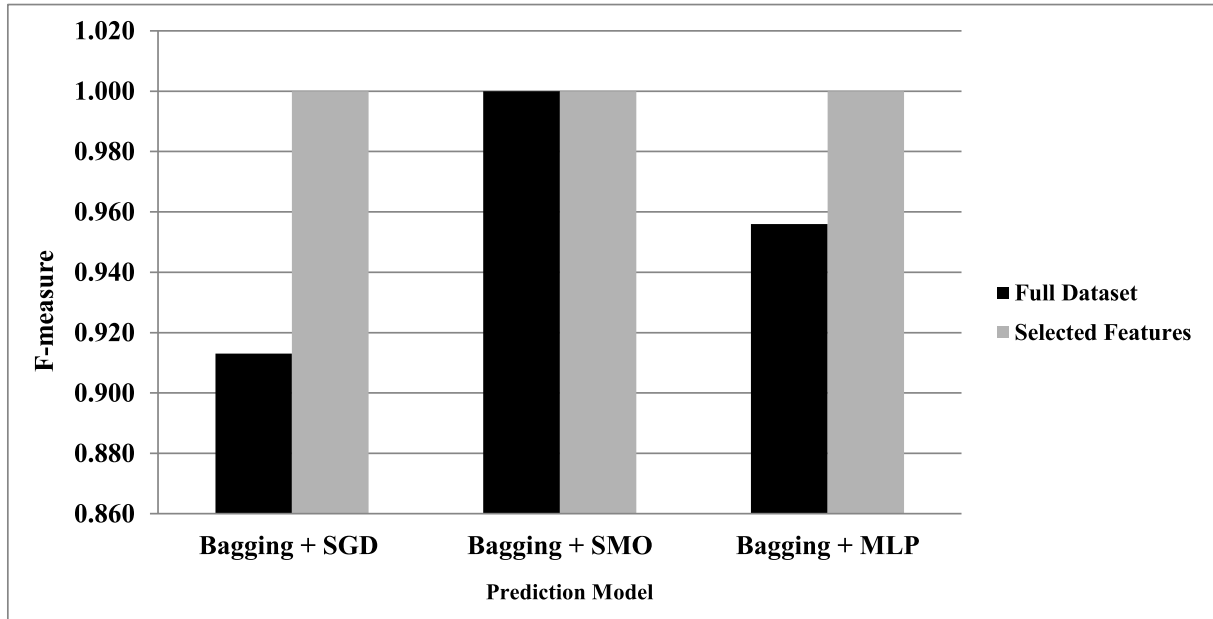


Figure 4.12: Performances of Bagging Prediction Models Before and After Feature Selection

would improve the quality of the component. Moreover, the effectiveness evaluation of the proposed estimation model showed that the reuse scores of the static metrics aligns its the theoretical interpretation. Therefore, static metrics analysis is an effective type of analysis to estimate the software reusability. The answer of this question in addition to the literature review accomplished research objective **O7**.

RQ6. *Can the GQM paradigm be used as a feature selection technique?*

The effectiveness and accuracy evaluation of the proposed estimation model showed that the GQM paradigm played an influential role in estimating the software component reusability. Also, to ensure that the extracted subset is adequate, we utilized Python and applied feature selection on the used dataset to select the best features. Figure 4.14 depicts the output of the feature selection application. The extracted features are the same features that we obtained using the GQM paradigm. This indicates that the GQM can be an effective feature selection technique on the used dataset. The answer of the question partially fulfills research objective **O8**

RQ7. *What are the metrics which can be used to model a software reusability estimator?*

According to the proposed GQM model, the selected metrics focus on the components' size, cohesion, coupling, complexity, documentation, inheritance, and documentation. The GQM

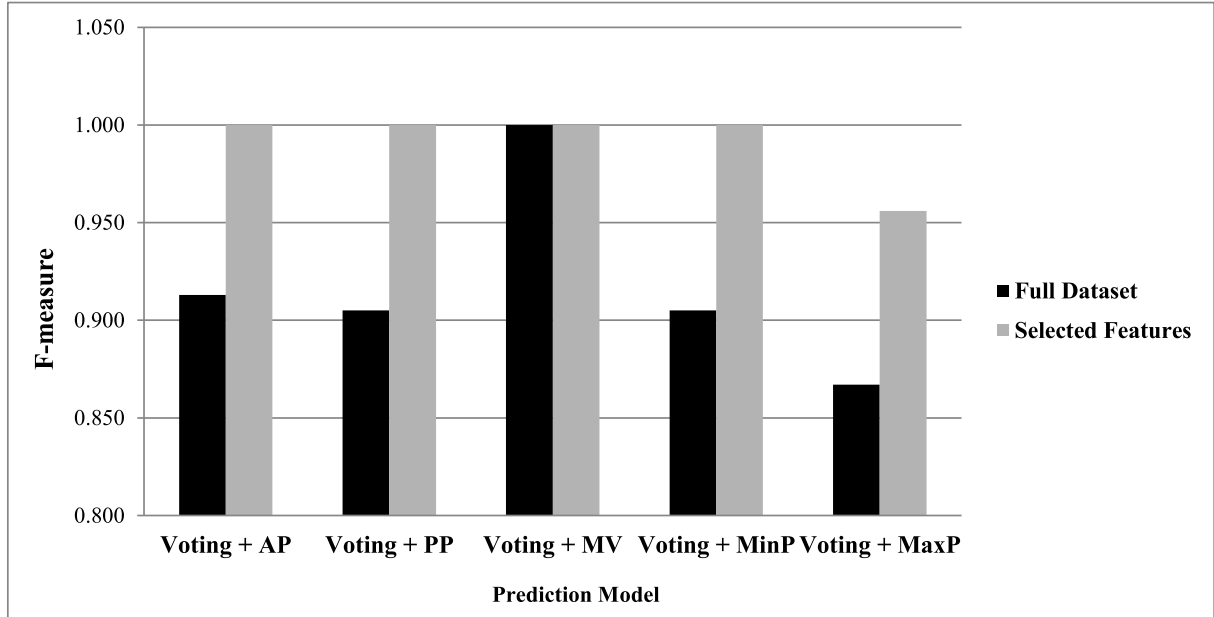


Figure 4.13: Performances of Voting Prediction Models Before and After Feature Selection

model identified twelve static metrics which were used in estimating java classes' reusability, depicted in Figure 3.5. The answer to this question fulfills research objectives **O8**

RQ8. *How can data clustering improve the software reusability estimation?*

The accuracy evaluation of the proposed estimation model showed that our model is more accurate than the proposed model in (Papamichail et al., 2019a) , which used the entire dataset's instances. By answering this question, we achieved research objectives **O9**.

4.5 Threats to Validity

In this section, we present the threats that could compromise our proposed models. The first part illustrates the threats to validate the software reuse prediction model. Meanwhile, the second part discusses the limitation of the proposed software reusability estimation model.

In light of the experiments performed to predict successful software reuse, some threats should be mentioned. The dataset used in these experiments was taken for a public repository (Promise Repository, 2004) with a limited number of 24 instances. Therefore, the size of the studied dataset is considered relatively small. A small dataset might negatively affect prediction models. For instance, it could affect the performance of feature selection as well as the classifiers' performance. To lower the effect of this threat, ensemble learning was chosen

	Specs	Score
4	tcloc	2.612361e+06
6	tloc	2.387039e+06
9	tnos	4.197347e+05
2	wmc	2.400809e+05
8	tnm	2.312939e+05
10	tnpm	1.826362e+05
7	tna	2.944520e+04
0	lcom5	2.307655e+04
3	cbo	2.144295e+04
1	nl	2.573549e+03
5	dit	5.079632e+02

Figure 4.14: Feature Extracted using SelectKBest in Python

rather than a single classifier. Also, the selected feature selection algorithm was tested on eight ML algorithms to ensure its effectiveness.

Furthermore, the used dataset was constructed by interviewing professionals in software development companies. These companies are located in Europe; therefore, there is a slight possibility of geographic influence on the collected dataset. Furthermore, the used dataset is quite unbalanced since the number of instances with success classification is more than the failure classification. In addition to that, there is no machine learning algorithm that superiorly classifies all the instances. Therefore, another classifier may obtain different outcomes while classifying the dataset.

For the estimation model, there are some factors that may threaten the validity of our proposed model. First, the dataset was a result of benchmarking the most popular projects in the Maven Repository. The reuse rate was calculated based on these projects only. Therefore, projects' popularity is the primary influencer of the reuse rate that could change over time. These changes may affect the effectiveness of our proposed model. Meanwhile, there might be other factors that could influence the reusability of the components, such as the popularity of the project provider. Furthermore, our proposed model depends on the analysis methodology

and machine learning model we decided to use while building our model. Adapting different analysis methodologies and machine learning algorithms may produce different results.

4.6 Summary

In this chapter, we presented the experimental results that led to the proposal of an ensemble prediction model. This model can recognize successful software reuse experiences. Also, we depicted the evaluation of the proposed reusability estimation model. When comparing the results of the ML assessment, it showed that SMO or SVM has the best results. Therefore, it is the appropriate algorithm to use as a classifier while estimating the accuracy of feature subsets. Also, the feature selection experiments have successfully identified the most relevant dataset's attributes to predict successful software reuse experience. Furthermore, ensemble learning was selected as a prediction model to minimize the negative influence of the size of the used dataset.

Moreover, the effectiveness evaluation of the proposed software reusability estimation model showed that the estimated reusability scores are interpretable. Also, the accuracy evaluation depicted that the proposed model is more accurate than an estimation modeled by (Papamichail et al., 2019a). The evaluation indicated that data clustering has the potentials to improve the estimation of software components' reusability.

Chapter 5

Conclusion and Recommendations

5.1 Overview

This chapter sums up what has been stated in the previous chapters. Software reuse became a prominent practice among software developers. The matter of identifying reusable assets can be challenging. In this thesis, we proposed two assessment models which utilized ML algorithms. These models can be integrated into the software assets assessment framework. The first model is concerned with the non-technical factors that could distinguish between successful and failure software experience. This model used classification machine learning algorithms. Meanwhile, the second model estimates java classes reusability. The proposed model incorporates data clustering and regression with static metric analysis to evaluate the reusability of java classes.

5.2 Conclusion

It has been more the half of a century since the NATO science committee expressed its concerns about software development productivity and the quality of the developed software. The experts and researchers in the software engineering discipline are determined to increase software development productivity and reduce the development time and effort. Also, software developers are trying to keep up with the market's high demand for sophisticated and high-quality functional software by reusing software components. Software reuse is one of the practices which software development professionals adopt to solve NATO concerns. However, some factors may prevent the success of reusing software assets.

The software engineering discipline has to consider the technical and non-technical factors

when assessing software artifacts. In the study, two ML based models were proposed as software asset assessment methodologies. The first model, which focuses on non-technical factors, started by assessing the potentials of ML in predicting successful software reuse experiences. Six ML algorithms were used on a public data set to predict the success of software reuse. These algorithms are NB, ANN, K-Star, HT, SVM, and KNN. The results showed that ML has the potential to predict successful software reuse. However, some features may affect prediction accuracy. We are planning to apply filters to explore these features that may reduce the training time.

Therefore, WEKA tool was used to apply six feature selection algorithms on a public dataset. The goal of these experiments was to explore the most effective attributes that predict successful reuse experiences. Also, five evaluation criteria were chosen to evaluate the prediction output. Then, the difference between results before and after feature selection was compared to assess the improvement. In general, the variance of the results was promising. The results showed an improvement in the prediction model accuracy. Moreover, there was a significant decrease in error rates. Some experiments showed a sustainable decline in RAE and RRSE around 68% and 65%, respectively. Therefore, it can be concluded that applying feature selection improves the accuracy and performance in predicting successful software reuse experiences. Furthermore, it extracted the essential attributes from the dataset.

The used data is considered a small dataset. Consequently, ensemble learning was utilized to ensure the success of the software reuse experience. Also, to reduce the negative influence of the dataset's size. Three ensemble learning algorithms were selected. These algorithms are stacking, bagging, and voting. Also, Wrapper Subset Evaluation was used to improve the outcome of the ensemble learning. The results showed that ensemble learning had performed very well in distinguishing successful software reuse experience. Moreover, feature selection positively impacted the ensemble learner's prediction where the accuracy was researched 100% in some experiments. The experimental results also showed that voting using the majority voting obtained the optimal results compared to other ensemble learning models. Meanwhile, bagging with SGD on the full dataset was the worst ensemble prediction model.

Software development methodologies have drastically changed, especially with the rapid increase of OSS availability. Also, the adaptation of agile development helped increase the software project productivity by reusing OSS components rather than developing the projects' components from scratch. However, the decision to reuse components is not blindly taken. A software component has thoroughly to be assessed to prior any reuse decision. Measuring the component's reusability by analyzing its static metric is one method that can be used in

the assessment phase. Therefore, the second proposed model was a reusability score model that statistically analyzed the static metrics of java classes. These metrics were collected by (Papamichail et al., 2019b) and published as a public dataset. The reuse rate of is dataset is measured by counting the number of the import lines. Furthermore, we explored the effect of feature selection using GQM, clustering, and regression in aggregating the reusability score. The error rates of the used ML algorithms were very minimal. Also, the findings of the effectiveness evaluation showed that the proposed model could effectively estimate the java classes' reusability.

5.3 Recommendations and Future Work

Software reuse influences to boost up the software projects productivity. However, the reuse decision must be taken systematically through software assets classification and assessment framework to reduce risks and cost. Therefore, software reuse practices have to follow a practical approach, especially in the early stages of the project. The incidents mentioned above of Ariane 5 and Therac-25 depicted how catastrophic a wrong software reuse decision can be.

The findings of our literature review show that machine learning is actively utilized in software reuse. Either in proposing software reuse approaches or estimating software reusability. The empirical contribution of this thesis is to find software reuse assessment models which ML algorithms as we believe that the software assessment is the most crucial and critical phase in the software reuse decision making. Moreover, the software reuse assessment process should never neglect the non-technical factors that influence the success of software reuse.

The experimental results and evaluation showed that the proposed models have effectively achieved their objectives. However, the dataset used to predict successful software reuse experiences considered imbalance since the portion of the failure software reuse instances to successful software reuse instances is around 1:2. The future work will concentrate on finding the appropriate techniques to solve the dataset imbalance.

Moreover, the effectiveness and accuracy evaluation of the proposed software reusability estimation model showed that the proposed model effectively estimated the java classes' reusability. We intend to explore the effectiveness of our estimation methodology in different components' levels. As a future work, we will explore the effectiveness of our methodology in estimating the reusability of the java methods and packages.

References

- Abbas, M. et al. (2020). Automated reuse recommendation of product line assets based on natural language requirements. In *International Conference on Software and Software Reuse*, pages 173–189. Springer.
- Aggarwal, C. and Reddy, C. (2013). *DATA CLUSTERING Algorithms and Applications*. Data Mining and Knowledge Discovery Series. CRC Press/Taylor & Francis Group.
- AL-BADAREEN, A. B. et al. (2021). In the process of software development: Available resources and applicable scenarios. *Journal of Theoretical and Applied Information Technology*, 99(1).
- Alabdulwahab, S. and Moon, B. (2020). Feature Selection Methods Simultaneously Improve the Detection Accuracy and Model Building Time of Machine Learning Classifiers. *Symmetry*, 12(9):1424.
- Ali, N. et al. (2020). A hybrid devops process supporting software reuse: A pilot project. *Journal of Software: Evolution and Process*, 32(7):e2248.
- Alves, V., Schwanninger, C., Barbosa, L., Rashid, A., Sawyer, P., Rayson, P., Pohl, C., and Rummler, A. (2008). An exploratory study of information retrieval techniques in domain analysis. In *2008 12th International Software Product Line Conference*, pages 67–76. IEEE.
- Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., and Stamelos, I. (2018). Reusability index: A measure for assessing software assets reusability. In *International Conference on Software Reuse*, volume 10826, pages 43–58. Springer.
- Anasuodei, M., Ojekudo, and Akpofure, N. (2021). Software Reusability: Approaches and Challenges. *International Journal of Research and Innovation in Applied Science*, 06(05):142–146.

- Awad, M. and Khanna, R. (2015). *Efficient learning machines: theories, concepts, and applications for engineers and system designers*. The expert’s voice in machine learning. Apress Open, Berkley?
- Barros-Justo, J. L., Pinciroli, F., Matalonga, S., and Martínez-Araujo, N. (2018). What software reuse benefits have been transferred to the industry? A systematic mapping study. *Information and Software Technology*, 103:1–21.
- Basili, V. R. (1993). The experimental paradigm in software engineering. In Goos, G., Hartmanis, J., Rombach, H. D., Basili, V. R., and Selby, R. W., editors, *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, volume 706, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Benesty, J., Jingdong Chen, and Yiteng Huang (2008). On the Importance of the Pearson Correlation Coefficient in Noise Reduction. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(4):757–765.
- Biddle, R., Noble, J., and Tempero, E. (2002). Supporting reusable use cases. In *International Conference on Software Reuse*, pages 210–226. Springer.
- Biggerstaff, T. J. and Perlis, A. J., editors (1989). *Software Reusability: Vol. 1, Concepts and Models*. Association for Computing Machinery, New York, NY, USA.
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.
- Brunese, L. et al. (2020). An ensemble learning approach for brain cancer detection exploiting radiomic features. *Computer methods and programs in biomedicine*, 185:105134.
- Bühlmann, P. and Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, 30(4):927–961.
- Cau, S. B. (2016). *Software Reuse and Its Effect on Software Quality for Real-Time Geometric Measurement*. PhD thesis, Faculty of Graduate Studies and Research, University of Regina.
- Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27. Publisher: IEEE.
- Deshpande, M. B. S., Kumar, D. B., and Kumar, D. A. (2021). Assessment of Software Reliability by Object Oriented Metrics using Machine Learning Techniques. *International Journal of Grid and Distributed Computing*, 14(1):10.
- Dimensions (2021). Dimensions - The next evolution in linked scholarly information.

- Doublait, S. (1997). Standard reuse practices: many myths vs. a reality. *StandardView*, 5(2):84–91.
- Ergasheva, S., Kruglov, A., and Shulhan, I. (2019). Development and evaluation of gqm method to improve adaptive systems. *computing*, 4:9.
- Ezran, M., Morisio, M., and Tully, C. (2002). *Practical software reuse*. Springer Science & Business Media.
- Fazal-e Amin, A. K. M. and Oxley, A. (2011). Reusability assessment of open source components for software product lines. *International Journal on New Computer Architectures and Their Applications (IJNCAA)*, 1(3):519–533.
- Feng, W. et al. (2018). Class imbalance ensemble learning based on the margin theory. *Applied Sciences*, 8(5):815.
- Frakes, W. B. and Fox, C. J. (1995). Sixteen questions about software reuse. *Communications of the ACM*, 38(6):75–ff.
- Gaffney, J. E., Cruickshank, R., et al. (1992). A general economics model of software reuse. In *14th International Conference on Software Engineering: Proceedings*, page 327. Association for Computing Machinery.
- Gao, X. et al. (2019). An adaptive ensemble machine learning model for intrusion detection. *IEEE Access*, 7:82512–82521.
- Ghogare, M. S. A. (2019). Performance comparison of tree based classifier using weka. *Journal of the Gujarat Research Society*, 21(3):70–75.
- Gkortzis, A., Feitosa, D., and Spinellis, D. (2021). Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. *Journal of Systems and Software*, 172:110653.
- Hammad, M., Babur, Ö. n., Basit, H. A., and Brand, M. v. d. (2020). Deepclone: Modeling clones to generate code predictions. In *International Conference on Software and Software Reuse*, pages 135–151. Springer.
- Harris, J. (2018). HEARTBLEED: A CASE STUDY. *Issues In Information Systems*, 19(2).
- Hasselbring, W. and Giesecke, S. (2006). *Research methods in software engineering*. Gito Verlag.

- Holmes, R. and Walker, R. J. (2012). Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):20.
- Imoize, A. L., Idowu, D., and Bolaji, T. (2019). A brief overview of software reuse and metrics in software engineering. *World Scientific News*, 122:56–70.
- Ito, S., Hayashi, S., and Saeki, M. (2017). How Can You Improve Your As-Is Models? Requirements Analysis Methods Meet GQM. In *Requirements Engineering: Foundation for Software Quality*, volume 10153, pages 95–111. Springer International Publishing. Series Title: Lecture Notes in Computer Science.
- Jalender, B., Govardhan, A., and Premchand, P. (2010). A pragmatic approach to software reuse. *Journal of Theoretical & Applied Information Technology*, 14.
- Jones, T. C. (1984). Reusability in programming: A survey of the state of the art. *IEEE Transactions on Software Engineering*, SE-10(5):488–494.
- Kakarontzas, G., Constantinou, E., Ampatzoglou, A., and Stamelos, I. (2013). Layer assessment of object-oriented software: A metric facilitating white-box reuse. *Journal of Systems and Software*, 86(2):349–366.
- Kaur, P. J. and Kaushal, S. (2020). A fuzzy approach for estimating quality of aspect oriented systems. *International Journal of Parallel Programming*, 48(5):850–869.
- Kaur, R. and Sharma, S. (2019). An ANN Based Approach for Software Fault Prediction Using Object Oriented Metrics. In Luhach, A. K., Singh, D., Hsiung, P.-A., Hawari, K. B. G., Lingras, P., and Singh, P. K., editors, *Advanced Informatics for Computing Research*, volume 955, pages 341–354. Springer Singapore, Singapore.
- Khan, M. Z. (2020). Hybrid Ensemble Learning Technique for Software Defect Prediction. *International Journal of Modern Education and Computer Science*, 12(1):1–10.
- Khuat, T. T. and Le, M. H. (2019). Ensemble learning for software fault prediction problem with imbalanced data. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(4):3241.
- Kittler, J., Hater, M., and Duin, R. (1996). Combining classifiers. In *Proceedings of 13th International Conference on Pattern Recognition*, pages 897–901 vol.2, Vienna, Austria. IEEE.

- Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.
- Krüger, J. and Berger, T. (2020). An empirical analysis of the costs of clone-and platform-oriented software reuse. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 432–444.
- Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. (2018). Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23(1):384–417.
- kumar, Y. and Sahoo, G. (2012). Analysis of Parametric & Non Parametric Classifiers for Classification Technique using WEKA. *International Journal of Information Technology and Computer Science*, 4(7):43–49.
- Lahmiri, S. et al. (2020). Performance assessment of ensemble learning systems in financial data classification. *Intelligent Systems in Accounting, Finance and Management*, 27(1):3–9.
- Lahouij, A. et al. (2020). Dynamic reconfiguration of cloud composite services using event-b. In *International Conference on Software and Software Reuse*, pages 69–84. Springer.
- Lampropoulos et al. (2018). React-a process for improving open-source software reuse. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 251–254. IEEE.
- Langley, P. (1994). Selection of Relevant Features in Machine Learning:.. Technical report, Defense Technical Information Center, Fort Belvoir, VA.
- Latour, L., Wheeler, T., and Frakes, B. (1991). Descriptive and predictive aspects of the 3cs model: Seta1 working group summary. In *Proceedings of the first international symposium on Environments and tools for Ada*, pages 9–17.
- Lehman, M. M. (1996). Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124. Springer.
- Leveson, N. and Turner, C. (1993). An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41.

- Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., and Ying, S. (2019). Heterogeneous defect prediction with two-stage ensemble learning. *Automated Software Engineering*, 26(3):599–651.
- Lions, J.-L. et al. (1996). Flight 501 failure. *Report by the Inquiry Board*, 190.
- Liu, B. (2011). *Web Data Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Lucrédio, D., dos Santos Brito, K., Alvaro, A., Garcia, V. C., de Almeida, E. S., de Matos Fortes, R. P., and Meira, S. L. (2008). Software reuse: The brazilian industry scenario. *Journal of Systems and Software*, 81(6):996–1013.
- Luts, J., Ojeda, F., Van de Plas, R., De Moor, B., Van Huffel, S., and Suykens, J. A. (2010). A tutorial on support vector machine-based methods for classification problems in chemometrics. *Analytica Chimica Acta*, 665(2):129–145.
- Maggo, S. and Gupta, C. (2014). A machine learning based efficient software reusability prediction model for java based object oriented software. *International Journal of Information Technology and Computer Science (IJITCS)*, 6(1):1–12.
- Malhotra, R. and Khanna, M. (2018). Prediction of change prone classes using evolution-based and object-oriented metrics. *Journal of Intelligent & Fuzzy Systems*, 34(3):1755–1766.
- Mangayarkarasi, P. (2020). Automated Software Design Reusability using a Unique Machine Learning Technique. *International Journal of Innovative Technology and Exploring Engineering*, 9(5):1825–1828.
- Manju and Bhatia, P. K. (2021). Dynamic Reusability Measurement Using Machine Learning Algorithms in Object-Oriented Environment. In *Intelligent Computing and Communication Systems*, pages 393–399. Springer Singapore. Series Title: Algorithms for Intelligent Systems.
- Manoj, H. and Nandakumar, A. (2016). Constructing relationship between software metrics and code reusability in object oriented design. *APTIKOM Journal on Computer Science and Information Technologies*, 1(2):63–76.
- Mateen, A., Kausar, S., and Sattar, A. R. (2017). A software reuse approach and its effect on software quality, an empirical study for the software industry. *arXiv preprint arXiv:1702.00125*.

- McIlroy, M. D., Buxton, J., Naur, P., and Randell, B. (1968). Mass-produced software components. In *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*, pages 88–98.
- Mohagheghi, P. and Conradi, R. (2007). Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12(5):471–516.
- Mohagheghi, P., Conradi, R., Killi, O. M., and Schwarz, H. (2004). An empirical study of software reuse vs. defect-density and stability. In *Proceedings. 26th International Conference on Software Engineering*, pages 282–291. IEEE.
- MohanGoel, B. and Kumar Bhatia, P. (2012). Analysis of Reusability of Object-Oriented System using CK Metrics. *International Journal of Computer Applications*, 60(10):32–36.
- Morisio, M., Ezran, M., and Tully, C. (2002). Success and failure factors in software reuse. *IEEE Transactions on software engineering*, 28(4):340–357.
- Murphy, K. P. and others (2006). Naive bayes classifiers. *University of British Columbia*, 18(60):1–8.
- Naur, P. and Randell, B. (1968). Report on the nato software engineering conference. *NATO Scientific Affairs Division*.
- Negi, P. and Tiwari, U. K. (2020). Machine learning algorithm for assessing reusability in component based software development. *EasyChair Preprint*, 4142:1–8.
- Pacheco, C. L., Garcia, I. A., Calvo-Manzano, J. A., and Arcilla, M. (2015). A proposed model for reuse of software requirements in requirements catalog. *Journal of Software: Evolution and Process*, 27(1):1–21.
- Padhy, N., Satapathy, S. C., and Singh, R. (2019a). Estimation of complexity by using an object oriented metrics approach and its proposed algorithm and models. *International Journal of Networking and Virtual Organisations*, 21(1):43.
- Padhy, N., Singh, R., and Satapathy, S. C. (2019b). Enhanced evolutionary computing based artificial intelligence model for web-solutions software reusability estimation. *Cluster Computing*, 22(4):9787–9804.

- Pahareeya, J. S., Patsariya, S., Jha, A., and Saxena, A. (2018). Finding successful and failure of software using intelligence techniques. *International Journal of Information Technology and Computer Science (IJITCS)*, 3.
- Painuli, S., Elangovan, M., and Sugumaran, V. (2014). Tool condition monitoring using K-star algorithm. *Expert Systems with Applications*, 41(6):2638–2643. Publisher: Elsevier.
- Panigrahi, R., Padhy, N., and Satapathy, S. C. (2019). Software reusability metrics estimation from the social media by using evolutionary algorithms: refactoring prospective. *International Journal of Open Source Software and Processes (IJOSSP)*, 10(2):21–36.
- Papamichail, M. D., Diamantopoulos, T., and Symeonidis, A. L. (2019a). Measuring the reusability of software components using static analysis metrics and reuse rate information. *Journal of Systems and Software*, 158:110423.
- Papamichail, M. D., Diamantopoulos, T., and Symeonidis, A. L. (2019b). Software reusability dataset based on static analysis metrics and reuse rate information. *Data in Brief*, 27:104687.
- Parhami, B. (1994). Voting algorithms. *IEEE Transactions on Reliability*, 43(4):617–629.
- Parkhomenko, A., Sokolyanskii, A., Shepelenko, V., Zalyubovskiy, Y., and Gladkova, O. (2016). Reusable solutions for embedded systems design. In *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pages 319–323. IEEE.
- Patrick, M. T. (2020). Exploring software reusability metrics with q&a forum data. *Journal of Systems and Software*, 168:110652.
- Pearson, K. (1896). VII. Mathematical Contributions to the Theory of Evolution.—III. Heredity, and Panmixia..pdf. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, (187):253–318.
- Polikar, R. (2006). Ensemble based systems in decision making. *IEEE*, 6(3).
- Ponta, S. E., Plate, H., and Sabetta, A. (2018). Beyond Metadata: Code-centric and Usage-based Analysis of Known Vulnerabilities in Open-source Software. *arXiv:1806.05893 [cs]*.
- Poulin, J. S. (1999). Technical opinion: reuse: been there, done that. *Communications of the ACM*, 42(5):98–100.

- Prakash, B. A., Ashoka, D., and Aradhya, V. M. (2012). Application of data mining techniques for software reuse process. *Procedia Technology*, 4:384–389.
- Programming, I. (1983). *Proceedings of the Workshop on Reusability in Programming: Newport, R.I., September 7-9, 1983*. publisher not identified, 1983.
- Promise Repository (2004). (PROMISE) PROMISE 2005: International Workshop on Predictor Models in Software Engineering.
- Sagi, O. and Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4).
- Sammut, C. and Webb, G. I. (2011). *Encyclopedia of Machine Learning*. Springer Science & Business Media.
- Sandhu, A. K. et al. (2020). Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm. *Software: Practice and Experience*.
- Sharma, A., Grover, P., and Kumar, R. (2009). Reusability assessment for software components. *ACM SIGSOFT Software Engineering Notes*, 34(2):1–6.
- Shri, A., Sandhu, P. S., Gupta, V., and Anand, S. (2010). Prediction of reusability of object oriented software systems using clustering approach. *World academy of science, Engineering and Technology*, 43:853–856.
- Sierra, M., Pabón, M. C., Rincón, L., Navarro-Newball, A., and Linares, D. (2019). A comparative analysis of game engines to develop core assets for a software product line of mini-games. In *International Conference on Software and Systems Reuse*, pages 64–74. Springer.
- Singh, P. K., Sangwan, O. P., Singh, A. P., and Pratap, A. (2015). A framework for assessing the software reusability using fuzzy logic approach for aspect oriented software. *IJ Information Technology and Computer Science*, 7(2):12–20.
- Smith, T. C. and Frank, E. (2016). Introducing Machine Learning Concepts with WEKA. In *Statistical Genomics*, volume 1418, pages 353–378. Springer New York, New York, NY.
- Song, Z., Wang, Y., Wang, W., and Zhang, J. (2020). An Empirical study of Exploring Relevant Metrics to Assess Software Product Quality. In *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*, pages 114–124. IEEE.

- U.S. House of Representatives Committee on Oversight and Government Reform, M. S. R. t. C. (2018). The Equifax Data Breach. Technical report, U.S. House of Representatives Committee on Oversight and Government Reform.
- Varnell-Sarjeant, J. and Andrews, A. A. (2015). Comparing reuse strategies in different development environments. In *Advances in Computers*, volume 97, pages 1–47. Elsevier.
- Waikato (2020). Downloading and installing Weka - Weka Wiki.
- Walker, R. and Cottrell, R. (2016). Pragmatic software reuse: A view from the trenches. Technical report, Science.
- Wangoo, D. P. and Singh, A. (2018). A Classification based Predictive Cost Model for Measuring Reusability Level of Open Source Software. *International Journal of Information Technology and Computer Science*, 5(1):5.
- Willmott, C. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30:79–82.
- Zahara, S. I., Ilyas, M., and Zia, T. (2013). A study of comparative analysis of regression algorithms for reusability evaluation of object oriented based software components. In *2013 international conference on open source systems and technologies*, pages 75–80. IEEE.
- Zhang, C. et al. (2020). Transformer fault diagnosis method using iot based monitoring system and ensemble machine learning. *Future Generation Computer Systems*, 108:533–545.
- Özdemir, A., Yavuz, U., and Dael, F. A. (2019). Performance evaluation of different classification techniques using different datasets. *International Journal of Electrical and Computer Engineering*, 9(5):7.

ملخص

يواجه تخصص هندسة البرمجيات تحدياً منذ الإعلان عن أزمة البرمجيات في أواخر الستينات، حيث تشهد هندسة البرمجيات صراعاً في السوق من أجل تطوير برامج فعالة في فترة زمنية قصيرة. في الوقت الحاضر، يقوم مطورو البرامج بإعادة استخدام أصول البرامج بدلاً من تطوير البرنامج بالكامل من البداية. لذلك، تعتبر إعادة استخدام البرامج ممارسة اعتيادية في صناعة تطوير البرمجيات. الهدف الأساسي من إعادة استخدام البرامج هو تقليل جهود التطوير وميزانية المشروع وزيادة الإنتاجية. ومع ذلك، قد تؤدي بعض العوامل إلى فشل إعادة استخدام البرنامج. كما يتعين على شركات تطوير البرمجيات مراعاة هذه العوامل لمنع فشل المشروع بسبب إعادة استخدام البرامج. ولقد لوحظ بوضوح في الماضي العقاقير الخيمة لإعادة استخدام البرامج بشكل خاطئ. لذلك يمكن أن يساعد توقع نجاح استخدام البرامج في تجنب أي خسارة ناتجة عن أي قرار خاطئ لإعادة استخدام البرنامج. ومع ذلك، حتى يومنا هذا، لا يوجد حل واضح يمكنه التنبؤ بتجربة إعادة استخدام البرامج ولا تقدير إمكانية إعادة استخدام مكونات البرامج.

تتناول هذه الأطروحة سيناريوهين من الممكن أن يساعدان أثناء اتخاذ قرار إعادة استخدام البرامج. ولا تركز هذين السيناريوهين على العوامل المتعلقة بالجوانب الفنية للمشروع فقط، ولكنها تعطي أيضاً القرارات الإدارية للشركات. السيناريو الأول هو توقع نجاح تجربة إعادة استخدام البرامج. حيث تم تجربة هذه الخوارزميات على مجموعة بيانات عامة. وقد أظهرت النتائج أن فعالية التعلم الآلي في توقع نجاح إعادة استخدام البرامج، وبدقة وصلت إلى 100%. كما تم استخدام تقنيات اختيار الميزة لاستخراج السمات الأكثر صلة من مجموعة البيانات. أظهرت النتائج التجريبية أن اختيار الميزة أدى إلى تحسين التوقع بتجربة إعادة استخدام البرامج. وأخيراً، حاولنا التطرق لحل ضعف مجموعة البيانات المستخدمة من خلال تقنية دمج التعلم الآلي. على حد علمنا، هذه هي المحاولة الأولى لاستخدام دمج التعلم الآلي في التوقع بنجاح إعادة استخدام البرامج. أظهرت الدراسة التجريبية نتائج ملحوظة وصلت إلى دقة تصل إلى 100%.

السيناريو الثاني يقترح نموذج لتقدير إمكانية إعادة استخدام البرمجيات. يحلل النموذج المقاييس الثابتة لبرامج الجافا باستخدام جميع البيانات والتعلم الآلي لتقدير إمكانية إعادة استخدامها. تم إنشاء هذا النموذج بناءً على مجموعة بيانات عامة تستخدم رموز الاستيراد البرامج كمعدل لإعادة استخدام. أيضاً، تم استخدام الهدف والسؤال والمقياس كأسلوب لاختيار الميزة لاستخراج المقاييس الأكثر صلة في مجموعة البيانات المستخدمة. نجحت نتائج التقييم الخاصة بالنموذج المقترح في تقييم قابلية إعادة استخدام برامج جافا بأقل معدلات خطأ.

جامعة البحرين
كلية تقنية المعلومات



توقع إعادة الاستخدام الناجم للبرامج باستخدام خوارزميات التعلم الآلي

أطروحة مقدمة في استيفاء جزئي لمتطلبات الحصول على درجة الماجستير في هندسة البرمجيات

إعداد

اسم الطالب: مريم خليفة أمين

الرقم الجامعي: ١٩٩٨٠٦٠٨

بإشراف

د. مصطفى حماد

(أستاذ مشارك)

جامعة البحرين

مملكة البحرين

February 28, 2022