Design of Early Reliability Prediction Models for Software Projects

Submitted in partial fulfillment of the requirements

for the award of the degree of

DOCTOR OF PHILOSOPHY

Submitted by

Manchala Pravali

(Roll No. 719086)

Under the guidance of

Dr. Manjubala Bisi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

TELANGANA - 506004, INDIA

October 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

THESIS APPROVAL FOR Ph.D.

This is to certify that the thesis entitled, Design of Early Reliability Prediction Models for Software Projects, submitted by Mrs. Manchala Pravali [Roll No. 719086] is approved for the degree of DOCTOR OF PHILOSOPHY at National Institute of Technology Warangal.

Examiner

Research Supervisor
Dr. Manjubala Bisi
Dept. of Computer Science and Engg.
NIT Warangal
India

Chairman
Prof. R. Padmavathy
Head, Dept. of Computer Science and Engg.
NIT Warangal
India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

CERTIFICATE

This is to certify that the thesis entitled, Design of Early Reliability Prediction Models for Software Projects, submitted in partial fulfillment of requirement for the award of degree of DOCTOR OF PHILOSOPHY to National Institute of Technology Warangal, is a bonafide research work done by Mrs. Manchala Pravali [Roll No. 719086] under my supervision. The contents of the thesis have not been submitted elsewhere for the award of any degree.

Research Supervisor

Dr. Manjubala Bisi
Dept. of Computer Science and Engg.
NIT Warangal
India.

Place: NIT Warangal
Date: October, 2024

DECLARATION

This is to certify that the work presented in the thesis entitled "*Design of Early Reliability Prediction Models for Software Projects*" is a bonafide work done by me under the supervision of Dr. Manjubala Bisi and was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Manchala Pravali
(Roll No. 719086)
Date: 28-10-2024

ACKNOWLEDGMENTS

It is with great pleasure that I acknowledge my sincere thanks and deep sense of gratitude to my supervisor Dr. Manjubala Bisi, Department of Computer Science and Engineering for her valuable guidance throughout the course. Her continuous support, timely feedback, and constructive discussions have played a pivotal role in helping me achieve my objectives. I am grateful for the ample time she dedicated to reviewing my work and providing insightful suggestions for improvement.

I extend my gratitude to all my Doctoral Scrutiny Committee members Prof. P. Radha Krishna, Prof. R. Padmavathy, Dr. S Ravi Chandra, Dr. Sanjaya Kumar Panda and Prof. Ramanamurthy for their insightful comments and suggestions during oral presentations.

I am immensely thankful to Prof. R. B. V. Subramaanyam, Prof. P. Radha Krishna, Dr. S. Ravi Chandra and Prof. R. Padmavathy, Heads of Dept. of CSE and chairmans of DSC, during my stay in the department, for providing adequate facilities. I wish to express my sincere thanks to Prof. N.V. Ramana Rao and Prof. Bidyadhar Subudhi, Directors of NIT Warangal, during my stay at the institute, for their official support and encouragement.

Gratitude to my parents, my in-laws, my husband and my only son for their love, support and prayers for my success in achieving the goal. I am very thankful to my mother for all her support during my PhD. I sincerely thank my husband for his consistent support, understanding, and motivation. I dedicate this achievement to him with all my love and appreciation.

I always thank God for guiding and taking me on the right path at every step of the way.

Manchala Pravali

Dedicated to

My Family & Teachers

ABSTRACT

The significance of software in modern civilization spans across social, political, financial, healthcare, and military domains. However, the increasing complexity and size of software pose challenges, including jeopardizing quality and driving up testing costs. Software reliability is crucial, especially for mission-critical and high-assurance applications. This thesis aims to enhance software reliability by predicting faulty modules and estimating development efforts early in the software development lifecycle. Existing prediction models fall into two categories: Software Reliability Growth Models (SRGMs) and Early Software Reliability Prediction (ESRP) Models. While reliability growth models offer estimates, relying solely on them for corrective actions can be costly and delayed. Early prediction facilitates refined project planning, timely delivery, and cost overruns, mitigates overestimation and underestimation, allocate resources effectively, and formulates the optimal development approaches. Software fault prediction (SFP), including Within-Project Fault Prediction (WPFP) and Cross-Project Fault Prediction (CPFP) models, improves reliability. Additionally, effort estimation reduces cost estimation uncertainty and enhances software quality by estimating required manpower early in development.

The main objectives of this thesis include: (i) To predict the software fault-prone modules in within-project through the Weighted Average Centroid based Imbalance Learning approach. (ii) To predict the software fault-prone modules in a cross-project through similarity based source project and training data selection techniques. (iii) To predict the software fault-prone modules in a cross-project using applicability based source project selection, resampling, and feature reduction. (iv) To estimate the software development effort for a project through a two-stage optimization technique.

Firstly, a diverse imbalance learning technique is designed for WPFP. The prediction performance is enhanced by diverse synthetic data generation and noisy data elimination. Secondly, this study utilizes the Wilcoxon Signed Rank (WSR) test to identify similar source projects for cross-project prediction. A novel oversampling technique is introduced to address distribution gaps and skewed distributions. Additionally, the performance of CPFP is improved through the use of the Binary-RAO algorithm. This algorithm explores diverse combinations of software features and hyperparameters within a specified search space to extract highly correlated features related to module faultiness. Thirdly, the second objective is extended by integrating applicability scores with similarity scores to improve the accuracy of source project selection. A novel resampling technique is devised to extract highly correlated and similar instances while discarding irrelevant ones from the source data, thus enhancing the efficiency of training data construction and reducing distribution discrepancies and class imbalances. Additionally, to tackle the high-dimensionality issue, an efficient deep learning-based Stacked Autoencoder (SAE) model is developed for feature reduction, leading to enhanced performance in CPFP. Lastly, an Adaptive Neuro-Fuzzy Inference System (ANFIS) estimation model is developed using multi-objective optimization techniques for efficient software development effort estimation. The multi-objective rank-based improved Social Network Search (SNS) algorithm is applied to extract optimal software project features and to identify ANFIS parameters.

The experimental results of this research demonstrate the importance of imbalance learning, source selection, instance selection, and feature optimization for software datasets and the effectiveness of metaheuristic optimization techniques in effort estimation models. The proposed methods outperform existing methods, offering improved software fault prediction and effort estimation performance, thereby improving reliability prediction overall.

Keywords: Software quality, Software reliability, Within-project fault prediction, Cross-project fault prediction, Software development effort estimation, Imbalance learning, Source project selection, Distribution difference, Instance filtering, Feature optimization, Resampling, Social network search, Wilcoxon signed-rank test

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

SRGMs	Software Reliability Growth Models
ESRP	Early Software Reliability Prediction
SFP	Soft Fault Prediction
WPFP	Within-Project Fault Prediction
CPFP	Cross-Project Fault Prediction
SDEE	Software Development Effort Estimation
KNN	K-nearest Neighbors
LR	Logistic Regression
NB	Naive Bayes
SVM	Support Vector Machine
DT	Decision Tree
RF	Random Forest
ANN	Artificial Neural Networks
DNN	Deep Neural Networks
CART	Classification and Regression Tree
LR	Linear regression
RR	Ridge regression
FOR	Fall Out Rate
AUC	Area Under the Curve
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
MMRE	Mean Magnitude of Relative Error

MdMRE	Median Magnitude of Relative Error
BMMRE	Balanced Mean Magnitude of Relative Error
PRED	Prediction
WACIL	Weighted Average Centroid based Imbalance Learning
WSR	Wilcoxon Signed-Rank
optimizedTC	optimized Training data Construction
WPS	WSR test based source project selection
WPSTC	WPS and optimizedTC
SRES	Similarity and applicability based source projects selection, REsampling, and Stacked autoencoder
SAPS	Similarity and Applicability-based source Project Selection
TSoptEE	Two-Stage optimization technique for software development Effort Estimation
SNS	Social Network Search
RiBSNS	Rank-based improved Binary SNS
RiCSNS	Rank-based improved Continuous SNS
ANFIS	Adaptive Neuro-Fuzzy Inference System
NOS	Not Oversampled
ROS	Random Over Sampling
SMOTE	Synthetic Minority Oversampling Technique
BSMOTE	Borderline SMOTE
SOTB	Semi-Supervised Oversampling Approach based on Trigonal Barycenter Theory
allCPFP	all projects CPFP
TCA	Transfer Component Analysis
NNFilter	Nearest Neighbor Filter
STrNN	Stratification embedded in Nearest Neighbor
TNB	Transfer Naive Bayes
TFSA	Transformation and Feature Selection Cross-Project Defect Prediction
TDS	Training Data Selection

CFPS	Collaborative Filtering based Source Project Selection
CAMEL	Correlation-based Selection of Multiple Source Projects and Ensemble Learning
ANFIS-SBO	ANFIS - Satin Bower Bird Optimization
BABE	artificial Bee colony guided Analogy-Based Estimation
CBR-GA	Case-Based Reasoning - Genetic Algorithm
WDL	Win-Draw-Loss

Chapter 1

Introduction

Over the last few years, the software has played a very crucial role in the current civilization's life cycle by increasing human dependency on software for various critical and non-critical applications, including smart homes, manufacturing companies, clinical support units, air transport management, education systems, shopping, military affairs, and a few more controlling systems [1]. Moreover, commercial and government organizations also rely on software systems for their projects.

In traditional software engineering practices, conventional wisdom suggests the importance of evaluating software quality during system implementation. Identifying significant quality issues during software application operation may necessitate extensive re-engineering efforts, which is significantly costly [2]. Hence, software quality holds utmost importance, particularly for mission-critical and high-assurance applications [3]. Software quality can be expressed by quality requirements or attributes such as reliability, availability, safety, security, and performance. Among these, software reliability stands out as a key quality factor. The assessment, estimation, and prediction of software reliability are increasingly essential in projects aiming to achieve highly reliable software systems. The software reliability is defined as follows:

Definition 1. (Software Reliability): Software Reliability is defined as the probability that software works without any failure over a specified period of time in a certain environment [4]. In mathematical terms, reliability, denoted as $Reliability(t)$, signifies the probability of a system successfully operating from time 0 to time t:

$$Reliability(t) = \text{Prob}(T > t) \quad \text{for} \quad t \geq 0 \tag{1.1}$$

Here, T represents a random variable indicating the time-to-failure or failure time as defined by the developer.

1.1 Overview of Software Reliability Prediction

The existing software reliability prediction models are categorized into two main groups. The first category involves methods that rely on failure data collected during testing. These models aim to estimate the current reliability of the software and determine the efforts needed to achieve a specific level of reliability before its release, called "Software Reliability Growth Models (SRGMs)". The second category consists of models that evaluate development efforts and predict fault-prone modules to achieve a desired level of reliability before the testing phase. These are referred to as "Early Software Reliability Prediction (ESRP) Models". Evaluating software reliability before the testing phase allows these models to enhance the planning of a project and efficient resource allocation. Early-phase prediction is essential for efficient and timely project management and for ensuring cost-effectiveness.

There are a few parameters that affect the early reliability prediction, namely Software Fault Prediction (SFP), Software Development Effort Estimation (SDEE), Software Testing Effort Estimation (STEE), Software Development Cost Estimation (SDCE), Software Testing Cost Estimation (STCE), to name a few [5]. In this thesis, we have concentrated on SFP and SDEE as reliability parameters. These models are particularly valuable during the early phases of the software development life cycle, as they enable reliability estimation before the software is fully implemented or released. By leveraging early development data, design information, and historical metrics, these models provide actionable insights. Implementing them offers significant advantages, including better risk management, optimized resource allocation, and enhanced software quality.

1.1.1 Overview of Software Fault Prediction

As the complexity and size of the software continue to increase over time, they might produce a variety of faults, which jeopardize the quality of the software and drive up the testing cost by demanding more resources for testing [6, 7]. Faulty modules in software are those with defects or bugs, while non-faulty modules are those without defects. Undiscovered faults can lead to unpredictable outcomes or failures after software deployment. Moreover, fault management often costs about 80% of the total budget [8].

SFP models are broadly divided into Within-Project Fault Prediction (WPFP) and Cross-Project Fault Prediction (CPFP) models. WPFP can be defined as both training and testing data belonging to the same project under the assumption of having the same distribution. In reality, collecting historical training data for new projects can be challenging due to small organizations or the product/service being released for the first time. In such circumstances, cross-project prediction is an alternative approach to allow multiple projects to share available historical data.

Definition 2. (Software Failure): It starts with a programmer mistake or oversight during the implementation phase of a product development process, leading to an error within the initial phase of the software development life cycle. If this error persists in the final software code, it is termed as a software fault. If the fault becomes evident during either testing or actual usage, it is recognized as a software failure. In other words, a software failure occurs when a software system does not perform as specified, leading to unexpected behavior because of the fault which occurs during development. A fault may remain hidden in the software without causing immediate issues, but it can trigger a failure when specific inputs or circumstances arise. A few examples of faults are logical, arithmetic, multi-threading, syntax, performance and interface defects, etc.

Definition 3. (Software Module): Although there is no universally accepted definition, a module is generally considered to be a logically independent component within a system that carries out a distinct function. Throughout the thesis, the terms "module" and "component" will be used interchangeably [9].

1.1.2 Overview of Software Development Effort Estimation

Software development effort estimation is a process of estimating the effort needed to build software and is quantified in terms of person-months or person-hours. Estimating effort in the preliminary stages can enhance project planning, budgeting, process monitoring, task scheduling, and resource distribution. The precise estimation of software effort holds a significant role in software engineering since it prevents both overestimation and underestimation. Overestimation could result in the misallocation of resources, adversely affect the progress of other vital projects, and exceed budget limits. Conversely, underestimation could result in budget overruns, delays in project delivery, and delivery of poor-quality projects, ultimately impacting software reliability and client satisfaction. A thorough comprehension and management of the software development process improves the software's reliability.

The rest of this chapter is organized as follows. The motivation behind this work is presented in Section 1.2 along with the objectives. In Section 1.3, the contributions of the thesis are discussed. The organization of the thesis has been presented in Section 1.4.

1.2 Motivation and Objectives

Currently, SFP is gaining more popularity. As per Boehm [10], the longer a bug stays in the software, the more expensive it is to fix it. The past studies on the prediction of software faults emphasized the eminence of many classification models, including K-Nearest Neighbors (KNN), Logistic Regression (LR), Naive Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and Artificial Neural Networks (ANN) and Deep Neural Networks (DNN). Due to the class imbalance problem in software datasets, the classifier trains to recognize non-faulty instances, potentially degrading the prediction rate of faulty modules. To cope with imbalance data, a large amount of research work has been raging on the oversampling approaches [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. Few models reproduce instances as new ones, leading to overfitting, where classifiers trained on redundant data perform well over training data but struggle over test data [15]. A few models generate synthetic data between closest neighbors, potentially introduced into a closed cluster, reduce the diversity of synthetic instances, and increase false positive outcomes [11, 12, 13]. As per Bennin et al. [14], the diversity of the synthetic faulty instances plays a key role in reducing false positive outcomes. Moreover, identifying and eliminating noisy instances during synthetic instance generation is crucial for classification model performance, as it may impact the model's overall efficiency. The generation of diverse synthetic data and the elimination of noisy samples can improve prediction performance and reduce false positives. This is the motivation behind the use of diverse synthetic data generation and noisy sample elimination mechanisms in our WPFP work.

Even though the existing CPFP strategies can produce noticeable prediction performance, there's still scope for improvement when comparing their overall performance to the basic WPFP model and current CPFP models. The distribution difference of cross-projects impacts the CPFP model's performance. The prominent models, such as transfer learning techniques [22, 23, 24, 25, 26, 27, 28] and training data selection models [29, 30, 31], are available to address the distribution difference problem. However, existing CPFP models are unstable, and the variety of source projects has a great influence on their performance. The prediction model trained on randomly selected single or set of source projects [25, 26, 32] can't generalize over target projects because the randomly selected source projects may not have an identical distribution as the target and the prediction model receives a very small amount of data from one source project, which isn't enough to build a model. In another scenario, the prediction model is trained on all available source projects, which may include a significant amount of irrelevant data. This can mislead the model and result in inaccurate predictions. Therefore, the selection of source projects becomes an imperative task, as the number of available open-source software projects expands day to day [33]. In literature, instead of building a prediction model randomly on available projects, a few studies concentrated on source project selection [30, 33, 34, 35, 36, 37, 38, 39, 40], and few studies assume that the similarly distributed source projects might be applicable to the target [36, 39]. However, Liu et al. [37] and Sun et al. [38] considered the applicability, but there is a restriction on the number of source projects to be selected. Liu et al. [36] model limited their selection to two source projects, while Sun et al. [37] model limited to three source projects for each target project.

There are two more problems associated with CPFP models. One of the issues is the imbalanced structure of software datasets, and another is the curse of dimensionality [25, 41, 42, 43, 44]. Existing literature for CPFP considered any one or two issues among similarity and applicability between projects, distribution dissimilarity, imbalanced data, and high dimensionality issues, potentially leading to inaccurate fault prediction in target projects. This motivates us to select source projects and reduce distribution dissimilarity along with imbalance learning and feature extraction, which can improve the possibility of providing more accurate predictions and reduce the false positive outcomes in the final classification over target data.

Most of the machine learning models in effort estimation fall under categories like regression models, Case-Based Reasoning (CBR) models, Analogy-Based Estimation (ABE), ANN, fuzzy-based neural networks, extreme learning, and ensemble learning models [45, 46, 47, 48, 49, 50, 51, 52]. However, conventional regression models like Linear Regression (LR), Ridge Regression (RR), Classification, and Regression Tree (CART) are notably sensitive to outliers and also necessitate considerably a large dataset for training, it is often difficult in the software effort estimation domain. Lately, the Adaptive Neuro-Fuzzy Inference System (ANFIS) is getting notable attention among effort estimation models, due to its rapid learning capacity, the ability to represent the nonlinear process structure, and adaptability to improperly specified data. Despite its widespread acceptance, the ANFIS model encounters constraints such as the curse of dimensionality and high computational costs, which restrict its use in applications involving a large number of inputs. The number of rules, premise, and consequent parameters increase exponentially with the number of inputs. Moreover, the standard learning process of ANFIS involves gradient learning, which is prone to converge at local minima. This justifies the importance of the feature selection and parameter optimization and adaptation processes as an essential part of classical ANFIS, as they play a pivotal role in generating accurate and efficient outcomes with minimal absolute estimation error. The SDEE datasets are defined by large numbers of metrics, which might escalate the computational cost of the ANFIS model. This motivates the importance of feature selection and parameter optimization as an essential part of classical ANFIS for SDEE.

By considering the above-mentioned research gaps, the following major objectives are formulated in this thesis:

Objective 1: To predict the software fault-prone modules in within-project through the weighted average centroid based imbalance learning approach.

Objective 2: To predict the software fault-prone modules in a cross-project through similarity based source project and training data selection techniques.

Objective 3: To predict the software fault-prone modules in a cross-project using applicability based source project selection, resampling, and feature reduction.

Objective 4: To estimate the software development effort for a project through a two-stage optimization technique.

1.3 Overview of the Contributions of the Thesis

This section provides a summary of the contributions made in each chapter of the thesis. Each subsection outlines the key points and findings of its respective chapter.

1.3.1 A Diverse Oversampling Technique for Within-Project Fault Prediction

In this work, a Weighted Average Centroid based Imbalance Learning (WACIL) approach is presented for WPFP. WACIL is an oversampling approach to reduce the ratio of class imbalance between faulty and non-faulty instances, which generates pseudo-data to make dataset balance. The novelty of the proposed framework lies in generating a synthetic set of diverse instances to maximize the recognition rate of faulty instances and minimize false positive outcomes. The proposed approach consists of two primary stages:

- In the first stage of WACIL, we designed a method to extract important hard-to-classify borderline instances of the faulty class and construct the borderline faulty instance set.

- The next stage aims to balance the dataset by introducing diverse pseudo-instances in faulty class and computing the weighted average pseudo-instances using Mahalanobis distance after extracting borderline instances. Among these pseudo-instances, few might be noisy samples. The filtered pseudo-instance stage eliminates the noisy samples parallel with pseudo-instance generation. The generation and elimination stages ensure equal faulty and non-faulty classes ratio and it can build a better classifier and improve prediction performance.

To observe the superiority of the proposed model, a widespread comparison with the other baseline imbalance learning models as well as with the Not Oversampled (NOS) original dataset, is conducted over a total of 24 PROMISE and NASA projects. The KNN, LR, NB, SVM, DT, and DNN classifiers are constructed on generated pseudo-data and measured the performance. The performance measures considered in this work are Fall Out Rate (FOR), Recall, F-measure, Area Under the Curve (AUC), and Geometric-mean (G-mean). On average, WACIL ranked 4, 1, 17, 2, 36, and 3 with KNN, LR, NB, SVM, DT, and DNN, respectively. We can conclude that the classifiers LR and DNN with WACIL perform the best against all other combinations. In addition, lower FOR and higher Recall outcomes indicate how effectively imbalanced data is handled by the generated diverse pseudo-instances.

1.3.2 Source Project and Optimized Training Data Selection Approach for Cross-Project Fault Prediction

To resolve some of the issues faced by CPFP, we proposed a novel optimized source data selection approach called WPSTC (Wilcoxon signed-rank (WSR) test based source project selection (WPS) and optimized training data construction(optimizedTC)). The novelty of the proposed framework lies in the similarly distributed source selection and a new way of the training data selection process. The proposed WPSTC model is a two-phase approach.

- In the first phase, for a particular new project, based on the distributional characteristics through the WSR test, the association between it and historical projects is investigated, and the corresponding similar source projects are recorded.
- In the second phase, in each iteration, one of the source projects is taken as training data and performs the instance filtering process, then appends it to the filtered data,

which can reduce the distribution gap and make the dataset balanced. To enhance the quality of training data even more, the Binary-RAO algorithm is employed to identify the optimal feature set from the filtered data.

We compared our WPSTC against WPFP, all projects CPFP, and a few previously succeeded cross-project prediction models over 24 PROMISE and NASA datasets using KNN, SVM, DT, LR, NB, and ensemble classifiers, and findings are measured through FOR, Recall, AUC, F-measure, G-mean, normalized Matthews correlation coefficient (nMCC), and Balance measures. Based on prediction performance and statistical comparison, FOR, Recall, G-mean, and Balance values indicate that WPS+CPFP can handle imbalance issues much better. The G-mean, AUC, Balance, and nMCC indicate the overall performance of WPS+CPFP is superior to allCPFP. The WPSTC's FOR values are high but the Recall, G-mean, and Balance values are higher than the WPFP. The results further indicate that WPSTC can handle the imbalance issue in cross-project prediction by improving the Recall values without deteriorating the FOR values.

1.3.3 A Cross-project Fault Prediction through Applicability based Source Project Selection

This study incorporates both similarity and applicability scores to choose comparable historical source projects. Additionally, following a thorough investigation of the issues affecting the cross-project prediction performance, we proposed a three-fold SRES (Similarity and applicability based source projects selection, REsampling, and Stacked autoencoder) model for CPFP. The novelty of the proposed framework lies in the effective source project selection process and novel resampling model. The major contributions to this work are listed below:

- To minimize the distribution gap between source and target projects, we developed the novel Similarity and Applicability-based source Project Selection (SAPS) method, then developed oversampling and undersampling techniques. Later, employed an efficient deep learning-based Stacked Autoencoder SAE) model for feature reduction.

- The similarity relationships between the specific target project and all the available historical projects are explored by comparing the five distributional characteristic measures, such as mean, mode, median, standard deviation and range. We assume the model trained on the nearest neighbors of target data in historical projects might produce superior prediction performance over target data. The applicability scores are computed by considering AUC as the applicability measure.

- Next, we introduce a novel data resampling method to address class imbalance and distribution gap issues by modifying source data through undersampling and oversampling. The above-generated balanced training data is fed into stacked autoencoders to extract deep representations of actual software features without losing the original information through unsupervised pre-training. Then the training and testing data characterized by a reduced feature set are used to predict labels of the data through SAE's supervised fine-tuning.

To assess the performance, FOR, Recall, Balance, AUC, G-mean, and normalized Matthews correlation coefficient (nMCC) are utilized and experimented on 24 projects. The performance of the SAPS model compared with the existing source project selection models. Furthermore, the SRES model was evaluated against the most recent and widely used CPFP models. We compare the SRES model with WPFP to prove that cross-project data works better than within-project data more systematically and quantitatively. We compare the SRES model with SAPS and confirm that resampling and feature reduction along with source project selection techniques can aid in accurate fault prediction.

1.3.4 A Two-Stage Optimization Technique for Software Development Effort Estimation

This work develops a Two-Stage Optimization Technique for Software Development Effort Estimation (TSoptEE). The novelty of the proposed framework lies in reducing the number of features and optimizing the premise and consequent parameters of the ANFIS model through an improved Social Network Search (SNS) algorithm. So far, based on our knowledge of the literature, no model has concentrated simultaneously on these two issues. The major contributions to this work are listed below:

- We proposed the TSoptEE model to enhance effort estimation accuracy. The TSoptEE model is developed in two stages: Stage 1 performs optimal feature selection through a rank-based improved binary social network search (RiBSNS) multi-objective decision-making Weighted Sum Method (WSM). It dynamically finds out ranks for moods of the SNS algorithm in each iteration, then accordingly selects moods and updates the solutions by considering the minimization of the number of features, maximization of the Adjusted R2 (R-squared) (Adj R^2) (R-squared or R^2 measures the proportion of variance in the dependent variable), and minimization of the MAE score, which can enhance the learning ability of the RiBSNS algorithm. In this context, "moods" refers to different social and behavioural patterns that influence user interactions and perceptions, affecting decision-making in the SNS algorithm. The four moods are imitation, conversation, disputation, and innovation. Then, the selected features are passed to the ANFIS model as input.

- Stage 2 performs ANFIS parameter optimization to accurately estimate effort, which searches among a wide range of classical ANFIS parameters through a rank-based improved continuous social network search (RiCSNS) multi-objective method and gives the most appropriate set of parameters with the least absolute estimation error and higher Adj R^2.

To demonstrate the superiority of our proposed model, we perform comparisons with a few fundamental regression models, including LR, RR, CART, and a simple ANN model. Additionally, compared with three recently developed estimation approaches. The proposed TSoptEE model has been evaluated over nine publicly accessible benchmark datasets, using a set of seven reliable evaluation metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), Median Magnitude of Relative Error (MdMRE), Balanced Mean Magnitude of Relative Error (BMMRE), Prediction (PRED), and Adj R^2 . It is observed that the estimation accuracy of the TSoptEE is typically better than or at least comparable to that of the other estimation models. As observed from experimental results, RMSE, MAE, MMRE, MdMRE, BMMRE, and PRED average values of datasets range between 1400 to 3000, 900 to 1200, 0.40 to 3.0, 0.3 to 1.2, 0.6 to 1.3, and 0.3 to 0.5, respectively. Where our TSoptEE achieves the lowest average over RMSE, MAE, and BMMRE and the highest PRED score. In terms of MMRE and MdMRE, the second lowest was achieved by TSoptEE because we optimized MAE and Adj R^2 as our objective functions in this study, but in the compared models, MMRE was considered as a minimizing objective. The Win-Draw-Lose (WDL) results reveal that TSoptEE stands out as a reliable model, with a large number of wins and zero losses.

1.4 Organisation of the Thesis

The main focus of this dissertation is to design early reliability prediction models for software projects by considering fault prediction and effort estimation. The efficacy of the proposed approaches has been shown through experimentation with publicly accessible software datasets. The thesis has been organized into seven chapters.

Chapter 1: This chapter presents a brief introduction to software reliability prediction models and reliability parameters, along with problems associated with the prediction models. A brief description of the objectives of the thesis is also provided in this chapter.

Chapter 2: In this chapter, a detailed literature review on reliability prediction models, WPFP, CPFP, and SDEE is presented. A detailed survey of existing approaches that reduce distribution differences, imbalance, and the curse of dimensionality issues in software datasets has been presented in this chapter. Additionally, development effort estimation models are also thoroughly discussed in this chapter.

Chapter 3: This chapter presents the detailed weighted average centroid based imbalance learning approach, which is designed to address class imbalance within the WPFP. The aim is to minimize the class imbalance between faulty and non-faulty instances, thereby optimizing the performance of the predictive model.

Chapter 4: This chapter presents an improved CPFP model to tackle distribution dissimilarity, imbalanced and high-dimensional data through Wilcoxon signed-rank test based source project selection and an optimized training data construction approach.

Chapter 5: This chapter presents the proposed SRES model for CPFP. Firstly, the similarity and applicability based source projects selection, then performed the resampling through oversampling and undersampling techniques. Later, stacked autoencoder based feature reduction is introduced to improve CPFP performance.

Chapter 6: This chapter presents a two-stage optimization technique for software development effort estimation. The aim is to improve the accuracy of software development effort estimation through multi-objective feature selection and optimization of parameters in the ANFIS model.

Chapter 7: This chapter summarises the work, the outcomes of the contributions, and the scope for future expansion of the work.

Chapter 2

Literature Survey

This chapter provides an overview of related work in the area of software reliability prediction, especially the work related to software fault prediction and software development effort estimation. Until the late 1960s, the focus was predominantly on hardware performance in systems. However, by the early 1970s, software also emerged as a significant concern. This shift was largely prompted by the escalating costs of software compared to hardware, both in development and operation. Producing reliable software applications has become one of the most challenging issues confronting the software industry. In the context of large-scale or international software enterprises, the effective development of a software system hinges on the reliability of its software components.

Software reliability prediction models are constructed from the input of software development phases. As illustrated in Figure 2.1, these models fall into two main types: Early Software Reliability Prediction (ESRP) models and Software Reliability Growth Models (SRGMs). Early prediction models aim to forecast software reliability in the requirement, design, and implementation phases of the life cycle. On the other hand, reliability growth models endeavour to anticipate software reliability during the testing phase by examining the failure patterns of the software during testing and projecting its performance during operation. Several other models may be included in either or both of these categories, such as architecture-based and input domain-based models. Architecture-based models forecast reliability from the design, implementation, and testing phases, thus integrating elements of both types. These models prioritize the software's architecture and calculate reliability

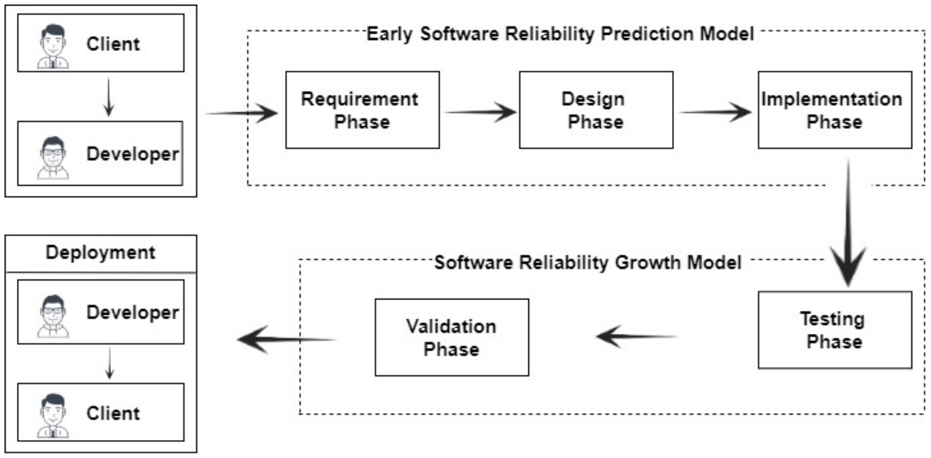


Figure 2.1: Software development life cycle with reliability prediction models

estimates by consolidating estimates derived from the various components of the software. Input domain-based models leverage the software's input domain (validated data) properties to deduce a correctness probability estimate from properly executed test cases.

2.1 Software Reliability Growth models

In this method, estimating reliability involves measuring the reliability of a software application by aligning the failure data gathered during the testing phase of the product development cycle with a suitable growth model [53, 54, 55, 56, 57, 58]. These models assess current and future software reliability based on failure data collected during the testing phase. As well as these models aim to establish statistical relationships between fault detection data and well-known functions such as exponential functions. If the correlation proves strong, these established functions can be employed to forecast future performance. These models quantify reliability by measuring the number of faults discovered versus the number of faults remaining in the software after a specified period or time interval between software failures.

Goel et al. [54] introduced a stochastic framework based on a non-homogeneous Poisson process (NHPP) for the software failure phenomenon. This approach involves analyzing the failure process to establish an appropriate mean value function for the NHPP, enabling the determination of software reliability and performance. Experiments were conducted using the Naval Tactical Data System (NTDS) from the US Navy Fleet Computer Programming Center. Musa et al. [55] developed a new software reliability model that predicts expected failures. Huang et al. [56], described how few NHPP growth models, such as Goel-Okumoto, Gompertz Growth Curve, Logistic Growth Curve, Generalized Goel NHPP, Yamada Delayed S-Shaped, Inflected S-Shaped, and Weibull-Type Testing-Effort Function Models, can be thoroughly established by applying the concept of weighted geometric, arithmetic, or harmonic mean. The purpose of Zhang et al. [59] is to include fault removal efficiency in the evaluation of software reliability. Debugging is imperfect because the faults found during the process may not be eliminated, and new faults may be introduced into the software. Khoshgoftaar & Woodcock proposed an approach [60] to choose a reliability model over several possible models through the log-likelihood function. An S-shaped model was determined to be the most suitable for the procedure. A deterministic distance-based approach was developed by Sharma et al. [58] and used to choose and rank over sixteen distinct NHPP growth models and concluded that no model is most effective for all contributing criteria.

SRGMs have certain assumptions regarding software failure and development processes, which are not valid in real scenarios. Exponential models assume a static failure rate over time, which may not reflect changing real-world scenarios where the rate can change, making them less effective for complex systems. Logarithmic models suggest that improvements in reliability will decrease over time, which may not hold if significant new faults are introduced, and they are highly sensitive to data quality. The Goel-Okumoto model often relies on a single parameter for fitting, oversimplifying complex failure behaviours. NHPP models are mathematically complex and require extensive historical data for accurate failure rate modelling. Weibull models face challenges in accurately estimating parameters, often needing significant data, while log-logistic models assume a specific failure distribution that may not be suitable for all software systems.

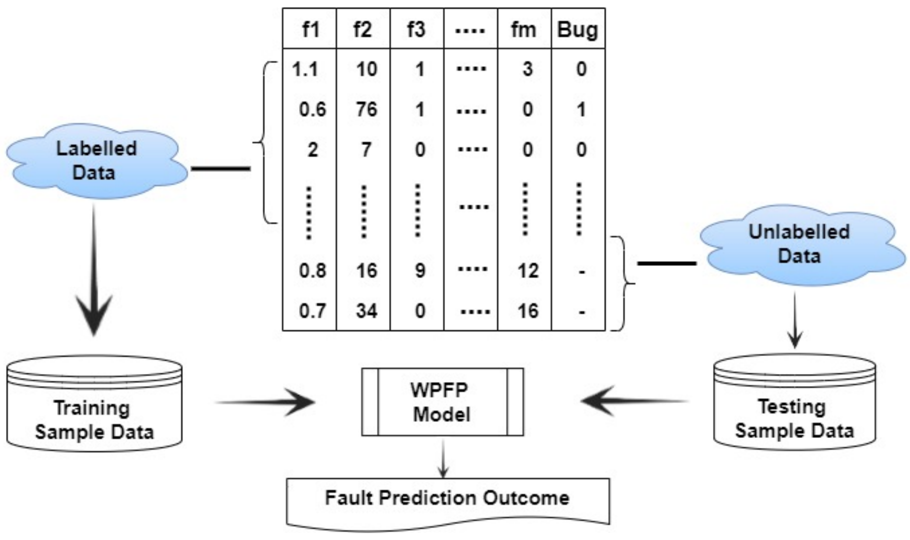
2.2 Early Software Reliability Prediction Models

Reliability growth models provide reliability estimates, but taking corrective actions based on these estimates often occurs too late and at considerable expense. Some reliability estimation approaches treat a software system as a single, undifferentiated entity, neglecting its internal structure, termed black-box approaches. Moreover, many pivotal design decisions concerning a software system are made well before the implementation phase [61]. Therefore, early-stage reliability prediction models hold significant importance, enabling the early detection of cost overruns, process discrepancies in software development, and the formulation of optimal development approaches [61]. Early reliability prediction methods utilize the reliability of individual components and their architectural arrangement to predict the reliability of the entire system during the initial stages of the software development lifecycle, spanning from requirements to coding, known as white-box approaches. In these early phases, precise failure data is absent, hindering the quantitative assessment of software reliability [2]. Hence, predictions rely on various factors such as reliability metrics, expert opinions, developer expertise, and historical failure data from analogous projects.

A standard reference, IEEE Guide for the Use of IEEE Standard Dictionary of Measures provided to produce reliable software [62]. This standard is used by clients, project managers, and developers to ensure the highest level of reliability in software products. A method for predicting early software reliability based on a determination of software process failure modes and modelling of the influence on the software product in the requirement-analysis phase was presented by Smidts et al. in [61]. Functional requirements, performance requirements, memory requirements, accuracy requirements, reliability and safety requirements, and portability requirements are a few examples of the requirement-analysis phase. The reliability prediction model introduced by the U.S. Air Force Rome Air Development Centre (RADC) [63] assesses a system's reliability based on its failure rate. Initially, the software application's fault density is approximated using metrics from software engineering. Subsequently, this fault density is translated into a failure rate using specific conversion ratios outlined by the RADC. However, it overlooks object-oriented metrics and treats the software as a single unit since utilizing this approach for predicting the reliability of software components within the system is unfeasible.

A reliability prediction model for component-based systems has been suggested by Cortellessa et al. [64]. Utilizing a Bayesian methodology, they calculated the model parameters based on historical data regarding component failure probability and system utilization. Later, they employed this model with an appropriately annotated use case and sequence diagrams to estimate system reliability throughout the design phase. Nevertheless, their method may cause an overestimation of system reliability because it ignores the reliability of the component interfaces. In [65], a component's reliability is determined by averaging its services' reliabilities. It is expected that the component services' reliability is known. Nevertheless, this approach overlooks the design structure of the component, potentially diminishing the accuracy of the component's reliability prediction. Ensemble models were constructed in [66] to predict software reliability. The ensembles comprise a range of approaches, such as multivariate adaptive regression, multiple linear regression, dynamic evolving neuro-fuzzy inference system, back propagation neural network, and TreeNet. The non-linear ensemble is trained using a back-propagation neural network, and it gives each strategy a weight according to its prediction capabilities. The nonlinear ensemble exhibited superior performance compared to all other ensembles and individual reliability techniques. Cheung et al. presented a methodology in [3] for estimating component reliability during architectural design. The values derived from this suggested component-level approach can be fed into other system-level reliability models that require such information. In [2], Mohanta et al. proposed a bottom-up approach that extracts design metrics, which have a strong correlation with the specific fault categories, and predicts the reliability of a system based on the reliabilities of its components, where classes are considered the basic components over a simple restaurant automation system.

Accurately predicting the reliability of modules in the early stages of product development poses a significant challenge due to the unavailability of actual field failure data or failure data obtained during testing. Hence, fault prediction models utilize class metrics (module features) of the historical data [67, 68, 69, 70] as the input and assume all modules are independent of each other. Several metrics, such as Coupling Between Objects (CBO), Depth Of Inheritance Tree (DIT), Response For Class (RFC), Weighted Method per Class



f1	f2	f3	....	fm	Bug
1.1	10	1	....	3	0
0.6	76	1	....	0	1
2	7	0	....	0	0
⋮	⋮	⋮	....	⋮	⋮
0.8	16	9	....	12	-
0.7	34	0	....	16	-

Figure 2.2: A typical within-project fault prediction model

(WMC), Lines of Code (LOC), Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), and Coupling Factor (CF) are used for predicting software faults. Effort estimation models utilize project attributes of the historical project's data [71, 72, 73] as the input to train the estimation model. Several attributes, such as analyst's capability, programmer's capability, application experience, modern programming practices, use of software tools, virtual machine experience, language experience, Manager Experience, Team Experience, Language, Turnaround time, Database size, Source lines of code, Raw function point counts, Adjusted function points are used for software development effort estimation. Notably, contemporary methodologies such as machine learning models, neural networks, fuzzy neural networks, ensemble models, advanced data preprocessing techniques, and hybrid approaches have demonstrated significant advancements in prediction of software reliability in terms of fault prediction and effort estimation compared to conventional statistical methods [66]. A detailed literature review of WPFP, CPFP, and SDEE models is given in the following sections.

2.3 Within-Project Fault Prediction

Currently, SFP is gaining more popularity in developing superlative-quality software systems cost-effectively. The basic process of prediction in a WPFP model is depicted in

Figure 2.2. Within the same project, a subset of labelled modules is employed to train, and the remaining unlabeled modules are utilized to test the WPFP model. Research studies on the prediction of software faults emphasized the eminence of many classification models, including LR [74, 75], SVM [76], RF [75], KNN [77], Bayesian Methods [78, 79], ANN and DNN models [80, 81, 82, 83]. A significant problem associated with predicting software faulty modules is the class imbalance issue [84, 85]. In practice, this can be defined where the ratio between faulty and non-faulty instances is very high in datasets used for WPFP, where the classifier yields results partial towards non-faulty instances [86]. For these reasons, fault prediction pays more attention to the class-imbalance issue and more research has been raging on this issue. The imbalance learning strategies are mainly grouped into three prevalent methods:

1) Data level strategies [11, 12, 13, 14, 15, 18, 87, 88, 89, 90]: The first and most preferable approach, with its simplicity, is a variation of the re-sampling (i.e. oversampling or undersampling) approach, whereby an imbalanced dataset with a skewed class distribution is transformed into a balanced dataset by wisely offering new pseudo-instances (i.e. synthetic or artificial instances) into the faulty class. A huge amount of research work has been raging on this approach, such as Random Under Sampling (RUS) [77, 91], Random Over Sampling (ROS) [15], Synthetic Minority oversampling Technique (SMOTE) [11], Borderline-Synthetic Minority oversampling Technique (BSMOTE) [12], Adaptive Synthetic Sampling Approach (ADASYN) [87], Diversity Based Oversampling Approach (MAHAKIL) [14] and Semi-Supervised Oversampling approach based on Trigonal Barycenter Theory (SOTB) [18], to name a few. These data-level approaches produce efficient outcomes with simple implementation, but the accurate outcome depends on the chosen problem and the classification algorithm.

2) Algorithm level strategies [92, 93, 94, 95, 96]: These approaches directly alter the classification mechanism to cope with imbalanced data, such as cost-sensitive learning techniques and one-class learning techniques, to name a few.

3) Ensemble learning strategies [97, 98, 99, 100, 101, 102, 103]: This approach outperforms the existing classifiers by working with a combination of multiple classifiers, namely bagging, boosting, SMOTEBoost, RAMOBoost, and AdaBoost.NC, to name a few.

Among these three approaches, we focused more on sampling strategies, especially oversampling techniques. The cost-sensitive approach works by assigning miss-classification costs [93, 104]. However, it is not clear how much it will cost. There is a lot of research has been done in the field of WPFP through ensemble learning approaches [99, 101, 102, 105], but these methods consume more time for the construction of fault prediction models because they have to combine multiple models. To resolve imbalance issues, the data-level strategies resample the actual dataset and construct synthetic information to alter the faulty class distribution (oversampling) or exclude data from non-faulty classes (undersampling) to prepare a balanced dataset. The data-level strategies benefit over the other two strategies in that they don't alter the classifier and obtain better prediction accuracy. For these reasons, we focused our research on data-level strategies.

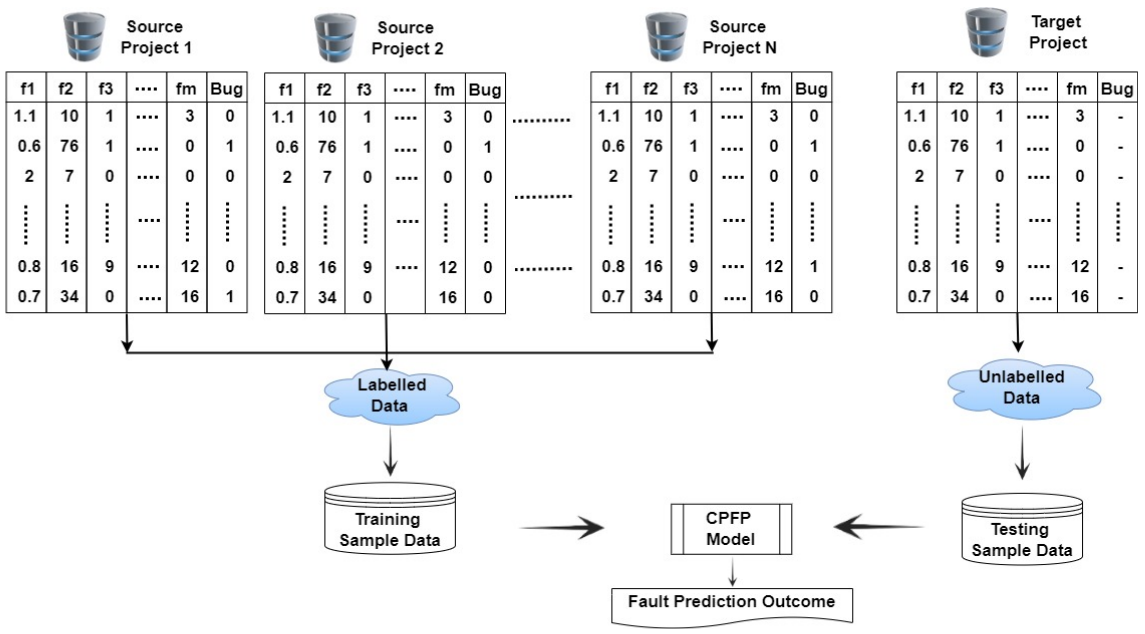
RUS [77, 91] eliminates instances from the non-faulty class of the datasets at random. However, the instances excluded from the non-faulty class might hold essential data for classification models; the usage of undersampling is the least frequent in SFP. ROS [15] randomly selects instances from faulty classes and reintroduces them into the same class so that it can maximize the strength of the class, which may result in severe overgeneralization of the model. Chawla et al. [11] introduced the SMOTE algorithm. Firstly, each faulty instance of a faulty class finds its K-closest neighbors, then it randomly selects one of the closest neighbors and computes a random synthetic instance with the selected neighbour in the second step. During the past two decades, a significant number of researchers have been motivated by the conception of SMOTE, particularly Borderline-SMOTE [12], ADASYN [87], MWMOTE [13], Geometric SMOTE [106], Stable SMOTE [21] and SMOTEFUNA [20], to name a few.

Han et al. [12] suggested Borderline-SMOTE; it selects near border faulty instances and then performs synthetic data generation on those border samples as like SMOTE to maximize the recognition rate of border faulty instances. Unlike SMOTE, Bennin et al. [14] suggested a new oversampling technique called MAHAKIL. It calculates the Mahalonobis distance between the mean vector and all other instances and arranges them in ascending order according to their Mahalonobis distance, then makes two partitions of the data and generates synthetic data according to the chromosomal inheritance theory concept. The synthetic data generated by MAHAKIL has more diversity than the SMOTE-based approaches, which can overcome the overgeneralization problem.

Liu et al. [18] presents SOTB, which uses the concept of trigonal barycenter theory to generate new synthetic instances. It forms non-intersecting triangles and takes an average of them as new synthetic data. Feng et al. [88] presented the Complexity-based Over Sampling Technique (COSTE). COSTE uses differential evolution to assign complexity to each feature of instances to produce synthetic data for further computation. Saez et al. [107] presented the SMOTE-Iterative Partitioning Filter (SMOTE-IPF) model, the oversampling process is the same as SMOTE. Later, it excludes the noisy and borderline samples from oversampled data. Lina Gong et al. [108], presented the KMFOS technique to mitigate imbalanced class issues along with a noise filter. Firstly, the K-means algorithm is utilized to generate these K clusters. Further, from those clusters, it generates synthetic data and then filters out noise instances. Khleel et al. [90] introduced a novel approach to fault prediction by employing bidirectional long short-term memory (Bi-LSTM) networks, a sophisticated deep learning method, combined with strategic oversampling techniques. Experiments conducted on standard datasets from the PROMISE repository validate the effectiveness of the proposed model and showed notable enhancements compared to unbalanced data outcomes. Arun et al. [89] introduced an oversampling method based on multipatch clustering to address class imbalance and handle smaller disjuncts in fault prediction. The performance of this approach was evaluated using five different machine learning models, demonstrating a reduction in false alarm rate.

Currently, deep learning is gaining popularity and for SFP. Zhao et al. [81] introduced a novel method for assessing code functional similarity called DeepSim. They created a semantic representation by encoding a matrix based on data and code control flow, then employed a DNN model to extract features from the matrix and perform fault prediction classification. Qiao et al. [82] analyze the performance of imbalanced software datasets through deep learning. They employ two variants of deep learning models (multi-layer perceptron and convolution neural networks) over four NASA datasets.

We observed that many proposed techniques tend to overlook the issue of class imbalance. However, studies that have specifically addressed this problem emphasize the pivotal



Source Project 1

f1	f2	f3	....	fm	Bug
1.1	10	1	....	3	0
0.6	76	1	....	0	1
2	7	0	....	0	0
⋮	⋮	⋮	....	⋮	⋮
0.8	16	9	....	12	0
0.7	34	0	....	16	1

Source Project 2

f1	f2	f3	....	fm	Bug
1.1	10	1	....	3	0
0.6	76	1	....	0	1
2	7	0	....	0	0
⋮	⋮	⋮	....	⋮	⋮
0.8	16	9	....	12	0
0.7	34	0		16	0

Source Project N

f1	f2	f3	....	fm	Bug
1.1	10	1	....	3	0
0.6	76	1	....	0	1
2	7	0	....	0	0
⋮	⋮	⋮	....	⋮	⋮
0.8	16	9	....	12	1
0.7	34	0	....	16	0

Target Project

f1	f2	f3	....	fm	Bug
1.1	10	1	....	3	-
0.6	76	1	....	0	-
2	7	0	....	0	-
⋮	⋮	⋮	....	⋮	⋮
0.8	16	9	....	12	-
0.7	34	0	....	16	-

Figure 2.3: A typical cross-project fault prediction model

role of data balancing methods in enhancing the accuracy of SFP [11, 12, 14, 20, 18]. Particularly, oversampling with diverse synthetic data is crucial for minimizing false positive rates. Recent research underscores that combining machine learning with data balancing techniques can effectively boost prediction accuracy. Consequently, our study aims to tackle the class imbalance problem in WPFP through the oversampling technique.

2.4 Cross-Project Fault Prediction

A newly launched or small project could not have adequate training data from the same project, these WPFP models are inappropriate in such cases. Therefore Nagappan et al. (2006) [109] investigated how a prediction model trained on one project's history is adaptable to other projects. Consequently, numerous studies have been recommended to develop CPFP models, in which training data is gathered from existing projects for a newly developed project [23, 25, 26, 29, 32, 36, 37, 38, 42, 109, 110, 111, 112]. The fundamental process of CPFP is illustrated in Figure 2.3. In this process, the CPFP model is trained using labeled historical source projects' data and then evaluated using the unlabeled data from the target project.

The majority of the CPFP studies focused on how to minimize the distributional difference between cross-project data. Transfer learning is the most commonly used technique to reduce distribution differences [23, 24, 25, 26, 27, 28, 37, 112], which explicitly maps training and testing projects into a common feature space where the dissimilarity between the two projects' data distribution is minimal. Later, training data selection models [29, 30, 31, 43, 113] utilize the k-NN concept to select similar characteristics training data. In this way, a machine learning model trained on appropriate source data can more accurately detect the target data. Another approach to address the distribution gap is through source project selection. Instead of randomly choosing source projects for training, models select source projects based on their similarity to the target project, thereby reducing the distribution gap [30, 36, 37, 39, 38, 40]. Two additional challenges are linked to software datasets. The first challenge concerns the imbalance within these datasets [17, 25, 41, 42]. The second issue is the curse of dimensionality in software datasets [41, 43, 44, 114].

For the past two decades, researchers have been working on these issues to improve CPFP performance. According to our research, Basili et al. [110] accomplished the first kind of cross-project fault prediction work. Based on a logistic regression prediction model that was trained using data retrieved from one open-source project and predicted from another project. Briand et al. [111] investigated the adaptability of fault prediction models across projects that came from an identical development setting. Current CPFP research, however, typically assumes that cross-project data originates from sources outside of an organization or a company. Zimmermann et al. [32] performed cross-project predictions for 622 pairs from 28 projects and found that only 21 pairs (3.4%) were successful in achieving Accuracy, Recall, and Precision measure values above 0.75. In addition, they discovered that the association among the projects—i.e., the mere fact that project A is appropriate for training an effective prediction model for project B does not automatically imply that project B is also appropriate for training an effective model for project A.

2.4.1 Source project Selection Models

To reduce the distribution gap between source and target projects, many studies first concentrate on the concept called source projects selection, which selects similar source projects to individual target projects. According to a study by He et al., [33], building a prediction model from the same project's data may not always result in more accurate predictions. Instead, the predictive performance of the cross-project prediction model can be significantly increased to 52.9% by using meticulously selected cross-project data. Moreover, they select the training data projects with 16 different distributional features of both the training and target data to accomplish this success rate. The exponential run time of this approach, however, means that it cannot scale to large data sets. Jureczko and Madeyski [34] attempted to group various software fault projects into multiple clusters of similar characteristics under the assumption that a prediction model should perform effectively for every project that comes under the same group, based on the approaches called Kohonen's neural network, k-means, and hierarchical clustering. Menzies et al. [35] suggest partitioning the projects into clusters with extremely comparable data can choose source projects wisely. Herbold [36] presented the nearest neighbour and EM clustering algorithms using distance-based and clustering-based approaches, respectively, which select the source projects based on the distributional properties of the data of the readily available source project and the target project. He assessed the model over 44 open source projects for a certain target project. A two-phase transfer learning model (TPTL), which combines the source project selection and transfer learning model for CPFP, was proposed by Liu et al. [37]. The TPTL selects two source projects with the highest distributional similarity to a target project, and then models are constructed using the approach used in [23] to enhance the CPFP performance. The results indicate that TPTL can resolve the instability of TCA+ over different source projects. Sun et al. [38] put forward a collaborative filtering-based source project selection (CFPS) method for CPFP. Computed the inter-project similarity score, followed by the inter-project applicability score, and then selected three appropriate source projects for a specific target by performing collaborative filtering recommendations over both scores. Kim et al. [39] proposed a source project selection approach for heterogeneous CPFP by controlling the imbalance issue called CorrelAtion-based selection of Multiple source projects and Ensemble Learning (CAMEL). Three phases of CAMEL's method are used to solve heterogeneous CPFP. First, features are matched using the Kolmogorov-Smirnov test to turn heterogeneous data into homogeneous data. Next, multiple source projects are selected using correlation analysis. Finally, the ensemble learning model is trained based on the selected source project data and predicts labels of the target data to deal with the issue of class imbalance.

2.4.2 Transfer Learning and Training Data Selection Models

The aforementioned studies are not altering the data but instead attempting to choose data that is already comparable under the same assumption that similar distributions perform more effectively for CPFP. Furthermore, alternative methods for managing training data and prediction models like transfer learning models, instance selection models, feature learning techniques, and imbalance learning techniques are probably required to overcome issues faced by CPFP. To build a fault prediction model that learns from a Java project and predicts over a C++ project, Watanabe et al. [115] transformed metric values of trained data in accordance with test data. It was discovered that the transformation can improve the recall of predictors applied in a different environment. Turhan et al. [29] explored cross-company fault prediction, which involves using data from other companies to build fault predictors for local projects. They investigated three Turkish and seven NASA projects. They presented a method to select the training data based on the k nearest neighbour technique by selecting 10 instances from all available project data that are most similar to each target instance, then concluded that cross-company data increases the true positive rate while raising the false positive rate. Meanwhile, Pan et al. [22] proposed a domain adaptation transfer learning algorithm called the Transfer Component Analysis (TCA) method. Instead of performing training data selection, TCA maps cross projects into common feature space to make distribution differences close. Nam et al. [23] further extended the TCA to TCA+ by including data preprocessing techniques to maximize CPFP performance. Considering cross-company fault prediction, Ma et al. [116] proposed the Transfer Naive

Bayes (TNB) method, where the source and target data come from distinct companies. TNB determines the distribution of the test data, transforms the cross-company data information into the weights of the training data, and then constructs a fault prediction model through weighted training data and investigated the NASA, PROMISE, and SOFTLAB projects. Bala et al. [117] address the distribution gap and high-dimensionality issues affecting the CPFP through transformation and feature selection approaches. Gong et al. [25] developed a novel class imbalance learning approach for WPFP and CPFP called STrNN (stratification embedded in nearest neighbor). First, TCA was used to transform the source and target data into the same distribution, and then STrNN was applied to reduce the imbalance between faulty and non-faulty classes. Limsettho et al. [118] developed a novel approach called Class Distribution Estimation with Synthetic Minority Oversampling Technique (CDE-SMOTE), which employs CDE to determine the class distribution of a certain target project. The training data's distribution is subsequently modified through SMOTE until it resembles the target project's approximate class distribution. Amasaki [119] looked at the effectiveness of CPFP models for cross-version prediction. Zhu et al. [120] developed a just-in-time fault prediction model based on denoising autoencoder (DAE) and convolution neural network (CNN) called DAECNN-JDP. First, DAE generates a more robust representation of features, and then CNN maps this basic feature representation into the abstract deep semantic feature representation to improve the recognition rate of both faulty and non-faulty instances accurately. To deal with the distribution gap and high-dimensionality issues in CPFP models, Ni et al. [121] developed the Feature Selection Using Clusters of Hybrid Data (FeSCH) model. FeSCH initially groups features into clusters through a density-based clustering technique and then, in the subsequent phase, uses various heuristic ranking techniques to choose features from each cluster. A two-layer approach was presented by Xia et al. [122] to capture similar features between the source and target projects through the genetic algorithm combined with the ensemble model and the advantages of various classifiers. Bhat et al. [31] proposed a combined data transformation and instance filtering model called BurakMHD. Initially, data transformation was applied to numerous source project data and target data, and then training data was constructed by selecting k-instances from transformed source data with the minimum hamming distance from each individual target project instance. Hosseini et al. [43] generated training data through the filtering method, which is the genetic algorithm integrated with the NN-filter [29].

In [42], kaliraj et al. explore different classifiers on diverse datasets sourced from multiple software projects. Subsequently, the effectiveness of CPFP should be assessed to gauge how well models trained on one project can predict faults in others. Additionally, it analyzes the impact of augmenting training samples from various projects on prediction accuracy. Khatri et al. [44] investigated the transfer of knowledge between a source and target through feature selection. Their proposed approach comprises two distinct feature selection strategies, each with its focus on balancing cost and performance. Both strategies were evaluated through 26 cross-project experiments involving eight software projects. In [123], Tong et al. proposed a new effective Adaptive Triple Feature-Weighted Transfer Naive Bayes (ARRAY) method. They introduced the idea of feature-weighted instance weight, determined by feature importance and weighted similarity among features, to limit the distribution gap. They conducted experiments on 34 datasets and evaluated performance using metrics such as AUC, F1-score, and MCC, with statistical testing methods applied.

From the literature, we observed that many proposed techniques don't focus on all the issues faced by CPFP. However, studies that have specifically addressed distribution difference, imbalance, and high dimensionality problems emphasize the accuracy of CPFP models [25, 27, 37, 38, 41, 44]. Recent research underscores that combining transfer learning and source project selection models with sampling and feature optimization techniques can effectively boost prediction accuracy. Consequently, our study aims to address the aforementioned issues in CPFP to improve prediction accuracy.

2.5 Software Development Effort Estimation

Due to the pivotal role that SDEE holds within the development process, a variety of estimation approaches have been established to improve the accuracy of estimation. The fundamental process of SDEE is illustrated in Figure 2.4. According to [124], software ef-

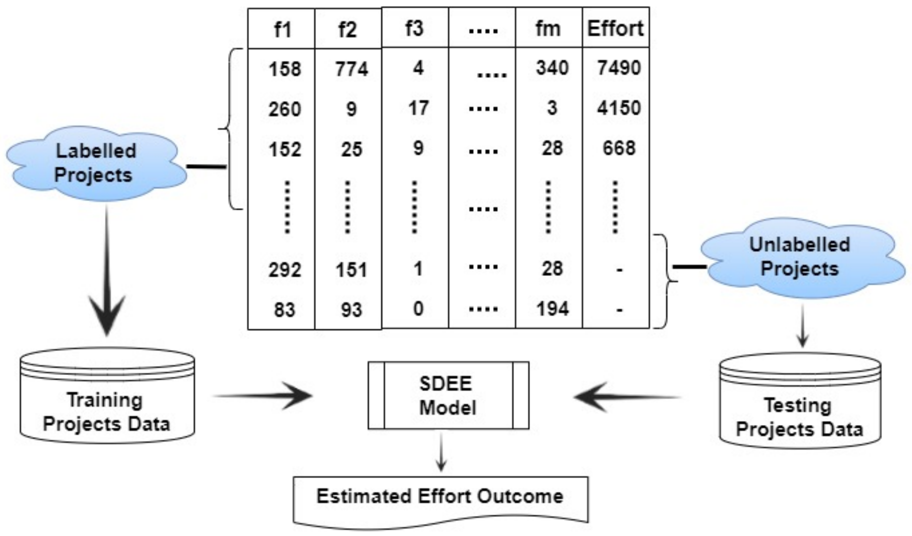


Figure 2.4: A typical software development effort estimation model

fort estimation techniques fall into six distinct categories, later, they are grouped into three categories such as expert judgment, algorithmic models, and machine learning processes. Conventional algorithmic models, such as the constructive cost model (COCOMO) [125], software life cycle management (SLIM) [126], and Function Point Analysis (FPA) [127] perform estimations based on a statistical analysis of project input data, which implies the effort is computed by a mathematical model from the numerical inputs of one or more projects. In the case of expert judgment, Delphi [128, 129] and work breakdown structure [130] methods rely on past experience of a domain expert to estimate the required effort. The problem here is the lack of an objective foundation and in situations where data or expertise in numerical techniques is limited.

Machine learning (ML) techniques have emerged as promising strategies, showing comparable accuracy to algorithmic methods while also potentially offering enhanced comprehensibility and application ease [131]. Most of the ML models in effort estimation fall under categories like regression models, CBR models, ABE models, ANN, fuzzy neural networks, extreme learning, and ensemble learning models. However, conventional regression models are notably sensitive to outliers and also necessitate considerably a large dataset for training, which is often difficult in the software effort estimation domain.

Optimization techniques are widely applied in SDEE to fine-tune the parameters in effort estimators like ABE [47, 48, 51, 132, 133, 134]. An artificial Bee colony guided

Analogy-Based Estimation (BABE) model incorporates the Artificial Bee Colony (ABC) algorithm with ABE to enhance estimation accuracy. While training, ABC generates and updates various feature weights and integrates them into the similarity function of ABE by reducing the MMRE value, out of which the optimal feature weight vector is utilized to accurately estimate the testing data [47]. In [52], Hameed et al. integrated the genetic algorithm (GA) into the CBR technique to address the challenge of tuning multiple parameters. GA optimizes the feature weights and a number of neighbors for CBR, aiming to achieve the most accurate effort estimates by minimizing absolute estimation errors. The results achieved in terms of absolute and relative errors are better than the fixed k-number of neighbors. In this study [135], a variety of ensemble methods have been explored for effort estimation, where stacking using a random forest ensemble approach yields superior results when compared to single-model approaches. In [136], De et al. selected influential features using Pearson's correlation coefficient and subsequently applied the Extreme Learning Machine (ELM) model over selected features for effort estimation. ELM demonstrated significantly better performance compared to conventional regression models. Kassaymeh et al. [137], employed a fully connected neural network (FCNN) model coupled with gray wolf optimizer (GWO), referred to as GWO-FC. This integration aims to optimize the weights and bias parameters of the FCNN, enhancing result accuracy. Khan et al. [138] enhanced the deep neural network by integrating strawberry and grey wolf optimizer algorithms to optimize both learning rate and initial weights for effort estimation.

In a pool of a wide variety of ML techniques, neuro-fuzzy frameworks prove effective in establishing complex relationships between diverse features of data [49]. Lately, the ANFIS is getting notable attention among effort estimation models [45, 46]. A neural network can train itself by learning from a dataset, a fuzzy system necessitates explicit user-defined parameters. Conversely, neural networks solely perform predictions without acquiring knowledge, while fuzzy inference systems effortlessly extract knowledge from user input. Thus, the combination of these two models is beneficial in effort estimation. Azzeh et al. [139] proposed a new software project similarity measure and adaptation technique in ABE through fuzzy numbers concept to improve the performance of software project effort estimation in the early stages, and empirical analyses were carried out using five benchmark data sets. In [140], classical and fuzzy analogy-based approaches are combined to enhance effort estimation. The results indicate that the ensemble based on fuzzy analogy outperforms the classical analogy model. Agrawal et al. [141] developed a neuro-fuzzy model to classify software projects and establish rules. The neuro-fuzzy model with trapezoidal, triangular, and Gaussian membership functions is compared with the neural network models, and it is observed that the trapezoidal membership function yielded superior results compared to all the other models. Moosavi et al. [46] proposed a novel metaheuristic algorithm known as Satin Bower Bird Optimization (SBO) and integrated it with ANFIS to optimize both premise and consequent parameters by reducing MMRE and enhancing PRED scores. ANFIS-SBO was assessed using three publicly available effort estimation data sets and obtained better outcomes. Karimi et al. [50] presented a hybrid model of ANFIS and the differential evolution algorithm for effort estimation. Sharma et al. [142] proposed a fusion of the Genetic Elephant Herding Optimization-based Neuro-Fuzzy Network (GEHO-NFN). A neuro-fuzzy network is used to estimate the effort in terms of cost, while the GEHO method optimizes NFN parameters. In [49], the neuro-fuzzy inference system examined the elements of an ANN with fuzzy logic to provide more accurate estimations. Nanda et al. [45] proposed a hybrid effort estimation model known as ANFIS-PSO. The ANFIS model uses the particle swarm optimization technique to optimize the parameters associated with triangular and Gaussian membership functions by considering the mean standard error as fitness and experiments performed over the NASA dataset. Edinson et al. [143] designed the ANFIS model using the fuzzy C-means clustering and subtractive clustering techniques to compute the software effort and compare it with the Elman neural network. Ali et al. [144] conducted an empirical investigation by integrating bio-inspired and non-bio-inspired feature selection algorithms with basic estimation models. The findings revealed that bio-inspired models outperformed non-bio-inspired models. Recently, the social network search (SNS) algorithm has been used across various optimization problems and demonstrated its superiority in different domains [145].

The research works mentioned above aimed to generate accurate estimates through various methods. However, many of them utilizing ANFIS didn't address the challenge associated with the complexity arising from the number of parameters generated by the number of features. In ANFIS, the number of rules, premise, and consequent function parameters increase exponentially with the number of inputs. This justifies the importance of the feature selection, parameter optimization, and adaptation processes as an essential part of classical ANFIS. Our research endeavours to address this gap by leveraging the SNS technique to optimize the number of features and ANFIS parameters to enhance effort estimation capability.

2.6 Summary

In this chapter, we explore various models for predicting software reliability, including early software reliability models and software reliability growth models. We pay particular attention to the parameters of the ESRP model, such as fault prediction and effort estimation models, and discuss their merits and limitations. Furthermore, it provides an in-depth literature review of studies conducted within the scope of this thesis. The next chapter presents a weighted average centroid based oversampling approach for within-project fault prediction.

Chapter 3

A Diverse Oversampling Technique for Within-Project Fault Prediction

Currently, fault prediction models are gaining popularity for predicting software reliability in the early phases of development. As discussed in Chapter 1 and Chapter 2, there is an important problem associated with the prediction of software faults, that is the class imbalance issue [84, 85]. In this chapter, we proposed an imbalance learning approach to reduce the percentage of class imbalance between faulty and non-faulty instances to maximize the prediction capability of the classifier.

The remainder of this chapter is organized as follows: Section 3.1 describes our proposed WACIL approach. The description of the experimental setup and a demonstration of the experimental results are provided in Section 3.2 and 3.3, respectively, while 3.4 provides a thorough discussion of those results. Section 3.5 discusses the threats to the validity of our approach. Finally, the chapter summary is provided in Section 3.6.

3.1 Weighted Average Centroid based Imbalance Learning (WACIL) for Within-Project Fault Prediction

The WACIL approach is accustomed to build balanced datasets through the introduction of pseudo-instances into the faulty class. The proposed oversampling algorithm comprises two stages: the first stage is the selection of hard-to-classify (borderline) instances from the faulty class (minority class). The second stage is the introduction of filtered pseudo-instances into the faulty class. The two stages are repeated until the training data becomes balanced. The comprehensive description of the two stages is listed in the following sub-sections.

3.1.1 Extraction of borderline Instances

Many traditional classifiers aim to extract and learn decision boundaries to achieve better performance. Instances that are far from the decision boundary are more likely to be classified accurately; conversely, instances near or on the decision boundary have a higher likelihood of being misclassified. We can refer to these as hard-to-classify instances. Therefore, it is important to generate pseudo-instances of hard-to-classify faulty instances to build a better classifier.

In our proposed approach (As shown in Algorithm 1), we designed a way to extract important hard-to-classify borderline instances of faulty class and construct the borderline faulty instance set, FS$_{binc}$. Suppose that the whole training set is 'T', the faulty class set is FS and the non-faulty class set is NFS and the whole process of extracting borderline instances in set FS$_{binc}$ (lines 1-13 of Algorithm 1) can be described as follows:

We have experimented with projects of high imbalance and moderate imbalance rates from the PROMISE and NASA datasets for different K (closest neighbour) values. The experimental results indicate that K = 5 produces better performance for both highly and moderately imbalanced projects, so we consider K = 5 for all the experiments.

1. The foremost step in our approach is the identification of the closest neighbors of each and every instance (CN(Inc_i)) of faulty modules in FS and NFS, differently, according to the cosine similarity measure by considering K = 5. For each instance (Inc_i), compute the sum of the distances of the closest neighbors from NFS and FS and store those distances in sets D_{NFS} and D_{FS}, respectively. For Inc_i, if the sum of the nearest neighbors distance ($dist_{Inci}$) $\in D_{FS}$ is greater than $dist_{Inci} \in D_{NFS}$ then append Inc_i to a new temporary set (FS$_{tmp}$), else the instance is considered as a

noise instance and not added to FS$_{tmp}$.

2. The further step is, for each $Inc_i \in$ FS$_{tmp}$, identify the closest neighbors (CN (Inc_i)) in the total original training dataset according to the cosine similarity measure by considering K = 5. Make a set of union of all the closest neighbors from NFS in NFS$_{tmp}$, for which Inc_i the closest neighbors from NFS are greater than or equal to three. Here, instances having three or more out of the five closest neighbors from the non-faulty class (NFS) will share more boundaries with the non-faulty class. Therefore, we have considered the count to be greater than or equal to three.

3. Now the conclusive move of extraction of borderline instances, for each $Inc_i \in$ NFS$_{tmp}$, identifies the closest neighbors (CN(Inc_i)) in FS according to the cosine similarity measure by considering K = 5. Take a set of union of all the closest neighbors in FS$_{binc}$. The borderline instances in set FS$_{binc}$ are used to generate filtered pseudo-instances of the faulty class.

3.1.2 Weighted Average Centroid based Pseudo-instance Generation

This stage aims to balance the dataset through diverse pseudo-instances introduction in faulty class. After identifying borderline faulty modules, the process proceeds with calculating the Mahalanobis distance and generating filtered pseudo-instances to oversample the faulty class data. This stage explains how the pseudo-faulty instances are generated to make the balanced dataset.

3.1.2.1 Mahalanobis Distance Computation

The first and foremost step is determining the distance of extracted borderline instances of a faulty class from their mean. The Euclidean distance has been a chart-topping measure for decades [146] for determining the distance between two points, but it is diplomatic towards highly correlated features and units of features. For example, if only one feature of two instances holds more difference, then the Euclidean distance between them is very high, irrespective of other less differenced features. This can mislead the results. For this reason, Mahalanobis distance (MD) is considered in this study as a distance measure.

MD can be defined as a distance measure between point x_i and Distribution μ. MD is calculated as follows in Eq. 3.1,

$$\text{MD}(x_i, \mu) = \sqrt((x_i - \mu)^T \times COV^{-1} \times (x_i - \mu)) \tag{3.1}$$

where $X = \{x_1, x_2, x_3, \ldots, x_n\}_{n\times m}$, COV^{-1} is the inverse co-variance matrix and $\mu = \{\mu_1, \mu_2, \mu_3. \ldots, \mu_m\}_{m\times 1}$. μ is the mean distribution of all features, μ_1 is mean of first feature, μ_2 is of second feature , ..., and μ_m is of m^{th} feature, respectively. The 'm' represents the total number of features, 'n' represents the total number of instances in the dataset, and x_i is an instance.

A few datasets produce singular covariance matrices. We can't generate the inverse of singular covariance matrices. In such circumstances, we computed Moore–Penrose inverse/generalized inverse [147] of a covariance matrix for Mahalanobis distance calculation.

3.1.2.2 Filtered Pseudo-instance Generation

Suppose the number of faulty modules is represented as Np and the number of non-faulty modules as Nn, then the required number of faulty pseudo-instances to be generated is calculated as N = Nn - Np. The set of borderline instances is arranged in ascending order with respect to their Mahalanobis distance and then divided into three equal cases or sets, followed by pseudo-instances generated. Particularly, for a few datasets the number of faulty modules that occur close to the boundary is extremely low, so the algorithm functions appropriately through three partitions. Instances are sorted in ascending order, so Case_1 contains samples that are too close to the mean distribution, Case_2 contains fair distanced samples and Case_3 contains far distanced samples.

After partition, to cope with the problem of nearest sample synthesis, we shuffle the instances of all three cases. The following step is the tagging process: In all three cases, instances are tagged with the same labels, sequentially. Case_2 and Case_3 pursue a consistent pattern of Case_1, which signifies case_2 and case_3 instances tagged with similar labels as Case_1.

Algorithm 1: WACIL Algorithm

```
    Require: Imbalanced training dataset ($\mathbf{T}$) and K;
    Ensure: Balanced training dataset ($\mathbf{T}'$);
1   Make a partition of $\mathbf{T}$ in to faulty class set ($\mathbf{FS}$) and non-faulty class set ($\mathbf{NFS}$)
2   For each faulty instance $Inc_i \in \mathbf{FS}$, compute K closest instances in ($\mathbf{FS}$) and store the sum of the
      distance of all K neighbors in $\mathbf{D_{FS}}$ according to cosine similarity measure
3   In similar way, for each faulty instance $Inc_i \in \mathbf{FS}$, compute K closest instances in $\mathbf{NFS}$ and store
      the sum of the distance of all K neighbors in $\mathbf{D_{NFS}}$ according to cosine similarity measure.
4   Initialize an empty set $FS_{tmp}$
5   if $dist_{Inci} \in \mathbf{D_{FS}} > dist_{Inci} \in \mathbf{D_{NFS}}$ then
6       $FS_{tmp}$.append($Inc_i$)
7   end
8   Initialize an empty set $NFS_{tmp}$
9   For each instance $Inc_i \in FS_{tmp}$, compute K closest instances ($CN(Inc_i)$ in $\mathbf{T}$
10  if count of $CN(Inc_i) \in$ NFS $\geq$ 3 then
11      Take union of those closest instances belongs to NFS in $NFS_{tmp}$
12  end
13  For each instance $Inc_i \in NFS_{tmp}$, compute K closest instances in FS , then take the union of all
      these closest instances in $FS_{binc}$
14  Compute the number of Pseudo-instances $(N)$ to be generated, $N = Nn - Np$
15  Initialize $N_{cnt}$, to keep track of generated pseudo-instances
16  while $N_{cnt} < N$ do
17      Compute Mahalanobis distance (MD) between $\mathbf{FS}_{binc}$ and mean $\mu$ using Eq. 3.1
18      Arrange $\mathbf{FS}_{binc}$ in ascending order with respect to its mahalanobis distance
19      Initialize an empty set $Inc_{pseudo}$
20      Initialize $n$ with $arraylength(\mathbf{FS}_{binc})$
21      Create three partitions from ($\mathbf{FS}_{binc}$)
22      Compute each partition length, j=n/3
23      Case_1 = $\{\mathbf{FS}_{binc(1)}, \mathbf{FS}_{binc(2)}, \ldots \mathbf{FS}_{binc(j)}\}$,
            Case_2 = $\{\mathbf{FS}_{binc(j+1)}, \mathbf{FS}_{binc(j+2)}, \ldots \mathbf{FS}_{binc(2j}\}$ and
            Case_3 = $\{\mathbf{FS}_{binc(2j+1)}, \mathbf{FS}_{binc(2j+2)}, \ldots \mathbf{FS}_{binc(3j)}\}$
24      for $i = 1, 2 \ldots \ldots j$ do
25          Inc1=Case_1[i], Inc2=Case_2[i] and Inc3=Case_3[i]
26          if $N_{cnt} < N$ then
27              $S \leftarrow$ min(Inc1,Inc2,Inc3)+rand(0,1)(max(Inc1,Inc2,Inc3)-min(Inc1,Inc2,Inc3))
28              Compute K closest instance of $S$ in $\mathbf{T}$
29              Calculate the number of non-faulty instances ($n1$) and the number of faulty instances
                  ($n0$) in K-closest neighbors
30              if $n0 > n1$ and $n0 \neq K$ then
31                  $Inc_{pseudo} \leftarrow Inc_{pseudo} + S$
32                  $N_{cnt} \leftarrow N_{cnt} + 1$
33              end
34          end
35      end
36      $\mathbf{FS}_{binc} = \mathbf{FS}_{binc} + Inc_{pseudo}$
37  end
38  $\mathbf{T}'$ = Union (NFS, FS, FS$_{binc}$)
39  return $\mathbf{T}'$
```

Tag_set= $\{tag_1, tag_2, \ldots tag_j\}$, this Tag_set is the same for all three cases for all instances, sequentially. Where j=n/3 and 'n' is the total number of instances.

The generated faulty modules from these three partitions are filtered and included in the training set. For one combination select instances with tag_1 from all cases (inc_1, inc_{j+1} and inc_{2j+1}), for the next combination select instances with tag_2 from all cases and repeat similar type combination selection for the remaining instances. It takes the weighted average of the selected combination of all three instances.

Suppose, Inc1 $\in$ Case_1, Inc2 $\in$ Case_2, and Inc3 $\in$ Case_3 have same tag.

Inc1 = $\{inc_{11}, inc_{12}, \ldots inc_{1m}\}$

Inc2 = $\{inc_{21}, inc_{22}, \ldots inc_{2m}\}$

Inc3 = $\{inc_{31}, inc_{32}, \ldots inc_{3m}\}$.

Where 'm' represents the number of features present in the training dataset. Unlike existing techniques, the minimum and maximum values belong to different instances, thus the generated pseudo-faulty module resides in between them and shares similar features with all of these modules. The pseudo instances are generated using the following equation Eq. 3.2:

$$\begin{aligned} Min &= [Minimum(Inc1, Inc2, Inc3)]_{1\times m} \\ Max &= [Maximum(Inc1, Inc2, Inc3)]_{1\times m} \\ rand &= [random(0,1)]_{1\times m} \\ New\_inc &= Min + rand(0,1) \times (Max - Min) \end{aligned} \tag{3.2}$$

After generating pseudo-faulty modules, there could be some undesirable or redundant instances generated during implementation. These instances can influence the performance of the prediction model. They often occur in the space of the non-faulty class, share many more boundaries with the non-faulty class, or are completely redundant with the actual faulty module. To find out these undesirable instances, a cosine similarity measure is employed. Firstly, we apply cosine similarity to get the closest neighbors of a pseudo-instance by considering K = 5. Then, take a different count of the nearest neighbors for faulty modules and non-faulty modules, respectively. If the number of faulty modules is less than the number of non-faulty modules, then that faulty instance is undesirable. Otherwise, if the number of faulty modules is equal to K, then that pseudo-instance is considered as a redundant instance. These instances should be excluded from the pseudo-instance set.

Repeat the above explained two stage process until the generated filtered pseudo-instances count equals N, which means the ratio of faulty and non-faulty modules becomes 1:1. Through these two stages, the training dataset is constructed. For example, assume a dataset with Nn = 680, Np = 70, and N becomes 610 (680-70=610). Our proposed algorithm iterates (lines 16–37 of Algorithm 1) until the number of pseudo-faulty class samples reaches 610. After all iterations, the ratio of both classes turns to 1:1 and the dataset becomes balanced. As a result, prediction models trained on this dataset can equally learn parameters from both classes, ensuring unbiased results and preventing model overgeneralization.

3.2 Experimental Setup

In this Section, we report an overall summary of the datasets and a detailed description of the model performance assessment measures used to evaluate our simulation work. Next, we briefly explain baseline methods, classifier selection, experimental framework design, and statistical comparison measures.

3.2.1 Experimental Objects

To assess our proposed approach, we conducted experiments using a total of 24 diverse datasets. Among these, 14 were sourced from PROMISE projects [34, 148], while the remaining 10 were obtained from NASA projects [25, 70, 149, 150]. Detailed information about the datasets is provided in Table 3.1, consisting of dataset names and their total number of instances, imbalance rate and total metrics (features). The PROMISE project datasets imbalance rate varies from 8.97% to 46.67% and the NASA project datasets varies from 2.15% to 35.2%. Each instance corresponds to a module in the software, with varying sizes and distinct characteristics. Although these modules share the same features, they exhibit different distributions, highlighting their unique properties within the dataset. In datasets, each instance (module) is associated with a specific number of faults; thus, we transformed the datasets for binary classification by labeling a value of 0 to modules with zero faults and a value of 1 to modules with one or more faults. The detailed descriptions of PROMISE and NASA features (metrics) are given in Tables 3.2 and 3.3.

Table 3.1: Summary of PROMISE and NASA project datasets

Project	Datasets	#Total instances	# faulty instances (%)	#Metrics	Project	Datasets	#Total instances	# faulty instances (%)	#Metrics
PROMISE	ant-1.7	745	166(22.28%)	20	PROMISE	xalan-2.4	723	110(15.21%)	20
	arc	234	27(11.54%)	20		xerces-1.3	453	69(15.23%)	20
	camel-1.6	965	188(19.48%)	20	NASA	CM1	327	42(12.84%)	37
	ivy-2.0	352	40(11.36%)	20		MW1	253	27(10.67%)	37
	jedit-4.2	367	48(13.10%)	20		KC3	194	36(18.56%)	39
	log4j-1.0	135	34(25.19%)	20		MC1	1988	46(2.31%)	38
	lucene-2.0	195	91(46.67%)	20		MC2	125	44(35.20%)	39
	poi-2.0	314	37(11.78%)	20		PC1	705	61(8.65%)	37
	redaktor	176	27(15.34%)	20		PC2	745	16(2.15%)	36
	synapse-1.2	256	86(13.85%)	20		PC3	1077	134(12.44%)	37
	tomcat	858	77(8.97%)	20		PC4	1287	177(13.75%)	37
	velocity-1.6	229	78(34.06%)	20		PC5	1711	471(27.53%)	38

3.2.2 Performance Assessment Measures

Table 3.4 illustrates the confusion matrix for a binary classification task, where faulty modules are labelled as positive and non-faulty modules as negative. The TP cell contains true positive results, defined as the number of faulty modules identified as faulty. The TN cell contains true negative results, defined as the number of non-faulty modules identified as non-faulty. The FP cell contains false positive results, defined as the number of non-faulty modules identified as faulty modules. The FN cell contains false negative results, defined as the number of faulty modules identified as non-faulty modules.

Most of the Software datasets are imbalanced in nature. Hence, we should go with multiple assessment measures. The performance assessment measures considered in this work are Fall Out Rate (FOR), Recall, F-measure, G-mean, and AUC [14, 25, 151, 152, 153]. The AUC is a balanced measure [154], it is a trade-off between FOR and Recall. The G-mean incorporates the classifier's accuracy into faulty and non-faulty class modules [151]. Measures are defined as follows in Eqs. 3.3 - 3.6:

$$FOR = \frac{FP}{FP + TN} \tag{3.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

$$F - measure = \frac{2TP}{2TP + FP + FN} \tag{3.5}$$

Table 3.2: List of features from 14 PROMISE projects

No.	Features	No.	Features
1	WMC (Weighted methods per class)	11	LOC (Lines of code)
2	DIT (Depth of Inheritance Tree)	12	DAM (Data access metric)
3	NOC (Number of children)	13	MOA (Measure of aggregation)
4	CBO (Coupling between object classes)	14	MFA (Measure of functional abstraction)
5	RFC (Response for a class)	15	CAM (Cohesion among methods)
6	LCOM (Lack of cohesion in methods)	16	IC (Inheritance coupling)
7	CA (Afferent couplings)	17	CBM (Coupling between methods)
8	CE (Efferent couplings)	18	AMC (Average method complexity)
9	NPM (Number of public methods)	19	MAX(CC) (Maximum of McCabe's cyclomatic complexity (CC) methods)
10	LCOM3 (Number of cohesion in methods)	20	AVG(CC) (Mean of the CC value)

Table 3.3: List of features from 10 NASA projects

No.	Features	No.	Features
1	BRANCH_COUNT	21	HALSTEAD_LEVEL
2	CALL_PAIRS	22	HALSTEAD_PROG_TIME
3	LOC_CODE_AND_COMMENT	23	HALSTEAD_VOLUME
4	LOC_COMMENTS	24	MAINTENANCE_SEVERITY
5	CONDITION_COUNT	25	MODIFIED_CONDITION_COUNT
6	CYCLOMATIC_COMPLEXITY	26	MULTIPLE_CONDITION_COUNT
7	CYCLOMATIC_DENSITY	27	NODE_COUNT
8	DECISION_COUNT	28	NORMALIZED_CYLOMATIC_COMPLEXITY
9	DESIGN_COMPLEXITY	29	NUM_OPERANDS
10	DESIGN_DENSITY	30	NUM_OPERATORS
11	EDGE_COUNT	31	NUM_UNIQUE_OPERANDS
12	ESSENTIAL_COMPLEXITY	32	NUM_UNIQUE_OPERATORS
13	ESSENTIAL_DENSITY	33	NUMBER_OF_LINES
14	LOC_EXECUTABLE	34	PERCENT_COMMENTS
15	PARAMETER_COUNT	35	LOC_TOTAL
16	HALSTEAD_CONTENT	36	LOC_BLANK
17	HALSTEAD_DIFFICULTY	37	DECISION_DENSITY
19	HALSTEAD_ERROR_EST	38	GLOBAL_DATA_COMPLEXITY
20	HALSTEAD_LENGTH	39	GLOBAL_DATA_DENSITY

Table 3.4: Confusion Matrix

	Predicted: Positive (faulty)	Predicted: Negative (non-faulty)
Actual: Positive (faulty)	TP	FN
Actual: Negative (non-faulty)	FP	TN

$$G - mean = \sqrt{Recall * (1 - FOR)} \tag{3.6}$$

A high value of F-measure along with a high AUC score indicates that the comprehensive performance of the predictive model is good and a high G-mean is also a good indication of the model's comprehensive performance.

3.2.3 Classifier Selection

In our experimental work, we have considered six classifiers to assess the performance of models. Among them, five are conventional machine learning models, namely KNN [77], LR [74, 75], NB [78, 79], SVM [76] and DT [155], and the other model is DNN [82, 83]. In the existing literature, these classifiers have been successfully applied in similar contexts and are recognized for their effectiveness in various scenarios, ensuring their relevance to our study. The DNN was chosen because, while conventional classifiers are often easier to interpret, the DNN offers scalability and the ability to capture complex relationships within the data. These classifiers are implemented using the scikit-learn package in Python. The fully connected DNN model is implemented in Python using Keras and uses TensorFlow as the back end.

3.2.4 Experimental Design

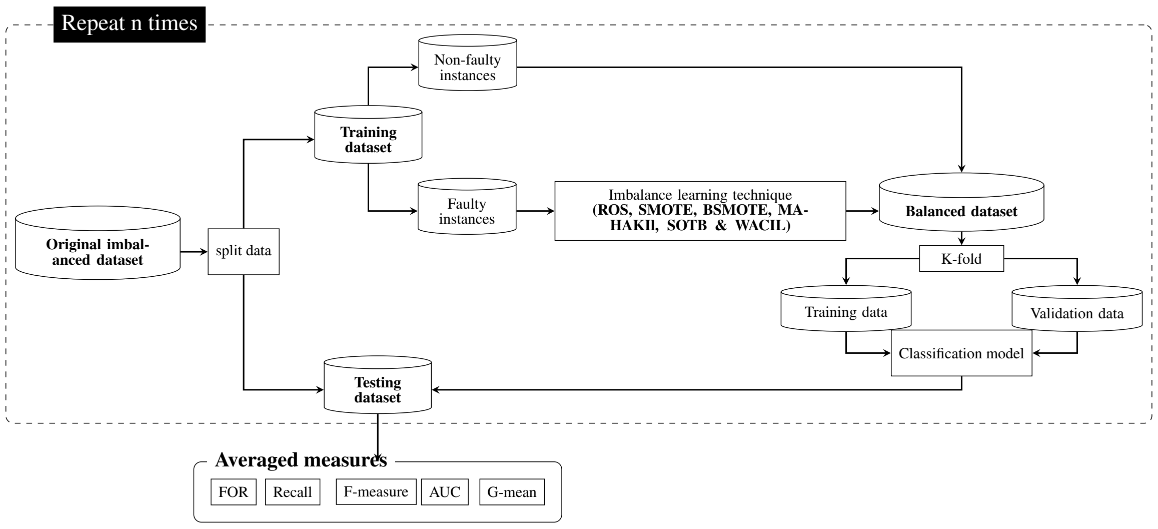


Figure 3.1: Experimental framework of WPFP model

To acquire extensive knowledge and also to notice the effectiveness of the proposed model, we conducted experiments on 24 diverse imbalanced datasets selected from PROMISE and NASA projects. To address the imbalance in the selected datasets, the proposed learning model generates pseudo-instances in the faulty class to achieve a balanced distribution with the non-faulty class. Subsequently, the observed outcomes were compared against various sampling techniques, including ROS, SMOTE, BSMOTE, MAHAKIL, and SOTB, as well as with the not oversampled dataset. ROS, SMOTE and BSMOTE are implemented using the imblearn library available in Python, while MAHAKIL and SOTB are implemented according to the algorithms outlined in [14] and [18]. The experimental framework is depicted in the Figure 3.1. According to empirical research [14, 18, 156], assigning 20–30% of the dataset to testing and the remaining 70–80% to training yields the greatest outcomes. We have experimented with both splits, 70/30% (training set/testing set) and 80/20% (training set/testing set) to assess the model's performance. The experimental results indicate that a 70% training set and a 30% testing set deliver the best performance. This is likely because the 70-30% split offers a balanced distribution of data for training and testing, ensuring that the testing set is sufficiently large to provide a statistically significant evaluation of the model's performance. In contrast, the 80-20% split leaves a smaller portion for testing, which may limit accurate assessment of the model's effectiveness. To evaluate the data accurately, we split the data into 70% training and 30% testing sets. The oversampling techniques were performed on the training set so that the training process happens equally well in both classes. After that, on the training set, we perform K-Fold (10-Fold) cross-validation to conquer issues such as selection bias or overfitting, as well as to provide insight into how the model can extrapolate to an unknown dataset. To reduce the impact of randomness, the average performance measures of 10 iterations are noted as the final measures.

3.2.5 Statistical Measures

In contemplation of measuring the statistical performance differences, we choose the WSR test [20, 157]. For example, the Dif is the difference between the two model's outcomes. Let W_+ be the rank sum of our model, W_- is the rank sum of the base model, and now 'n' is the count of non-zero differences. W_+ and W_- are represented below in Eqs. 3.7 and 3.8,

$$W_+ = \sum_{i=0}^{n} \text{Rank}(\text{Abs}(Dif_i)), \quad \text{where } \; Dif > 0 \tag{3.7}$$

$$W_{-} = \sum_{i=0}^{n} \text{Rank}(\text{Abs}(Dif_i)), \quad \text{where } \; Dif < 0 \tag{3.8}$$

The assumption of the null hypothesis (H_0) is the results obtained from the two approaches are statistically relative. At a confidence level of 95%, the WSR test assumes H_0 is true. If the P-value is greater than the significance level α (α = 0.05), then we fail to reject H_0, such as observed statistical evidence is inadequate to reject it, else there is a considerable difference in rejecting the default hypothesis H_0. However, the P-value doesn't indicate the strength of the relationship. The matched-pairs rank biserial correlation coefficient (W_c) can be used to measure effect-size for the WSR test [158, 159]. Eq. 3.9 represents the W_c:

$$\text{W}_c = \frac{4\left|T - \left(\frac{W_{+}+W_{-}}{2}\right)\right|}{n(n+1)} \tag{3.9}$$

Where 'T' is the minimum (W_{+}, W_{-}) and 'n' is the count of non-zero difference samples. The effect-size can be rendered using three labels [20], small ($\text{W}_c \leq 0.1$), medium ($0.1 < \text{W}_c < 0.5$) and large ($\text{W}_c \geq 0.5$). Furthermore, Win-Draw-Loss (WDL) statistics [160] are adopted to compare the overall performance of each predictive model with all other predictive models (*classification models * imbalance learning approaches - 1*) for all performance assessment measures. If the P-value is greater than the α then both models' win counter increases by one, else win counter increases for the greater rank sum model, and the loss counter increases for the lesser rank sum model.

3.3 Experimental Results

To demonstrate the effectiveness of our proposed method, we conducted experiments and addressed the following three research questions:

RQ1: Does WACIL contribute to the diverse nature of generated pseudo-instances of faulty class data?

RQ2: How effectively does WACIL tackle the class imbalance problem?

RQ3: How does WACIL's overall performance compare to state-of-the-art strategies?

In order to answer the research questions RQ1, RQ2, and RQ3, we computed WACIL's performance and compared it with ROS [15, 32], SMOTE [11], BSMOTE [12], MAHAKIL [14], and SOTB [18], and also with the Not Oversampled (NOS) original dataset.

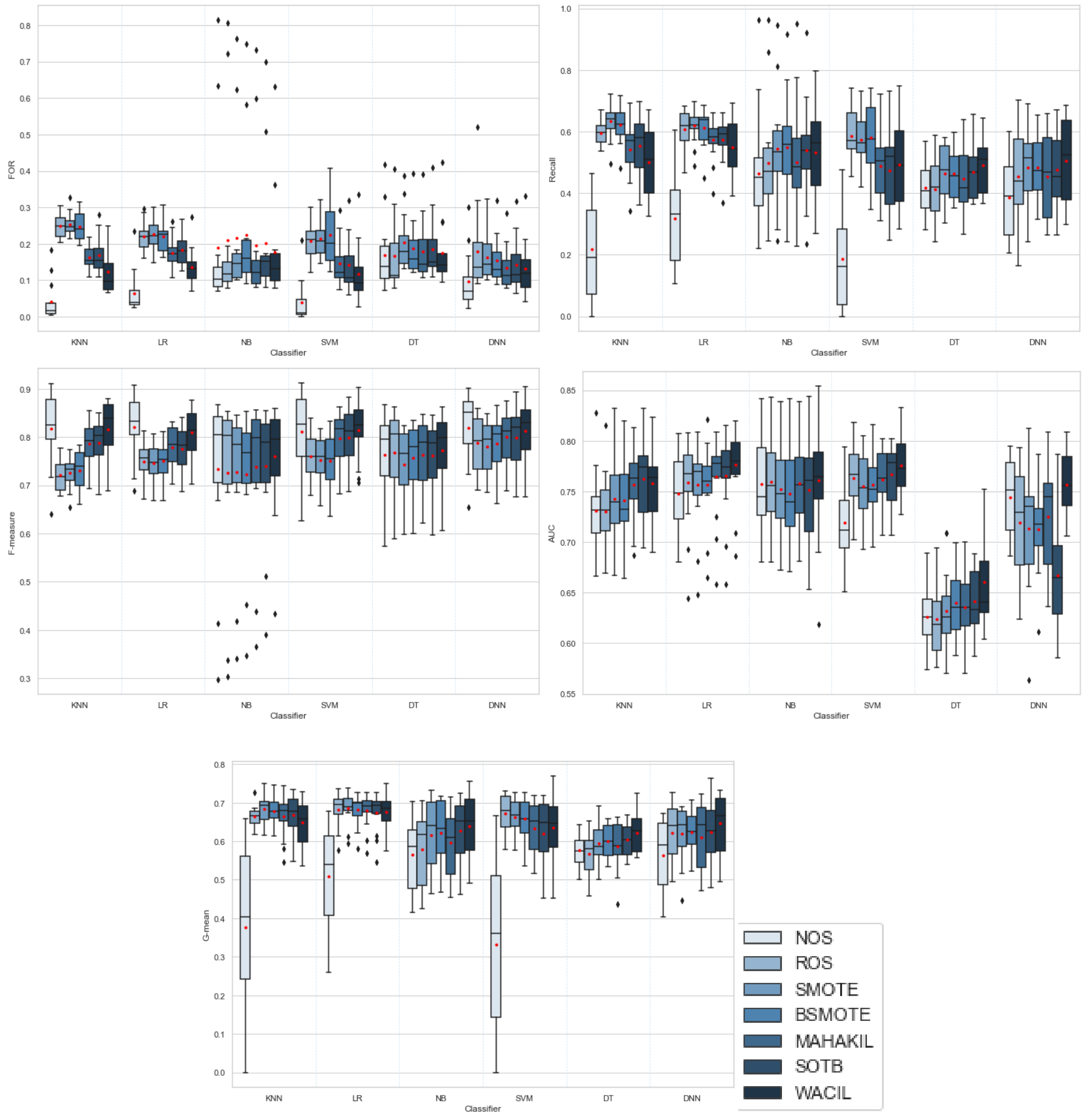


Figure 3.2: Boxplots of six imbalance learning techniques and NOS on six classifiers over 14 PROMISE project datasets in terms of FOR, Recall, F-measure, AUC and G-mean

3.3.1 Overall Results

Figures 3.2 and 3.3 represent the results of individual classifiers in box plots for PROMISE and NASA projects, respectively. Furthermore, Figure 3.4 shows the distribution of corre-

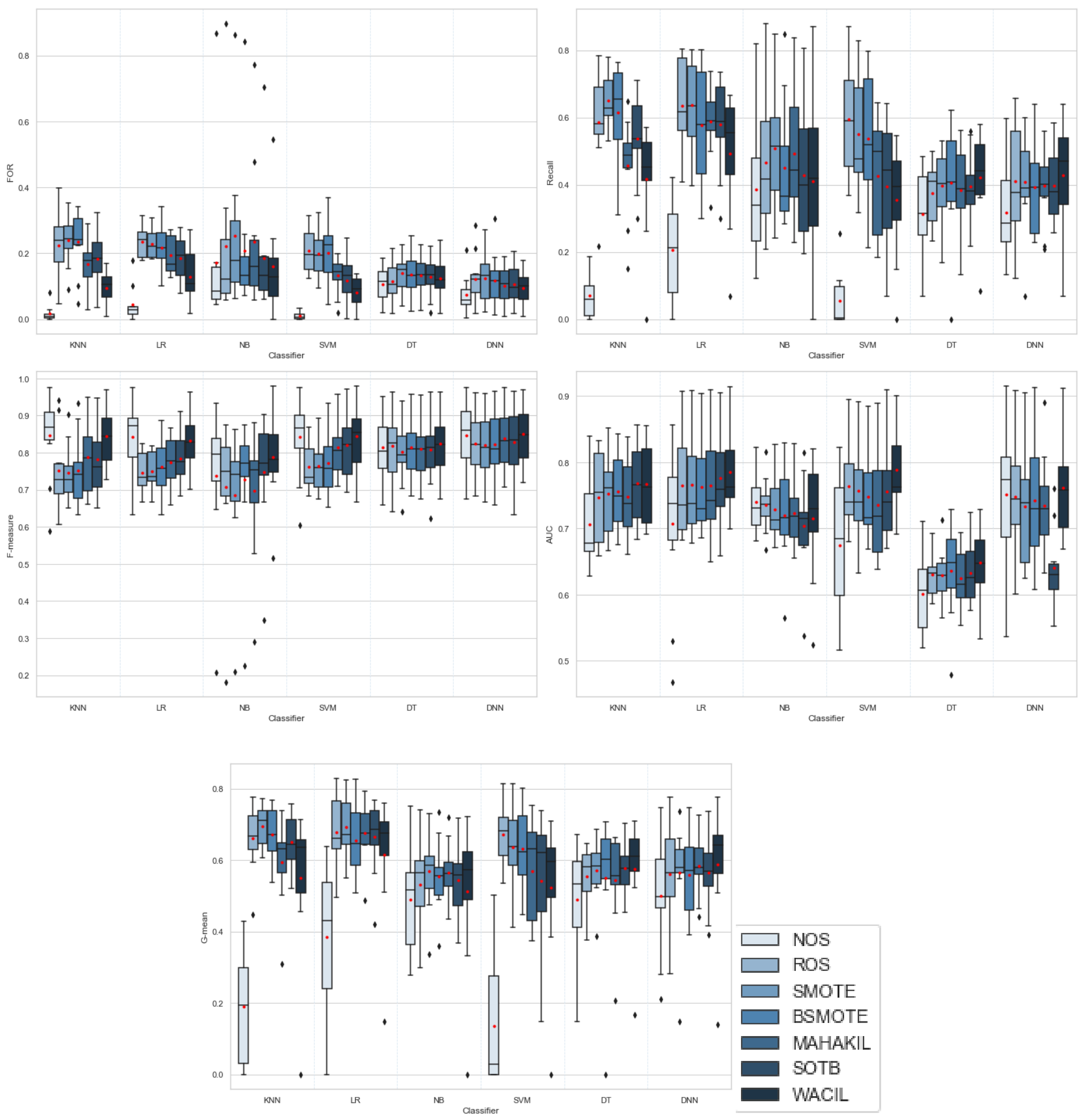


Figure 3.3: Boxplots of six imbalance learning techniques and NOS on six classifiers over 10 NASA project datasets in terms of FOR, Recall, F-measure, AUC and G-mean

sponding average (average of all classifier results) results across 24 datasets. The little red circle in each box represents the mean value of each method. According to the results, we can conclude the following:

1. Across all datasets and classification models, NOS and WACIL outperform all the other methods through the best FOR values with a mean of 0.088 and 0.131, respectively. While ROS (0.198), SMOTE (0.206), and BSMOTE (0.199) offer the worse

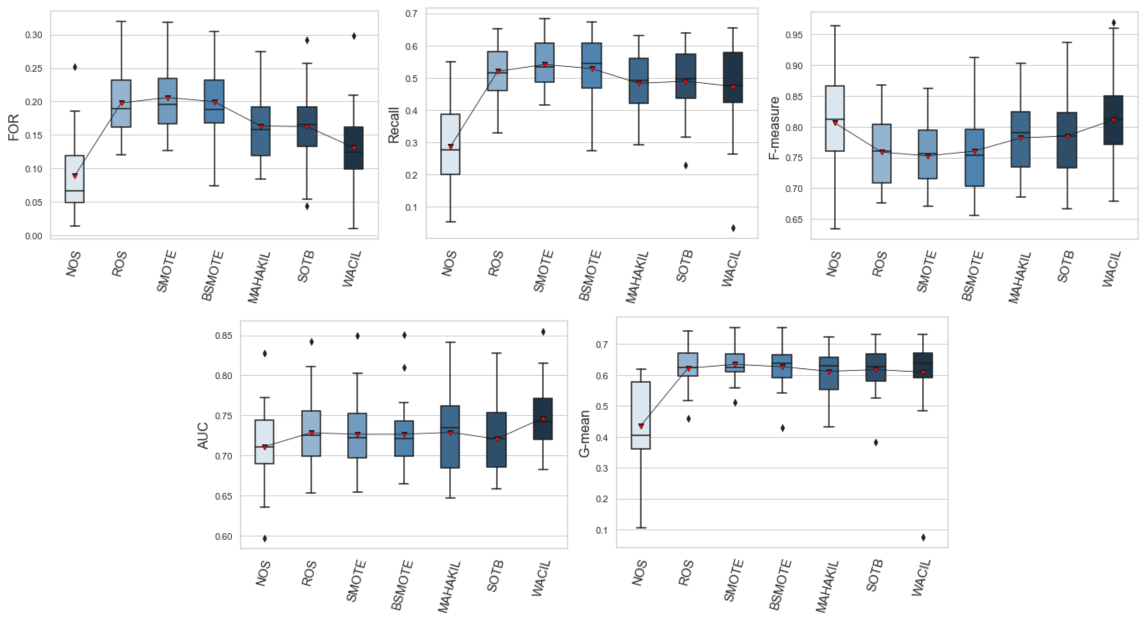


Figure 3.4: Boxplots of six imbalance learning techniques and NOS on averaged results over 24 PROMISE and NASA datasets in terms of FOR, Recall, F-measure, AUC and G-mean

FOR values. MAHAKIL (0.163) and SOTB (0.162) produce moderately better results than those methods.

2. In terms of Recall, WACIL with DNN and DT produce better Recall values than other sampling approaches. Although the average Recall values of WACIL (0.48) are not always the best, our approach can archive a similar distribution of Recall results to MAHAKIL (0.483) and SOTB (0.489). While NOS (0.286) produces the lowest Recall values, BSMOTE (0.528) and SMOTE (0.541) produce the best.

3. For almost all datasets and all the classification models, NOS and WACIL outperform all the competitive approaches over F-measure with highest means of 0.806 and 0.811, respectively. While ROS (0.758), SMOTE (0.752) and BSMOTE (0.760) produce the worse outcomes. MAHAKIL (0.781) and SOTB (0.785) produce moderately better values.

4. For the vast majority of datasets, the WACIL technique outperforms the other compared approaches in terms of AUC with an average of 0.747. This implies that the model's ability to discriminate between faulty and non-faulty classes is better, while

other models mean results are 0.711, 0.728, 0,726, 0.726, 0.729, 0.721 for NOS, ROS, SMOTE, BSMOTE, MAHAKIL and SOTB, respectively.

5. WACIL outperforms all the competitive approaches over PROMISE datasets with respect to G-mean outcome. However, over NASA datasets, WACIL with DNN and DT gives comparable results to the competitive methods. On KNN, LR, and NB, SMOTE performs better, and on SVM, ROS produces moderately better G-mean values. On the average results of both project datasets WACIL (0.609) gives comparable results to other models. While other models mean results are 0.436, 0.622, 0.634, 0.627, 0.611, 0.617 for NOS, ROS, SMOTE, BSMOTE, MAHAKIL and SOTB, respectively.

As per Benin [14], the diverse nature of generated pseudo-instances can be defined by FOR values. The lower the FOR, the greater the diversity. We found that the WACIL approach achieves a lower FOR than the other oversampling techniques. As a result, we can answer the **RQ1**, yes, WACIL contributes towards diversity.

The predictive model built on the SMOTE and BSMOTE generated training datasets resulted in higher mean Recall and higher mean FOR. Furthermore, MAHAKIL and SOTB are better alternatives to ROS, SMOTE, and BSMOTE to generate diverse instances. Over DT and DNN, WACIL results have higher mean Recall over all the datasets than those of other sampling approaches. The other classification models with WACIL get comparable results over PROMISE datasets and fewer steps away from the competitive approaches over NASA datasets. The fundamental reason is that various data have various distributions, and it's possible that the data distribution is only acceptable for certain classifiers. As a result, we can respond to **RQ2**. WACIL tackles imbalanced data by introducing pseudo-instance with lower FOR values and with comparable Recall values.

In accordance with FOR and Recall, one can't justify a particular approach being superior to others. Therefore, to assess the comprehensive performance, we consider the F-measure, AUC, and G-mean. The greater these values, the better the overall performance. Therefore, we compared these measures between different imbalance learning techniques to answer **RQ3**. Figures 3.2, 3.3 and 3.4 demonstrate that WACIL outperforms all other approaches in terms of AUC and in terms of F-measure, outperforms all other approaches except NOS. In terms of G-mean, WACIL gives a decent output. Hence, we can conclude that the comprehensive performance of WACIL is superior to other methods.

3.3.2 Statistical Performance Comparison

To further examine the outcomes as well as to determine if there is any statistically significant difference between the proposed approach and the other methods, a WSR test is applied to the WACIL outcomes versus each other method.

To answer research questions, we statistically analyze each performance measure of all competitive approaches over all the classification models. Tables 3.5, 3.6 and 3.7 summarize the WSR test results of WACIL compared to competitive approaches. The tabular results compare methods in terms of P-value, effect size, and rank sums. For each performance measure, it shows the P-value (<0.05 or >0.05), the effect size (W_c), and the positive rank sum (W_+)/negative rank sum (W_-). The positive rank sum is of WACIL and the negative rank sum is of competitive approaches. In the Tables, the bold part P-value **<0.05** represents WACIL is superior to the competitive approach, and the Italian font bold part P-value ***>0.05*** represents WACIL is statistically identical to the competitive approach.

In order to answer **RQ1**, we compared the FOR results of our proposed approach with the competitive approaches. In Tables 3.5, 3.6 and 3.7, the first row of all the tables shows the comparison of FOR values. We can notice the following points:

1. WACIL with KNN, LR, and SVM statistically outperforms all the compared sampling approaches with high effect sizes except NOS.

2. WACIL with NB statistically outperforms ROS, SMOTE, and BSMOTE, and for NOS, MAHAKIL, and SOTB, the rank sum is less than our approach, but the P-value is > 0.05, so it's statistically proven our approach has an identical distribution of results with those models on the NB classifier.

3. WACIL with DT statistically outperforms SMOTE and BSMOTE. For NOS, ROS, MAHAKIL, and SOTB, the P-value is > 0.05, so it is statistically proven that our approach has an identical distribution of results with them on the DT classifier.

Table 3.5: Statistical comparison of WACIL with other competitive approaches using KNN, LR, and NB

				K-Nearest Neighbor			
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	1.82E-05(< 0.05)	2.07E-05 (**<0.05**)	1.82E-05 (**<0.05**)	1.82E-05 (**<0.05**)	0.000255 (**<0.05**)	1.82E-05 (**<0.05**)
	W_c	1 (0/300)	0.993 (299/1)	1 (300/0)	1 (300/0)	0.857 (278.5/21.5)	1 (300/0)
Recall	P-value	2.70E-05 (**<0.05**)	7.14E-05 (<0.05)	1.82E-05 (<0.05)	2.07E-05 (<0.05)	0.01263 (<0.05)	2.35E-05 (<0.05)
	W_c	1 (276/0)	0.927 (11/289)	1 (0/300/)	0.993 (1/299)	0.591 (56.5/219.5)	0.987 (2/298)
F-measure	P-value	0.4489411 (***>0.05***)	1.82E-05 (**<0.05**)	1.82E-05 (**<0.05**)	1.82E-05 (**<0.05**)	0.000162 (**<0.05**)	3.98E-05 (**<0.05**)
	W_c	0.173 (17/124)	1 (300/0)	1 (300/0)	1 (299/1)	0.899 (262/14)	1 (253/0)
AUC	P-value	3.64E-05 (**<0.05**)	4.02E-05 (**<0.05**)	0.004673 (**<0.05**)	0.001549 (**<0.05**)	0.051986 *(**>0.05**)*	0.338372 *(**>0.05**)*
	W_c	0.963 (294.5/5.5)	0.978 (273/0)	0.678 (231.5/44.5)	0.771 (224/29)	0.453 (218/82)	0.223 (116.5/183.5)
G-mean	P-value	3.09E-05 (**<0.05**)	2.88E-02 (<0.05)	0.0018439 (<0.05)	0.001755 (<0.05)	0.061933 *(**>0.05**)*	0.001673 (<0.05)
	W_c	0.993 (275/1)	0.513 (73/227)	0.723 (41.5/258.5)	0.73 (40.5/259.5)	0.455 (69/184)	0.733 (40/260)
				Logistic Regression			
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	2.07056E-05 (<0.05)	2.66915E-05 (**<0.05**)	3.8838E-05 (**<0.05**)	2.07056E-05 (**<0.05**)	0.00013647 (**<0.05**)	9.05972E-05 (**<0.05**)
	W_c	0.993 (1/299)	0.98 (297/3)	0.96 (294/6)	0.993 (299/1)	0.89 (283.5/16.5)	0.917 (287.5/12.5)
Recall	P-value	1.82153E-05 (**<0.05**)	0.001636198 (<0.05)	0.000828696 (<0.05)	0.001016031 (<0.05)	0.02986533 (<0.05)	0.015576468 (<0.05)
	W_c	1 (300/0)	0.767 (29.5/223.5)	0.777 (33.5/266.5)	0.767 (35/265)	0.507 (74/226)	0.589 (52/201)
F-measure	P-value	0.259005754 ((***>0.05***))	1.81974E-05 (**<0.05**)	1.82153E-05 (**<0.05**)	1.82153E-05 (**<0.05**)	4.38232E-05 (**<0.05**)	3.02696E-05 (**<0.05**)
	W_c	0.267 (110/190)	1 (300/0)	1 (300/0)	1 (300/0)	0.953 (293/7)	0.977 (296.5/3.5)
AUC	P-value	0.000787036 (**<0.05**)	0.003100486 (**<0.05**)	0.002670949 (**<0.05**)	0.001242669 (**<0.05**)	0.008504642 (**<0.05**)	0.072686595 ((***>0.05***))
	W_c	0.787 (268/32)	0.69 (253.5/46.5)	0.731 (219/34)	0.753 (263/37)	0.627 (224.5/51.5)	0.417 (195.5/80.5)
G-mean	P-value	1.82153E-05 (**<0.05**)	0.201451172 ((***>0.05***))	0.170197795 ((***>0.05***))	0.265059089 ((***>0.05***))	0.391341771 ((***>0.05***))	0.423687147 ((***>0.05***))
	W_c	1 (300/0)	0.308 (95.5/180.5)	0.317 (102.5/197.5)	0.263 (110.5/189.5)	0.203 (119.5/180.5)	0.19 (121.5/178.5)
				Naive Bayes			
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	0.3155264 ((***>0.05***))	0.897692 ((***>0.05***))	0.0177021 (**<0.05**)	0.02062 (**<0.05**)	0.737954 (((***>0.05***))	0.077703 (((***>0.05***))
	W_c	0.239 (105/171)	0.03 (145.5/154.5)	0.553 (233/67)	0.54 (231/69)	0.08 (149/127)	0.42 (196/80)
Recall	P-value	0.0396624 (**<0.05**)	0.241427 ((***>0.05***))	0.4662411 ((***>0.05***))	0.475051 ((***>0.05***))	0.345754 ((***>0.05***))	0.753304 ((***>0.05***))
	W_c	0.48 (222/78)	0.273 (191/109)	0.17 (124.5/175.5)	0.167 (125/175)	0.22 (183/117)	0.07 (139.5/160.5)
F-measure	P-value	0.1102107 ((***>0.05***))	0.103404 ((***>0.05***))	0.0018226 (**<0.05**)	0.003903 (**<0.05**)	0.124506 ((***>0.05***))	0.003404 (**<0.05**)
	W_c	0.38 (190.5/85.5)	0.38 (207/93)	0.739 (240/36)	0.667 (250/50)	0.37 (189/87)	0.687 (253/47)
AUC	P-value	0.4661956 ((***>0.05***))	0.587209 ((***>0.05***))	0.2415541 ((***>0.05***))	0.280234 ((***>0.05***))	0.235469 ((***>0.05***))	0.042491 (**<0.05**)
	W_c	0.17 (175.5/124.5)	0.123 (168.5/131.5)	0.279 (176.5/99.5)	0.257 (173.5/102.5)	0.286 (177/98.5)	0.473 (221/79)
G-mean	P-value	0.0078806 (**<0.05**)	0.014566 (**<0.05**)	0.3896018 ((***>0.05***))	0.345754 ((***>0.05***))	0.041046 (**<0.05**)	0.715111 ((***>0.05***))
	W_c	0.617 (242.5/57.5)	0.57 (235.5/64.5)	0.209 (153/100)	0.22 (183/117)	0.477 (221.5/78.5)	0.087 (150/126)

4. WACIL with DNN statistically outperforms ROS, SMOTE, BSMOTE, and SOTB except for NOS and MAHAKIL (> 0.05), which has an identical distribution as our proposed approach.

5. Table 3.7 demonstrates that on average, our method outperforms all the other sampling approaches. The original data gives lower FOR results than any of the sampling techniques.

In reference to the aforementioned observations, we can conclude that, in terms of FOR, WACIL is superior to competitive approaches except NOS. As per Benin. [14], it's statistically proven that the FOR results are good enough to say WACIL contributes towards the diversity of generated pseudo-instances.

To answer **RQ2**, we compared the Recall results of our proposed approach with the competitive approaches, and the results are as follows:

1. The statistical performance of WACIL with KNN and LR is poor in terms of Recall

Table 3.6: Statistical comparison of WACIL with other competitive approaches using SVM, DT and DNN

Support Vector Machine							
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	2.70E-05 (<0.05)	2.67E-05 (**<0.05**)	2.07E-05 (**<0.05**)	2.07E-05 (**<0.05**)	0.000162 (**<0.05**)	9.07E-05 (**<0.05**)
	W_c	1 (0/276)	0.98 (297/3)	0.993 (299/1)	0.993 (299/1)	0.88 (282/18)	0.913 (287/13)
Recall	P-value	2.70E-05 (**<0.05**)	5.61E-05 (<0.05)	0.0001288 (<0.05)	0.000285 (<0.05)	0.24719 ((***>0.05***))	0.692535 ((***>0.05***))
	W_c	1 (276/0)	0.94 (9/291)	0.893 (16/284)	0.847 (23/277)	0.27 (109.5/190.5)	0.094 (125/151)
F-measure	P-value	0.7494278 ((***>0.05***))	9.61E-05 (**<0.05**)	3.52E-05 (**<0.05**)	2.67E-05 (**<0.05**)	0.002828 (**<0.05**)	0.002038 (**<0.05**)
	W_c	0.072 (148/128)	0.91 (286.5/13.5)	0.986 (274/2)	0.98 (297/3)	0.703 255.5/44.5	0.747 (221/32)
AUC	P-value	1.82E-05 (**<0.05**)	0.008574 (**<0.05**)	0.0031035 (**<0.05**)	0.000914 (**<0.05**)	0.001591 (**<0.05**)	0.007039 (**<0.05**)
	W_c	1 (300/0)	0.613 (242/58)	0.693 (254/46)	0.786 (246.5/29.5)	0.737 (260.5/39.5)	0.66 (210/43)
G-mean	P-value	2.70E-05 (**<0.05**)	0.000262 (<0.05)	0.0042747 (<0.05)	0.005581 (<0.05)	0.539006 ((***>0.05***))	0.951492 ((***>0.05***))
	W_c	1 (276/0)	0.87 (18/258)	0.667 (50/250)	0.647 (53/247)	0.147 (128/172)	0.004 (137.5/138.5)
Decision Tree							
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	0.0151325 (<0.05)	0.116083 ((***>0.05***))	0.0005456 (<0.05)	0.006191 (<0.05)	0.119399 ((***>0.05***))	0.068017 ((***>0.05***))
	W_c	0.567 (65/235)	0.363 (95.5/204.5)	0.803 (270.5/29.5)	0.656 (228.5/47.5)	0.363 (204.5/95.5)	0.431 (197.5/78.5)
Recall	P-value	2.67E-05 (**<0.05**)	0.000747 (**<0.05**)	0.0537719 ((***>0.05***))	0.016384 (**<0.05**)	0.010128 (**<0.05**)	0.071861 ((***>0.05***))
	W_c	0.98 (297/3)	0.787 (268/32)	0.45 (217.5/82.5)	0.56 (234/66)	0.6 (240/60)	0.42 (213/87)
F-measure	P-value	0.0225179 (**<0.05**)	0.136001 ((***>0.05***))	7.12E-05 (**<0.05**)	0.006784 (**<0.05**)	0.002865 (**<0.05**)	0.008135 (**<0.05**)
	W_c	0.551 (214/62)	0.355 (187/89)	0.93 (289.5/10.5)	0.641 (226.5/49.5)	0.714 (236.5/39.5)	0.634 (225.5/50.5)
AUC	P-value	3.09E-05 (**<0.05**)	0.000748 (**<0.05**)	0.0018439 (**<0.05**)	0.004668 (**<0.05**)	0.001017 (**<0.05**)	0.016265 (**<0.05**)
	W_c	0.993 (275/1)	0.783 (267.5/32.5)	0.727 (259/41)	0.674 (231/45)	0.767 (265/35)	0.576 (217.5/58.5)
G-mean	P-value	1.82E-05 (**<0.05**)	0.000919 (**<0.05**)	0.032107 (**<0.05**)	0.005579 (**<0.05**)	0.005642 (**<0.05**)	0.072654 ((***>0.05***))
	W_c	1 (300/0)	0.773 (266/34)	0.5 (225/75)	0.643 (246.5/53.5)	0.659 (229/47)	0.435 (198/78)
Deep Neural Network							
Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	0.0011827 (**<0.05**)	0.001242 (**<0.05**)	0.0006383 (**<0.05**)	0.005579 (**<0.05**)	0.587171 ((***>0.05***))	0.04297 (**<0.05**)
	W_c	0.757 (263.5/36.5)	0.77 (265.5/34.5)	0.797 (269.5/30.5)	0.653 (248/52)	0.13 (169.5/130.5)	0.482 (204.5/71.5)
Recall	P-value	3.43E-05 (**<0.05**)	0.161492 ((***>0.05***))	0.2123927 ((***>0.05***))	0.05742 ((***>0.05***))	0.003252 **<0.05**	0.030814 **<0.05**
	W_c	0.967 (295/5)	0.327 (199/101)	0.297 (179/97)	0.443 (216.5/83.5)	0.683 (252.5/47.5)	0.518 (209.5/66.5)
F-measure	P-value	0.3141425 ((***>0.05***))	0.000127 (**<0.05**)	6.32E-05 (**<0.05**)	2.66E-05 (**<0.05**)	0.010979 (**<0.05**)	0.000122 (**<0.05**)
	W_c	0.245 (95.5/157.5)	0.913 (264/12)	0.933 (290/10)	0.98 (297/3)	0.6 (240/60)	0.957 (226/5)
AUC	P-value	0.1102725 ((***>0.05***))	0.002455 (**<0.05**)	3.88E-05 (**<0.05**)	0.000828 (**<0.05**)	0.000965 (**<0.05**)	1.82E-05 (**<0.05**)
	W_c	0.377 (190/86)	0.703 (255.5/44.5)	0.963 (294.5/5.5)	0.783 (267.5/32.5)	0.786 (246.5/29.5)	1 (300/0)
G-mean	P-value	5.61E-05 (**<0.05**)	0.013755 (**<0.05**)	2.64E-02 (**<0.05**)	0.007871 (**<0.05**)	0.004273 (**<0.05**)	1.06E-02 (**<0.05**)
	W_c	0.94 (219/9)	0.583 (218.5/57.5)	0.525 (210.5/65.5)	0.617 (242.5/57.5)	0.667 (250/50)	0.609 (222/54)

Table 3.7: Statistical comparison of WACIL with other competitive approaches in terms of average FOR, Recall, F-measure, AUC and G-mean

Measure	measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	0.000162316 (<0.05)	3.02413E-05 (**<0.05**)	2.07056E-05 (**<0.05**)	2.07056E-05 (**<0.05**)	0.000374651 (**<0.05**)	2.6987E-05 (**<0.05**)
	W_c	0.88 (18/282)	0.97 (295.5/4.5)	0.993 (299/1)	0.993 (299/1)	0.827 (274/26)	0.977 (296.5/3.5)
Recall	P-value	2.07056E-05 (**<0.05**)	0.021434925 (<0.05)	0.000776477(<0.05)	0.00057523 (<0.05)	0.897634428 ((***>0.05***))	0.174603819 ((***>0.05***))
	W_c	0.993 (299/1)	0.537 (69.5/230.5)	0.801 (27.5/248.5)	0.803 (29.5/270.5)	0.03 (154.5/145.5)	0.317 (102.5/197.5)
F-measure	P-value	0.447004889 ((***>0.05***))	2.6987E-05 (**<0.05**)	1.82153E-05 (**<0.05**)	1.81795E-05 (**<0.05**)	6.32333E-05 (**<0.05**)	3.9959E-05 (**<0.05**)
	W_c	0.178 (162.5/113.5)	1 (276/0)	1 (300/0)	1 (300/0)	0.933 (290/10)	1 (253/0)
AUC	P-value	1.8019E-05 (**<0.05**)	0.000312397 (**<0.05**)	3.42058E-05 (**<0.05**)	2.6987E-05 (**<0.05**)	8.01794E-05 (**<0.05**)	1.81259E-05 (**<0.05**)
	W_c	1 (300/0)	0.859 (256.5/19.5)	0.967 (295/5)	1 (276/0)	0.92 (288/12)	1 (300/0)
G-mean	P-value	2.07056E-05 **<0.05**	0.310397062 ((***>0.05***))	0.317261121 ((***>0.05***))	0.303656438 ((***>0.05***))	0.010614157 (**<0.05**)	0.315330456 ((***>0.05***))
	W_c	0.993 (299/1)	0.237 (185.5/114.5)	0.23 (115.5/184.5)	0.24 (114/186)	0.601 (221/55)	0.239 (221/55)

score; it outperforms only NOS.

2. Over NB, our approach is statistically superior to NOS and statistically has an identical distribution to all other competitive approaches.

3. Over SVM, our approach statistically outperforms NOS and has identical distribu-

tion results as MAHAKIL and SOTB. ROS, SMOTE, and BSMOTE are better than WACIL.

4. Over DT, our approach statistically outperforms NOS, ROS, BSMOTE, and MAHAKIL and has an identical distribution of results to SMOTE and SOTB.

5. Over DNN, our approach statistically outperforms NOS, MAHAKIL, and SOTB and has identical distribution results as ROS, SMOTE, and BSMOTE.

6. Table 3.7 demonstrates that on average our method outperforms NOS, is statistically identical to MAHAKIL and SOTB, and is unable to outperform the rest of the competitive approaches.

In reference to the aforementioned observations of FOR and Recall, we can conclude that in terms of FOR, our approach is superior to all the compared sampling techniques except NOS. In terms of Recall, our approach is superior to NOS, MAHAKIL, and SOTB except ROS, SMOTE, and BSMOTE on average. So, it's statistically proven that WACIL produces lower FOR values with comparable Recall values on average.

In order to answer research question Q3, we compared the F-measure, AUC, and G-mean results of our proposed approach with the competitive approaches and the results are as follows:

1. In terms of F-measure, our approach outperforms all the competitive sampling approaches and is statistically identical to NOS over KNN, LR, SVM, and DNN. WACIL with NB outperforms SMOTE, BSMOTE, and SOTB and is statistically identical to NOS, ROS, and MAHAKIL. WACIL with DT outperforms NOS, SMOTE, BSMOTE, MAHAKIL, and SOTB and is statistically identical to ROS.

2. In terms of AUC, our approach outperforms all the competitive approaches over SVM and DT. WACIL with KNN outperforms NOS, ROS, SMOTE, and BSMOTE and is statistically identical to MAHAKIL and SOTB. WACIL outperforms SOTB and is statistically identical to the rest of the methods over NB. WACIL is statistically identical to SOTB and outperforms the rest of the methods over LR. WACIL is statistically identical to NOS and outperforms the rest of the methods over DNN.

3. In terms of G-mean, WACIL outperforms all the competitive approaches over DT and DNN. WACIL is superior to NOS, and statistically identical to MAHAKIL over KNN. WACIL outperforms NOS and is statistically identical to the rest of the competitive approaches over LR. WACIL outperforms NOS, ROS, MAHAKIL, and SOTB and is statistically identical to SMOTE and BSMOTE over NB. WACIL outperforms NOS, statistically identical to MAHAKIL and SOTB over SVM.

4. Table 3.7 demonstrates that on average WACIL outperforms all the other competitive approaches in terms of AUC. In terms of F-measure, WACIL outperforms all the sampling approaches and is statistically identical to NOS. In terms of G-mean, WACIL outperforms NOS and MAHAKIL, statistically identical to the rest of the methods.

We can conclude that in terms of F-measure and AUC, WACIL is superior to almost all competitive approaches on average, and in terms of G-mean, most of the time, WACIL is superior to all methods over DT and DNN classifiers. For the rest of the classifiers, WACIL outperforms NOS, MAHAKIL, and SOTB are identical to ROS, SMOTE, and BSMOTE on average. Hence, it's statistically proven that the comprehensive performance of WACIL is superior to the competitive approaches. In fault detection, missing a faulty module (a false negative) can lead to severe consequences, including system failures or safety issues. Conversely, flagging a non-faulty module as faulty (a false positive) may lead to unnecessary maintenance or investigation. The importance of faulty and non-faulty modules depends upon the applications. Few applications like medical (cancer) diagnostics, autonomous vehicle object detection systems, and fault detection in critical systems (airplane engine monitoring) give more priority to high recall. Conversely, few applications like network security (intrusion detection systems), email spam filters and credit card fraud detection give more priority to low FOR. Our methodology prioritizes maximize balancing between FOR and recall, so the AUC scores of our model are getting better.

The Table 3.8 reports the comparison results of WDL comparisons over each predictive model (*classification model + imbalance learning techniques*). We have computed the total number of wins, draws, and losses for each model over all five performance measures. For

Table 3.8: Statistical comparison of all the predictive models over all the performance measures. W-Wins, D-Draws, and L-Losses

Model	FOR					Recall					F-measure					AUC					G-mean					Total				
	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank
KNN+WACIL	29	5	7	24	6	8	14	19	-6	23	34	0	7	34	2.5	18	2	21	16	10	11	11	19	0	16	100	32	73	68	4
KNN+SOTB	9	17	15	-8	27.5	23	7	11	16	12	9	12	20	-3	19.5	21	2	18	19	7	29	2	10	27	10	91	40	74	51	7
KNN+MAHAKIL	10	14	17	-4	23	21	12	8	9	14	9	12	20	-3	19.5	14	7	20	7	20	25	10	6	15	14	79	55	71	24	13
KNN+BSMOTE	0	33	8	-33	41	36	0	5	36	2	0	30	11	-30	40	10	10	21	0	25.5	30	0	11	30	5.5	76	73	56	3	19
KNN+SMOTE	0	34	7	-34	42	37	0	4	37	1	0	32	9	-32	42	12	11	18	1	23.5	33	0	8	33	2.5	82	77	46	5	18
KNN+ROS	0	30	11	-30	39.5	30	2	9	28	6	0	31	10	-31	41	9	17	15	-8	29	27	3	11	24	11	66	83	56	-17	28
KNN+NOS	40	0	1	40	1.5	1	40	0	-39	41	33	0	8	33	5.5	8	23	10	-15	33	1	40	0	-39	41	83	103	19	-20	29
LR+WACIL	18	7	16	11	12	22	9	10	13	13	30	0	11	30	7.5	39	0	2	39	1	29	0	12	29	7	138	16	51	122	1
LR+SOTB	9	22	10	-13	32.5	29	5	7	24	8	9	15	17	-6	23	29	0	12	29	3	29	1	11	28	8.5	105	43	57	62	5
LR+MAHAKIL	9	20	12	-11	30.5	28	5	8	23	9	9	15	17	-6	23	22	1	18	21	4.5	29	1	11	28	8.5	97	42	66	55	6
LR+BSMOTE	2	29	10	-27	37	32	0	9	32	5	3	26	12	-23	36	18	2	21	16	10	31	0	10	31	4	86	57	62	29	10
LR+SMOTE	1	30	10	-29	38	34	0	7	34	4	0	26	15	-26	38	20	3	18	17	8	34	0	7	34	1	89	59	57	30	9
LR+ROS	0	30	11	-30	39.5	35	0	6	35	3	0	27	14	-27	39	18	2	21	16	10	33	0	8	33	2.5	86	59	60	27	11.5
LR+NOS	39	2	0	37	3	2	39	0	-37	40	34	0	7	34	2.5	8	3	30	5	21	2	39	0	-37	40	85	83	37	2	20.5
NB+WACIL	11	5	25	6	18	7	9	25	-2	18.5	11	6	24	5	15	10	2	29	8	18.5	7	14	20	-7	23.5	46	36	123	10	17
NB+SOTB	9	5	27	4	20	7	9	25	-2	18.5	3	11	27	-8	27.5	9	12	20	-3	27	7	14	20	-7	23.5	35	51	119	-16	26.5
NB+MAHAKIL	1	6	34	-5	25	6	8	27	-2	18.5	1	9	31	-8	27.5	10	6	25	4	22	5	17	19	-12	32	23	46	136	-23	30
NB+BSMOTE	0	9	32	-9	29	7	7	27	0	16	0	16	25	-16	32	9	9	23	0	25.5	5	12	24	-7	23.5	21	53	131	-32	34
NB+SMOTE	0	11	30	-11	30.5	11	4	26	7	15	0	15	26	-15	31	10	9	22	1	23.5	6	12	23	-6	18.5	27	51	127	-24	31
NB+ROS	1	5	35	-4	23	5	9	27	-4	21	0	9	32	-9	29.5	12	3	26	9	17	4	21	16	-17	34	22	47	136	-25	32
NB+NOS	15	3	23	12	10.5	3	19	19	-16	36	0	9	32	-9	29.5	11	3	27	8	18.5	3	31	7	-28	39	32	65	108	-33	35
SVM+WACIL	35	4	2	31	5	4	17	20	-13	34.5	33	0	8	33	5.5	37	0	4	37	2	7	16	18	-9	28.5	116	37	52	79	2
SVM+SOTB	18	6	17	12	10.5	5	15	21	-10	31.5	20	6	15	14	11	23	2	16	21	4.5	5	15	21	-10	30.5	71	44	90	27	11.5
SVM+MAHAKIL	16	8	17	8	15.5	8	14	19	-6	23	19	8	14	11	12	14	4	23	10	16	8	14	19	-6	18.5	65	48	92	17	15.5
SVM+BSMOTE	2	28	11	-26	35.5	23	3	15	20	10	3	26	12	-23	36	14	3	24	11	15	24	5	12	19	13	66	65	74	1	22
SVM+SMOTE	3	29	9	-26	35.5	23	6	12	17	11	3	26	12	-23	36	16	2	23	14	12.5	25	5	11	20	12	70	68	67	2	20.5
SVM+ROS	4	28	9	-24	34	28	1	12	27	7	4	22	15	-18	33.5	22	2	17	20	6	30	0	11	30	5.5	88	53	64	35	8
SVM+NOS	40	0	1	40	1.5	0	41	0	-41	42	30	0	11	30	7.5	8	27	6	-19	34	0	41	0	-41	42	78	109	18	-31	33
DT+WACIL	13	13	15	0	21	8	14	19	-6	23	14	10	17	4	16.5	6	34	1	-28	35	10	14	17	-4	17	51	85	69	-34	36
DT+SOTB	12	16	13	-4	23	6	16	19	-10	31.5	9	16	16	-7	25.5	1	36	4	-35	38.5	7	16	18	-9	28.5	35	100	70	-65	37
DT+MAHAKIL	11	17	13	-6	26	5	18	18	-13	34.5	10	16	15	-6	23	1	36	4	-35	38.5	4	23	14	-19	35	31	110	64	-79	39.5
DT+BSMOTE	10	18	13	-8	27.5	6	15	20	-9	29.5	9	16	16	-7	25.5	2	35	4	-33	37	6	16	19	-10	30.5	33	100	72	-67	38
DT+SMOTE	8	21	12	-13	32.5	6	14	21	-8	26.5	3	21	17	-18	33.5	0	36	5	-36	40	4	17	20	-13	33	21	109	75	-88	41
DT+ROS	17	9	15	8	15.5	3	29	9	-26	37	9	11	21	-2	18	0	37	4	-37	41	3	25	13	-22	36	32	111	62	-79	39.5
DT+NOS	18	8	15	10	13.5	3	35	5	-32	38.5	9	14	18	-5	21	0	39	2	-39	42	3	29	9	-26	37	33	125	49	-92	42
DNN+WACIL	27	5	9	22	7	12	14	15	-2	18.5	34	0	7	34	2.5	16	2	23	14	12.5	20	10	11	10	15	109	31	65	78	3
DNN+SOTB	22	7	12	15	9	6	15	20	-9	29.5	26	6	9	20	10	5	34	2	-29	36	8	15	18	-7	23.5	67	77	61	-10	24.5
DNN+MAHAKIL	24	5	12	19	8	6	17	18	-11	33	29	4	8	25	9	8	17	16	-9	30	8	15	18	-7	23.5	75	58	72	17	15.5
DNN+BSMOTE	19	9	13	10	13.5	6	14	21	-8	26.5	17	10	14	7	14	8	20	13	-12	32	8	15	18	-7	23.5	58	68	79	-10	24.5
DNN+SMOTE	15	10	16	5	19	6	14	21	-8	26.5	15	11	15	4	16.5	8	18	15	-10	31	8	15	18	-7	23.5	52	68	85	-16	26.5
DNN+ROS	16	9	16	7	17	6	14	21	-8	26.5	19	10	12	9	13	8	15	18	-7	28	8	15	18	-7	23.5	57	63	85	-6	23
DNN+NOS	37	3	1	34	4	3	35	3	-32	38.5	34	0	7	34	2.5	15	3	23	12	14	3	30	8	-27	38	92	71	42	21	14

each performance measure, each single predictive model computes WDL statistics against 41 other predictive models (*6 classification models * 7 imbalance learning techniques - 1*). The ranks are assigned to predictive models on the basis of wins-draws. The predictive models that have the highest wins-draws are assigned higher ranks (starting from rank 1). In Table 3.8, gray part cells in columns represent the top seven predictive models with respect to individual measures. WACIL ranked according to individual performance measures in the following way, FOR of WACIL ranked 6, 12, 18, 5, 21, and 7. The Recall of WACIL ranked 23, 13, 18.5, 34.5, 23, and 18.5. The F-measure of WACIL ranked 2.5, 7.5, 15, 5.5, 16.5, and 2.5. The AUC of WACIL ranked 10, 1, 18.5, 2, 35, and 12.5 and the G-mean of WACIL ranked 16, 7, 23.5, 28.5, 17, and 15 on KNN, LR, NB, SVM, DT, and

DNN, respectively. The total (all performance measures combined) WDL comparisons are reported in the last column of Table 3.8. On average, WACIL ranked 4, 1, 17, 2, 36, and 3 over KN, LR, NB, SVM, DT, and DNN, respectively. From the aforementioned observations, we can conclude that the classifiers LR and DNN with WACIL perform the best against all other combinations. WACIL with SVM performed the best over FOR, F-measure, and AUC. WACIL with NB performed the best over all the measures except G-mean. All the sampling techniques combined with DT, work poorly. Overall, the total results ranks show that WACIL clearly outperforms all other predictive models with 1, 2, 3, and 4 ranks.

3.4 Discussion of Results

In this section, the analysis of experimental results is presented, exploring why WACIL achieves superior performance compared to other models, which classifier demonstrates superior performance and providing insights into the reasons behind these findings.

3.4.1 Why does proposed WACIL perform better than other sampling techniques?

The imbalance problem can be overwhelmed by the oversampling approaches by introducing new synthetic modules into the faulty class. While a few synthetic instances overlap with the non-faulty class because the SMOTE approach uses the nearest neighbors of instances to generate synthetic information, sometimes these may cross the boundary of the faulty class and overlap with the non-faulty class. On the contrary, MAHAKIL and SOTB ensure that the synthetic data doesn't cross the boundary of a faulty class, but the problem is that the diversity of generated samples is poor.

The modules which are close to the boundary are misclassified most frequently [12, 13]. WACIL wisely selects border instances, which are used to generate pseudo-instances. The selected instance is neither near nor too far, and to mitigate the limitation of closet sample selection, it divides the pool of borderline faulty class instances into three clusters (case_1, case_2, and case_3) according to Mahalanobis distance and shuffles them. WACIL generates pseudo-instances by considering the weighted average of the minimum and maximum values of those cluster instances. So, the generated pseudo data is distributed in a wide range and is rich in diversity. Furthermore, generated pseudo data is filtered to exclude out-of-boundary information, ensuring that the distribution of pseudo data is managed to remain within the boundary of the faulty class, and WACIL doesn't consider the closest modules to generate pseudo-instances. So it avoids falling into the trap of sub-cluster enhancement. All these challenges are wisely handled by WACIL, resulting in better overall performance.

3.4.2 Does filtration of generated pseudo-instance affect the WACIL performance?

During the generation of pseudo-instances, there could be some noisy modules generated. To maximize the performance of the prediction model, they have to be excluded from the set of pseudo-instances. In the filtration step, we excluded instances with more closest neighbors from the non-faulty class than the faulty class. When a classification model is trained on pseudo-instances that have a large number of closest neighbors from the non-faulty class, it misclassifies non-faulty modules as faulty modules during testing. As a result, excluding them reduces the number of false positives and increases the true negatives, which can improve the FPR and TNR of the predictive model (Discussed in Section 3.3). In the filtration step, we excluded instances, which have all K closet neighbors from the faulty class; this can reduce the predictive model's over-generalization of the faulty module class.

3.4.3 What effect does the choice of classifiers have on WACIL's performance?

As mentioned in Section 3.2.3, we have chosen a total of six classifiers as prediction models, including KNN, LR, NB, SVM, DT, and DNN. We have noticed the influence of these classifiers on WACIL and other compared approaches over 24 datasets. We have analyzed classifier performance using various performance measures, including FOR, Recall, F-measure, AUC, and G-mean. From Table 3.8, on the basis of ranks, one can conclude that WACIL with KNN, NB, and SVM provides the best results, despite showing a slight difference over a few measures and it statistically achieves superior rankings with LR and DNN classifiers in comparison to other classifiers over all the measures. Thus, LR and DNN are the supremely appropriate classifiers to build a model on training data generated by WACIL.

3.5 Threats to Validity

This section addresses the ascertained potential threats to our experimental determinations. Different assessment measures cause different results, which could lead to construction threats in this study. In this work, five distinct performance assessment measures are used to minimize threat to the model performance. The considered datasets in this study are diverse in the number of features, instances, and the percentage of faulty modules, thus it might be helpful to acquire the generalization of our determinations over divergent circumstances, which can limit the external threat. The main assumption of WACIL is that those borderline instances form three clusters (case_1, case_2, and case_3). The datasets we utilized in this study yield a minimum of three border instances. Datasets that do not produce at least three border instances may jeopardize our study's internal validity. The experiments are repeated 10 times and consider the average performance measures to avoid biased results, and the results of WACIL are compared to those of other sampling approaches. Furthermore, we have used the WSR test for WDL analysis. As a result, the conclusion validity threat to our research has a limited impact.

3.6 Summary

The WPFP confronts a challenge which is the class imbalance issue. Many prominent oversampling approaches generated synthetic data that is not diverse. Simultaneously, they ignored noisy data, which raises the misclassification rate of classifiers. Exploiting these issues, we demonstrated the imbalance learning for WPFP through the proposed WACIL approach, which introduces diverse filtered pseudo-instances around the border of the faulty class through weighted average centroid and noise filtering techniques. WACIL reduces false positives and increases the diversity of newly generated pseudo-instances. We evaluated our proposed approach based on empirical results, by comparing it with not oversampled data and the five oversampling approaches, namely ROS, SMOTE, BSMOTE, MAHAKIL, and SOTB over 24 imbalanced PROMISE and NASA projects, using six classification models, namely KNN, LR, NB, SVM, DT and DNN and five performance measures, namely FOR, Recall, F-measure, AUC, and G-mean. The experimental outcomes and statistical measures show that the performance of our method is superior to the compared approaches. In particular, WACIL outperforms all other oversampling methods in terms of FOR, F-measure, and AUC while yielding comparable results in terms of Recall and G-mean. WACIL with LR and DNN outperforms all the other predictive models over all the performance measures. The next chapter presents a novel source project selection and optimized training data construction model for cross-project fault prediction.

Chapter 4

Source Project and Optimized Training Data Selection Approach for Cross-project Fault Prediction

The Within-project prediction techniques function effectively when models have sufficient distinct and homogeneously distributed training data. In such circumstances, CPFP is an alternative approach to allow multiple projects to share available historical data [23, 25, 26, 29, 37, 109, 110, 111, 161]. We proposed a novel optimized source data selection approach called WSR test based source project selection and optimized training data construction for CPFP.

The remainder of this Chapter is organized as follows: The implementation details of the proposed WPSTC are presented in Section 4.1. Section 4.2 consists of an experimental setup. Section 4.3 reports and compares the experimental findings, While a discussion of those results is given in Section 4.4, and the possible threats to the validity of the study are discussed in Section 4.5. Finally, a summary of the chapter is discussed in Section 4.6.

4.1 Wilcoxon Signed-rank test based Source Project Selection and Optimized Training Data Construction (WPSTC)

The proposed WPSTC model is a two-phase approach, which includes source project selection in the first phase and construction of the training data in the second phase. The first phase concerns how similar distributed source projects have been selected for a particular target project. Next, the proposed fault prediction model introduced instance filtering and feature selection techniques in the second phase to reduce the distribution difference between source and target, maintain both the class count ratio, and control the curse of dimensionality issue. In section 4.1.1, we first explain the notations used in this chapter. Next, a thorough explanation of the two phases is given in the sections 4.1.2 and 4.1.3, as well as in Algorithms 2, 3, 4 and 5. The framework of the proposed WPSTC approach is shown in the figure 4.1.

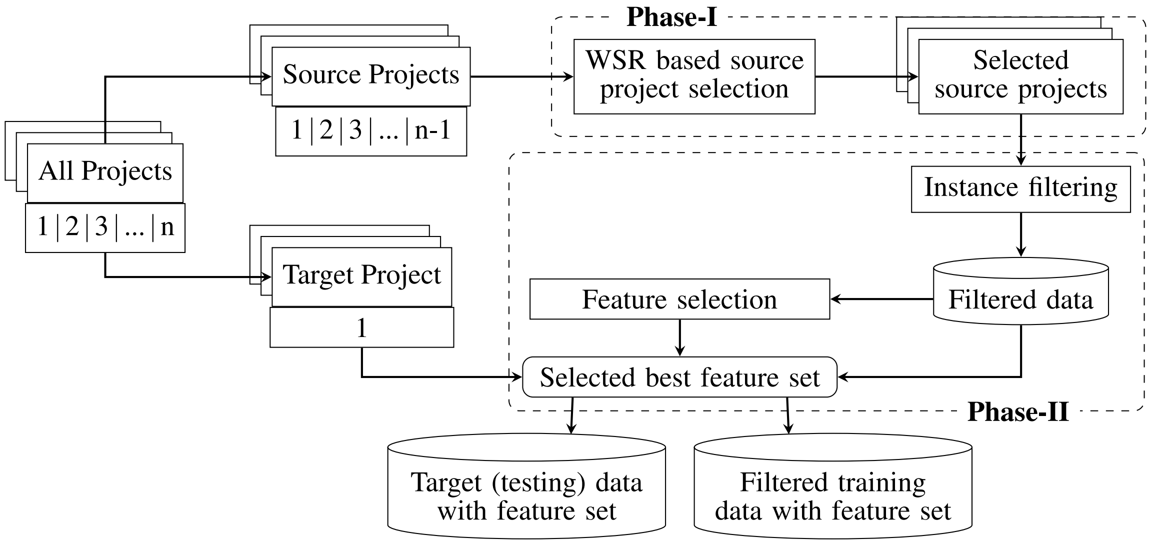


Figure 4.1: An overview of WPSTC model framework

4.1.1 Notations

To maintain a hassle-free explanation of the proposed work, we have given detailed notations of the terms used in this work.

Suppose, P_S is the set of source (training datasets) projects with labels and P_T is the set of the target (test dataset) projects without labels. P_S contains a project (training project) $D_S = \{(x_i^S, y_i^S)|i = 1, 2, \dots, n_S\}$, where x_i^S indicates the i^{th} instance in a project D_S, y_i^S is the corresponding class label and n_S is the total number of instances in that particular source project D_S. y_i^S represents fault proneness of instance, which means $y_i^S \in \{0, 1\}$, where $y_i^S = 0$ and $y_i^S = 1$ represents non-faulty modules (instances/samples) and faulty modules, respectively. P_T contains a project (test project) without labels is $D_T = \{x_i^T|i = 1, 2, \dots, n_T\}$, where x_i^T indicates the i^{th} instance in a project D_T and n_T is the total number of instances in that particular target project D_T. Since we focused on homogeneous CPFP, source and target projects possess the same features. So, each instance belongs to D_S and D_T is described as $x_i^S = [a_{ij}|j = 1, 2, \dots, m]$ and $x_i^T = [b_{ij}|j = 1, 2, \dots, m]$, respectively, where a_{ij} & b_{ij} indicates the value of *j-th* feature in the i^{th} instance of D_S and D_T, respectively and '*m*' is count of features present in D_S and D_T.

4.1.2 Phase-I: WSR based Source Project Selection

Selection of source projects is the fundamental prerequisite of training data construction since the availability of open source software datasets is multiplying from day to day. Our proposed source project selection approach is feasible when we have sufficient historical data that shares similar features with the target data, even if the distribution of the data may not be the same. When direct data is unavailable, teams can consider using proxy projects that share similar characteristics to the desired projects. They can also search for external data sources, employ manual data collection methods, and create simulations based on general industry standards to help estimate project metrics. The majority of prior research focuses on training instance filtering while overlooking the significance of selecting source projects [25, 26, 29, 116, 117, 162]. Herbold [36] states that a CPFP model can achieve a reasonable success rate with related (similarly distributed) source and target projects. Thus, in our proposed approach, we primarily pay attention to the source project selection.

Algorithm 2: WSR based Source Project Selection

```
   Require: Target set ($P_T$) and Source set ($P_S$), measure, $\alpha$;
   Ensure: Similar source projects ($Sim_{D_T}[measure]$);
1  Initialize: $D_T \leftarrow$ one of target project from $P_T$
2      $Measure_{D_T} \leftarrow$ measure of target project ($D_T$)
3      $Sim_{D_T}[measure] \leftarrow$ NULL
4  for each $D_S$ in $P_S$ do
5      $Measure_{D_S} \leftarrow$ measure of source project ($D_S$)
6      p_val $\leftarrow$ WSR ($Measure_{D_T}$,$Measure_{D_S}$)
7      if p_val > $\alpha$ then
8          Update: $Sim_{D_T}[measure]$.append ($D_S$)
9      end
10 end
11 return $Sim_{D_T}[measure]$
```

Motivated by the WSR test [157, 163], we propose WSR test based source project selection. The rationale behind using the WSR test is that its a non-parametric statistical test used to compare two related samples or paired observations. It is specifically designed to evaluate the effectiveness of different models or algorithms by analyzing their performance metrics on the same subjects. The WSR test is particularly effective for this purpose, as it assesses the median differences between paired results, offering clear insights into significant changes. Additionally, WSR is robust against outliers, relying on ranks rather than actual data values, which helps provide a more reliable assessment when the data contains extreme values. The main steps included in the choice of source projects are explained in Algorithm 2. For a new target project, based on the distributional characteristics through the WSR test, the association between it and each historical project is investigated, and the corresponding similar source projects with respect to a particular distributional characteristic are recorded. Here, we considered four distributional source and target data characteristics: mean, median, standard deviation, and maximum. These four measures are independent and show similarity when both projects are related.

As an initial step, a set of target projects (P_T), a set of source projects (P_S), a distributional characteristic measure, and alpha (α) are provided as inputs. Here, at a time, the

Algorithm 2 takes a single distributional measure as input and finds similar source projects for a target project with respect to the measure, while all other measures follow the same procedure and stores the similar source projects for a particular target in terms of individual distributional characteristic measures. The algorithm's output is a set of similar projects with respect to a specific measure ($Sim_{D_T}[measure]$) for a target project.

We calculate the above-mentioned four characteristic measures for all the features in the target project ($D_T \in P_T$, $[Measure_{D_T}]_{m\times1}$). A project's measure is a row vector with the size of the number of features (m) present in a project, and each cell contains the characteristic measure value of the respective feature. Similarly, we calculate the measure value ($[Measure_{D_S}]_{m\times1}$) of each source project ($D_S \in P_S$). Then, the WSR test is performed on both target and source measures and returns a P-value. We conducted a WSR statistical test with a confidence level of 95% (α=0.05). If the P-value exceeds the significance level (α), the test failed to reject the null hypothesis, indicating that the two have statistically similar distribution of measure values, then the particular source project will be added to the set of similar sources for a particular measure of the target project. When all source projects' WSR tests are completed, it will return similar sources of target with that measure. The same process follows for all the other measures and returns similar sources for all of them. Then, we take the count of each source project from four sets of similar sources, and the source projects that have a higher count than the average count will be considered as the final source projects for the target project. In this process, the source projects set ($Source_{D_T}$) is selected for the target project D_T.

4.1.3 Phase-II: Optimized Training Data Construction (optimizedTC)

The training data is constructed for a target project using the above-extracted source projects in this phase. Constructing optimized training data consists of the proposed new instance filtering technique and the Binary-RAO algorithm feature selection technique. The comprehensive interpretation of these steps can be found in Algorithms 3, 4 and 5.

```
Algorithm 3: Optimized Training Data Construction
  Require: Target project ($D_T$),Source of $D_T$ ($Source_{D_T}$);
  Ensure: Final training data;
1 Calls the instance filtering function with target project and sources of target project
    as parameters
2 $S_{filt} \leftarrow$ INSTANCE_FILTERING ($D_T$, $Source_{D_T}$)
3 Calls the feature selection function with target project and filtered training data as
    parameters
4 $S_{fin} \leftarrow$ FEATURE_SELECTION ($D_T$, $S_{filt}$)
5 return $S_{fin}$
```

4.1.3.1 Instance Filtering

Even after source project selection, cross-project prediction is still challenging because the distributions of a few instances from the selected projects are not a perfect match to the target project and the chosen source projects still pose a class imbalance nature. We introduce a new instance filtering phase as a part of our proposed approach to address these two issues. Algorithm 4 presents the instance filtering process in detail. *Step 1:* Target project (D_T) and the selected source projects of D_T ($Source_{D_T}$) are provided as input parameters and return filtered training data (S_{filt}).The algorithm iterates until $Source_{D_T}$ is NULL. In each iteration, we take one of the source projects as training data (S) and perform the instance filtering process, then append the filtered data to S_{filt}.

Step 2: The instance filtering process begins here, test_data (T) (target project data) and training_data (S) (source project data). Divide the training dataset (S) into a faulty instances class set (S_{min}) (minority class) and a non-faulty instances class set (S_{maj}) (majority class). we perform a partition of the test data (T) into faulty (T_{min}) and non-faulty (T_{maj}) sets with artificially generated class labels using the nearest neighbours (NNs) concept. To begin, compute K1, which is the square root of the number of instances in S_{min}, and then compute K1-NNs for each instance of the testing dataset (T) in the training dataset (S). If the number of computed NNs belonging to S_{min} exceeds the number of computed NNs belonging to S_{maj} for each instance in T, then update the testing faulty class set (T_{min}) with that instance of T; otherwise, update the testing non-faulty class set (T_{maj}) with that instance of T. All instances from T are divided based on their close relationships, yielding

Algorithm 4: Instance Filtering Phase

```
Require: $D_T, Source_{D_T}$;
Ensure: $S_{filt}$;
function Instance_Filtering$D_T, Source_{D_T}$
    Initialize: test_data ($T$) $\leftarrow D_T$ without labels
        $S_{filt} \leftarrow$ NULL
    for each $S$ in $Source_{D_T}$ do
        $S_{min} \leftarrow$ faulty instances of $S$
        $S_{maj} \leftarrow$ non-faulty instances of $S$
        K1 $\leftarrow$sqrt(len($S_{min}$))
        Find K1-NNs of each instance of $T$ in $S$
        $T_{min} \leftarrow$ NULL
        $T_{maj} \leftarrow$ NULL
        for each inc in $T$ do
            if #NN(inc)$\in S_{min} \geq$ #NN(inc)$\in S_{maj}$ then
                Update: $T_{min}$.append (inc)
            else
                Update: $T_{maj}$.append (inc)
            end
        end
        Find one NN of each instance of $T_{maj}$ in $S_{maj}$
        Take union of all the NNs in Array1 along with counts
        for each inc in Array1 do
            if Count(inc) $\geq$ Avg(count) then
                Update: $Maj\_set$.append (inc)
            end
        end
        SMOTE is used to generate len($Maj\_set$) number of new faulty instances in to training
          data ($S$)
        $S_{min} \leftarrow$ faulty instances of new $S$
        K2 $\leftarrow$ ceil(len($Maj\_set$/len($T_{min}$))
        Find K2-NNs of each instance of $T_{min}$ in $S_{min}$
        Take union of all the NNs in Array2 along with counts
        for each inc in Array2 do
            if Count(inc) $\geq$ Avg(count) then
                Update: $Min\_set$.append (inc)
            end
        end
        Update: $S_{filt}$.append($Maj\_set$, $Min\_set$)
    end
    return $S_{filt}$
end
```

T_{min} and T_{maj} sets (lines 5–15 in Algorithm 4).

Step 3: To make data balance, we compute one NN of each instance of T_{maj} in S_{maj} because non-faulty instances outnumber the faulty class instances. Take the union of all the neighbours in a set, along with the number of times that instance occurred as a neighbor. For which neighbor instance the count exceeds the average count append it to the final source project majority set (Maj_set), which means we are holding non-faulty source project instances that are identical to the target project (lines 16-22 in Algorithm 4).

Step 4: The source projects also suffer from an imbalance issue. That means there are fewer faulty instances than non-faulty instances. If we generate a minority class set from the source project, the generated training data will be unbalanced. So, before generating a similar minority class set, we employed SMOTE [11] to generate the len(Maj_set) number of new faulty instances into S. Now S_{min} is a faulty instance set of newly generated S, and k2 is the number of instances in Maj_set divided by the number of instances in T_{min}. Compute the K2 number of NNs of each instance of T_{min} in S_{min} and take a union of all the neighbours in a set along with the number of times that instance occurred as a neighbor. For which neighbor instance the count exceeds the average count, append it to the final source project minority set (Min_set), which means we are holding faulty source project instances similar to the target project into one set. Update S_{filt} in each iteration with computed maj_set and Min_set (lines 23-33 in Algorithm 4). S_{filt} is the filtered instances dataset after all iterations.

4.1.3.2 Feature Selection

After source projects and training data selection, CPFP performance could be further improved via feature selection. RAO algorithm [164, 165] is a simple and algorithm-specific parameter-less optimization technique that can effectively solve complex problems. We employ a Binary-RAO optimization technique for feature selection in this study to mitigate the issue of the curse of dimensionality over software fault datasets. Algorithm 5 explains the step-by-step procedure of the feature selection algorithm. In the algorithm, the feature selection function takes the target project and the above generated filtered source data (S_{filt}) as input parameters and returns the best feature subset for a particular target project.

Step 1: we define the size of the population as 'n', the number of features present in both source and target is 'm', and created a random 2D binary population (POP) with the size of 'n' number of rows and 'm' number of columns. In $POP_{n\times m}$, zero denotes the absence of a feature and one denotes its presence. We experimented on all the datasets to determine

Algorithm 5: Feature Selection Phase

```
Require: $D_T$,$S_{filt}$;
Ensure: $p_{best}$;
function Feature_Selection$D_T, S_{filt}$
    Initialize: n ← population size
        m ← #features of project
        $POP_{n\times m}$ ← random binary population
        $fit\_val$ ← NULL
    for each $i$ in $len(POP)$ do
        $fit\_val[i]$ ← Fit_fun($POP[i]$,$S_{filt}$)
    end
    while $(termination\_condition)$ do
        $p_{best} \leftarrow POP[max[fit\_val]]$
        $p_{worst} \leftarrow POP[min[fit\_val]]$
        for each $i$ in $len(POP)$ do
            $X_{old} \leftarrow POP[i]$
            $X_{ran}$ ← randomly selected population
            Find $X_{new}$ using $X_{old}$ and $X_{ran}$ via Eqs. 4.2, 4.3 and 4.4
            $New_v$ ← Fit_fun($X_{new}$,$S_{filt}$)
            if (Objective_Function) then
                Update $POP[i]$ with $X_{new}$
                Update $fit\_val[i]$ with $New_v$
            else
                Keep $X_{old}$ at $POP[i]$
                Keep $fit\_val[i]$ as of $X_{old}$
            end
        end
    end
    return $p_{best}$
end
```

the maximum number of iterations needed to converge the RAO algorithm and regard the iteration count as the termination criteria. Almost all the datasets converge near 50 iterations, so we consider termination criteria to be 50 iterations for all the projects.

Step 2: Define the objective function: our Binary-RAO algorithm has two consecutive objectives, such as maximization of the fitness function and minimization of the total number of features with maximum fitness value. Thus, the algorithm updates the population's solution if the solution satisfies these two objectives, the formal equation of objective function given in Eq. 4.1.

Define fitness function ($Fit_{f}un$): In our approach, we used the AUC score as the fitness value. $Fit_{f}un$ takes the one solution from the population and source data as inputs and returns all the classifiers' average AUC scores. If we consider only one classifier to find the AUC score, the selected feature set shows partiality towards a particular classifier. Thus, we considered the average score of all the classifiers as the return score to ensure the chosen feature set performs equally well on all the classifiers.

$$\begin{aligned} Objective\_Function = &\Bigg(\Big(fit\_val[X_{new}] \geq fit\_val[X_{old}]\Big) \\ &OR \\ &\Big(fit\_val[X_{new}] == fit\_val[X_{old}] \\ &AND \\ &Count(X_{new}) \leq Count(X_{old})\Big)\Bigg) \end{aligned} \tag{4.1}$$

Step 3: We calculated the fitness value for all the solutions in POP (lines 3-5 in Algorithm 5).

Step 4: The iteration process begins here, and we identify the best (p_{best}) and worst (p_{worst}) solution from the population based on their fitness function values. p_{best} is the solution (feature set) having the maximum fitness value and p_{worst} is the solution having the minimum fitness value (lines 7 and 8 in Algorithm 5).

Step 5: We update each population ($i = 1, 2, \ldots, len(POP)$) by following Eqs. 4.2, 4.3 and 4.4. For that, selects a random solution (X_{ran}) from POP. If the current solution's fitness value $fit\_val[X_{old}]$ is greater than the randomly selected solution's fitness value $fit\_val[X_{ran}]$, then X_{new} follows the below given: Eq. 4.2

$$Temp_{m\times 1} = X_{old} + r1(p_{best} - p_{worst}) + r2(X_{old} - X_{ran}) \tag{4.2}$$

If the randomly selected solution's fitness value $fit\_val[X_{ran}]$ is greater than the current solution's fitness value $fit\_val[X_{old}]$, then X_{new} follows the below given: Eq.4.3

$$Temp_{m\times 1} = X_{old} + r1(p_{best} - p_{worst}) + r2(X_{ran} - X_{old}) \tag{4.3}$$

$$[X_{new}]_{m\times 1} = \begin{cases} 0, & \text{if } Temp_{m\times 1} < 0.5. \\ 1, & \text{otherwise.} \end{cases} \tag{4.4}$$

Where $r1$ and $r2$ are two random numbers selected in the range [0, 1] for the current iteration. The last terms in Eqs. 4.2 and 4.3 indicate the random cooperation between the present solution and the randomly selected solution (lines 9–13 in Algorithm 5).

Step 6: If the objective function is satisfied with X_{new}, then replace X_{old} in POP with X_{new} and update $fit\_val[X_{old}]$ with $fit\_val[X_{new}]$, otherwise discard the new solution and keep POP and fit_val as previous (lines 14-20 in Algorithm 5). The in-detailed objective function is given in the Eq. 4.1.

Step 7: In the last step, if the termination criterion is satisfied, we report the optimum solution (p_{best}). Else, we followed the process from Step 4. Finally, S_{filt} data with selected p_{best} features are used to build the Classification model.

4.2 Experimental Setup

This section includes the details of utilized projects, their features, and the evaluation metrics we used to assess the proposed work. Next, a concise description of the base models, classifier selection, and an overview of statistical comparison measures are explained.

4.2.1 Experimental Objects

In the experimental work, we used 24 diverse projects to evaluate WPSTC, including 14 projects from PROMISE [34, 148] and ten projects from NASA [25, 37, 43, 70, 149, 150]. A total of 13 projects from NASA and approximately 30 projects from the PROMISE repository are available for analysis. We selected specific projects that have undergone extensive research, ensuring that they possess a robust dataset and relevant performance metrics. Focusing on specific projects with established performance metrics allows for meaningful comparisons among different prediction models, leading to more robust conclusions. The project's detailed summary is explained in Table 3.1 of Chapter 3. For each PROMISE project experiment, an extracted similar project set from the rest of the PROMISE projects serves as the source projects. The same type of experiments were performed by NASA. The PROMISE projects have the same feature sets, whereas NASA projects have different feature sets [43, 150]. All 20 features from PROMISE and 35 common features from all NASA projects were used to perform a homogeneous CPFP.

4.2.2 Performance Assessment Measures

To assess the performance of the proposed model, the FOR, Recall, micro-averaged F-measure, G-mean, AUC, nMCC and Balance are considered as evaluation metrics in our experiments [25, 26, 29, 31, 37, 149]. The evaluation metrics FOR and Recall are used to assess the model over faulty class, and F-measure, AUC, G-mean, Balance, and nMCC are used to assess the model's overall performance. These measures are extracted from the confusion matrix (represented in Table 3.4 of Chapter 3), which is built from the actual labels and prediction model generated labels.

The utilized evaluation metrics are defined as follows:

$$FOR = \frac{FP}{FP + TN} \tag{4.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.6}$$

$$F - measure = \frac{2TP}{2TP + FP + FN} \tag{4.7}$$

$$G - mean = \sqrt{Acc_{+} * ACC_{-}} = \sqrt{Recall * (1 - FOR)} \tag{4.8}$$

$$Balannce = 1 - \frac{\sqrt{(0 - FOR)^2 + (1 - Recall)^2}}{\sqrt{2}} \tag{4.9}$$

MCC: MCC calculates the Pearson product-moment correlation coefficient between actual and predicted values. MCC values range from [-1, 1], so used normalized MCC (nMCC) for statistical comparisons between experiments [31]. nMCC ranges between [0, 1].

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TN + FN)(TP + FN)(FP + TN)}} \tag{4.10}$$

$$nMCC = \frac{MCC + 1}{2} \tag{4.11}$$

AUC: AUC is a performance metric that measures the tradeoff between FOR and Recall.

All measures except FOR, the higher the values, the better the classifier's performance. As per FOR, a lower value indicates better performance. All the measures range between [0, 1] except MCC.

4.2.3 Classifier Selection

In this work, we employed five standard ML algorithms as classifiers. These ML algorithms are KNN, LR, SVM, NB, and DT, which have been extensively used in similar CPFP models [23, 37, 121]. In addition, the voting-based ensemble learning algorithm was used as a classification model. These five conventional ML classifiers are used as estimators in that ensemble classifier. The estimated probability of the output class of each estimator is used to calculate voting. To implement the aforementioned classifiers, we use the Python Scikit-learn library.

4.2.4 Statistical Measures

To analyze WPSTC's experimental outcome and demonstrate its superiority, we choose the WSR as a statistical performance measure [25, 37, 157]. The matched-pairs rank biserial correlation coefficient measure is used to find the strength of the relationship (effect size) between the pairs [20, 158, 166]. The effect size (E_c) ranges between [0, 1], and it can be interpreted using three labels [20], large ($E_c \geq 0.5$), medium ($0.1 < E_c < 0.5$) and small ($E_c \leq 0.1$). Furthermore, to ensure the proposed model superiority, we compare each base model and our model with that of the remaining predictive models (*classifiers * fault prediction models-1*) using widely popular WDL [26, 166] comparison technique.

4.3 Experimental Results

In this section, we addressed the computed FOR, recall, F-measure, AUC, G-mean, nMCC, and Balance values of each model for all 24 projects with respect to each classification model. We compared our WPSTC findings with six baseline models, including WPFP, allCPFP, WPS+CPFP, WPS+TCA [22], WPS+NNFilter [29], TCA+STrNN [25]. In allCPFP approach, for every experiment, one project operates as testing data and the rest of the projects operate as training data for the prediction model. Basic WPFP and allCPFP models implemented in Python. We utilised the source code provided by its author for TCA implementation [22], while NNFilter and STrNN are implemented in Python according to the algorithms outlined in [29] and [25], respectively.

Firstly, we reported the WPS selected similar source projects in Table 4.1. Few projects don't have any characteristics similar to other projects, considering all the available homogeneous projects as sources.

Table 4.1: selected source projects for each target project

Target Project	Selected source projects
Ant-1.7	Ivy-2.0, Poi-2.0, Tomcat, Xalan-2.4, Xerces-1.3
Arc	Lucene-2.0, Redaktor, Synapse-1.2
Camel-1.6	Lucene-2.0, Redaktor, Tomcat, Velocity-1.6, Xerces-1.3
Ivy-2.0	Ant-1.7, Lucene-2.0, Xerces-1.3
Jedit-4.2	Ant-1.7
Log4j-1.0	Arc, Redaktor
Lucene-2.0	Arc, Camel-1.6, Ivy-2.0, Synapse-1.2
Poi-2.0	Ant-1.7, Camel-1.6, Velocity-1.6, Xalan-2.4
Redaktor	Arc, Camel-1.6, Lucene-2.0, Synapse-1.2
Synapse-1.2	Arc, Lucene-2.0, Redaktor
Tomcat	Ant-1.7, Camel-1.6, Velocity-1.6, Xerces-1.3
Velocity-1.6	Camel-1.6, Tomcat, Xerces-1.3
Xalan-2.4	Ant-1.7, Poi-2.0, Tomcat
Xerces-1.3	Ant-1.7, Camel-1.6, Ivy-2.0, Tomcat, Velocity-1.6
CM1	MC2, KC3
MW1	PC2, PC4
KC3	CM1, MC2, PC1, PC2
MC1	PC2, PC4
MC2	CM1, KC3, PC2
PC1	CM1, KC3, PC2, PC3
PC2	KC3, MC1, MW1, PC1, PC3, PC4
PC3	MC1, PC1, PC2
PC4	MC1, MW1, PC2
PC5	CM1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4

Furthermore, the obtained results are depicted through scatter plots in Figures 4.2 and 4.3. The Y-axis and X-axis of the figures represent the projects and their performance measure values, respectively. The results of all models are compared in the Tables 4.2, 4.4 and 4.5. Table 4.2 shows the averaged measures of all the projects for each prediction model, the bold value represents the highest measure value. Table 4.4 shows the statistical comparison between WPSTC and the other fault prediction models over all 24 projects for each classifier and the average of all classifiers. For each measure, it shows the P-value and the effect size (Ec), the bold part (P-value < 0.0.5) represents WPSTC as statistically superior, and the bold italic part (P-value > 0.05) represents WPSTC statistically similar to other models in terms of a particular measure. Table 4.5 shows the WDL statistics of each fault prediction model combined with each classifier over other 41 (*7 (fault prediction models) * 6 (classifiers)-1*) models. Each measure shows W-L (the difference between the number of wins and losses), and the rank was assigned to W-L values. The sum of wins minus losses of all the measures and ranks assigned to the sum of wins - the sum of losses are represented in the last column of Table 4.5. The grey part represents the top 7 ranked models for each performance measure. We performed an extensive study and explored the following three research questions to assess the effectiveness of the WPSTC model:

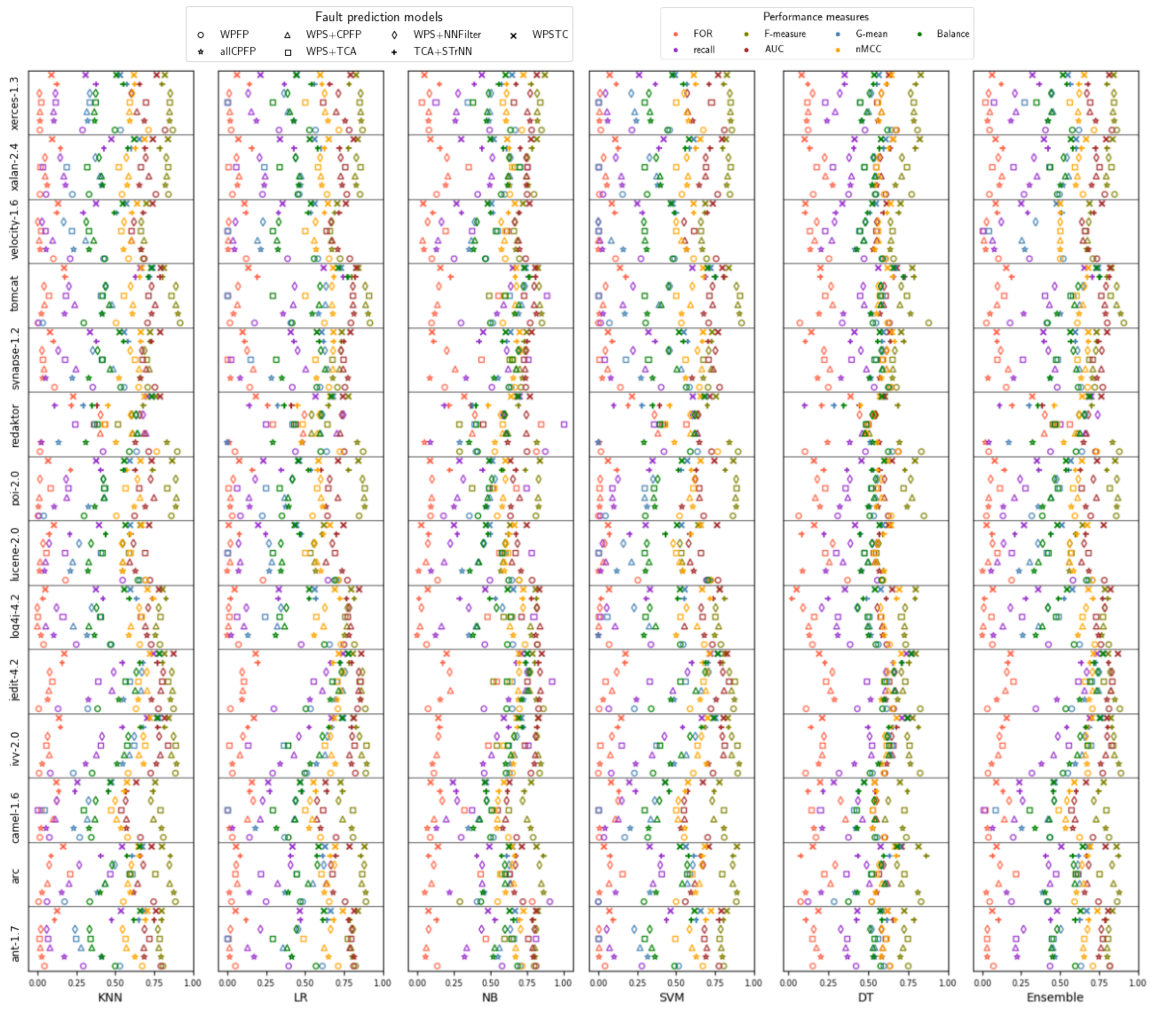


Figure 4.2: Scatter plots of fault prediction models over each classifier across 14 PROMISE projects

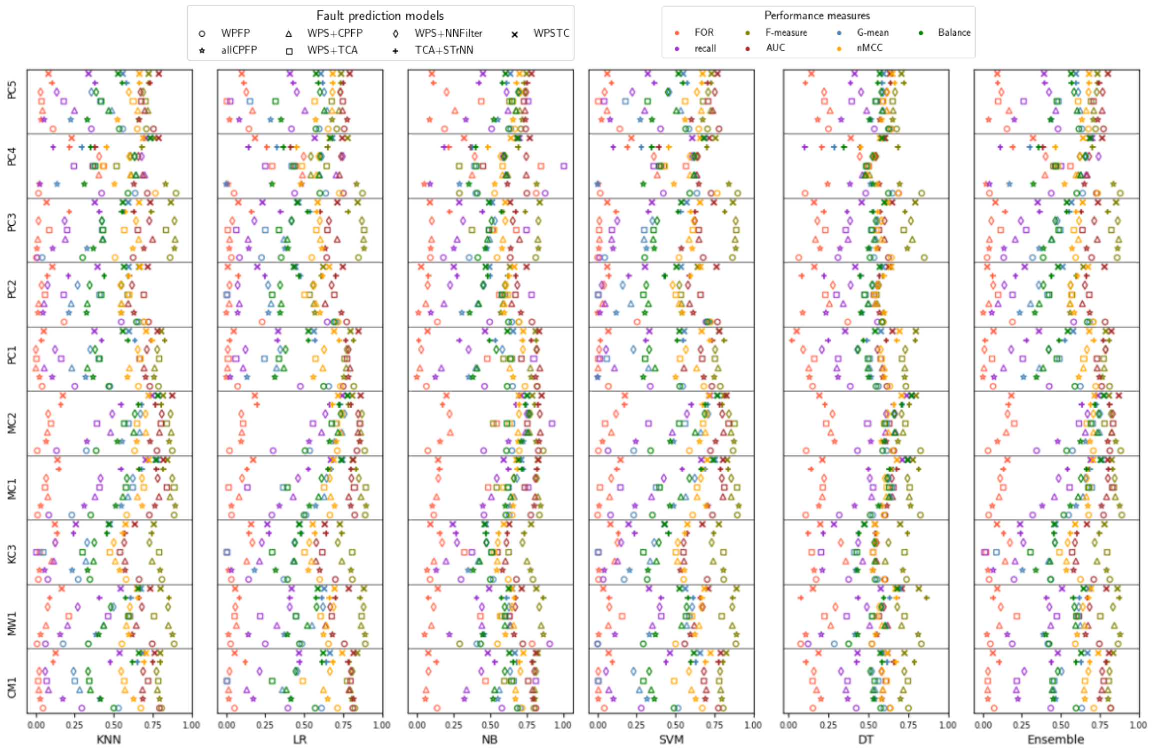


Figure 4.3: Scatter plots of fault prediction models over each classifier across 10 NASA projects

4.3.1 RQ1: How's the WPS+CPFP performance compared with allCPFP?

Figures 4.2 and 4.3 provide more insights into the WPS+CPFP and allCPFP results. The symbols $\triangle$ and $\star$ represent WPS+CPFP and allCPFP results, respectively. As we can observe from the figures, WPS+CPFP surpasses allCPFP across the following count of projects out of 24 projects with different classification models: WPS+CPFP in combination with KNN 11, 14, 13, 14,14,15 and 14 projects, with LR 4, 16, 6, 8,17,12 and 17 projects, with NB 1, 24, 3, 12,24,20 and 24 projects, with SVM 8, 17, 7, 18,17,15 and 17 projects, with DT 14, 15, 16, 16,15,18 and 14 projects, with ensemble learner 6, 18, 5, 16,18,17, and 19 projects achieve greater scores out of 24 projects against allCPFP in terms of FOR, Recall, F-measure, AUC, G-mean, nMCC, and Balance measures. As we can see from Table 4.2, on average across 24 experiments, WPS+CPFP outperforms the allCPFP by 58.38%, 1.65%, 22.64%, 2.45%, and 14.43% in terms of Recall, AUC, G-mean, nMCC, and Balance, respectively. The WPS+CPFP is unable to outperform with FOR and F-measure, because allCPFP uses more data for training, so the misclassification rate of faulty modules is less for it, but the recognition of faulty and non-faulty modules should be balanced for a better prediction model. The WPS+CPFP achieves good results over balanced measures like G-mean and Balance, and overall performance measures AUC and nMCC are also superior compared to allCPFP.

The statistical comparison of both models over each classifier's predicted results are given in Table 4.3. The bold P-value and the bold italic P-value represent WPS+CPFP as statistically superior and statistically have no significant difference (similarly distributed) to allCPFP, respectively. The models' KNN and DT generated data on FOR and F-measure are statistically similar. Nevertheless, in most cases, the WPS+CPFP outperforms CFPF with medium to large effect sizes for the rest of the measures.

Based on prediction performance and statistical comparison, FOR, Recall, G-mean, and Balance values indicate WPS+CPFP can handle imbalance issues much better. The G-mean, AUC, Balance, and nMCC indicate the overall performance of WPS+CPFP is superior to allCPFP.

Table 4.2: Average results of 24 projects with respect to each fault prediction models

Models	Evaluation measures						
	FOR	Recall	F-measure	AUC	G-mean	nMCC	Balance
WPFP	0.085	0.278	**0.815**	0.726	0.427	0.618	0.473
allCPFP	**0.063**	0.173	0.804	0.668	0.349	0.572	0.409
WPS+CPFP	0.108	0.274	0.773	0.679	0.428	0.586	0.468
WPS+TCA	0.161	0.258	0.730	0.645	0.323	0.551	0.417
WPS+NNFilter	0.110	0.277	0.771	0.678	0.429	0.585	0.470
TCA+STrNN	0.138	0.370	0.768	0.670	0.548	0.609	0.540
WPSTC	0.127	**0.423**	0.784	**0.733**	**0.589**	**0.638**	**0.573**

4.3.2 RQ2: Is WPSTC performing better than the WPFP?

The supposition is that WPFP can accomplish the highest performance than the CPFP models. We compare our WPSTC performance with the WPFP approach's performance to investigate the supposition. The WPSTC and WPFP's in-detail results are demonstrated

Table 4.3: Statistical assessment of WPS+CPFP with allCPFP model over six classifiers for each evaluation measure

Models	Evaluation measures													
	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
KNN	***0.386010***	0.214	***0.071846***	0.420	***0.223215***	0.296	***0.845534***	0.040	***0.368071***	0.210	***0.18866***	0.300	***0.071861***	0.417
LR	0.002031	0.723	**0.008143**	0.630	0.025805	0.527	***0.091851***	0.390	**0.022541**	0.543	***0.283903***	0.247	**0.008512**	0.627
NB	3.03E-05	0.973	**1.82E-05**	1.000	0.000301	0.843	***0.903163***	0.029	**1.82E-05**	1.000	**0.001549**	0.783	**1.82E-05**	1.000
SVM	0.005122	0.693	**0.009613**	0.645	0.007393	0.652	**0.009322**	0.610	**0.015707**	0.602	***0.306369***	0.245	**0.010104**	0.641
DT	***0.977206***	0.007	***0.170198***	0.320	***0.615755***	0.116	***0.109562***	0.373	***0.324249***	0.230	***0.145036***	0.340	***0.265132***	0.260
Ensemble	0.001673	0.730	**0.001517**	0.743	0.001754	0.727	***0.106430***	0.380	**0.003565**	0.680	**0.049777**	0.467	**0.001403**	0.764

in scatter plot Figures 4.2 and 4.3. The symbols $\times$ and $\circ$ represent WPSTC and WPFP results, respectively. The scatter plots show that WPSTC obtains greater scores than the WPFP for most projects. Comprehensively, WPSTC surpasses WPFP across the following count of projects out of 24 projects with different classification models: the WPSTC in combination with KNN 8, 22, 9, 13,20,19 and 21 projects; with LR 9, 14, 8, 12,15,13 and 15 projects; with NB 15, 13, 14, 14,15,13 and 18 projects; with SVM 7, 18, 9, 12,19,17 and 18 projects; with DT 11, 12, 9, 14,13,13 and 14 projects; with ensemble learner 7, 16, 7, 13,16,14 and 19 projects achieve greater scores out of 24 projects against WPFP in terms of FOR, Recall, F-measure, AUC, G-mean, nMCC and Balance measures respectively. As shown in Table 4.2, on average across 24 experiments, WPFP outperforms WPSTC in terms of FOR and F-measure by 33.07% and 3.95%, respectively, whereas WPSTC outperforms WPFP by 52.16%, 0.96%, 37.94%, 3.24%, and 21.14% in terms of Recall, AUC, G-mean, nMCC and Balance, respectively.

The pair-wise WSR test provides a better insight into the results. The statistical comparison of both models over each classifier predicted results is given in Table 4.4. The first row of each classifier in Table 4.4 shows the statistical comparison between WPSTC and WPFP. With the majority of the classifiers Recall, AUC, G-mean, nMCC, and Balance results show that WPSTC is superior to WPFP. On average Recall, G-mean and Balance with P-values 0.00373, 0.000606, and 0.002235, as well as effect sizes of 0.677 (large), 0.8 (large), and 0.713 (large), provide evidence for the statistical superiority of WPSTC over WPFP. On average, WPSTC's FOR and F-measure with P-values of 0.015158 and 0.007881 could not outperform the WPFP. On average, AUC and nMCC with P-values of 0.33131 and 0.055347 as well as effect sizes of 0.227 (medium) and 0.453 (medium), provide evidence for such insignificant differences.

Based on prediction performance and statistical comparison, the WPSTC's FOR values are high, but the Recall, G-mean and Balance values are higher than the WPFP, which indicates WPSTC can handle imbalance issues much better than the WPFP. The G-mean, AUC, Balance, and nMCC results are much better than WPFP, which means the overall performance of WPSTC is superior to WPFP.

4.3.3 RQ3: How's the overall WPSTC performance compared with base CPFP models?

The overall results of 24 projects for each fault prediction method, classifier, and performance measures are depicted in Figures 4.2 and 4.3. For each individual classifier, the symbols $\star$, $\triangle$, $\square$, $\diamond$, + and $\times$ represent allCPFP, WPS+CPFP, WPS+TCA, WPS+NNFilter, TCA+STrNN, and our WPSTC. The different colours indicate the different performance measures. We can note from the plots that our approach produces better scores than the other cross-project prediction models for more projects with most of the measures and gives comparable results for a few measures like FOR. Comprehensively, on average, WPSTC surpasses WPFP across the following count of projects out of 24 projects with different classification models: the WPSTC has greater scores over allCPFP 2, 24, 10, 22,24,24 and 24 times; over WPS+CPFP 9, 23, 15, 22,23,24 and 23 times; over WPS+TCA 17, 24, 21, 23,24,24 and 24 times; over WPS+NNFilter 7, 23, 15, 22,24,24 and 24 times; over TCA+STrNN 15, 13, 18, 21,15,19 and 14 times out of 24 experiments in terms of FOR, Recall, F-measure, AUC, G-mean, nMCC and Balance measures respectively. We can see the average of all 24 projects' measure values for each fault prediction model in Table 4.2, WPSTC outperforms TCA+STrNN by -7.97%, 14.32%, 2.08%, 9.4%, 7.48%, 4.76% and 6.11%; WPS+NNFilter by 15.45%, 52.77%, 1.69%, 13.64%, 37.3% and 9.06%, 21.91%; WPS+TCA by -21.12%, 63.95%, 7.4%, 13.64%, 82.35%, 15.79% and 37.41%; WPS+CPFP by 17.59%, 54.38%, 1.42%, 7.95%, 37.62%, 8.87% and 22.44%; allCPFP by 98.44%, 144.51%, -2.49%, 9.73%, 68.77%, 11.54% and 40.1% in terms of FOR, Recall, F-measure, AUC, G-mean, nMCC and Balance. The negative and positive percentages indicate a decrease in value and an increase in value, respectively. For a better prediction

Table 4.4: Statistical comparison of WPSTC with other fault prediction models using KNN, LR, NB, SVM, DT and Ensemble classifiers

K-Nearest Neighbour														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	7.14E-05	0.927	**4.96E-05**	0.947	0.000673	0.793	***0.423663***	0.187	**6.33E-05**	0.933	**0.001517**	0.743	**4.39E-05**	0.953
allCPFP	1.82E-05	1	**1.82E-05**	1	0.032116	0.5	**0.000255**	0.853	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+CPFP	0.000285	0.847	**1.82E-05**	1	***0.113725***	0.373	**0.000129**	0.897	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+TCA	0.000441	0.82	**1.82E-05**	1	***0.331335***	0.223	**5.61E-05**	0.94	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+NNFilter	0.000318	0.84	**1.82E-05**	1	***0.235711***	0.277	**5.61E-05**	0.94	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
TCA+STrNN	***0.466127***	0.17	**0.000526**	0.83	***0.212366***	0.293	**0.000115**	0.9	**0.000285**	0.847	**0.000228**	0.86	**0.000318**	0.84
Logistic Regression														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	0.010128	0.597	**0.059333**	0.44	0.000192	0.87	***0.731666***	0.077	**0.042502**	0.467	***0.52963***	0.143	***0.052033***	0.453
allCPFP	0.000182	0.873	**3.02E-05**	0.973	***0.103386***	0.38	**0.005826**	0.643	**2.67E-05**	0.98	**0.000162**	0.88	**2.84E-05**	0.977
WPS+CPFP	***0.071861***	0.42	**0.000269**	0.85	***0.657836***	0.103	**0.000285**	0.843	**2.67E-05**	0.98	**0.000918**	0.773	**3.02E-05**	0.973
WPS+TCA	0.000145	0.887	**1.82E-05**	1	***0.492893***	0.16	**0.001069**	0.763	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+NNFilter	0.034472	0.493	**0.000182**	0.873	***0.720971***	0.08	**0.000255**	0.853	**2.50E-05**	0.983	**0.001592**	0.737	**2.35E-05**	0.987
TCA+STrNN	**0.022257**	0.537	***0.886397***	0.03	**0.014561**	0.567	**0.000441**	0.82	***0.689157***	0.09	**0.017675**	0.562	***0.753304***	0.067
Navie Bayes														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	***0.483905***	0.163	***0.737954***	0.08	***0.852654***	0.04	***0.784276***	0.065	***0.050319***	0.457	***0.259054***	0.263	***0.065737***	0.435
allCPFP	0.000546	0.807	**0.000102**	0.907	***0.174714***	0.317	**0.042502**	0.473	**4.96E-05**	0.947	**8.54E-05**	0.917	**8.05E-05**	0.92
WPS+CPFP	***0.797058***	0.06	**0.042502**	0.473	***0.129953***	0.357	***0.094603***	0.393	**0.007877**	0.62	**0.02582**	0.52	**0.009319**	0.61
WPS+TCA	**2.07E-05**	0.993	**8.05E-05**	0.92	**1.82E-05**	1	**0.000546**	0.807	**0.019137**	0.553	**8.53E-05**	0.917	**0.043969**	0.47
WPS+NNFilter	***0.324224***	0.23	***0.18875***	0.307	***0.137315***	0.347	**0.013448**	0.573	***0.078878***	0.41	***0.082944***	0.417	***0.112786***	0.37
TCA+STrNN	***0.137274***	0.347	***0.689157***	0.093	***0.063277***	0.437	**0.006642**	0.637	***0.668203***	0.097	**0.015148**	0.567	***0.830316***	0.047
Support Vector Machine														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	0.004675	0.678	**0.001517**	0.74	0.00148	0.757	***0.338471***	0.223	**0.000491**	0.813	**0.013755**	0.591	**0.000829**	0.78
allCPFP	1.82E-05	1	**1.82E-05**	1	***0.059332***	0.449	**2.07E-05**	0.993	**1.82E-05**	1	**3.02E-05**	0.973	**1.82E-05**	1
WPS+CPFP	0.004275	0.667	**3.65E-05**	0.963	***0.797048***	0.06	**7.69E-05**	0.942	**2.07E-05**	0.993	**1.82E-05**	1	**2.20E-05**	0.99
WPS+TCA	0.001843	0.727	**2.07E-05**	0.993	***0.637318***	0.107	**5.61E-05**	0.94	**2.07E-05**	0.993	**1.82E-05**	1	**2.07E-05**	0.993
WPS+NNFilter	0.003905	0.677	**3.02E-05**	0.973	***0.466241***	0.167	**0.000102**	0.907	**2.07E-05**	0.993	**2.07E-05**	0.993	**2.67E-05**	0.98
TCA+STrNN	***0.091834***	0.39	***0.977206***	0.007	**0.014**	0.573	**6.33E-05**	0.933	***0.764165***	0.07	**0.008135**	0.63	***0.830316***	0.05
Decision Tree														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	***0.647568***	0.107	***0.567709***	0.133	***0.345705***	0.213	***0.415457***	0.187	***0.360567***	0.213	***0.731706***	0.077	***0.647568***	0.107
allCPFP	***0.074131***	0.42	**0.001307**	0.75	**0.000918**	0.773	**4.39E-05**	0.953	**0.000318**	0.837	**2.35E-05**	0.987	**0.000441**	0.82
WPS+CPFP	**0.001405**	0.761	**0.006358**	0.64	**9.07E-05**	0.913	**3.88E-05**	0.96	**0.000162**	0.88	**3.03E-05**	0.973	**0.00049**	0.813
WPS+TCA	**0.011583**	0.601	**3.43E-05**	0.967	**0.000129**	0.893	**2.07E-05**	0.993	**2.35E-05**	0.987	**2.35E-05**	0.987	**2.67E-05**	0.98
WPS+NNFilter	**0.004676**	0.66	**0.011919**	0.59	**0.000145**	0.887	**7.14E-05**	0.927	**0.000441**	0.82	**4.97E-05**	0.947	**0.000674**	0.79
TCA+STrNN	***0.492871***	0.16	***0.10958***	0.373	***0.954431***	0.013	**0.030977**	0.503	***0.071846***	0.42	**0.03324**	0.507	***0.067422***	0.423
Ensemble														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	0.004676	0.663	**0.016395**	0.56	0.000102	0.907	***0.587228***	0.127	**0.011923**	0.587	***0.903168***	0.029	**0.016395**	0.56
allCPFP	2.67E-05	0.98	**2.07E-05**	0.993	***0.091785***	0.393	**0.000517**	0.81	**2.07E-05**	0.993	**2.70E-05**	1	**2.07E-05**	0.993
WPS+CPFP	***0.05932***	0.437	**5.61E-05**	0.94	***0.277605***	0.253	**0.000491**	0.813	**2.67E-05**	0.983	**5.94E-05**	0.957	**4.67E-05**	0.95
WPS+TCA	***0.345754***	0.217	**2.07E-05**	0.993	**0.029817**	0.541	**7.57E-05**	0.923	**1.82E-05**	1	**2.70E-05**	1	**1.82E-05**	1
WPS+NNFilter	***0.133595***	0.353	**6.33E-05**	0.933	***0.290348***	0.25	**0.00049**	0.813	**2.35E-05**	0.987	**9.90E-05**	0.928	**3.03E-05**	0.973
TCA+STrNN	***0.290397***	0.247	***0.265157***	0.26	***0.129934***	0.353	**8.04E-05**	0.92	***0.06745***	0.427	**0.000477**	0.874	***0.120863***	0.366
Average measures														
Models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance	
	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec	P-value	Ec
WPFP	0.015158	0.567	**0.00373**	0.677	0.007881	0.623	***0.33131***	0.227	**0.000606**	0.8	***0.055347***	0.453	**0.002235**	0.713
CPFP	6.77E-05	0.946	**1.82E-05**	1	***0.196111***	0.308	**0.000285**	0.847	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+CPFP	***0.078878***	0.413	**2.67E-05**	0.98	***0.097491***	0.387	**0.000162**	0.88	**2.07E-05**	0.993	**1.82E-05**	1	**2.07E-05**	0.993
WPS+TCA	**0.034491**	0.493	**1.82E-05**	1	**0.000318**	0.84	**3.43E-05**	0.967	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
WPS+NNFilter	***0.091768***	0.393	**2.35E-05**	0.987	***0.089098***	0.4	**0.000162**	0.88	**1.82E-05**	1	**1.82E-05**	1	**1.82E-05**	1
TCA+STrNN	***0.317286***	0.233	***0.219233***	0.287	***0.07272***	0.428	**0.000192**	0.87	***0.081359***	0.407	**0.000708**	0.793	***0.174714***	0.317

model, the FOR value should be lower and the other measure values should be high. Thus, the negative percentage for FOR and positive percentage for other measures indicate that WPSTC outperforms the competitive approaches.

To thoroughly examine WPSTC's performance and verify that it is effective in achiev-

Table 4.5: W(number of wins)/D(number of draws)/L(number of losses) statistics of each model compared to all the other models

fault prediction models	FOR		Recall		F-measure		AUC		G-mean		nMCC		Balance		Average	
	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank	W/D/L	Rank
WPFP+KNN	27/13/1	6	2/14/25	31	37/4/0	2	21/16/4	12	1/14/26	32	3/28/10	24	2/15/24	30	93/104/90	21
CPFP+KNN	25/13/3	10	3/8/30	38	21/14/6	8	8/12/21	26	3/8/30	37	2/15/24	34	3/8/30	38	65/78/144	30
(WPS+CPFP)+KNN	26/12/3	8	3/10/28	32	18/18/5	9	8/11/22	27	5/9/27	31	7/14/20	29	3/11/27	32	70/85/132	29
(WPS+TCA)+KNN	25/13/3	10	3/9/29	36	14/22/5	11	4/11/26	34	3/9/29	34	2/13/26	36	3/9/29	34	54/86/147	37
(WPS+NNFilter)+KNN	26/11/4	10	3/10/28	32	12/22/7	18	8/11/22	27	3/9/29	34	2/16/23	30	3/9/29	34	57/88/142	33
(TCA+STrNN)+KNN	7/13/21	27	19/15/7	19	5/19/17	30	9/9/23	27	19/15/7	17	16/16/9	17	19/16/6	17	94/103/90	19
(WPSTC)+KNN	5/12/24	31	36/4/1	2	7/26/8	20	26/14/1	7	40/1/0	1	35/6/0	1	39/2/0	1	188/65/34	2
WPFP+LR	21/16/4	14	10/19/12	23	38/3/0	1	35/6/0	2	12/15/14	23	30/11/0	5	12/20/9	21	158/90/39	6
CPFP+LR	29/10/2	4	3/8/30	38	28/9/4	6	23/14/4	9	3/9/29	34	6/19/16	26	3/8/30	38	95/77/115	26
(WPS+CPFP)+LR	17/16/8	18	10/10/21	28	5/30/6	20	19/13/9	19	10/9/22	28	13/16/12	23	10/10/21	28	84/104/99	25
(WPS+TCA)+LR	33/8/0	3	0/4/37	40	14/22/5	11	20/15/6	16	0/4/37	41	0/1/40	41	0/4/37	40	67/58/162	38
(WPS+NNFilter)+LR	19/14/8	16	11/11/19	26	5/30/6	20	19/14/8	18	10/10/21	27	13/18/10	19	11/9/21	27	88/106/93	23
(TCA+STrNN)+LR	5/10/26	34	23/14/4	9	5/15/21	35	12/18/11	22	25/13/3	8	17/17/7	16	23/15/3	10	110/102/75	14
(WPSTC)+LR	11/13/17	24	21/18/2	9	13/22/6	17	35/6/0	2	24/16/1	7	31/10/0	4	23/17/1	7	158/102/27	4
WPFP+NB	1/17/23	36	23/17/1	7	2/30/9	28	25/12/4	8	20/20/1	13	21/20/0	8	21/19/1	10	113/135/39	9
CPFP+NB	21/10/10	16	11/9/21	27	16/18/7	11	23/13/5	11	13/9/19	26	14/15/12	22	11/10/20	26	109/84/94	18
(WPS+CPFP)+NB	5/15/21	29	22/16/3	9	5/19/17	30	23/14/4	9	24/14/3	10	20/16/5	12	22/17/2	10	121/111/55	11
(WPS+TCA)+NB	0/0/41	42	41/0/0	1	0/0/41	42	7/10/24	32	22/16/3	13	6/17/18	28	23/16/2	8	99/59/129	28
(WPS+NNFilter)+NB	5/12/24	31	23/16/2	8	5/19/17	30	21/15/5	13	25/15/1	6	23/16/2	8	25/15/1	6	127/108/52	8
(TCA+STrNN)+NB	1/12/28	37	29/11/1	4	5/14/22	36	18/12/11	20	30/10/1	4	23/13/5	11	31/10/0	3	137/82/68	10
(WPSTC)+NB	5/15/21	29	32/8/1	3	5/28/8	23	28/11/2	6	33/8/0	2	32/9/0	3	33/8/0	2	168/87/32	3
WPFP+SVM	27/14/0	4	0/16/25	32	37/4/0	2	19/19/3	13	1/14/26	32	2/30/9	24	0/17/24	32	86/114/87	22
CPFP+SVM	38/3/0	1	0/3/38	41	32/8/1	5	4/5/32	36	1/3/37	40	2/12/27	38	0/3/38	41	77/37/173	39
(WPS+CPFP)+SVM	28/10/3	7	3/9/29	36	15/20/6	11	7/12/22	31	3/8/30	37	2/15/24	34	3/9/29	34	61/83/143	31
(WPS+TCA)+SVM	37/4/0	2	0/3/38	41	13/24/4	11	5/11/25	33	0/1/40	42	0/1/40	41	0/3/38	41	5/47/185	41
(WPS+NNFilter)+SVM	26/12/3	8	3/10/28	32	10/24/7	19	8/11/22	27	3/8/30	37	2/13/26	36	3/9/29	34	55/87/145	35
(TCA+STrNN)+SVM	9/14/18	25	20/15/6	17	9/20/12	23	10/9/22	25	21/14/6	16	21/14/6	12	20/15/6	16	110/101/76	15
(WPSTC)+SVM	17/8/16	20	20/15/6	17	15/22/4	10	28/12/1	5	23/15/3	12	31/9/1	5	21/15/5	15	155/96/36	6
WPFP+DT	3/21/17	27	19/21/1	12	5/23/13	29	4/5/32	36	22/18/1	10	17/21/3	14	22/18/1	8	92/127/68	16
CPFP+DT	1/6/34	38	15/16/10	20	1/3/37	41	0/3/38	41	15/11/15	21	2/12/27	38	15/11/15	23	49/62/176	40
(WPS+CPFP)+DT	1/5/35	41	20/17/4	15	1/4/36	38	1/2/38	39	20/13/8	17	3/14/24	30	20/14/7	17	66/69/152	34
(WPS+TCA)+DT	1/6/34	38	15/7/19	24	1/4/36	38	0/1/40	42	14/8/19	25	2/9/30	40	14/8/19	24	47/43/197	42
(WPS+NNFilter)+DT	1/6/34	38	20/17/4	15	1/4/36	38	1/2/38	39	19/11/11	19	3/14/24	30	19/14/8	19	64/68/155	36
(TCA+STrNN)+DT	6/9/26	33	22/15/4	12	5/14/22	36	4/3/34	38	22/14/5	15	16/12/13	19	21/17/3	14	96/84/107	24
(WPSTC)+DT	4/12/25	34	27/12/2	5	5/16/20	33	5/6/30	35	26/14/1	5	24/14/3	8	27/13/1	5	118/87/82	13
WPFP+Ensemble	21/15/5	15	13/18/10	21	37/4/0	2	35/6/0	2	14/19/8	20	29/12/0	7	13/19/9	20	162/93/32	5
CPFP+Ensemble	25/13/3	10	5/9/27	30	25/12/4	7	18/11/12	21	5/10/26	30	11/9/21	26	5/9/27	30	94/73/120	27
(WPS+CPFP)+Ensemble	15/13/13	19	14/8/19	25	7/24/10	23	20/16/5	15	15/8/18	24	15/14/12	19	13/10/18	24	99/93/95	19
(WPS+TCA)+Ensemble	13/15/13	22	10/10/21	28	5/16/20	33	11/11/19	24	8/11/22	29	3/14/24	30	9/10/22	29	59/87/141	31
(WPS+NNFilter)+Ensemble	14/14/13	20	16/8/17	22	7/24/10	23	20/15/6	16	15/11/15	21	15/17/9	18	16/10/15	22	103/99/85	17
(TCA+STrNN)+Ensemble	8/13/20	26	21/16/4	14	8/20/13	27	17/8/16	22	25/13/3	8	20/14/7	15	22/16/3	13	121/100/66	12
(WPSTC)+Ensemble	12/12/17	23	25/14/2	6	13/24/4	11	38/3/0	1	31/9/1	3	33/8/0	2	28/12/1	4	180/82/25	1

ing enhanced (significant) performance measures, we performed a WSR test of WPSTC against the competitive CPFP approaches with individual classifier results and average results of classifiers, Table 4.4 presents these results. We also present the win, draw, and loss results of WPSTC against those of WPFP, CPFP, WPS+CPFP, WPS+TCA, WPS+NNFilter, and TCA+STrNN per each performance measure across all 24 projects in Table 4.5. From the Table 4.4, we can observe the following:

1. WPSTC is statistically superior or similar to TCA+STrNN with individual classifiers for all measures. On average, WPSTC is statistically superior to TCA+STrNN over AUC and nMCC with P-values of 0.000192 and 0.000708 and high effect sizes of 0.87 and 0.793, respectively. On average, P-values of 0.317286, 0.219233, 0.07272, 0.081359, and 0.174714 are statistically similar in terms of FOR, Recall, F-measure, G-mean, and Balance with medium effect sizes. TCA+STrNN can somehow handle the imbalance issue, but overall performance measures like AUC and nMCC are unable to succeed over our approach.

2. For all the measures except FOR and F-measure, WPSTC is statistically superior to WPS+NNFilter for individual classifier generated results. On average, with P-values of 0.0000235, 0.0000162, 0.0000182, 0.0000182 and 0.0000182 and with high effect sizes of 0.987, 0.88, 1, 1 and 1, WPSTC statistically outperforms the NNFilter in terms of Recall, AUC, G-mean, nMCC and Balance, respectively. Possibly because our method maintains both class count ratios and finds the similarly distributed data through source project selection and also removes irrelevant features.

3. In terms of FOR, our approach was unable to outperform the WPS+TCA with KNN, LR and SVM. Our approach is statistically superior except with FOR, regarding all the other measures. On average, with P-values of 0.034491, 0.0000182, 0.000318, 0.0000342, 0.0000182, 0.0000182 and 0.0000182 and with large effect sizes of 0.493, 1, 0.84, 0.967, 1, 1 and 1, WPSTC statistically outperforms the WPS+TCA in terms of all the measures.

4. In terms of Recall, AUC, G-mean, nMCC and Balance, our approach outperforms the allCPFP and WPS+CPFP with large effect sizes. Regarding F-measure, WPSTC is statistically similar to allCPFP and WPS+CPFP with medium effect sizes. Merely with FOR, our approach was unable to outperform. allCPFP gets lower FOR values but at the cost of degradation of the faulty module prediction rate. Our approach gets a little higher FOR values than the allCPFP, but it doesn't cost faulty modules prediction rate. our model manages the balance between FOR and Recall rate. As we can see, the Balance measure is much higher for WPSTC and other overall performance indicators are also much better than the competitive cross-project prediction models.

Table 4.5 shows the ranks obtained by each fault prediction method combined with a classifier. Each model Compared with the other 41 models and reported the number of wins, losses, and draws, the rank is determined by the win-loss count. We can observe from Table 4.5, the ranks obtained by our approach for FOR are 31, 24, 29, 20,34 and 23; for Recall are 2, 9, 3, 17, 5 and 6; for F-measure are 20,17, 23,10, 33 and 11; for AUC are 7, 2, 6, 5, 35 and 1; for G-mean are 1, 7, 2, 12, 5 and 3; for nMCC are 1, 4, 3, 5, 8 and 2; for Balance are 1, 7, 2, 15, 5 and 4 combined with KNN, LR, NB, SVM, DT and ensemble classifiers, respectively. Except for F-measure and FOR, WPSTC is supposed to be in the highest top 7 ranks. The FOR and F-measure ranks are not much better than WPFP but comparable with other cross-project prediction models. Moreover, WPSTC ranks higher with FOR and F-measure than the most recent approach, TCA+STrNN. Based on single individual measures, we can not justify the superiority of a particular model, so we consider the average wins, draws, and losses of all the measures and assign a rank for them. The ranks obtained by our approach combined with KNN, LR, NB, SVM, DT, and ensemble classifiers are 2, 4, 3, 6, 13, and 1, respectively. Based on the individual measure ranks and average ranks, we can conclude that WPSTC combined with ensemble, KNN, and NB performs better than the other classifiers. The DT with WPSTC rank is 13, which is less compared to other classifiers, but DT with other fault prediction models ranks much less than our approach.

Although a few projects FOR values are not better than some other approaches, faulty modules are still more important than non-faulty modules in software testing. The results further indicate that WPSTC can handle the imbalance issue in cross-project prediction by improving the Recall values without deteriorating the FOR values, this can be viewed by the Balance measure. The Balance measure is about maintaining faulty and non-faulty module recognition rates equally well, so WPSTC obtained Balance values are much better than the other fault prediction models. The overall performance assessment measures G-mean, AUC, Balance, and nMCC values are much better for WPSTC than the other fault prediction models and statistically outperforms all the other methods. We can conclude that the overall performance of faulty and non-faulty module prediction rates is better.

4.4 Discussion of Results

4.4.1 Why our proposed WPSTC's overall performance is better than the other cross-project prediction models?

The distribution difference between historical data and target data is the main issue in CPFP [23, 25]. Furthermore, the majority of historical software projects are imbalanced in nature [25, 166] and also plagued by the curse of dimensionality issues [43, 167]. Most of the existing fault prediction models concentrate on one of the issues only. When a classification model uses a large amount of irrelevant data from all the available projects for training, it may degrade the overall prediction performance. We carefully selected similar source projects through the WSR test to reduce the distribution difference between testing (target project data) and training (selected source projects data) data instances through an instance filtering technique. The overall performance of a binary prediction model depends on the perfect recognition of positive and negative samples. This method trained the model with similarly distributed data, which can increase the recognition rate of both classes. Numerous software defect datasets have a low proportion of faulty instances and a high proportion of faulty-free instances, which leads to exceptionally low performance on faulty class data. While performing instance filtering, WPSTC uses SMOTE [11] to generate faulty modules, so the Balance measure gives good results over our approach. We employed the Binary-RAO algorithm to select relevant feature subsets. Our proposed WPSTC performs better because we trained the classification model on selected training data with selected features to maximize prediction performance.

4.4.2 Does the choice of classifiers affect the WPSTC performance?

As reported in Section 4.2.3, we employed six classification models, including KNN, NB, LR, DT, SVM, and ensemble learner classifiers. Over 24 projects, we examined the influence of these classification models on WPSTC and the other fault prediction models. On the basis of overall experimental results, we can conclude that WPSTC combined with ensemble learner, KNN, and NB performs best. Despite a little discrepancy in a few measures, WPSTC with LR, SVM, and DT yields the best possible results. An ensemble classifier is more robust towards overfitting. The ensemble method can use each learner's (classifier) probability over predicted classes and predict the most appropriate label for test data. In our approach, we selected training data instances based on the nearest neighbours of testing data instances. On the contrary, the KNN classifier performs prediction on test data by selecting closer instances in training data, so KNN performance is better with our approach. The NB classifier gives equivalent weight to each feature, so the irrelevant features may bring down the NB prediction performance. NB performs better with WPSTC because it removes irrelevant and redundant features through a feature subset selection technique. Therefore, ensemble, KNN, and NB classifiers are suitable for building a prediction model on training data extracted by WPSTC.

4.5 Threats to Validity

In this section, we review the potential threats that may influence the validity of this study's experimental findings. The selected feature subsets for each project in our approach might not produce identical subsets and results for the next time the code is run. This could be an internal threat to our study. NNFilter [29] and STrNN [25] are implemented carefully and classification models use default parameters, which can limit the internal threat to our model. The proposed approach performed effectively on publicly available projects, but we cannot claim the generalizability to other commercial projects, we provided a detailed explanation of algorithm it can be used for other projects to limit external threats. For outcomes comparison, the WSR statistical test and WDL comparisons are executed, so the conclusion threat has a minimal effect on our experimental findings.

4.6 Summary

In this study, we proposed a two-phase WPSTC model for CPFP. WPSTC first develops a source project selection technique utilizing the concept called WSR statistical test to choose similar source projects for each target project using four (mean, median, max, and standard deviation) statistical properties of projects. The next phase is optimized training data construction, where the model selects similarly distributed training data from the source projects followed by selecting relevant feature subsets from the filtered training data. We build six classification models on filtered training data with selected feature sets and make predictions on target project data with the same feature sets. We evaluate our WPSTC model on 24 projects and compare it with WPFP, allCPFP, WPS+CPFP, WPS+TCA, WPS+NNFilter, TCA+STrNN in terms of FOR, Recall, micro-averaged F-measure, G-mean, AUC, nMCC and Balance. Experimental results show that the proposed approach outperforms the competitive CPFP models in terms of all measures except FOR and F-measure. In terms of FOR and F-measure, our approach gives comparable results to WPS+CPFP, allCPFP, and WPFP. Further, the WDL statistics show that our approach with ensemble learner, KNN, and NB gives better outcomes than the other classifiers. The next chapter presents an improved source project selection using an applicability score and addresses the imbalance and high-dimensionality issues in cross-project fault prediction.

Chapter 5

A Cross-project Fault Prediction through Applicability based Source Project Selection

Earlier studies put forward alternative solutions for cross-project prediction that could overcome the limitations of CPFP models. However, existing CPFP models are unstable, and the variety of source projects greatly impacts their performance. As discussed in Chapter 4, we assume that similarly distributed source projects might be applicable to the target. In this study, as an extension to the work reported in Chapter 4, we proposed a three-fold SRES model for CPFP, which selects sources using similarity and applicability scores, following imbalance and feature learning methods.

A well-known industry example of source project selection and cross-project fault prediction is the Apache Software Foundation (ASF). ASF maintains hundreds of open-source projects. Fault prediction models have been applied using data from similar Apache projects to predict which parts of new projects are likely to contain faults. For example, faulty data from the Apache Tomcat web server could be used to predict potential defects in newer web-based projects with similar architectures. By selecting the right source projects, then choosing appropriate balanced data and extracting informative features from source projects of Apache can improve the accuracy of their fault prediction models and focus testing resources on the areas of new projects that are most at risk of bugs. Benefits of source project selection in industry are improved prediction accuracy, resource optimization, and applicability across domains.

The remainder of this Chapter is organized as follows: The proposed approach is explained in Section 5.1. The experimental settings are described in Section 5.2, while Section 5.3 presents the experimental result analysis along with a discussion, and Section 5.4 discusses the threats to the validity of our study. Finally, Section 5.5 concludes the summary of the Chapter.

5.1 SRES Approach for CPFP

We proposed a three-fold SRES model for CPFP. To minimize the distribution gap between source and target projects, the SRES model develops the SAPS (Similarity and Applicability based Source Project Selection) method in the first stage. The second phase performs the proposed resampling technique over combined selected source projects data to overcome the issue of imbalanced data and the distribution gap. Finally, to address the high-dimensionality problem, the third stage employs an efficient Stacked Autoencoder (SAE) model for feature reduction. Figure 5.1 depicts the overall framework of this study. The implementation phases are explained more particularly in subsequent subsections along with Algorithms 6, 7, 8 and 9, respectively.

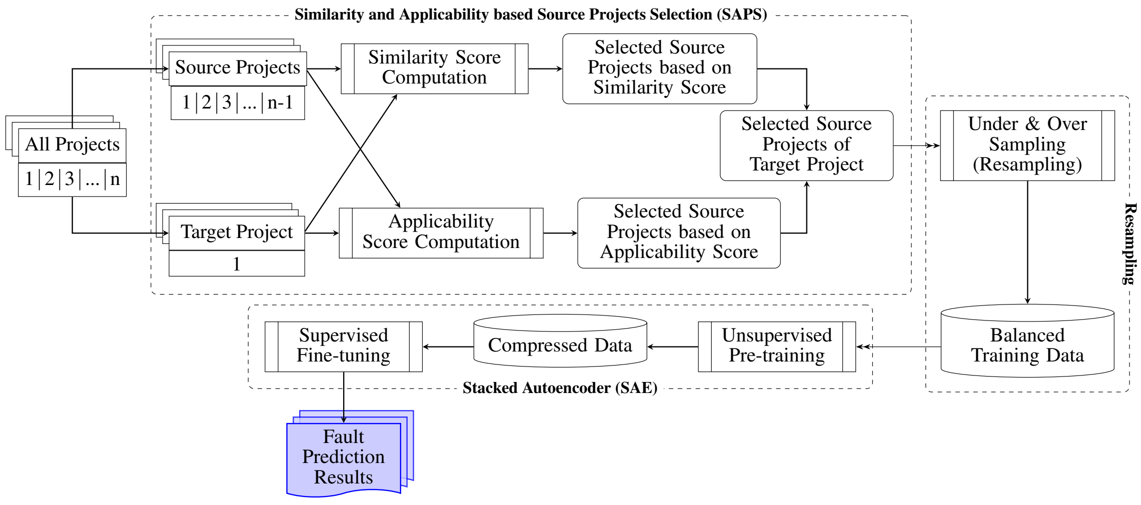

Similarity and Applicability based Source Projects Selection (SAPS)
All Projects
1|2|3|...|n
Source Projects
1|2|3|...|n-1
Target Project
1
Similarity Score Computation
Applicability Score Computation
Selected Source Projects based on Similarity Score
Selected Source Projects based on Applicability Score
Selected Source Projects of Target Project
Under & Over Sampling (Resampling)
Resampling
Balanced Training Data
Unsupervised Pre-training
Compressed Data
Supervised Fine-tuning
Stacked Autoencoder (SAE)
Fault Prediction Results

Figure 5.1: An overview of SRES model framework

Algorithm 6: SRES model framework

```
  Require: Target project without labels ($D_T$) and set of available projects with
           labels ($P$);
  Ensure: Target project labels;
1 $Sim\_scores$ ← SIMILARITY_SCORE ($D_T$, $P$)
2 $App\_scores$ ← APPLICABILITY_SCORE ($D_T$, $P$)
3 $Sim\_sources$ ← Set of projects having $Sim\_score$ greater than the
    Average($Sim\_scores$)
4 $App\_sources$ ← Set of projects having $App\_score$ greater than the
    Average($App\_scores$)
  /* Selected source projects of $D_T$ */
5 $Source_{D_T}$ ← Intersection of $Sim\_sources$ and $App\_sources$
6 $Balanced\_train\_data$ ← RESAMPLING ($Source_{D_T}$,$D_T$)
7 $Compressed\_train\_data$, $Compressed\_test\_data$ ← SAE performs the
    unsupervised pre-training over $Balanced\_train\_data$ and $D_T$
8 test_data labels ← SAE performs the supervised fine-tuning over
    $Compressed\_train\_data$ and $Compressed\_test\_data$
9 return test_data labels
```

5.1.1 Similarity and Applicability based Source Projects Selection (SAPS)

In CPFP approaches, the performance of a prediction model over the target project may differ when using various source projects as a training set. According to Zhou et al. [168], training a prediction model with suitable source projects may enhance the performance of current CPFP approaches, so it is crucial and essential to design an automatic source projects selection technique for a specific target. According to He et al. [33], a correlation exists between the distributional characteristics of feature data and the suitability of the source project data. Herbold [36] suggested that the CPFP models trained on closely distributed source project data can enhance the prediction performance over the target. Accordingly, using similarity and applicability scores, the proposed SRES model chooses source projects with similar data distribution and higher applicability to the target. SAPS includes two stages: similarity score computation and applicability score computation. These two stages are thoroughly explained in the following two subsections.

Algorithm 7: Similarity Score Computation

```
Require: $D_T$, $P$;
Ensure: $Sim\_score_{D_T}$;
function SIMILARITY_SCORE($D_T, P$)
    Initialize: $m \leftarrow \#features$ in $D_T$
                $n \leftarrow \#projects$ in $P$
                $[Sim\_score_{D_T}]_{1\times n} \leftarrow$ NULL
    for each project $D_S$ in $P$ do
        $Dist \leftarrow 0$
        $Count \leftarrow 0$
        $T_1 \leftarrow Mean(D_T)$
        $S_1 \leftarrow Mean(D_S)$
        $T_2 \leftarrow Mode(D_T)$
        $S_2 \leftarrow Mode(D_S)$
        $T_3 \leftarrow Median(D_T)$
        $S_3 \leftarrow Median(D_S)$
        $T_4 \leftarrow Standard\_deviation(D_T)$
        $S_4 \leftarrow Standard\_deviation(D_S)$
        $T_5 \leftarrow Maximum(D_T) - Minimum(D_T)$
        $S_5 \leftarrow Maximum(D_S) - Minimum(D_S)$
        for $i$ = 1 to 5 do
            $P-value \leftarrow$ WSRTest($T_i, S_i$)
            if $P-value \geq \alpha$ then
                $Count \leftarrow Count + 1$
                $Effect$ = EffectSize($T_i, S_i$)
                $Dist \leftarrow Dist + Effect$
            end
        end
        if $Count > 0$ then
            Update: $Sim\_score_{D_T}[D_S]$.append $(1 - \frac{Dist}{Count})$
        else
            Update: $Sim\_score_{D_T}[D_S]$.append (0)
        end
    end
    return $Sim\_score_{D_T}$
end
```

5.1.1.1 Similarity Score Computation

The similarity relationships between the specific target project and all the available projects are explored, and the corresponding similarity scores are calculated by comparing the below-mentioned five distributional characteristic measures. Algorithm 7 demonstrates the detailed process of obtain-ing similarity scores.

Distributional Measures: Different distributional characteristic measures can uniquely represent each project since they can quantify both the similarities and differences between projects. We choose the following five distributional characteristics to measure the similarity between projects:

1. **Mean:** Mean signifies the average of a set of values and is derived by dividing the sum of all the elements in a set by the total number of elements. For example consider a set $P = \{p_1, p_2, p_3, \ldots, p_k\}$ and length (total number elements) of P is 'k', now the $Mean(P)$ can be calculated using Eq. 5.1,

$$Mean(P) = \frac{1}{k}\sum_{i=1}^{k} p_i = \frac{p_1 + p_2 + p_3 + \cdots + p_k}{k} \tag{5.1}$$

2. **Mode:** Mode is a value with the highest frequency in a specific dataset, which means it appears most of the time compared to other values in the dataset.

3. **Median:** Median is a value that falls in the middle of a specific dataset when arranged in an ordered fashion, i.e., from smaller to larger values or vice versa. At least half of the values are greater than or equal to the median value, while the other half are less than or equal to the median value. The $Median(P)$ can be computed using Eq. 5.2,

$$Median(P) = \begin{cases} p_{\left[\frac{k+1}{2}\right]}, & \text{if } k \text{ is odd} \\ \frac{p_{\left[\frac{k}{2}\right]} + p_{\left[\frac{k}{2}+1\right]}}{2}, & \text{if } k \text{ is is even} \end{cases} \tag{5.2}$$

4. **Standard deviation:** The standard deviation measures how widely the data is distributed from the mean. The higher and lower standard deviation scores imply that the dataset values are more dispersed and grouped around the mean. The $Standard\_deviation(P)$ can be computed using Eq. 5.3,

$$Standard\_deviation(P) = \sqrt{\frac{\sum_{i=1}^{k}(p_i - \mu)^2}{k}} \tag{5.3}$$

Where p_i are individual values and μ is the $mean$ of P.

5. **Range:** Range can be defined as the difference between the greatest and least values in a

specific dataset.

The inputs used to calculate similarity scores are a set of available projects (P) and a specific target project (D_T). We compute a similarity score for each project ($D_S \in P$). As an initial step, we represent each project with its five distributional feature vectors; T1-T5 and S1-S5 are the feature vectors of the five distributional measures of D_T and D_S, respectively (lines 8-17 in Algorithm 7).

We consider two distributions to be relative if their difference is statistically significant and the effect size is small. We apply the WSR test [157] to compare the distribution of feature values between D_S and D_T data with 95% confidence level (i.e., significance level (α) =0.05). The WSR test evaluates the statistical significance of two paired distributions. When the WSR test returned P-value greater than α, there is evidence to confirm that both distributions are statistically relative. If the P-value is greater than 0.05, we calculate the matched-pairs rank biserial correlation coefficient value (Line 22 in Algorithm 7) as the effect size [159] to quantify the difference between the D_S and D_T for five individual distributional measures. The detailed computation of the WSR test and matched-pairs rank biserial correlation coefficient test is provided in [13, 20, 166]. The computed effect size represents the degree of difference between two sample distributions. It ranges from 0 (the difference between two distributions is low) to 1 (the difference between two distributions is high). If the P-value exceeds alpha, we update the $Count$ value by one and the distance ($Dist$) value by the computed effect size for the particular distributional measure. This means source and target projects are statistically relative with a confidence level of 95% (lines 18–25 in Algorithm 7) for particular distributional measures. If the $Count$ value is greater than zero after the completion of all five distributional measures, we then update the similarity score of the source project ($sim\_score_{D_T}[D_S]$), which indicates that at least one measure in the source project has to be statistically significant; otherwise, similarity will be zero (lines 26–30 in Algorithm 7). Finally, it returns the similarity scores of all source projects concerning a specific target project D_T.

5.1.1.2 Applicability Score Computation

According to [38], the applicability of various available source projects to target project prediction performance is highly inconsistent. So, in addition to similarity based selection, we also considered how well each available source project is applicable to the target project. According to [29], the model trained on its nearest neighbors can perform well on that particular project data. So, while computing the applicability score, we assume the model trained on the nearest neighbors of target

Algorithm 8: Applicability Score Computation

```
   Require: $D_T$, $P$;
   Ensure: $App\_score_{D_T}$;
1  function APPLICABILITY_SCORE($D_T, P$)
2      Initialize: m ← #features in $D_T$
3                  n ← #projects in $P$
4                  $[App\_score_{D_T}]_{1\times n}$ ← NULL
5      $Source\_data$ ← Union of all projects ($P$) data
       /* Calculating NN of each instance of $D_T$ in
          $Source\_data$                                      */
6      NN$_{D_T}$, $Count$ = Nearest_Neighbours ($D_T$, Source_data) // NN$_{D_T}$ stores
        Union of NN's and count stores the instance
        repeated count
7      Sort the instances in NN$_{D_T}$ in descending order according to their count
8      for each project $D_S$ in $P$ do
9          S_len = 30% of length($D_S$)
10         train_data = $D_S$
11         test_data = NN$_{D_T}$[S_len]
12         $Score$ = Classification_Model_Score (train_data, test_data)
13         Update: $App\_score_{D_T}[D_S]$.append ($Score$)
14     end
15     return $App\_score_{D_T}$
16 end
```

data in available projects might produce superior prediction performance over target data. Based on it, we compute the applicability score using Algorithm 8.

The inputs used to calculate the applicability scores are the same as similarity score computation, P and D_T.We first take the union of all the projects data into $Source\_data$ (line 3 in Algorithm 8). Then, it finds one nearest neighbor (NN) of each instance of D_T in $Source\_data$. The nearest neighbors set may contain redundant instances; therefore, we transform them into a unique set (NN$_{D_T}$) and keep a count of how frequently each instance occurs as NN. Based on the count value, we sort NN$_{D_T}$ in descending order (lines 6-7 in Algorithm 8). A portion of the generated NN data serves as testing data. A prediction model is trained on D_S to predict the labels of part of the NN$_{D_T}$ data and compute the AUC. The AUC score is the applicability score of a particular D_S to the target project. As per how much NN$_{D_T}$ data should be taken for testing, prior studies suggested that a prediction model trains with 70% and tests with 30% of the data give better performance [156]. So, we choose the size of test data as 30% of the instances in D_S. The model builds on each D_S, predicts the labels of the 30% NN$_{D_T}$ data, and then evaluates the model's applicability to the target project. The same holds for all available projects, and applicability scores ($App\_score_{D_T}[D_S]$) are updated (lines 8–14 in Algorithm 8). The algorithm returns the applicability scores of all source projects with respect to a specific target project D_T ($App\_score_{D_T}$).

From the aforementioned similarity and applicability scores, we derive the target's corresponding most similar and applicable projects, respectively. The scores of the projects greater than the average are regarded as the most appropriate projects. Finally, the relevant sources ($Source_{D_T}$) of a target project D_T computed by considering the intersection of similarity based sources and applicability based sources (lines 1 - 5 in Algorithm 6).

5.1.2 Resampling Source data

A skewed data distribution may reduce the prediction performance of a model. Bennin et al. [169] suggested that the resampling technique can mitigate class imbalance issues and enhance the performance of the prediction model. We use a combination of selected source projects ($Source_{D_T}$) to train the model, but that is imbalanced too, and even after source project selection, there is still some distribution gap between $Source_{D_T}$ and D_T. Turhan et al. [29] suggested that selecting similar instances of the target data lowers the distribution gap and improves the CPFP model's prediction ability. We introduce a novel data resampling method to alleviate the class imbalance and distribution gap issues by modifying the $Source_{D_T}$ data. Our resampling method is done in two ways: *Undersampling* is the process of keeping instances that are statistically comparable to the target data while eliminating instances that are not similar, and *oversampling* is the process of generating new synthetic non-faulty data that share values of both source and target data. The prediction model will now have an equal likelihood of predicting both the faulty and non-faulty modules and exhibit a positive influence on prediction performance after resampling. The step-by-step implementation details are described in Algorithm 9.

$Train\_data$ contains the integrated sources data (lines 5-7 in Algorithm 9). As we don't have labels for the D_T (test data), we assign pseudo-labels to the test data through unsupervised K-Means clustering. K-Means divides the D_T into two clusters: faulty instances set (min_test) and non-faulty instances set (maj_test) (line 8 in Algorithm 9). Furthermore, both undersampling and oversampling techniques utilize this pseudo-labeled data.

Undersampling: In order to perform undersampling, we split the $Train\_data$ into a faulty class set (min_train) and a non-faulty class set (maj_train). Each instance of maj_train statistically

Algorithm 9: Resampling Source data

```
   Require: $Source_{D_T}$, $D_T$;
   Ensure: balanced_train_data;
1  function RESAMPLING($Source_{D_T}, D_T$)
2      Initialize: $Train\_data \leftarrow$ NULL
3                  $New\_maj\_train \leftarrow$ NULL
4                  $New\_min\_train \leftarrow$ NULL
5      for each project $D_S$ in $Source_{D_T}$ do
6          $Train\_data$.Append($D_S$ data)
7      end
8      $maj\_test$, $min\_test$ = K-Means_clustering($D_T$)
       Undersampling:
9      $maj\_train$, $min\_train \leftarrow Train\_data$
10     $len\_maj \leftarrow$ length($maj\_train$)
11     [Count]$_{1 \times len\_maj} \leftarrow 0$
12     for i in range(len($maj\_train$)) do
13         for j in range(len($maj\_test$)) do
14             $P-value$ = WSR($maj\_train$[i], $maj\_test$[j])
15             if $P-value > \alpha$ then
16                 Count[i]=Count[i]+1
17             end
18         end
19     end
20     Sort $maj\_train$, according to their Count values
21     $New\_maj\_train \leftarrow$ Top half of the instances from sorted $maj\_train$
       Oversampling:
22     T $\leftarrow$ length ($New\_maj\_train$) - length ($min\_train$)
23     temp $\leftarrow$ 0
24     while temp < T do
25         $min\_inc1 \leftarrow$ random instance from $min\_train$
26         $min\_inc2 \leftarrow$ random instance from $min\_test$
27         $New\_min\_train$.append(rand[0,1] $\times$ ($min\_inc1$ + $min\_inc2$) )
28         temp $\leftarrow$ temp + 1
29     end
30     $New\_min\_train$.Append ($min\_train$)
31     balanced_train_data $\leftarrow New\_maj\_train \cup New\_min\_train$
32     return balanced_train_data
33 end
```

compared to maj_test data using the WSR test. If the computed P-value is higher than the significance value ($alpha$), we increase the maj_train instance count value by one. Then, We sort the instances of maj_train descending order based on the count value and select the top half of the instances as under-sampled data (New_maj_train) (lines 9-21 in Algorithm 9). Thus, the under-sampled data is statistically similar to the target data.

Oversampling: Now, to make the ratio of two classes equal, we need to generate a 'T' number of synthetic instances. Where T is the number of instances in the New_maj_train minus the number of instances in min_train. We randomly choose an instance from min_train and another from min_test. Synthetic data is generated by multiplying a random number between 0 and 1 with the sum of those randomly selected instances. Thus, the generated over-sampled data shares both target and source distributions. The process repeats until the number of generated synthetic instances becomes T (lines 22-29 in Algorithm 9).

Finally, the resampling technique returns the balanced training data ($Balanced\_train\_data$), which is a combination of under-sampled non-faulty instances, over-sampled faulty instances, and actual faulty instances. Thereby, the distribution gap between the training data and target data is reduced, and the resampled training data have an equal ratio of both classes.

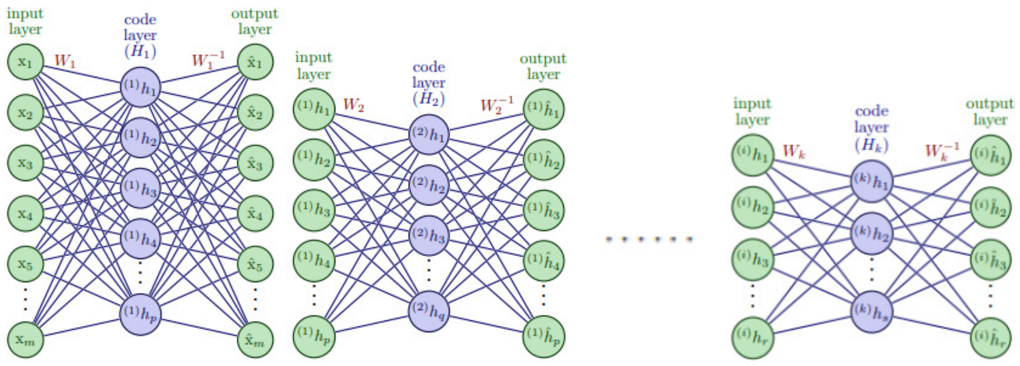
5.1.3 Unsupervised Feature Reduction and Supervised Prediction via Stacked Autoencoder

According to Kondo et al. [170], unsupervised feature reduction techniques perform better than supervised techniques in fault prediction models. Accordingly, we employ SAE to perform unsupervised feature reduction in our study to mitigate the issue of the curse of dimensionality. A basic autoencoder has a symmetrical structure composed of an encoding phase and a decoding phase. For a particular instance of training data $X = \{x_1, x_2, \ldots, x_m\}$, the goal of training is to approximate $\hat{X} \approx X$. Where $\hat{X}$ is the reconstructed output representation of the input. For a single-layer autoencoder, the encoding and decoding are done by following Eqs. 5.4 and 5.5.

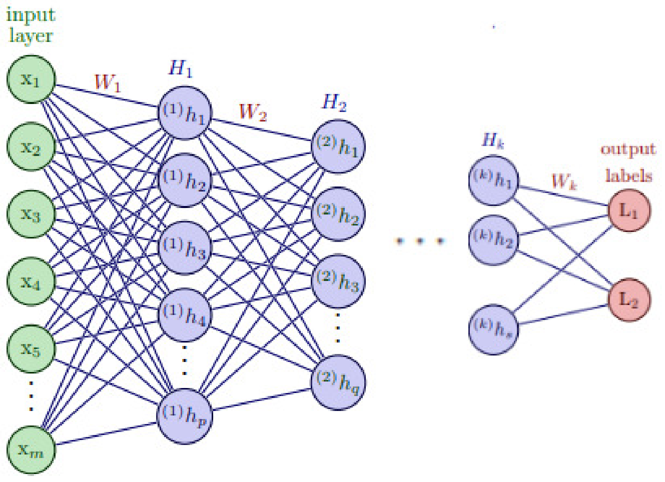
$$H : H(X) = f(W * X + b) \tag{5.4}$$

$$\hat{X} = f(W^{-1} * H(X) + b^{-1}) \tag{5.5}$$

Where 'X', 'H', and '$\hat{X}$' are autoencoder input, hidden, and output data. W and b are the encoder's weight and bias parameters, while 'f' is a nonlinear activation function. The decoder's weight (W^{-1}) and bias (b^{-1}) parameters are the inverse representations of W and b, respectively. All these parameters are optimized by minimizing the error ($J(\Theta)$) between X and $\hat{X}$ as shown in



(a) Feature reduction through unsupervised pre-training



(b) Predicting overall performance through supervised fine-tuning

Figure 5.2: Stacked Autoencoder architecture a) Feature reduction through unsupervised pre-training b) Predicting overall performance through supervised fine-tuning

Eqs. 5.6 and 5.7 .

$$MSE(\Theta) = \frac{1}{n}\sum_{i=1}^{n}(\parallel \hat{x_i} - x_i)^2 \parallel) \tag{5.6}$$

$$J(\Theta) = MSE(\Theta) + \lambda \sum_{i=1}^{n}(\Theta)^2 \tag{5.7}$$

Where $\Theta = \{W, b, W^{-1}, b^{-1}\}$), 'n' denotes the total number of instances in a dataset and $MSE(\Theta)$ is mean squared error represents the loss function. The loss function added by a weight attenuation term (L2-regularization) controls weight reduction.

As we can see from Figure 5.2, the whole training process of the SAE model includes two steps: unsupervised pre-training and supervised fine-tuning.

The above-generated balanced training data and the target project's testing data are fed into stacked autoencoders to extract deep representations of actual software features without losing the original information through unsupervised pre-training. Then, the training data characterized by a reduced feature set along with labels is used to train the SAE through supervised fine-tuning. Next, we use testing data characterized by a reduced feature set to evaluate trained SAE.

The balanced training data have the 'n' number of instances and the 'm' number of features. In Figure 5.2a, the first autoencoder receives the instances with 'm' number of features without labels as input, then encodes it to 'p' number of reduced representations of those features, and the decoder attempts to reconstruct the original input with that of reduced data by back-propagating the mean squared error. Then, the reduced representation from the first autoencoder serves as input for the second autoencoder. The second autoencoder's input layer takes the previously encoded data as input and encodes it into a more abstract form of reduced representation with a 'q' number of dimensions. The second autoencoder's decoder then attempts to reconstruct the original input served in the first layer of the second autoencoder with that of reduced data by back-propagating mean squared error. Similarly, each autoencoder trains independently until the hidden representation of the data is compelling enough, and the sequence of autoencoders should have the node count of the hidden layer in decreasing order. It results in compressed data with fewer dimensions and produces optimized parameters (i.e., weights and biases).

The figure 5.2b illustrates that SAE undergoes a supervised fine-tuning process. This involves adding a softmax classifier with two classes (faulty and non-faulty) on top of the unsupervised pre-trained network. The hidden layers $\{H_1, H_2, ..., H_k\}$ are utilized in the encoding process as input parameters for the softmax classifier. The network is then trained with labelled data to optimize parameters, propagating through subsequent layers towards the softmax layer. With labels available, supervised fine-tuning occurs by back-propagating cross-entropy loss and L2-regularizer. The objective is to minimize loss and optimize parameters using labelled training data. After unsupervised pre-training and supervised fine-tuning, the SAE model trains with balanced training data.

The trained network model validates the model with test data. The network structure and parameters (for example, the number of nodes in the hidden layers, the number of epochs, and the batch size of each epoch) are adjusted until the desired prediction performance is reached, varying across datasets. The number of nodes in each hidden layer, the number of epochs, and the batch size are different for each project. However, most of the projects performed effectively when the number of dimensions reduces to 20–30% of their original data. When the optimal parameters finalize for a project, the SAE model outputs the labels of target data. We then compute the performance of the proposed SRES model based on the SAE returned labels and the actual labels of target data.

5.2 Experimental Setup

As part of this study, we collected a few publicly accessible software fault projects and then chose the fault prediction models. A few performance assessment measures are collected and evaluated using statistical tests to study the performance of our SRES model. Then, we compare the outcome of our proposed approach with a few base models.

5.2.1 Experimental Objects

We chose 24 projects from two open-source datasets to experimentally validate the SRES model. Among them, 14 projects are extracted from the PROMISE repository [34], and the other ten projects are extracted from the NASA MDP dataset [70], in which both have been extensively used in various empirical CPFP studies [25, 26, 29, 33, 38, 116]. As listed in Table 3.1 in Chapter 3, each project from the PROMISE repository has 20 and NASA MDP has 36 to 39 features, respectively. NASA projects don't have the same feature, so we use all 35 shared features to perform homogeneous CPFP. Along with those independent features, one label feature is there for each project, which represents whether the module is faulty or non-faulty. We purposefully extracted single versions of different open-source software projects during the experiments since they have distinct class distributions. These datasets have an imbalanced ratio of 2.15% to 46.67%. This is performed to verify the experiments' generalizability because the employed datasets have different distributions and consist of significantly varying imbalance ratios. For each experiment, one project served as the target project, and the rest served as source projects for PROMISE and NASA datasets separately.

5.2.2 Prediction Models

Our proposed SRES CPFP model is compared with base models in two variants. We first compare the SAPS experimental outcome with the existing source project selection models, where we employ five well-known and extensively used conventional classification models as fault prediction models, including Random Forest (RF), Logistic Regression (LR), Naive Bayes (NB), Support Vector Machine (SVM) and Decision Tree (DT) classifiers [25, 32, 36, 37, 39, 116, 117]. While building a fault prediction model, each classification model has unique benefits. Therefore, we consider the aggregated performance of all five prediction models for the experimental comparison. The performance of the proposed SRES model can be predicted using the finely measured SAE model parameters.

5.2.3 Performance Assessment and Statistical Measures

We employ two sets of evaluation indicators to assess the performance of our proposed SRES model against existing baseline models. One set of measures is used to determine the fault-prone modules accurately. As suggested by Turhan et al. [29] and Menzies et al. [35], Recall and Fall Out Rate (FOR) are more reliable measures for assessing the majority and minority classes in severely unbalanced datasets, respectively. The other set includes four extensively used Balance, AUC, G-mean, and nMCC measures to assess the overall performance of fault prediction models [24, 26, 31, 116, 171]. These measures are extracted from the confusion matrix, built from the actual and predicted labels, as shown in Table 3.1 in Chapter 3.

$$FOR = \frac{FP}{FP + TN}$$

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

$$Balance = 1 - \frac{\sqrt{(0 - FOR)^2 + (1 - Recall)^2}}{\sqrt{2}}$$

$$G - mean = \sqrt{Acc_+ * ACC_-} = \sqrt{Recall * (1 - FOR)}$$

$$MCC = \frac{TP.TN - FP.FN}{\sqrt{(TP + FP)(TN + FN)(TP + FN)(FP + TN)}}$$

$$nMCC = \frac{MCC + 1}{2}$$

AUC is a performance metric that measures the tradeoff between FOR and Recall. All the measures range between [0, 1]. For all measures except FOR, the higher the values, the better the classifier's performance. As per FOR, a lower value indicates better performance.

To analyze the proposed model's experimental outcomes and demonstrate its superiority, we choose the WSR test as a statistical comparison measure [37, 157, 163]. In this study, the null hypothesis (H_0) is defined as two populations being compared are statistically similar to one another, and the assumption is that H_0 is valid under a confidence level of 95% (significance level $\alpha = 0.05$). The matched-pairs rank biserial correlation coefficient measure finds the strength of the relationship (effect size) [158, 159]. Furthermore, to ensure our proposed model superiority, we compare each base model and proposed model with all the measures using the widely popular Win-Draw-Loss (WDL) comparison technique.

5.3 Experimental Results and Discussion

In this section, we analyze the prediction performance of the proposed SRES model by comparing it with a few earlier cross-project prediction studies. We first compare the proposed SRES CPFP model with four various types of CPFP methods, including NNFilter [29], TCA [22], TNB [116], and TSFA [117]. The SRES model is compared with the basic WPFP model too. Further, the SAPS model is compared with three different source project selection methods, including TDS [36], CFPS [38], and CAMEL [39]. The SAPS model is also compared with All_CPFP. All_CPFP is a basic cross-project prediction, a specific target project serves as testing data, and the combination of the rest of the projects serves as training data for the prediction model. The following subsections describe the performance of our model through the findings of the following four research questions.

5.3.1 RQ1: How much improvement is there in the SAPS based source projects selection model over state-of-the-art source projects selection models?

Applying various SPS models to different software projects results in varying outcomes. The SAPS method is a part of the proposed SRES model, which selects the appropriate source projects in the first phase of our SRES model. RQ1 discusses how preferable our proposed SAPS is over the three existing SPS models and the all-projects CPFP model. To demonstrate how effectively SAPS selected source projects improve CPFP performance, we compared SAPS with three base SPS models and a basic CPFP model including TDS, CFPS, CAMEL, and All_CPFP. Figure 5.3 depicts the box plots of the six performance measures on the SPS prediction models. Table 5.1 provides the average, minimum, and maximum values of SPS models with respect to individual performance measures across 24 projects. The results of the WSR test and effect size differences between the proposed SAPS model and the base SPS models are outlined in Table 5.2.

Table 5.1: The Average, Minimum, and Maximum values of SPS based CPFP models across all 24 projects

Methods	FOR	Recall	Balance	AUC	G-mean	MCC
	Average					
All_CPFP	0.037	0.144	0.393	0.685	0.307	0.574
TDS	**0.011**	0.063	0.337	0.624	0.201	0.562
CFPS	0.043	0.142	0.386	0.693	0.273	0.566
CAMEL	0.032	0.144	0.394	0.714	0.332	0.582
SAPS	0.057	**0.209**	**0.437**	**0.719**	**0.391**	**0.594**
	Minimum					
All_CPFP	0.002	0.014	0.303	0.549	0.068	**0.496**
TDS	**0**	0	0.293	0.458	0	0.489
CFPS	**0**	0.028	0.313	0.550	0.088	0.505
CAMEL	**0**	0	0.293	0.514	0	0.473
SAPS	0.012	**0.063**	**0.337**	**0.569**	**0.173**	0.494
	Maximum					
All_CPFP	0.145	**0.477**	0.613	0.782	0.630	0.675
TDS	0.087	0.246	0.465	0.783	0.480	0.626
CFPS	0.177	0.347	0.534	0.813	0.544	0.657
CAMEL	**0.084**	0.400	0.574	0.832	0.614	**0.711**
SAPS	0.125	**0.477**	**0.618**	**0.842**	**0.639**	0.683

Table 5.2: The statistical comparison of SAPS with base SPS based CPFP models using WSR test

Measures	All_CPFP		TDS		CFPS		CAMEL	
	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)
FOR	0.001842	0.717(42.5/257.5)	2.67E-05	0.980(3.0/297.0)	*0.094506*	*0.407(75.0/178.0)*	0.000821	0.804(27.0/249.0)
Recal	**3.43E-05**	**0.967(295.0/5.0)**	**2.67E-05**	**0.980(297.0/3.0)**	**0.002972**	**0.723(218.0/35.0)**	**0.007237**	**0.627(244.0/56.0)**
Balance	**3.09E-05**	**0.993(275.0/1.0)**	**3.03E-05**	**0.973(296.0/4.0)**	**0.000651**	**0.834(232.0/21.0)**	**0.006639**	**0.633(245.0/55.0)**
AUC	**9.60E-05**	**0.910(286.5/13.5)**	**3.88E-05**	**0.960(294.0/6.0)**	**0.001236**	**0.787(226.0/27.0)**	*0.988599*	*0.000(150.0/150.0)*
G-mean	**3.43E-05**	**0.967(295.0/5.0)**	**4.97E-05**	**0.947(292.0/8.0)**	**0.000136**	**0.929(244.0/9.0)**	*0.086467*	*0.400(210.0/90.0)*
nMCC	**0.000517**	**0.822(251.5/24.5)**	**0.001244**	**0.753(263.0/37.0)**	**0.000427**	**0.854(234.5/18.5)**	*0.247189*	*0.277(191.5/108.5)*

Prediction performance: Figure 5.3 depicts the distribution of each SPS model over a specific performance measure across 24 projects. Each box in the subfigure represents a particular measure outcome distribution with that of a specific SPS model, and the red circle in that box represents the mean measure value. The black diamond-shaped symbol above and below each box denotes the outliers among those outcomes. Each box's bottom and top lines represent the minimum and maximum values, respectively. As we can notice from Figure 5.3 and Table 5.1, our SAPS selected sources substantially outperform baseline SPS based CPFP models. On average across 24 projects, SAPS outperforms All_CPFP, TDS, CFPS, and CAMEL by 45.14%, 231.75%, 47.18%, and 45.14% in terms of Recall, 11.2%, 29.67%, 13.21%, and 10.91% in terms of Balance, 4.96%, 15.22%, 3.75%, and 0.7% in terms of AUC, 27.36%, 94.53%, 43.22%, and 17.77% in terms of G-mean, and 3.48%, 5.69%, 4.95%, and 2.06% in terms of nMCC, respectively. Furthermore, the minimum in terms of Recall, Balance, AUC, and G-mean for SAPS has the highest scores with 0.063, 0.337,

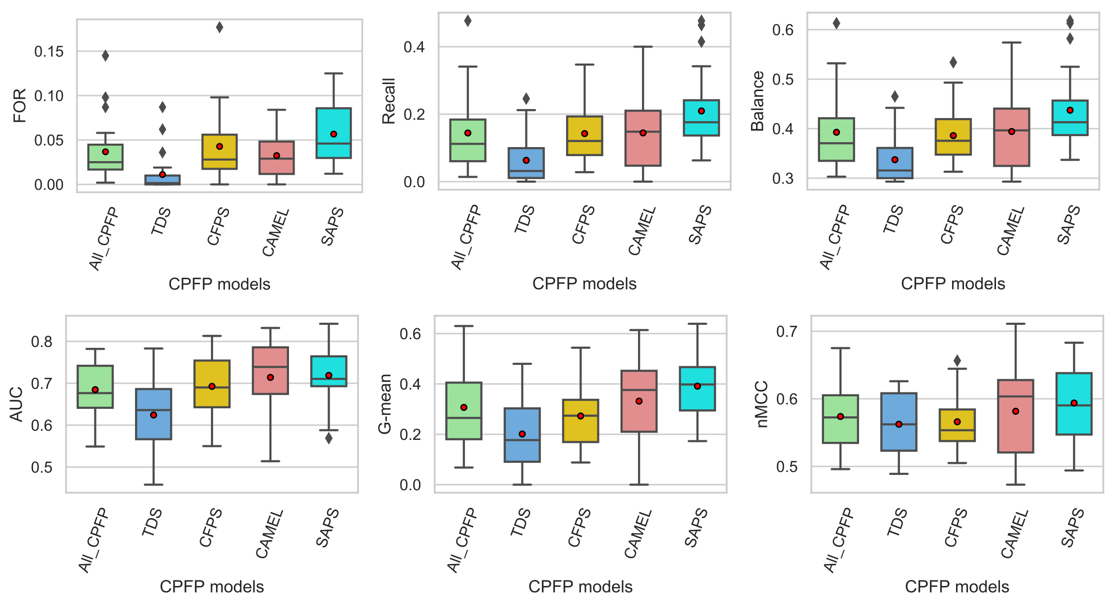


Figure 5.3: Box plots of five SPS based CPFP models outcomes across 24 projects

0.569, and 0.173 values. In terms of nMCC, CFPS gets the highest minimum value with just an excess of 2.27%. As per the maximum scores, in terms of Recall, Balance, AUC, and G-mean, SAPS has the highest scores with 0.477, 0.618, 0.842, and 0.639 values. In terms of nMCC, CAMEL gets the highest maximum value with just an excess of 4.09%.

As per FOR, SAPS didn't get a good enough performance. The compared base models achieve less FOR at the expense of a lower Recall measure value. According to Bennin et al. [169], both an increase in FOR with an increase in Recall and a decrease in FOR with a decrease in the Recall are improper recommendations for a specific application. To compute the combined performance of FOR and Recall, we use the Balance measure. In terms of the Balance measure, our SAPS outperforms the other All_CPFP, TDS, CFPS, and CAMEL models with average increases of 4.96%, 15.22%, 3.75%, and 0.7%, respectively. As we can see, TDS is unable to surpass the basic All_CPFP model, while CFPS and CAMEL performance is comparable to All_CPFP. Nevertheless, our SAPS exhibits significant improvement over All_CPFP with respect to all considered measures except FOR. Because the All_CPFP model is built with a lot of information by combining all the available projects, it can only reduce the false positives but is unable to maximize the overall performance.

Statistical Test: Table 5.2 presents the statistical comparison of SAPS against base SPS models. It contains the WSR test's returned P-values, the positive sum (W+) of our proposed model, and the negative sum (W-) of the compared base model, as well as the matched-pairs rank biserial correlation coefficient measure's returned effect size (Ec). As we can see from Table 5.2, the bold cells indicate that our model is statistically superior to the others, while the italic cells indicate that both models have a statistically similar distribution of results across 24 projects. The SAPS statistically outperforms (as the P-value is <0.05 and the W+ is more than the W-) the All_CPFP, TDS, and CFPS models with very large effect sizes (Ec from 0.723 to 0.993) in terms of Recall, Balance, AUC, G-mean, and nMCC. SAPS statistically wins over CAMEL in terms of Recall and Balance while being statistically similar (as the P-value is >0.05) in terms of AUC, G-mean, and nMCC. As per FOR, SAPS is statistically identical to CFPS with an effect size of 0.407, but it doesn't show significant improvement against other models. Regarding overall performance measures like Balance, AUC, G-mean, and nMCC, SAPS statistically outperforms all the compared models.

Besides, the SAPS's advantage over TDS and CAMEL indicates that although a suitable source project can be chosen by comparing its feature distribution to that of the target project, a preferable alternative is to learn their relationship through applicability among them (Section 5.1.1.2). Although the CFPS model considers both similarity and applicability into account, it is limited to choosing only three sources for each target, which might degrade the prediction performance over a few project datasets. Our SAPS model doesn't limit the number of sources to be selected. The All_CPFP model trains with all of the available projects data; too much unnecessary data might degrade the performance of the prediction model.

Results RQ1: The SAPS, the part of the proposed SRES model, shows significant improvement over previously succeeded SPS based CPFP models, including TDS, CFPS, CAMEL, and the basic All_CPFP model, in terms of overall performance measures.

5.3.2 RQ2: How effective is the proposed SRES model compared to the state-of-the-art CPFP models?

RQ2 discusses whether the suggested SRES approach can improve prediction performance over the target project by addressing CPFP-related issues such as distribution gaps, class imbalances, and high-dimensional data. We propose the SRES model to overcome these issues by introducing SPS, resampling, and feature reduction techniques. To demonstrate how effectively SRES improves CPFP performance, we compare the SRES model with four widely popular CPFP models, including NNFilter, TCA, TNB, and TSFA.

Table 5.3: The Average, Minimum, and Maximum values of CPFP models acorss 24 projects

Methods	FOR	Recall	Balance	AUC	G-mean	nMCC
	Average					
NNFilter	0.101	0.332	0.519	0.728	0.526	0.614
TCA	0.138	0.211	0.431	0.537	0.400	0.532
TNB	0.140	0.374	0.535	0.703	0.540	0.612
TFSA	**0.036**	0.124	0.379	0.698	0.290	0.566
SRES	0.126	**0.472**	**0.614**	**0.740**	**0.635**	**0.655**
	Minimum					
NNFilter	0.032	0.065	0.339	0.593	0.251	0.514
TCA	0.038	0.019	0.300	0.447	0.090	0.437
TNB	0.016	0.019	0.306	0.441	0.137	0.489
TFSA	**0**	0	0.293	0.495	0	0.455
SRES	0.063	**0.250**	**0.466**	**0.641**	**0.477**	**0.592**
	Maximum					
NNFilter	0.215	0.637	0.707	0.841	0.715	0.729
TCA	0.250	0.482	0.593	0.616	0.601	0.616
TNB	0.580	**0.727**	0.738	**0.858**	0.749	0.752
TFSA	**0.152**	0.442	0.601	0.819	0.636	0.681
SRES	0.235	0.688	**0.755**	0.812	**0.764**	**0.755**

Table 5.4: The statistical comparison of proposed SRES model with Base CPFP models using WSR test

Measure	NNFilter		TCA		TNB		TFAS	
	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)
FOR	0.015149	0.570(64.5/235.5)	*0.308222*	*0.250(172.5/103.5)*	*1.000*	*0.000(138.0/138.0)*	6.33E-05	0.933(10.0/290.0)
Recal	**6.33E-05**	**0.933(290.0/10.0)**	**2.67E-05**	**0.980(297.0/3.0)**	**0.004137**	**0.714(198.0/33.0)**	**1.82E-05**	**1.000(300.0/-0.0)**
Balance	**5.28E-05**	**0.943(291.5/8.5)**	**2.35E-05**	**0.987(298.0/2.0)**	**0.000228**	**0.860(279.0/21.0)**	**1.82E-05**	**1.000(300.0/-0.0)**
AUC	**0.059302**	**0.449(200.0/76.0)**	**1.82E-05**	**1.000(300.0/-0.0)**	**0.004882**	**0.657(248.5/51.5)**	**0.000203**	**0.867(280.0/20.0)**
G-mean	**5.61E-05**	**0.943(291.5/8.5)**	**1.82E-05**	**1.000(300.0/-0.0)**	**0.000102**	**0.907(286.0/14.0)**	**1.82E-05**	**1.000(300.0/-0.0)**
nMCC	**3.88E-05**	**0.960(294.0/6.0)**	**1.82E-05**	**1.000(300.0/-0.0)**	**3.03E-05**	**0.973(296.0/4.0)**	**2.70E-05**	**1.000(276.0/-0.0)**

Prediction performance: Figure 5.4 depicts the box plots of the distribution of each CPFP model with a specific performance measure across 24 projects. Table 5.3 outlines the precise numerical average, minimum, and maximum values of each CPFP model over individual measurements across 24 projects. We can observe from Figure 5.4 and Table 5.3 that our proposed SRES model performs significantly better than baseline CPFP models. On average over 24 software fault projects, our SRES model outperforms the NNFilter, TCA, TNB, and TFSA models in terms of Recall by 42.17%,123.69%,26.20%, and 280.65%, Balance by 18.30%, 42.46%, 14.77%, and 62.01%, AUC by 1.65%, 37.80%, 5.27%, and 6.08%, G-mean by 20.72%, 58.75%, 17.59%, and 118.96%, and nMCC by 6.68%, 23.12%, 7.03%, and 15.72%. On average, with respect to all the performance measures except FOR, SRES shows superiority over the compared models. Regarding FOR, SRES is unable to get a good enough outcome compared to the NNFilter and TFSA, but on average, SRES achieves an 8.70% and 10% lower score than the TCA and TNB models, respectively. In our SRES

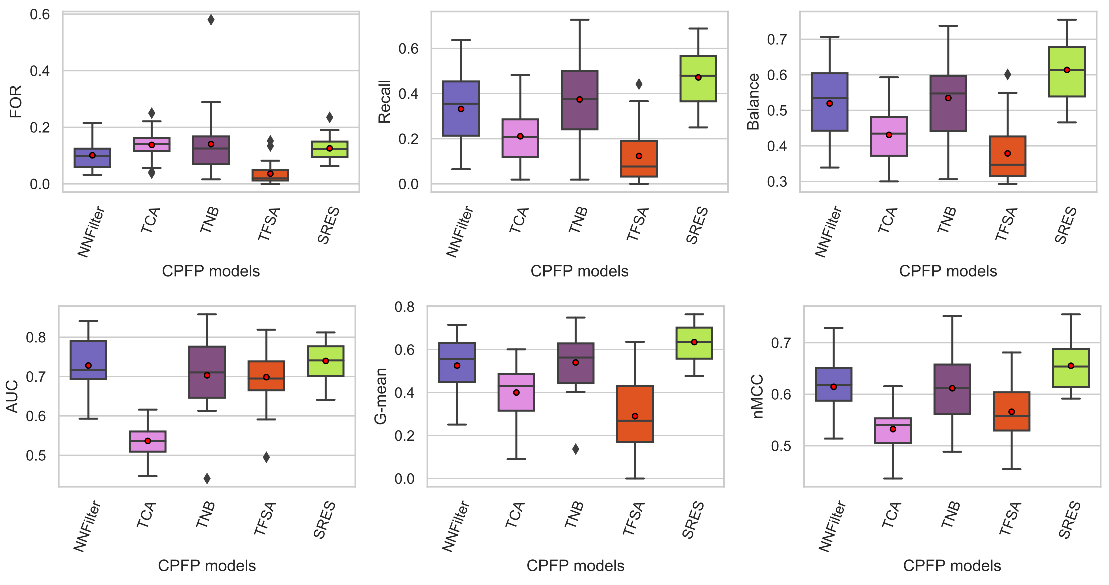


Figure 5.4: Box plots of five CPFP models outcomes across 24 projects

approach, while performing oversampling we introduce a few synthetic instances into the faulty instances class that might increase the false positives; due to this reason, SRES is unable to outperform the baseline models in terms of the FOR measure. In imbalanced datasets, the FOR and Recall measures should be balanced. Thus, we considered the Balance measure between FOR and Recall. As we can observe from Table 5.3 and Figure 5.4, the SRES model average score is significantly superior to the base models. In terms of FOR, SRES is unable to outperform the NNFilter and TFSA models, while their Balance scores and other metrics of overall performance, such as AUC, G-mean, and nMCC scores, are substantially lower than our approach. Moreover, we can compare the minimum and maximum of each CPFP model's performance scores over 24 projects in Table 5.3. The minimum in terms of Recall, Balance, AUC, G-mean, and nMCC for SRES has the highest scores with 0.250, 0.466, 0.641, 0.477, and 0.592 values, respectively. As per the maximum scores, SRES has the highest scores with 0.755, 0.764, and 0.755 values in terms of Balance, G-mean, and nMCC, respectively. Regarding Recall and AUC, TNB gets the highest maximum value with little excess of 5.67% and 5.66%, respectively. As per FOR, SRES is not superior but gives slightly comparable outcomes to those models.

Statistical Test: Table 5.4 presents the statistical comparison of SRES against base CPFP models. The bold cells indicate that our SRES model is statistically superior to the others, while the italic cells indicate that both the SRES and compared models have a statistically similar distribution of results across 24 projects. The SRES statistically outperforms the TCA, TNB, and TFSA models with large effect sizes (Ec from 0.714 to 1.000) in terms of Recall, Balance, AUC, G-mean, and nMCC. SAPS statistically defeats NFilter with large effect sizes (Ec from 0.933 to 0.960) in terms of Recall, Balance, G-mean, and nMCC, while being statistically similar in terms of AUC. As per FOR, SAPS is statistically identical to TCA and TNB with effect sizes of 0.250 and 0, respectively, but it doesn't significantly improve against the TFSA model. In terms of Recall and overall performance measures such as Balance, AUC, G-mean, and nMCC, our proposed SRES model statistically outperforms all the compared models with very large effect sizes.

Results RQ2: The proposed SRES outperforms previously established state-of-the-art CPFP models, including NNFilter, TCA, TNB, and TFSA, in terms of overall performance measures.

5.3.3 RQ3: Is prediction performance over the target project improved through SRES based cross-project prediction compared to within-project prediction performance?

It is a widespread misconception to believe the training data from the same project will enhance the accuracy of predictions. In [23, 24, 32, 33], reported that due to the different distributions between various projects, the prediction models learned from cross-project data exhibit significantly lower performance than the models learned from the same project data. To address the RQ3, we analyzed how much performance improvement is achieved by our proposed SRES model by comparing the prediction outcomes with the WPFP model more systematically and quantitatively.

Table 5.5: The Average, Minimum, and Maximum values of WPFP, SAPS, and SRES models across all 24 projects

Methods	FOR	Recall	Balance	AUC	G-mean	nMCC
	Average					
WPFP	0.101	0.305	0.488	0.717	0.454	0.624
SAPS	**0.057**	0.209	0.437	0.719	0.391	0.594
SRES	0.126	**0.472**	**0.614**	**0.740**	**0.635**	**0.655**
	Minimum					
WPFP	0.024	0.087	0.345	0.560	0.121	0.509
SAPS	**0.012**	0.063	0.337	0.569	0.173	0.494
SRES	0.063	**0.250**	**0.466**	**0.641**	**0.477**	**0.592**
	Maximum					
WPFP	0.315	0.610	0.706	**0.852**	0.726	0.724
SAPS	**0.125**	0.477	0.618	0.842	0.639	0.683
SRES	0.235	**0.688**	**0.755**	0.812	**0.764**	**0.755**

Table 5.6: The statistical comparison of SRES with WPFP and SAPS models using WSR test

Measures	WPFP		SAPS	
	P-value	Ec(W+/W-)	P-value	Ec(W+/W-)
FOR	0.042491	0.473(79.0/221.0)	2.66E-05	0.980(3.0/297.0)
Recal	**0.000747**	**0.790(268.5/31.5)**	**1.82E-05**	**1.000(300.0/0.0)**
Balance	**0.000129**	**0.893(284.0/16.0)**	**1.82E-05**	**1.000(300.0/0.0)**
AUC	*0.067985*	*0.435(198.0/78.0)*	*0.081203*	*0.407(211.0/89.0)*
G-mean	**0.000129**	**0.893(284.0/16.0)**	**1.82E-05**	**1.000(300.0/0.0)**
nMCC	**0.009322**	**0.610(241.5/58.5)**	**3.43E-05**	**0.967(295.0/5.0)**

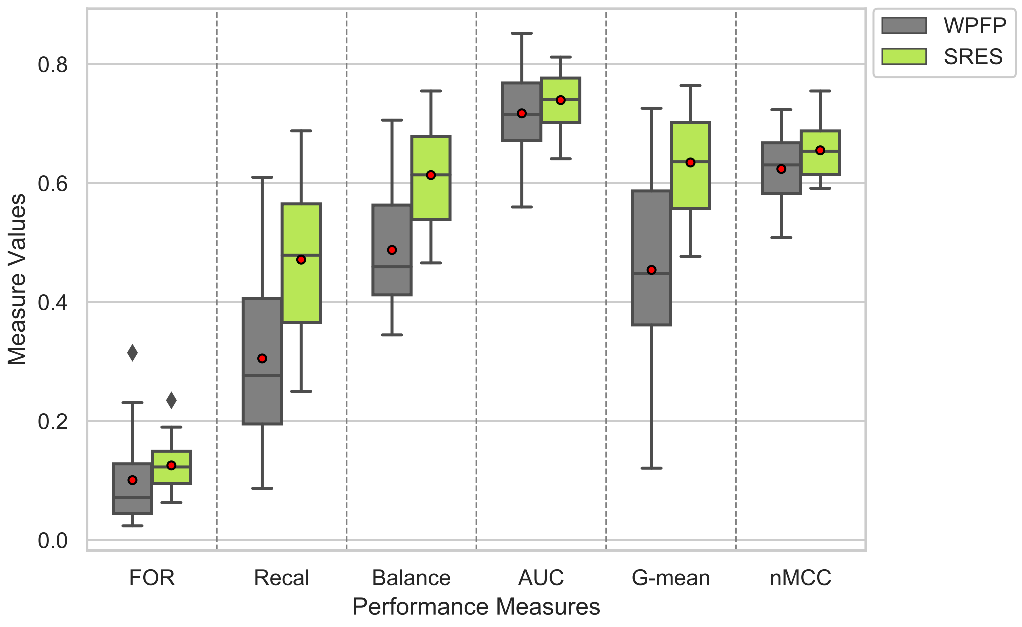


Figure 5.5: Box plot of WPFP and SRES models outcomes across 24 projects

Prediction performance: Figure 5.5 depicts the WPFP and SRES outcomes in box plots across 24 projects. Table 5.5 outlines both models' precise numeric average, minimum, and maximum values over individual measurements. On average across 24 software fault projects, SRES approach outperforms the WPFP model in terms of Recall, Balance, AUC, G-mean and nMCC by 54.75% (i.e., from 0.305 to 0.472), 25.82% (i.e., from 0.488 to 0.614), 3.21% (i.e., from 0.717 to 0.740), 39.87%(i.e., from 0.454 to 0.635), and 4.97% (i.e., from 0.624 to 0.655), NASA, and PROMISE datasets, respectively. According to previous studies, the AUC of the CPFP models doesn't improve much compared to the WPFP model over imbalanced data. Our proposed model showed a 3.21% improvement in AUC over the WPFP model. Even though AUC and nMCC's improvement is lower compared to Recall, Balance, and G-mean, it is still superior to the other CPFP models that are currently in use. While other models are unable to improve the performance compared to WPFP, our model gains considerably better improvement in terms of Recall, Balance and G-mean and a slight improvement in terms of AUC and nMCC. The training data we consider for the SRES model integrates all the selected sources data, and we oversample the faulty instances of training data by introducing synthetic data, which could be the reason for the FOR performance degradation. As we discussed in RQ2, only FOR doesn't determine the overall performance of a prediction model in imbalanced datasets. As we can observe from Table 5.5 and Fig 5.5, even though FOR score of our SRES model is not superior to WPFP, while the Balance scores of WPFP and other overall performance measures, such as AUC, G-mean, and nMCC scores, are substantially lower than our approach. Moreover, we compared the minimum and maximum performance scores of the WPFP and SRES models across 24 projects in Table 5.5. The minimum in terms of Recall, Balance, AUC, G-mean, and nMCC for SRES has the highest scores with 0.250, 0.466, 0.641, 0.477, and 0.592 values, respectively. As per the maximum scores, SRES has the highest scores with 0.688, 0.755, 0.764, and 0.755 values in terms of Recall, Balance, G-mean, and nMCC, respectively. Regarding AUC, WPFP gets the highest maximum value with a slight excess of 5.93%. As per FOR, SRES gives the highest minimum value for a project but reports the lowest maximum value.

Statistical Test: The statistical comparison of SRES against the WPFP model is presented in the first column of Table 5.6. The bold cells indicate that our SRES model is statistically superior to the WPFP, while the italic cells indicate that both the SRES and WPFP have statistically similar results across 24 projects. The SRES statistically outperforms the WPFP model with very large effect sizes (Ec from 0.610 to 0.893) in terms of Recall, Balance, G-mean, and nMCC, while being statistically similar in terms of AUC. As per FOR, WPFP statistically outperforms the SRES model with a medium effect size of 0.473. Even though, in terms of single-class measure Recall and overall performance measures such as Balance, G-mean, and nMCC, SRES statistically outperforms the WPFP model with very large effect sizes. Hence, we can assert that the overall performance of our proposed SRES based CPFP model is superior to the WPFP.

Results RQ3: We can observe from the prediction performance and statistical comparison between the SRES and WPFP models that the overall prediction performance of the proposed SRES model improved on the target projects compared to the within-project prediction performance.

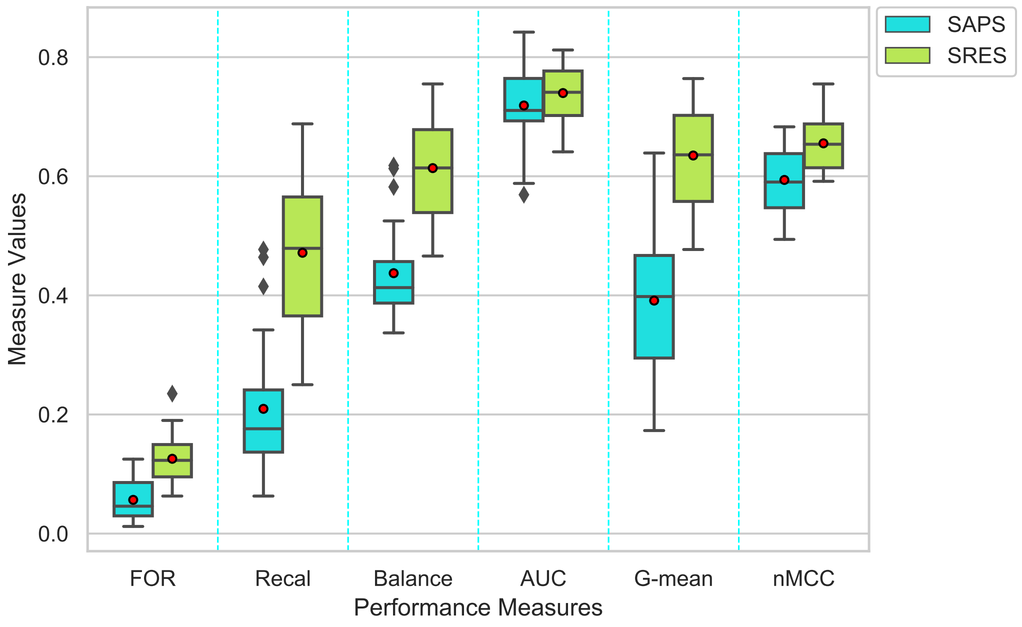


Figure 5.6: Box plot of SAPS and SRES models outcomes across 24 projects.

5.3.4 RQ4: How much improvement does the proposed SRES model gain over the SAPS model?

The SAPS method is a part of the proposed SRES model, which selects the appropriate source projects prior to resampling and feature reduction. In this RQ, we explore whether only source projects selection is sufficient to improve the cross-project prediction performance? To give a worthy explanation for this, we compare the SRES model with the SAPS model, where SRES is the combination of SAPS along with resampling and feature reduction techniques. We computed and compared the prediction outcomes of both the SAPS and SRES models.

Prediction performance : The box plots of the distribution of both the SAPS and SRES model's performance measures (i.e., FOR, Recall, Balance, AUC, G-mean, and nMCC) are depicted in Figure 5.6. The red circle in each box indicates the mean score of individual measures with respect to a particular prediction model. Table 5.5 outlines both models' precise numeric average, minimum, and maximum values over individual measurements across 24 projects. On average across 24 NASA and PROMISE software fault projects, SRES outperforms the SAPS model in terms of Recall, Balance, AUC, G-mean and nMCC by 125.84% (i.e., from 0.209 to 0.472), 40.50% (i.e., from 0.437 to 0.614), 2.92% (i.e., from 0.719 to 0.74), 62.40% (i.e., from 0.391 to 0.635), and 10.27%(i.e., from 0.594 to 0.655). In terms of FOR, the performance of SRES is not much better than SAPS because we utilize the combination of selected sources data without any modifications as training data for the SAPS model. Unlike SAPS, to address issues that influence the CPFP, like imbalanced data, a distribution gap between training and target data, and high-dimensionality issues, SRES takes data generated from SAPS as training data and applies resampling and feature reduction techniques. While resampling, we introduce fault-prone synthetic data into training data to equal faulty and non-faulty class instances. As a result, the prediction performance of the faulty modules (true positives) increases at the cost of increasing false positives. Afterwards, the feature set is reduced through the SAE model to address the high dimensionality issue. As a result, the overall performance of the prediction model quietly improved compared to SAPS. Moreover, we compared the minimum and maximum performance scores of the SAPS and SRES models across 24 projects in Table 5.5. The minimum in terms of Recall, Balance, AUC, G-mean, and nMCC for SRES has the highest scores with 0.250, 0.466, 0.641, 0.477, and 0.592 values, respectively. As per the maximum scores, SRES has the highest scores with values of 0.688, 0.755, 0.764, and 0.755 in terms of Recall, Balance, G-mean, and nMCC, respectively. Regarding AUC, SAPS gets the highest maximum value with a slight excess of 3.69%. As we discussed, SAPS reported the highest minimum and maximum values in terms of FOR.

Statistical Test: The statistical comparison of SRES against the SAPS model is presented in the second column of Table 5.6. The bold cells indicate that the SRES model is statistically superior to the SAPS, while the italic cells indicate that both the SRES and SAPS have a statistically similar distribution of results across 24 projects. The SRES statistically outperforms the SAPS model with very large effect sizes (Ec from 0.967 to 1.0) in terms of Recall, Balance, G-mean, and nMCC while being statistically similar in terms of AUC. As per FOR, SAPS statistically outperforms the SRES model with a large effect size of 0.980. However, in terms of Recall and overall performance measures such as Balance, G-mean, and nMCC, SRES statistically outperforms the SAPS model with very large effect sizes. Hence, we can assert that the overall performance of the SRES model is superior to the SAPS.

Results RQ4: From analyzing the prediction performance and statistical comparisons, we can assert that the SRES model improves over the SAPS model after applying resampling and feature reduction techniques by overcoming the imbalance and high dimensionality issues.

To analyze the performance of each fault prediction model, we perform WDL comparison, where each fault predictor is compared with 10 other models. The results (Shown in Table 5.7)

Table 5.7: WSR test based Win-Draw-Loss comparison performance across all fault prediction models per each performance measure

Models	Win	Draw	Loss	Win-Loss	Rank	CPFP	Win	Draw	Loss	Win-Loss	Rank
	FOR						Recall				
WPFP	1	3	6	-5	9	WPFP	5	4	1	4	4
All_CPFP	6	3	1	5	3	All_CPFP	1	3	6	-5	8.5
TDS	10	0	0	10	1	TDS	0	0	10	-10	11
CFPS	5	4	1	4	5	CFPS	1	3	6	-5	8.5
CAMEL	6	3	1	5	3	CAMEL	1	3	6	-5	8.5
NNFilter	2	2	6	-4	7.5	NNFilter	7	2	1	6	2.5
TCA	0	3	7	-7	11	TCA	5	2	3	2	5.5
TNB	0	4	6	-6	10	TNB	7	2	1	6	2.5
TFSA	6	3	1	5	3	TFSA	1	3	6	-5	8.5
SAPS	5	1	4	1	6	SAPS	5	2	3	2	5.5
SRES	2	2	6	-4	7.5	SRES	10	0	0	10	1
	Balance						AUC				
WPFP	5	4	1	4	4	WPFP	3	7	0	3	5
All_CPFP	1	3	6	-5	8.5	All_CPFP	2	2	6	-4	9
TDS	0	0	10	-10	11	TDS	1	0	9	-8	10
CFPS	1	3	6	-5	8.5	CFPS	2	4	4	-2	7.5
CAMEL	1	3	6	-5	8.5	CAMEL	5	5	0	5	3.5
NNFilter	7	2	1	6	2.5	NNFilter	6	4	0	6	1.5
TCA	5	2	3	2	5.5	TCA	0	0	10	-10	11
TNB	7	2	1	6	2.5	TNB	3	5	2	1	6
TFSA	1	3	6	-5	8.5	TFSA	2	4	4	-2	7.5
SAPS	5	2	3	2	5.5	SAPS	5	5	0	5	3.5
SRES	10	0	0	10	1	SRES	6	4	0	6	1.5
	G-mean						nMCC				
WPFP	5	3	2	3	4	WPFP	6	3	1	5	3.5
All_CPFP	2	2	6	-4	7.5	All_CPFP	1	4	5	-4	8.5
TDS	0	0	10	-10	11	TDS	1	4	5	-4	8.5
CFPS	1	2	7	-6	10	CFPS	1	4	5	-4	8.5
CAMEL	1	4	5	-4	7.5	CAMEL	1	5	4	-3	6
NNFilter	7	2	1	6	3	NNFilter	7	2	1	6	2
TCA	5	2	3	2	5	TCA	0	0	10	-10	11
TNB	8	1	1	7	2	TNB	6	3	1	5	3.5
TFSA	1	3	6	-5	9	TFSA	1	4	5	-4	8.5
SAPS	4	3	3	1	6	SAPS	5	3	2	3	5
SRES	10	0	0	10	1	SRES	10	0	0	10	1

are then aggregated across the prediction models, resulting in 110 (*11(single prediction model) * 10 (other prediction models)*) comparisons for each performance measure. Ranks were assigned based on wins-losses values. The highest wins-losses value receives the top rank (Rank 1), while the lowest values receive the worst rank. The better the model the higher the wins-losses ratio. Out of the 110 comparisons, the SRES approach consistently ranked first across all performance measures except FOR. Apart from our proposed SRES, NNFilter and TNB consistently rank in the top three positions for all measures except FOR. The part of the proposed SRES model which is SAPS based source projects selection approach, has always ranked in the middle. We can see from the Table 5.7 that the ranks of SAPS are 6, 5.5, 5.5, 3.5, 6, and 5 in terms of FOR, Recall, Balance, AUC, G-mean, and nMCC, respectively. Moreover, SAPS never gets a negative wins-losses ratio. Next, in terms of FOR, TDS, All_CPFP, CAMEL, and TFSA are in the top 3 ranks, while getting the lower ranks by consistently attaining negative wins-losses values for the rest of the measures. From Table 5.7, we can draw average rankings as 4.9, 7.5, 8.8, 8, 6.2, 3.2, 8.2, 4.4, 7.5, 5.3, and 2.2 for WPFP, All_CPFP, TDS, CFPS, CAMEL, NNFilter, TCA, TNB, TFSA, SAPS, and SRES, respectively. We can observe that, on average, our proposed SRES model achieved the top rank, followed by NNFilter and TNB.

From RQ1, RQ2, RQ3, RQ4, and overall rankings, we conclude that our proposed SRES performs superior to WPFP, basic CPFP, source projects selection models, and some popular CPFP models. The results are consistent with findings observed by [29, 36, 116], WPFP performs better than the CPFP models when assessed using the FOR performance measure. As well as our results are consistent with [31] and [33], in which CPFP models can outperform the WPFP models by cautiously selecting source projects and reducing the distribution gap between source and target projects.

5.4 Threats to Validity

This section reviews the potential threats that jeopardize the validity of this study's experimental findings. SRES approach selects source projects based on two assumptions: 1) Two projects with a statistically similar distribution can predict each other well, which might not be the case with a few projects. 2) While verifying applicability, we select test data from the entire set of available projects. However, when the set of available projects changes, there might be a change in the selected source projects of our study. Additionally, we used the trial-and-error method to determine the ideal number of hidden nodes, epochs, and batch size for SAE while performing feature reduction, which could lead to biased findings with various projects at different times. These could be internal threats to our study. In this study, we examined a large number of projects of various sizes from two different open-source datasets, limiting external threats to the model.

5.5 Summary

Over the past few years, CPFP models have received considerable attention and demonstrated acceptable prediction performance, although there is still scope for improvement. We addressed the issue of how to predict software faults using cross-project data using a novel cross-project fault prediction model called SRES. SRES is a combination of three phases, including a new source projects generation using applicability and similarity scores, a novel resampling of the source data through oversampling and undersampling, and it performs feature reduction using the employed SAE network. It makes the SRES model robust enough to address the aforementioned issues.

In this study, 24 software projects are selected for the experimental study and the effectiveness of the proposed SRES model is assessed with six recommended performance measures, including FOR, Recall, Balance, AUC, G-mean, and nMCC. In the experiments, we first compare the SPS CPFP models with our SAPS model. We also compared the proposed SRES model with the WPFP and baseline CPFP models. Experimental findings demonstrate that the SRES approach outperforms the competitive CPFP models in terms of all measures except FOR. The next chapter presents another early reliability prediction model called software development effort estimation through a two-stage optimization technique.

Chapter 6

Two-stage Optimization Technique for Software Development Effort Estimation

Due to the pivotal role that effort estimation holds within the development process, a variety of estimation approaches have been established to improve its accuracy. By considering the limitations in existing estimation models, we proposed a two-stage optimization effort estimation (TSoptEE) technique to improve the prediction performance.

The remainder of this chapter is organized as follows: Section 6.1 and 6.2 presents the background works and research methodology, respectively. Section 6.3 shows the details of experimental setup objects. Section 6.4 demonstrates the obtained results and the discussion. Finally, the chapter ends with a summary in Section 6.5.

6.1 Background

This section discusses the basic architecture of the ANFIS model and Social Network Search (SNS) meta-heuristic algorithm.

6.1.1 ANFIS: Adaptive Neuro-Fuzzy Inference System

The ANFIS model introduced by Jang [172] explains the basic ANFIS model's architecture and learning procedure. The ANFIS model has gained significant attention from researchers as an effective estimation model among neuro-fuzzy systems and other machine learning models. ANFIS combines fuzzy logic with neural networks, demonstrating notable improvement in effort estimation. ANFIS follows a five-layer architecture consisting of two kinds of nodes: fixed (layer 2, 3, and 5 nodes) and tunable (layer 1 and 2 nodes). The ANFIS architecture is depicted in Figure 6.1. ANFIS learns by adjusting all its tunable parameters to effectively map input data to the desired output with minimal error. The layers of the ANFIS model are described as follows:

Layer 1: In this layer, every input of node i is adaptive with a membership function and generates

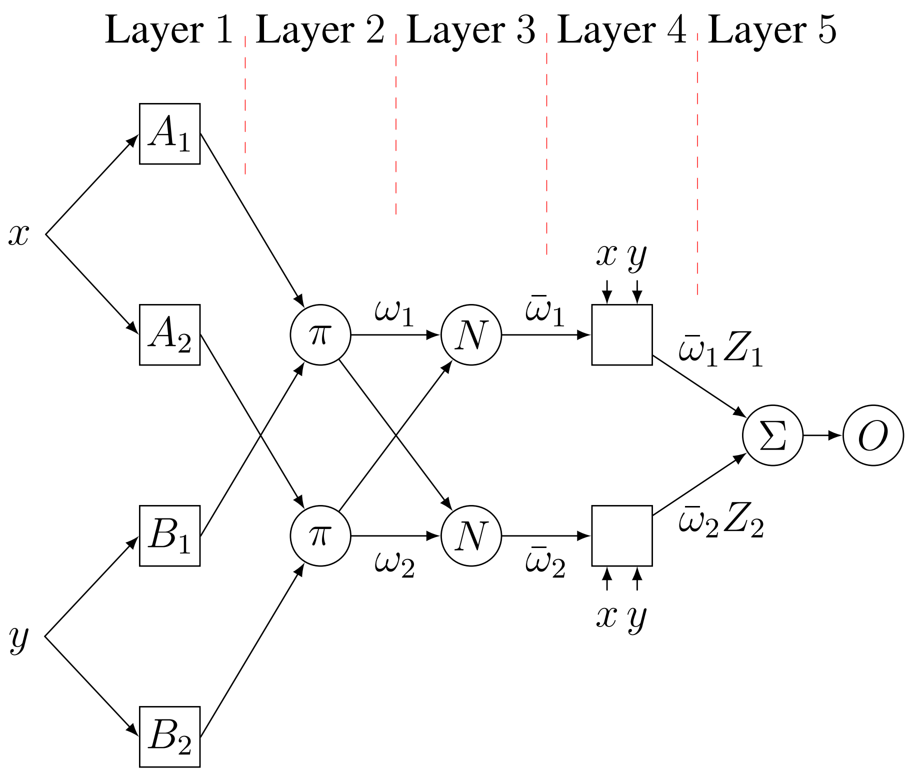


Figure 6.1: ANFIS model architecture

the membership degree of input values. Eq. 6.1 shows the output of membership functions:

$$\begin{aligned} O^1(x_i) &= A_i = \mu_{A_i}(x), \quad i = 1, 2 \\ O^1(y_i) &= B_i = \mu_{B_i}(y), \quad i = 1, 2 \end{aligned} \tag{6.1}$$

Where x and y are the inputs to the first layer nodes, A_i and B_i are their respective membership values, 'i' is the number of membership functions for each feature, and μ_{A_i} and μ_{B_i} are membership functions associated with each node and parameters in those functions called premise parameters. This process is called fuzzification. In this study, we use a generalized Bell (Gbell) membership function, which computes as follows in Eq. 6.2:

$$\mu(x) = \frac{1}{1 + \left(\frac{|x-c|}{a}\right)^{2b}} \tag{6.2}$$

Where 'x' is the input value, and a, b, and c are Gbell premise parameters to be tuned.

Layer 2: All potential rules between the membership values are formulated by applying fuzzy intersection (AND) as shown in Eq. 6.3, called weights:

$$\omega_i = \mu_{A_i}(x) \times \mu_{A_i}(y) \quad i{=}1, 2 \tag{6.3}$$

Layer 3: In the third layer, weights obtained from the second layer are normalized by following Eq. 6.4, called firing strengths:

$$\bar{\omega_i} = \frac{\omega_i}{\omega_1 + \omega_2} \quad i{=}1, 2 \tag{6.4}$$

Layer 4: Each node computes the contribution of the i^{th} rule to the overall output. This process is called defuzzification. Eq. 6.5 shows the computation:

$$\bar{\omega_i} Z_i = \bar{\omega_i}(a_i x + b_i y + c_i) \quad i{=}1, 2 \tag{6.5}$$

Where $\bar{\omega_i}$ is the output of layer three and a_i, b_i, c_i is the consequent parameter set to be tuned.

Layer 5: The final layer computes the overall output as the summation of all incoming values from layer 4 using Eq. 6.6:

$$O_{final} = \sum_i \bar{\omega_i} Z_i = \frac{\sum_i \omega_i Z_i}{\sum_i \omega_i} \tag{6.6}$$

Figure 6.1 explains the architecture of the ANFIS model used for software development effort estimation. When it comes to real-time project management, the project manager provides features (attributes) of the software dataset to the ANFIS model, and then optimal premise and consequent parameters are computed in layers 1 and 4, respectively. Layer 5 returns the estimated effort, while the accuracy of the estimated effort is decided by how the parameters are optimized. The project manager takes the returned effort as an estimated effort and makes project management and budget plans accordingly. This approach helps to mitigate overestimation and underestimation issues, which can significantly impact the project's cost estimation.

6.1.2 SNS: Social Network Search Algorithm

Social networks serve as virtual platforms where users can interact with one another. This interaction and influence among users follow an optimized process where individuals consistently strive to enhance their popularity within the network. Talatahari et al. [173] proposed the SNS algorithm, which replicates this interactive behaviour among users. As users interact, they might be influenced by the perspectives of other network users in different scenarios, including imitation, conversation, disputation, and innovation. The following is an explanation and mathematical modelling of these moods:

Mood 1 (Imitation): Imitation means that the views of other users are attractive, and usually, users try to imitate each other in expressing their opinions. One user imitates another user to update its view as follows in Eq. 6.7:

$$\begin{aligned} U_{inew} &= U_i + rand(-1,1) * P \\ P &= rand(0,1) * p \\ p &= U_j - U_i \end{aligned} \tag{6.7}$$

Where U_{inew} is the new view vector of i^{th} user, U_j is the randomly chosen user view vector. $rand(-1,1)$ and $rand(0,1)$ are two random vectors in intervals [-1, 1] and [0, 1], respectively, and P is the difference in the U_j and U_i users.

Mood 2 (Conversation): In social networks, users communicate with each other and benefit from their conversations. One user interacts with another user and updates its view as follows in Eq. 6.8:

$$\begin{aligned} U_{inew} &= U_k + d \\ d &= rand(0,1) * D \\ D &= sign(f_i - f_j) * (U_j - U_i) \end{aligned} \tag{6.8}$$

Where U_{inew} is the new view vector of i^{th} user, U_k is the randomly defined view vector, and U_j is the randomly chosen user in the network, $rand(0,1)$ is a random vector in the interval [0, 1]. D is the difference between i^{th} and j^{th} users where $sign(f_i - f_j)$ tells the effect of j^{th} view on U_i view, it can be positive or negative.

Mood 3 (Disputation): In disputation, users see different views from a group of close-circle users and may be influenced by the expressed opinions. One user interacts with a group of users and updates its view vector as follows in Eq. 6.9:

$$\begin{aligned} U_{inew} &= U_i + rand(0,1) * (\mu - F * U_i) \\ \mu &= \frac{\sum_t^{N_r} U_t}{N_r} \\ F &= 1 + round(rand(0,1)) \end{aligned} \tag{6.9}$$

Where U_{inew} is the new view vector of i^{th} user, $rand(0,1)$ is a random vector in the interval [0,1], N_r is the random number selected to decide the group size, μ is mean of N_r user viewpoints in the network. F is the influence factor, which can be either 1 or 2, and $round$ is a function that rounds its input to the nearest integer number

Mood 4 (Innovation): In Innovation mood, a part of user opinion changes according to their perspective modification and random user perspective. The new viewpoint of a user develops by changing a randomly selected feature as follows in Eq. 6.10:

$$\begin{aligned} U_{inew}^d &= t_1 * U_j^d + (1 - t_1) * n_{new}^d \\ n_{new}^d &= lb_d + t_2 * (ub_d - lb_d) \end{aligned} \tag{6.10}$$

Where U_{inew}^d is the new view of the particular 'd^{th}' feature of i^{th} user viewpoint, d is the feature that is selected randomly in the interval $[1, M]$, and M is the number of features. t_1 and t_2 are random numbers in the interval [0, 1]. Also, ub_x and lb_x are maximum and minimum values for the d^{th} feature.

The selection and handling of these four moods influence each user's perspective, prompting them to adopt a new view vector. If the new view vector surpasses the existing one, users accept and communicate it within the network.

6.2 TSoptEE: Two-Stage Optimization Technique for Software Development Effort Estimation

The objective of this study is to accurately estimate the software development effort in terms of manpower per month or hour. This study employs the neuro-fuzzy ANFIS model to estimate the software development effort. Despite its widespread acceptance, the ANFIS model encounters constraints such as the curse of dimensionality and high computational costs, which restrict its use in

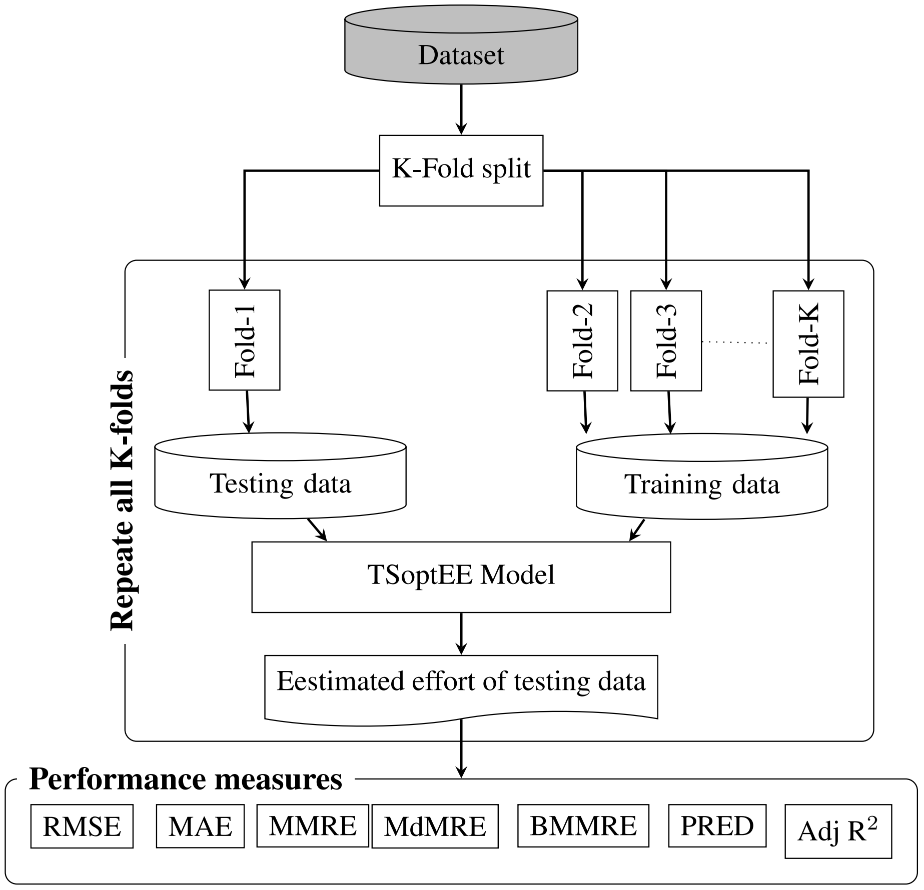


Figure 6.2: An overview of TSoptEE model framework

```
Algorithm 10: Two stage optimization technique for SDEE
    Require: Dataset ($D$);
    Ensure: $Effort_{pred}$;
  1 train_data($Tr$), test_data ($Te$) $\leftarrow$ split projects of dataset into training and testing
      sets
  2 $optimal\_feature\_set \leftarrow$ RiBSNS (train_data($Tr$))
  3 $Effort_{pred} \leftarrow$ RiCSNS_ANFIS ($Tr[optimal\_feature\_set]$,
      $Te[optimal\_feature\_set]$)
  4 return $Effort_{pred}$
```

applications involving large numbers of inputs. ANFIS generates a maximum number of rules by combining all possible fuzzy values, which escalates the computational cost due to the large number of consequent function coefficient parameters of these rules. The total number of premise and consequent parameters computed as follows below Eq. 6.11 :

$$F(m, nf, pf) = (m \times nf \times pf) + (nf^m \times (m + 1)) \tag{6.11}$$

Where 'm' denotes the number of features in the dataset, each feature value 'm' is divided into 'nf' number of membership functions, and 'pf' denotes the number of parameters that need to be tuned for each membership function. Thus, the total number of premise and consequent parameters to be tuned are '$m \times nf \times pf$' and '$nf^m \times (m+1)$', respectively. In the field of effort estimation, a frequently used publicly available dataset is 'Maxwell', comprising 26 independent features. In this study, each feature value is split into two membership functions, utilizing a generalized bell function as the membership function, which necessitates three parameters. So, the total number of parameters that require tuning is $(26 \times 2 \times 3) + (2^{26} \times (26+1)) = 1811939484$. This is an exceedingly large value, presenting significant complexity. ANFIS demonstrates its effectiveness when the input count is limited to five or fewer [174]. Considering these limitations, we have proposed a two-phase optimization approach for the software development effort estimation model, which is structured into two stages. These stages are thoroughly outlined in Algorithms 10, 11, and 12.

We employed a three-fold cross-validation strategy to assess the model's realistic accuracy. The model evaluates the data over each fold, making three possible experiments possible. In each experiment, two folds are used to train the model, while the remaining fold serves as the testing data. Firstly, multi-objective RiBSNS feature selection is carried out on training data to obtain an optimal feature set. Then, the training and testing data reduce their feature data to the selected optimal feature set. The nearest neighbour samples of each test sample are computed from the training data. The ANFIS model trains over the selected neighbour samples to obtain optimal premise and consequent parameters through the multi-objective RiCSNS algorithm, and the obtained optimal parameters are used to estimate the effort of the particular test sample. The process continues until all the testing samples have been evaluated. The average performance metrics across all three experiments are then considered as the final results. The process is depicted in Figure 6.2, where each iteration involves a different fold for testing, while the remaining folds are used for training. Further details about the two optimization stages are discussed in the subsequent subsections.

6.2.1 Stage 1: Multi-objective RiBSNS for feature selection

Firstly, TSoptEE selects optimal feature subsets using a multi-objective RiBSNS algorithm, which is detailed in Algorithm 11. To select optimal features through the RiBSNS algorithm, moods are assigned by ranks. Ranks are assigned to moods in the RiBSNS and RiCSNS algorithms. Out of

Algorithm 11: Multi-objective RiBSNS Feature Selection

```
Require: train_data($Tr$);
Ensure: $optimal\_feature\_set$;
function RiBSNS($train\_data(Tr)$)
    Initialize population: $CPOP_{[N \times M]}$ using Eqs. 6.12 and 6.13
    Convert the population ($CPOP_{[N \times M]}$) into binary population ($BPOP_{[N \times M]}$)
      using Eq. 6.14
    Calculate the fitness value $f_{1 \times N}$ for all solutions in $BPOP_{[N \times M]}$ using WSM
      method
    $Rank_{1 \times 4} \leftarrow$ NULL
    while $Termination$ do
        for each $solution(i)$ in $CPOP$ do
            Select the best mood out of four moods based on the assigned rank
            if $Mood == 1$ then
                Calculate new solution $CPOP_{inew}$ and $BPOP_{inew}$ using Eqs. 6.7
                  and 6.15
            end
            if $Mood == 2$ then
                Calculate new solution $CPOP_{inew}$ and $BPOP_{inew}$ using Eqs. Eqs.
                  6.8 and 6.15
            end
            if $Mood == 3$ then
                Calculate new solution $CPOP_{inew}$ and $BPOP_{inew}$ using Eqs. Eqs.
                  6.9 and 6.15
            end
            if $Mood == 4$ then
                Calculate new solution $CPOP_{inew}$ and $BPOP_{inew}$ using Eqs. Eqs.
                  6.10 and 6.15
            end
            f_inew ← Compute the fitness value of updated $BPOP_{inew}$ using
              WSM method
            if $f_{inew} \geq f_i$ then
                $f_i = f_{inew}$
                $CPOP_i = CPOP_{inew}$ and $BPOP_i = BPOP_{inew}$
                Rank[mood] ++
            else
                Rank[mood] - -
            end
        end
        $fbest = max(WSM(f))$
    end
    $optimal\_feature\_set \leftarrow fbest$
    return $optimal\_feature\_set$
end
```

four moods (Shown in Section 6.1.2), the mood with the best rank is selected to update the particular population's solution in that iteration, which can enhance the learning ability of the SNS algorithm. Ranks are assigned to moods in RiBSNS and RiCSNS algorithms to improve the feature selection process and ANFIS parameters optimization, respectively. The experimental results analysis is provided for selected features to predict software development effort. The first step of RiBSNS is the initialization of the population. As we are performing feature selection with a binary population, we first generate a population with continuous values and then convert them into binary. To maintain diversity in generated binary feature data, we perform mathematical computations over the continuous population and then convert it to binary while calculating fitness value. The dimension of the population is $N \times M$, where 'N' represents the randomly decided number of solutions and 'M' represents the number of features present in that particular dataset. The initialization of the population is done as follows in Eqs. 6.12 and 6.13:

Initialization: Continuous population ($CPOP_{[N\times M]}$) is initialized as follows Eq. 6.12

$$CPOP_{[N\times M]} = Min + rand(0,1) * (Max - Min) \tag{6.12}$$

Where $rand(0,1)$ is a random vector in the interval of [0,1]. The maximum and minimum values of each feature are denoted by $Max_{[1\times M]}$ and $Min_{[1\times M]}$, respectively.
Updating $CPOP$ into the range of 0 to 1 by following Eq. 6.13:

$$CPOP_{N\times M} = \begin{cases} rand(0,0.5), & \text{if } CPOP < Mean(CPOP). \\ rand(0.5,1), & \text{otherwise.} \end{cases} \tag{6.13}$$

Computing Binary population ($BPOP_{[N\times M]}$) by following Eq. 6.14:

$$BPOP_{N\times M} = \begin{cases} 1, & \text{if } rand(0,1) < CPOP \\ 0, & \text{otherwise} \end{cases} \tag{6.14}$$

Updation: The fitness score for each solution within the population is computed using a multi-objective decision-making technique called the weighted sum method (WSM). Next, a mood is selected for each solution based on rank and updated according to the Eqs. 6.7 - 6.10 in every iteration, resulting in updated solutions. The process involves conducting mathematical operations on the continuous solution before converting it into a binary solution. The transfer function defines the probability of updating a solution from continuous to binary. This approach transforms the updated solution within each mood into a binary solution using the sinusoidal (wave-shaped) transfer function as given in Eq. 6.15. Subsequently, the fitness score of the updated binary solution is calculated and compared with the current solution, if it is superior to the current one it will be updated. After each iteration, a locally optimal feature subset is identified, and ultimately, the RiBSNS outputs the optimal feature set for a specific dataset, which is then utilized in the subsequent optimization stage.

$$\begin{aligned} Probt &= \frac{1 + Sin(2 * \pi * CPOP_{inew})}{2} \\ BPOP_{inew} &= \begin{cases} 1 - BPOP_i, & \text{if } rand(0,1) < Probt \\ BPOP_i, & \text{otherwise} \end{cases} \end{aligned} \tag{6.15}$$

Further, the modifications employed in RiBSNS are explained, such as the ranking moods method, multi-objective fitness function, and weighted sum method.

Ranking Moods: To expedite the reach of the global optimum value, we introduced dynamic selection within the SNS algorithm. Despite choosing moods at random, assigning ranks to moods and selecting them based on ranks will enhance the learning ability of the SNS algorithm. All mood ranks are initially set to zero, resulting in an equal random chance of mood selection in the first iteration. In the first iteration, the rank of the specific selected mood increments by one for each solution if the updated solution's fitness value is better than the current solution; otherwise, the rank decreases by one. This procedure establishes individual ranks for all four moods by the end of the first iteration. Subsequently, moods with the highest ranks are chosen from the second iteration onwards to update each solution. It's because moods with higher ranks are preferred for updating solutions, as they are more likely to lead to improved fitness. In the following iterations, every solution undergoes a similar process to update ranks until the termination criterion is met. In our study, the termination criterion is the number of iterations. The training performance determines the iteration count for each dataset and varies from one dataset to another.

Multi-objective Fitness function: Our motivation behind performing feature selection in this study is to reduce as many features as possible, which has to maximize the estimation accuracy and reduce the complexity of the ANFIS model. In the RiBSNS optimization technique, the fitness function comprises three objectives: minimizing MAE, maximizing Adjusted R^2, and minimizing the number of features. Linear regression estimates the effort for each binary solution within the population. During regression model training, only the features corresponding to 1's in the binary solution are utilized, and the corresponding effort is estimated. The resultant estimated effort is used to calculate MAE and Adjusted R^2 values. The WSM [175] method computes the aggregated fitness values to satisfy these three objectives of the problem.

6.2.2 Stage 2: Multi-objective RiCSNS for ANFIS Parameters Optimization

Algorithm 12: Multi-objective RiCSNS for ANFIS parameters optimization

```
   Require: $Tr[optimal\_feature\_set], Te[optimal\_feature\_set]$;
   Ensure: $Effort_{pred}$;
1  function
    RiCSNS_ANFIS($Tr[optimal\_feature\_set], Te[optimal\_feature\_set]$)
2      $Testing\_data$ ← Te[optimal_feature_set]
3      $Training\_data$ ← Tr[optimal_feature_set]
4      Test_len ← len($Testing\_data$)
5      Traint_len ← len($Training\_data$)
6      for each $project(P)$ in $Testing\_data$ do
7          $NN\_set$ ← Nearest projects of $P$ in $Training\_data$
8          $optimal\_parameters$ ← training RiCSNS based ANFIS model over
            $NN\_set$ with multi-objective(MAE and Adj R2) optimization
9          $Effort_{pred}[P]$ ← ANFIS model with $optimal\_parameters$ estimated
            effort for testing project $P$
10     end
11     return $Effort_{pred}$
12 end
```

The default ANFIS learning algorithm employs gradient descent and least squares estimation to fine-tune the premise and consequent parameters. However, gradient descent encounters the limitation of getting stuck in local minima and slower convergence. The RiCSNS algorithm is utilized in this study to enable efficient ANFIS parameter optimization. Algorithm 12 gives a comprehensive outline of ANFIS training with RiCSNS. The training and testing data are pruned with the selected optimal feature set. The ANFIS model trains using pruned data reduces the complexity of the ANFIS model. The ANFIS model trains with its nearest neighbours from training data for each testing sample by adjusting the premise and consequent parameters. Then, the ANFIS model with optimal parameters estimates the testing sample effort. As per how much neighbourhood has to be considered, large datasets (>100 projects) took 20%, and small datasets (<100 projects) took up to 50% nearest samples from training data. The process repeats for each test sample until the testing set ends. The ANFIS model trained with the nearest samples will enhance the estimation capability by precisely adjusting the tunable parameters rather than training with the entire data. In the training phase, the ANFIS model's parameters are optimized by the RiCSNS algorithm. The operations of RiCSNS are analogous to the RiBSNS, with the exception of binary population conversion. Population size varies across datasets, and the number of attributes is equal to the number of premise and consequent parameters. The ranges of these parameters are manually adjusted according to the performance observed during the training phase. The objective is to minimize MAE and maximize Adj R^2; thus, the fitness value used in RiCSNS is computed using the WSM with two objectives. Throughout the training phase, for each solution (comprising a set of tunable parameters) within the population and during solution updates, these solutions are inputted into the ANFIS model for training, returning estimated effort. From estimated effort, MAE and Adj R^2 are calculated, subsequently serving as the fitness score within the RiCSNS algorithm. Then, the trained ANFIS model estimates the required effort for each testing sample using those optimal parameters. The estimated effort values of all the testing samples are then used to compute performance measures by comparing them with the actual effort values.

6.3 Experimental Setup

As part of the study, a few performance assessment measures are collected and evaluated using statistical tests across a few publicly accessible effort estimation projects to assess the performance of the proposed TSoptEE model. Then, we compare the outcome with a few base models.

6.3.1 Experimental Objects

We extracted nine benchmark effort estimation datasets to assess the effectiveness of the proposed model, which exhibits different characteristics [46, 47, 52, 176, 177]. A descriptive summary of all nine datasets, including the total number of projects (instances or size), the number of features (attributes), and units of software effort, is provided in Table 6.1. The effort of each dataset is not evenly distributed and presents a challenge for the advancement of software project development effort estimation.

Table 6.1: Description of the Datasets

No.	Datasets	#project	#feature	Units
1	albrecht	24	8	months
2	china	499	18	hours
3	cocomo81	63	17	months
4	desharnais	81	11	hours
5	kemerer	15	7	months
6	kitchenham	145	4	months
7	maxwell	62	27	hours
8	miyazaki	48	8	months
9	Telecom1	18	3	months

6.3.2 Performance Assessment Measures

This section describes various performance indicators such as RMSE, MAE, MMRE, MdMRE, BMMRE, PRED (0.25), and Adj R^2 to validate the efficiency of the proposed TSoptEE model compared to other estimation models [46, 47, 52, 134, 135, 136, 176]. Due to the asymmetric distribution of MRE, measures which depended on MRE are often claimed to be biassed by numerous studies [178, 179]. The model can't be chosen just based on these measures. Despite this, our study incorporated them for comparative purposes since many studies have reported. All these performance assessment measures are defined in Eqs. 6.16 - 6.24:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(AE_i - EE_i)^2} \tag{6.16}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|AE_i - EE_i| \tag{6.17}$$

$$\text{MRE} = \frac{|(AE_i - EE_i)|}{AE_i} \tag{6.18}$$

$$\text{MMRE} = \frac{1}{n}\sum_{i=1}^{n}\text{MRE}_i \tag{6.19}$$

$$\text{MdMRE} = \text{Median}(\text{MRE}) \tag{6.20}$$

$$\text{BMMRE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{AE_i - EE_i}{\text{Min}(AE_i, EE_i)}\right| \tag{6.21}$$

$$\text{PRED} = \frac{A}{n}, \qquad \text{A} = \sum_{i=1}^{n} \begin{cases} 1, & \text{if } \text{MRE}_i \leq 0.25 \\ 0, & \text{otherwise.} \end{cases} \tag{6.22}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(AE_i - EE_i)^2}{\sum_{i=1}^{n}(AE_i - \bar{AE})^2} \tag{6.23}$$

$$\text{Adj } R^2 = 1 - \frac{(1-R^2)(n-1)}{n-m-1} \tag{6.24}$$

Here, AE_i and EE_i are the actual and estimated efforts, respectively; 'n' and 'm' represent the count of projects and the count of independent features in a dataset. The parameter 'A' in PRED represents the number of projects with less than or equal to 0.25 MRE value. A model must produce lower values for employed measures except PRED and Adj R^2 to achieve relative estimation accuracy.

Interpreting these measures without the support of statistical tests can introduce instability in conclusions. Therefore, we employed WSR [157, 180, 181] test to compare the performance statistically. In this study, the null hypothesis assumes that two compared models are statistically similar under a 95% confidence level (significance level α = 0.05) if the computed P-value is greater than α, otherwise, a significant difference exists. Moreover, to measure relationship strength (effect size), the matched-pairs rank biserial correlation coefficient (Ec) technique [159, 182] is adopted. To summarize the comparison results among estimation models, we employed Win-Draw-Loss (WDL) statistics.

6.4 Experimental Results and Analysis

This section presents the findings of our experiments. To demonstrate the superiority of the proposed model, we perform comparisons with a few fundamental regression models, including linear regression (LR), ridge regression (RR), classification and regression tree (CART), and a simple artificial neural network (ANN) model. Additionally, compared with three recently developed estimation approaches such as the neuro-fuzzy based ANFIS-SBO model [46], the analogy-based BABE model [47], and the CBR-GA model [52]. The results are reported in Tables 6.2 - 6.8, in terms of RMSE, MAE, MMRE, MdMRE, BMMRE, PRED, and Adj R^2, the bold cell in each row represents the best measure score of the model for a particular dataset. The last three rows represent the statistical comparison between the TSoptEE and other base models. We conducted statistical comparisons using the WSR test, which returns the P-value, positive rank sum (W+) of TSoptEE, negative rank sum (W-) of other compared models, and effect size (Ec). The bold P-value (< **0.05**) signifies that our model is statistically superior to the compared model, while bold and italic P-value (> ***0.05***) indicate that our model and the compared models are statistically similar. In addition, the bar plots shown in Figure 6.3 for the comparison of average measure values over all the datasets.

Table 6.2: RMSE results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	14.996	17.693	16.863	13.697	20.909	9.1905	8.3734	**6.3429**
china	1169.3	1074.5	1766.5	**1025.7**	2558.8	1987.1	3119.7	1045.5
cocomo81	1814.3	1423.7	1480.2	1504.9	3665.8	1410.5	1385.1	**1192.4**
desharnais	3009.8	**2789.9**	4049.7	4140.8	3585.3	3137.5	3892.7	3489.6
kemerer	321.51	222.39	270.02	226.83	140.15	209.32	231.69	**117.32**
kitchenham	1838.2	1956.5	5833.2	2217.8	3124.1	5560.0	**1377.9**	1600.0
maxwell	7639.1	7922.7	7656.9	6303.9	7196.5	7298.4	7065.2	**5275.6**
miyazaki	118.83	119.19	152.27	111.45	118.73	86.314	158.23	**94.861**
Telecom1	132.25	145.26	139.80	135.14	112.38	119.81	**98.594**	101.51
P-value	***0.066316***	***0.085831***	**0.007686**	**0.020879**	**0.0076858**	***0.085831***	***0.085831***	
Effect size	0.689	0.644	1	0.867	1	0.644	0.644	
W+/W-	38/7	37/8	45/0	42/3	45/0	37/8	37/8	

Table 6.3: MAE results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	11.286	11.853	11.049	11.951	11.173	7.613	5.597	**4.8850**
china	407.92	413.59	573.71	389.81	1250.6	762.54	1515.8	**355.26**
cocomo81	1104.6	711.06	667.03	539.25	2348.5	569.54	575.48	**432.44**
desharnais	2128.3	**2111.1**	2781.8	2715.8	2301.6	2118.3	2722.7	2419.5
kemerer	263.91	166.72	169.75	155.77	**95.123**	133.44	143.32	97.009
kitchenham	816.69	827.24	1459.8	995.54	918.63	1240.7	827.20	**795.52**
maxwell	5623.6	5613.2	4671.7	3760.8	4008.9	4231.1	4410.5	**3450.3**
miyazaki	46.382	48.932	60.344	51.629	47.750	43.813	48.550	**41.820**
Telecom1	106.71	110.04	89.264	89.928	76.276	82.971	67.759	**62.664**
P-value	***0.066316***	***0.085831***	**0.007686**	**0.007686**	**0.049103**	**0.049103**	**0.007686**	
Effect size	0.689	0.644	1	1	0.733	0.733	1	
W+/W-	38/7	37/8	45/0	45/0	39/6	39/6	45/0	

The results of applying the proposed model to all datasets with respect to the RMSE evaluation measure compared with the other base models are shown in Table 6.2. It's notable that TSoptEE consistently produces lower RMSE values compared to other estimation models across five of the nine datasets, specifically Albrecht, Cocomo81, Kemerer, Maxwell, and Miyazaki. The results of the remaining four datasets are comparable. Moreover, even though with some datasets, other models have slightly better values, they aren't statistically superior to our model. As we can see,

Table 6.4: MMRE results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	1.3368	0.8554	1.0667	1.3668	0.6241	**0.4070**	0.8231	0.6624
china	0.2262	0.2437	0.1322	0.1354	0.3024	0.2261	0.4452	**0.0986**
cocomo81	19.823	11.429	1.5635	0.6968	**0.5797**	2.3414	1.5875	1.2945
desharnais	0.5980	0.6510	0.6731	0.5463	0.5447	0.4646	0.7250	**0.4422**
kemerer	2.0579	1.1420	0.7914	0.8225	**0.4712**	0.7335	0.7937	0.5692
kitchenham	0.5256	0.5052	0.3501	0.5092	**0.2581**	0.2994	0.3665	0.2668
maxwell	1.4219	1.6956	0.7210	0.5168	**0.4029**	0.5614	0.7412	0.5453
miyazaki	0.7372	0.7734	0.6897	0.7223	0.5894	**0.3085**	0.6270	0.3849
Telecom1	0.7264	0.6894	0.3842	0.3767	0.3236	0.3557	**0.2157**	0.3909
P-value	**0.007686**	**0.007686**	**0.010862**	***0.173071***	***0.7670969***	***0.51467***	**0.020879**	
Effect size	1	1	0.956	0.511	0.111	0.244	0.867	
W+/W-	45/0	45/0	44/1	34/11	20/25	28/17	42/3	

Table 6.5: MdMRE results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	0.6675	0.7381	0.5391	0.7672	0.3619	0.4341	**0.3224**	0.6013
china	0.1001	0.1046	0.0837	0.0928	0.2882	0.1449	0.3803	**0.0748**
cocomo81	5.5056	3.3888	0.7522	0.5337	**0.3509**	0.7435	0.8336	0.6421
desharnais	0.3644	0.3915	0.3811	0.4668	0.3199	0.3194	0.3945	**0.3028**
kemerer	1.8401	0.6467	0.5741	0.5205	0.3357	0.4605	0.6521	**0.3020**
kitchenham	0.3074	0.2976	0.2427	0.3711	**0.1768**	0.1827	0.2732	**0.1768**
maxwell	0.9116	0.9770	0.5088	0.4699	**0.3973**	0.4507	0.5658	0.4009
miyazaki	0.4028	0.5223	0.5016	0.5194	**0.2229**	0.2390	0.4458	0.2961
Telecom1	0.4618	0.3773	0.2414	0.2236	0.2232	0.2017	0.2377	**0.2005**
P-value	**0.007686**	**0.007686**	**0.020879**	**0.028402**	***0.767097***	***0.313938***	***0.066316***	
Effect size	1	1	0.867	0.822	0.111	0.378	0.689	
W+/W-	45/0	45/0	42/3	41/4	20/25	31/14	38/7	

our model exhibits statistical superiority (P-value <0.05 and W+ value is greater than the W-) over CART, ANN, and ANFIS-SBO while being similar to other models with medium range effect sizes, but any other model is not statically superior to TSoptEE. Even in cases with P-value > 0.05, our model's rank sum (W+) is much higher than the other model's rank sum (W-) (TSoptEE compared to LR, RR, BABE and CBR-GA, 31 highest rank sums are there, respectively). Each dataset yields different outcomes over a single measure, so we conducted a comparison of the average outcomes across all datasets, as depicted in Figure 6.3. When considering the average RMSE values of all the models in Figure 6.3, it is evident that TSoptEE consistently outperforms the other models, producing significantly lower average RMSE values.

Table 6.3 presents the results of the MAE evaluation measure across all models. As we can observe, the proposed model consistently produces lower MAE values than the other estimation models across seven of the nine datasets, namely Albrecht, China, Cocomo81, Kitchenham, Maxwell,

Table 6.6: BMMRE results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	**0.6203**	0.6587	1.7625	1.6474	0.9266	0.6979	1.3527	0.6658
china	0.2940	0.3962	0.1516	0.1382	0.4144	0.2623	0.4513	**0.0991**
cocomo81	4.5077	3.4031	2.5152	2.6676	**0.8206**	6.0743	3.3011	1.7504
desharnais	0.3697	**0.2789**	0.8386	1.0999	0.7909	0.5819	0.9190	0.6331
kemerer	0.8415	1.0926	1.2042	0.9583	0.5959	1.0539	1.2225	**0.5692**
kitchenham	0.3529	2.5782	0.5057	0.5304	0.3417	0.4145	0.4634	**0.3373**
maxwell	**0.0419**	0.4557	0.8994	0.7928	0.9716	0.9802	1.3439	0.7848
miyazaki	**0.4449**	2.3445	1.0838	0.7957	0.8436	0.6183	0.8937	0.4946
Telecom1	0.4786	0.3739	0.4415	0.4156	0.3764	0.3920	**0.2934**	0.4147
P-value	*0.678402*	*0.213524*	**0.007686**	**0.007686**	*0.2135244*	**0.028402**	**0.010862**	
Effect size	0.156	0.467	1	1	0.467	0.822	0.956	
W+/W-	26/19	33/12	45/0	45/0	33/12	41/4	44/1	

Table 6.7: PRED results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	0.2083	0.2083	0.2417	0.25	0.4167	0.1250	**0.4583**	0.3750
china	0.7495	0.7455	0.8858	0.8979	0.3668	0.7417	0.6086	**0.9879**
cocomo81	0.1270	0.1270	0.1905	0.1746	0.3210	0.1111	0.0794	**0.2063**
desharnais	0.2963	0.3457	0.3333	0.1975	0.2710	0.4074	0.2840	**0.4444**
kemerer	0.0667	0.1333	0.1333	0.2667	0.3333	0.2000	0.2667	**0.4000**
kitchenham	0.4766	0.4756	0.5378	0.3043	**0.6620**	0.6410	0.4742	0.6349
maxwell	0.1758	0.2121	0.2697	0.2424	**0.3561**	0.2881	0.2303	0.3045
miyazaki	0.3333	0.2292	0.2917	0.2083	**0.5**	**0.5000**	0.2533	0.3750
Telecom1	0.3889	0.3333	0.4444	0.5556	**0.6667**	**0.6667**	0.5000	**0.6667**
P-value	**0.007686**	**0.007686**	**0.007686**	**0.007686**	*0.952765*	*0.109745*	**0.015156**	
Effect size	1	1	1	1	0.056	0.667	0.911	
W+/W-	45/0	45/0	45/0	45/0	19/17	30/6.0	43/2	

Miyazaki, and Telecom1. For the remaining two datasets, our model yields results that are very close to the best-performing model. In addition, our proposed model demonstrates statistical superiority over CART, ANN, ANFIS-SBO, BABE, and CBR-GA with large effect sizes (1, 1, 0.733, 0,733, and 1, respectively) while being similar to other models with medium effect sizes. In terms of MAE, TSoptEE is statistically similar to LR and RR with medium effect sizes (0.689 and 0,644) and a high positive rank sum (38 and 37). Moreover, we can observe from Figure 6.3 that the average MAE value for TSoptEE is consistently lower than the other models. We can see significant differences between TSoptEE and other models from Tables 6.2 and 6.3, the lowest RMSE and MAE values are achieved by TSoptEE, and the following lowest values are obtained by CBR-GA, due to the fitness function which is considered while optimizing the number of features and ANFIS parameters.

Table 6.4 presents the results with respect to the MMRE evaluation measure across all models.

Table 6.8: Adj R^2 results in comparison with other models

Datasets	LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
albrecht	-2.1439	-2.4542	-3.6725	-0.9692	-3.6729	0.3426	0.3243	**0.4978**
china	0.9611	0.9693	0.9098	0.9662	0.8146	0.9174	0.3224	**0.9716**
cocomo81	-5.4596	-3.4167	-2.3444	-3.3073	-0.1956	-3.2546	-3.8767	**0.5086**
desharnais	0.1809	0.3691	-0.5001	-0.5276	-0.1917	0.0674	0.0749	**0.3772**
kemerer	29.6706	8.9230	16.7523	9.0235	2.5911	5.4699	-0.4546	**-0.1579**
kitchenham	0.8743	**0.8781**	0.4169	0.7930	0.7762	0.5748	0.7518	0.8428
maxwell	8.9791	9.4819	3.9165	2.6046	2.9528	3.3445	3.4575	**0.6039**
miyazaki	0.2070	0.1043	0.18306	-0.4715	**0.8099**	0.1061	0.2140	0.2776
Telecom1	0.2480	0.2514	0.2803	0.3619	0.3657	0.4093	0.4231	**0.5668**

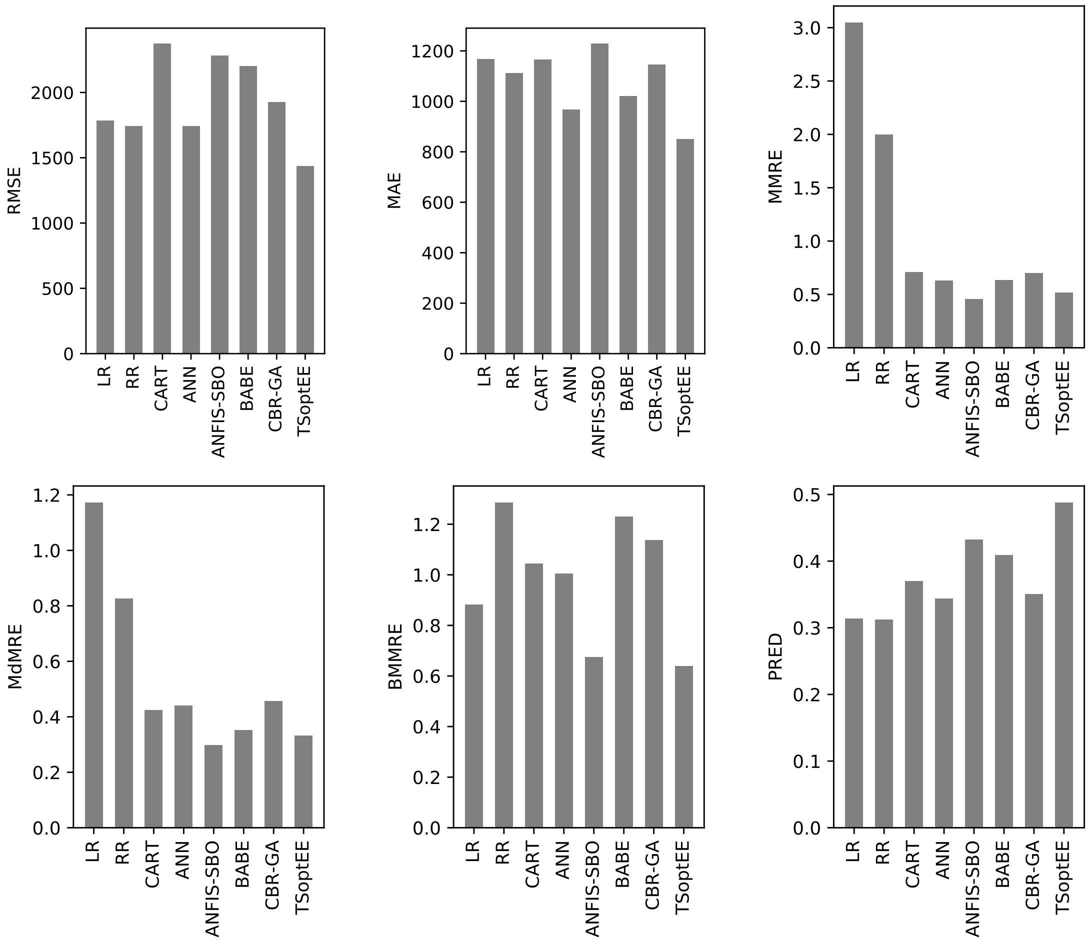


Figure 6.3: Average values across all datasets for each model and performance measure

We can generally notice that the proposed model consistently produces lower MMRE values than the other estimation models across two of the nine datasets, namely China and Desharnais. For the remaining five datasets, our results are comparable and not inferior to the best-performing models. In addition, our model demonstrates statistical superiority over LR, RR, CART, and CBR-GA with

Table 6.9: Win (W), draw (D), loss (L) comparison of all models

Measures		LR	RR	CART	ANN	ANFIS-SBO	BABE	CBR-GA	TSoptEE
RMSE	W	0	0	0	0	0	1	0	**3**
	D	7	7	5	6	6	6	7	4
	L	0	0	2	1	1	0	0	0
MAE	W	0	0	0	1	0	1	0	**5**
	D	7	7	4	5	6	5	6	2
	L	0	0	3	1	1	1	1	0
MMRE	W	0	0	2	1	**5**	2	1	4
	D	1	3	3	5	2	5	4	3
	L	6	4	2	1	0	0	2	0
MdMRE	W	0	0	2	0	**4**	**4**	0	**4**
	D	3	3	2	6	3	3	5	3
	L	4	4	3	1	0	0	2	0
BMMRE	W	1	0	0	0	1	0	0	**4**
	D	6	7	6	6	6	5	5	3
	L	0	0	1	1	0	2	2	0
PRED	W	0	0	2	0	0	1	0	**5**
	D	4	5	4	6	7	6	6	2
	L	3	2	1	1	0	0	1	0
All Measures	W	1	0	6	2	10	9	1	25
	D	28	32	24	34	30	30	33	17
	L	13	10	12	6	2	3	8	0
	W-L	-12	-10	-6	-4	8	6	-7	**25**

large effect sizes (1, 1, and 0.956, respectively) while being similar to ANN, ANFIS-SBO, and BABE models with small to medium effect sizes (0.511, 0.111, and 0.244, respectively). Additionally, when analyzing Figure 6.3, it is observed that TSoptEE achieves the second lowest average MMRE value compared to other models. The ANFIS-SBO model achieves the lowest average MMRE value, as it prioritizes the minimization of MMRE while optimizing ANFIS parameters through the SBO algorithm.

Table 6.5 presents the results with respect to the MdMRE evaluation measure across all models. We can generally notice that the proposed model consistently produces lower MdMRE values than the other estimation models over five of nine datasets, namely China, Desharnais, Kemerer, Kitchenham, and Telecom1. At the same time, the other four dataset's results are very close to the best-performing model. In addition, our model demonstrates statistical superiority over LR, RR, CART, and ANN with large effect sizes (1, 1, 0.867, and 0.822, respectively) while being similar to ANFIS-SBO, BABE, and CBR-GA models with small to medium effect sizes (0.111,0.378, and 0.689, respectively). Moreover, when analyzing Figure 6.3, it is observed that TSoptEE achieves the second lowest average MdMRE value compared to other models. ANFIS-SBO model attains

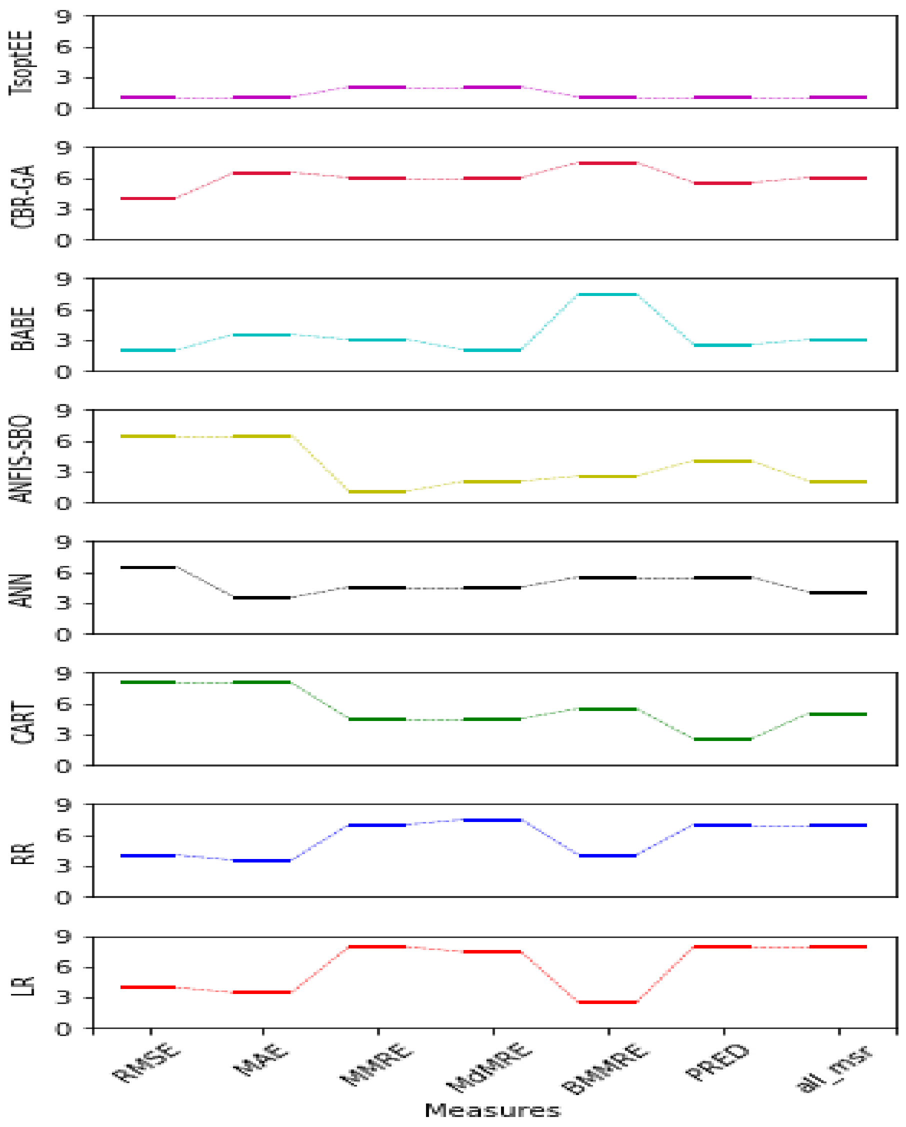


Figure 6.4: Ranks of all models

the lowest average MdMRE value.

Table 6.6 presents the results with respect to the BMMRE evaluation measure across all models. We can generally notice that the proposed model consistently produces lower BMMRE values than the other estimation models over three of nine datasets, namely China, Kemerer, and Kitchenham. We obtained results comparable to those of the best model for the remaining six datasets. In addition, we can observe that our model is statistically superior to CART, ANN, BABE, and CBR-GA with large effect sizes (1, 1, 0.822, and 0.956, respectively) while being similar to LR, RR, and ANFIS-SBO models with small to medium effect sizes (0.156, 0.467, and 0.467, respectively) and having highest rank sums comparatively. Moreover, we can observe from Figure 6.3 that the average BMMRE value is lower for TSoptEE compared to other estimation models.

Table 6.7 provides an overview of the results with respect to the PRED evaluation measure across all models. In the case of PRED, a higher value indicates better model performance. We can generally notice that the proposed TSoptEE model generated better predictions than other estimation models over five out of nine datasets, namely China, Cocomo81, Desharnais, Kemerer, and Telecom1. The remaining four datasets obtained almost similar results to the best-performing model. In addition, we can observe that our model is statistically superior to LR, RR, CART, ANN, and CBR-GA with large effect sizes (1, 1, 1, 1, and 0.911, respectively) while being similar to ANFIS-SBO and BABE models with small and medium effect sizes (0.056 and 0.667, respectively) and high positive rank sums. Moreover, we can observe from Figure 6.3 that the average PRED value for TSoptEE is the highest among all the estimation models, indicating its overall superior predictive performance.

Table 6.8 presents an overview of the results with respect to the Adjusted R^2 evaluation measure across all models. The higher Adjusted R^2 value indicates better model performance, it should ideally range between 0 and 1. We can generally notice that the proposed TSoptEE model is able to generate better predictions than other estimation models over seven out of nine datasets, namely Albrecht, China, Cocomo81, Desharnais, Kemerer, Maxwell, and Telecom1, in terms of Adj R^2. We obtained almost identical results to the best-performing model for the remaining two datasets. On the other hand, it's essential to address the presence of negative values and values exceeding 1 for the Adj R^2. Adj R^2 measures the percentage of variance in the target explained by the input variables [183]. When the number of samples and independent variables is close, the denominator approaches zero, which can result in exceeding one or becoming negative [184]. An increase in the Adj R^2 value only occurs when the inclusion of a new independent variable significantly improves the model fit beyond what it had previously; otherwise, it decreases [183]. The proposed TSoptEE model achieves superior Adj R^2 values compared to other models without encountering negative or greater than one value. This can be attributed to its feature selection capability, which enables it to select features that enhance the model's fit. TSoptEE also tries to maximize Adj R^2 during feature selection and ANFIS parameter optimization by considering it as part of the fitness score of the optimization technique. Consequently, TSoptEE demonstrates strong predictive performance in terms of Adj R^2 and other estimation measures. It's worth noting that statistical analysis over negative values is not possible. Instead, a straightforward comparison is made in Table 6.8 without statistical testing due to the prevalence of negative values in the base models.

As we can observe from Figure 6.3, RMSE, MAE, MMRE, MdMRE, BMMRE, and PRED average values range between 1400 to 3000, 900 to 1200, 0.40 to 3.0, 0.3 to 1.2, 0.6 to 1.3, and 0.3 to 0.5, respectively. Our TSoptEE achieves the lowest average over RMSE, MAE, and BMMRE and the highest PRED score. Regarding MMRE and MdMRE, the second lowest was achieved by TSoptEE because we optimized MAE and Adj R^2 as our objective functions in this study, but in the compared models, MMRE was considered a minimizing objective. According to past studies, MMRE is a biased measure [185, 47]. So, in this study, minimizing MMRE isn't chosen as the objective. Moreover, statistical comparison among all the measures and their ranks are depicted in Figure 6.4. To provide a comprehensive overview of our analysis and compare the predictive performance of our model against the base models, we employed the WDL technique [182]. The results of applying the WDL algorithm to all possible experimental outcomes are detailed in Table 6.9. These results reveal that our proposed TSoptEE outperformed other models with substantial wins across all measures except for MMRE. Regarding MMRE, ANFIS-SBO has the highest wins with only one higher value than TSoptEE. The last row summarizes the aggregated wins, draws, losses, and wins-losses over all the combined measures. It becomes evident that TSoptEE stands out as a reliable model, with a large number of wins and zero losses. Particularly, the TSoptEE achieved 25 wins, 17 draws, and 0 losses out of 42 comparisons. This large number of wins indicates the significant differences between the proposed TSoptEE and other base models and shows its competitive performance capability. Notably, we also observed that only ANFIS-SBO and BABE managed to attain positive win-losses.

Furthermore, we established ranks for all models by analyzing each model's performance using the wins-losses values depicted in Figure 6.4. The X-axis and Y-axis in Figure 6.4 represent each model's performance metrics and their corresponding ranks. The 'all_msr' signifies the rank achieved through averaging wins-losses across all performance metrics. The first rank is assigned to the highest wins-losses value, followed by ascending ranks for subsequent values. Figure 6.4 illustrates that TSoptEE consistently ranks within the top two positions across all performance measures, and on average (all_msr) obtained the top rank. Additionally, it is noticeable that LR and RR share similar rankings, while ANN obtains average ranks across all measures. Regarding RMSE and MAE, CART and ANFIS-SBO obtain lower ranks, but their ranks in terms of other measures are comparable. The BABE and CBR-GA achieve average rankings for all performance measures, except for BMMRE, where its rank is notably higher compared to others. Overall, while some models exhibit superior performance in specific measures only, the proposed TSoptEE model stands out by delivering the best performance across all the performance measures through accurate estimation of the required effort.

6.5 Summary

In the field of software engineering, accurately estimating the effort required for software development has always been a challenging task. In this chapter, we proposed a two-stage TSoptEE model for effort estimation. Firstly, the multi-objective RiBSNS algorithm performs a feature selection, which selects the most influential features contributing to effort estimation. The selected features are fed into ANFIS and then the multi-objective RiCSNS algorithm performs ANFIS parameter optimization to evaluate the accuracy of the TSoptEE model. We evaluate our approach using nine publicly accessible benchmark datasets and measure performance using metrics like RMSE, MAE, MMRE, MdMRE, BMMRE, PRED (0.25), and Adj R^2. The results were compared with those basic regression models and recently published effort estimation models. Based on the findings, it can be concluded that the TSoptEE model is more accurate than the existing models. Finally, the statistical WDL analysis of the results supported the superiority of the proposed model over the other models. According to the overall findings, it can be concluded that the method introduced in this study can enhance SDEE accuracy. In the next chapter, the conclusion and future scope of the thesis are presented.

Chapter 7

Conclusion and Future Directions

This chapter presents the summary of this thesis, the conclusion of each objective and the future scope for further direction of research.

7.1 The Major Contributions of the Thesis

This thesis explores and endeavours challenges related to software fault prediction and development effort estimation models. The research aims to create more accurate prediction models by investigating various imbalance learning techniques, feature extraction methods, transfer learning techniques, source project selection approaches, and exploration of different types of effort estimation models and optimization of model parameters. The ultimate goal is to develop an efficient early software reliability prediction model that ensures the production of reliable software without exceeding project budgets and testing costs.

Chapter 3 introduces the WACIL approach for WPFP, focusing on addressing imbalance learning. WACIL involves extracting diverse pseudo-instances around the border of the faulty class through the extraction of borderline instances, pseudo-instance generation, and noisy data elimination. Extraction of borderline instances for synthetic data generation to overcome the problem of faulty instance misclassification and increase the recognition rate. Pseudo-instance generation aids in introducing diverse pseudo-data, while undesirable or redundant data from generated pseudo-instances is eliminated in the noisy data elimination process to reduce negative influence on the performance. The WACIL technique's main objective is to reduce false positive outcomes and increase the recognition rate of faulty modules. Experimental results demonstrate the superiority of WACIL over existing approaches.

In Chapter 4, the WPSTC approach is proposed for CPFP. It begins with employing the WSR test to select source projects based on statistical properties, maximizing the relevance of selected sources. Optimized training data construction follows, involving novel instance filtering to select similarly distributed training data to minimize the distribution gap and imbalance rate and feature extraction using the Binary-RAO algorithm to further improve the model's prediction capability. WPSTC aims to maximize fault prediction performance in cross-project scenarios while balancing false positive and true positive outcomes. WDL statistics demonstrate that WPSTC with ensemble, KNN, and NB classifiers yields better predictions.

In Chapter 5, the novel SRES model is introduced for predicting software faults using cross-project data. SRES comprises three phases: SAPS for source project selection, resampling of training data, and SAE feature reduction. SAPS considers similarity (as mentioned in Chapter 4) and applicability among source and target projects to enhance the source projects selection process. Resampling involves oversampling data with similar characteristics to the target and undersampling data that doesn't address imbalance and distribution differences. The resampled data is then compressed using SAE to obtain optimal training data. The main objective of the SRES model is to obtain optimal training data to improve the prediction performance. SRES is evaluated on 24 software datasets, showing effectiveness in experimental comparisons against other CPFP models and basic WPFP.

In Chapter 6, the TSoptEE model is developed for SDEE. The multi-objective RiBSNS algorithm performs feature selection using an improved binary SNS algorithm, which explores the search space more effectively and selects the most influential features contributing to effort estimation. RiCSNS algorithm then explores the search space and optimizes the ANFIS parameters, which identifies the most appropriate premise and consequent parameters. We evaluate our approach using nine publicly accessible benchmark datasets and measure performance using metrics such as RMSE, MAE, MMRE, MdMRE, BMMRE, PRED (0.25), and Adj R^2. Evaluation of nine benchmark datasets shows TSoptEE's superiority over existing models, supported by statistical Win-Draw-Loss analysis.

7.2 Future Directions

The future scope for the proposed fault prediction and development effort estimation models can focus on several directions in the context of early reliability prediction:

- Future research should focus on enhancing the scalability and efficiency of the proposed models to cope with the increasing size and complexity of software datasets. Assessing extensive different development environments, the software will offer insights into the scalability, generalization ability, and reliability of these techniques in real-world scenarios.

- Collaborating with software developing companies and conducting large-scale studies on diverse real-time commercial datasets from different fields can provide valuable insights into the models' performance and impact on accurate decision-making.

- The proposed WACIL approach can be enhanced by considering software feature optimization techniques along with imbalance learning. This would enable the discovery of more diverse and optimum training data to further enhance the performance of the employed prediction models.

- The proposed fault prediction models can be modified and will apply to other areas in software reliability prediction, such as finding the number of faults during different stages of the early software development process. Finding the number of faults associated with a module can generalize the severity of a module, which can be used to give priority to modules while testing the whole software.

- Real-time newly developed software projects pose unique challenges due to their development environment, where the software develops in different environments with different module metrics that will be different from historical projects. Chapter 4 and Chapter 5 focus on recommendations between homogeneous data, where the source project and target project have similar software features in a cross-project environment. Future work can be extended to handle heterogeneous data by incorporating metric matching techniques and developing specialized architectures that can effectively process and analyze heterogeneous CPFP.

- A few datasets used for development effort estimation in Chapter 6 consist of a very small number of projects such as Telecom1, Kemerer, Albrecht, and Miyazaki, which aren't enough to effectively train the estimation model. Hence, effort estimation models should be evaluated on large-scale databases to validate their effectiveness and significance in real-world scenarios.

- In Chapter 6, the ANFIS premise and consequent parameters are optimized. In future works, ANFIS rule optimizations can be enabled to reduce the complexity of the model and improve the accuracy of software development effort estimation.

- In this thesis, we have proposed a framework for early reliability models such as fault prediction and effort estimation. However, the other reliability parameters may be considered for future work.

Bibliography

[1] Mehwish Riaz, Emilia Mendes, and Ewan Tempero. A systematic review of software maintainability prediction and metrics. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 367–377. IEEE, 2009.

[2] Sirsendu Mohanta, Gopika Vinod, AK Ghosh, and Rajib Mall. An approach for early prediction of software reliability. *ACM SIGSOFT Software Engineering Notes*, 35(6):1–9, 2010.

[3] Leslie Cheung, Roshanak Roshandel, Nenad Medvidovic, and Leana Golubchik. Early prediction of software component reliability. In *Proceedings of the 30th international conference on Software engineering*, pages 111–120, 2008.

[4] Hoang Pham. *System software reliability*. Springer Science & Business Media, 2007.

[5] Manjubala Bisi and Neeraj Kumar Goyal. *Artificial neural network applications for software reliability prediction*. John Wiley & Sons, 2017.

[6] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[7] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[8] Strategic Planning. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, 2002.

[9] Katerina Goševa-Popstojanova and Kishor S Trivedi. Architecture-based approaches to software reliability prediction. *Computers & Mathematics with Applications*, 46(7):1023–1036, 2003.

[10] Barry W. Boehm. Software risk management: principles and practices. *IEEE software*, 8(1):32–41, 1991.

[11] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[12] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*, pages 878–887. Springer, 2005.

[13] Sukarna Barua, Md Monirul Islam, Xin Yao, and Kazuyuki Murase. Mwmote–majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on knowledge and data engineering*, 26(2):405–425, 2012.

[14] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden, and Solomon Mensah. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering*, 44(6):534–550, 2017.

[15] Nutthaporn Junsomboon and Tanasanee Phienthrakul. Combining over-sampling and under-sampling techniques for imbalance dataset. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, pages 243–247, 2017.

[16] Lin Chen, Bin Fang, Zhaowei Shang, and Yuanyan Tang. Tackling class overlap and imbalance problems in software defect prediction. *Software Quality Journal*, 26:97–125, 2018.

[17] Qinbao Song, Yuchen Guo, and Martin Shepperd. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12):1253–1269, 2018.

[18] Dingxiang Liu, Shaojie Qiao, Nan Han, Tao Wu, Rui Mao, Yongqing Zhang, Chang-An Yuan, and Yueqiang Xiao. Sotb: semi-supervised oversampling approach based on trigonal barycenter theory. *IEEE Access*, 8:50180–50189, 2020.

[19] Yong Zhang, Tingting Zuo, Lichao Fang, Jun Li, and Zongyi Xing. An improved mahakil oversampling method for imbalanced dataset classification. *IEEE Access*, 9:16030–16040, 2020.

[20] Ahmad S Tarawneh, Ahmad BA Hassanat, Khalid Almohammadi, Dmitry Chetverikov, and Colin Bellinger. Smotefuna: Synthetic minority over-sampling technique based on furthest neighbour algorithm. *IEEE Access*, 8:59069–59082, 2020.

[21] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, and Miao Zhang. Investigation on the stability of smote-based oversampling techniques in software defect prediction. *Information and Software Technology*, page 106662, 2021.

[22] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010.

[23] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *2013 35th international conference on software engineering (ICSE)*, pages 382–391, 2013.

[24] Duksan Ryu, Jong-In Jang, and Jongmoon Baik. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Software Quality Journal*, 25:235–272, 2017.

[25] Lina Gong, Shujuan Jiang, Lili Bo, Li Jiang, and Junyan Qian. A novel class-imbalance learning approach for both within-project and cross-project defect prediction. *IEEE Transactions on Reliability*, 69(1):40–54, 2019.

[26] Shiqi Tang, Song Huang, Changyou Zheng, Erhu Liu, Cheng Zong, and Yixian Ding. A novel cross-project software defect prediction algorithm based on transfer learning. *Tsinghua Science and Technology*, 27(1):41–57, 2021.

[27] Hongwei Tao, Lianyou Fu, Qiaoling Cao, Xiaoxu Niu, Haoran Chen, Songtao Shang, Yang Xian, et al. Cross-project defect prediction using transfer learning with long short-term memory networks. *IET Software*, 2024, 2024.

[28] Osayande P Omondiagbe, Sherlock A Licorish, and Stephen G MacDonell. Improving transfer learning for software cross-project defect prediction. *Applied Intelligence*, pages 1–24, 2024.

[29] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14:540–578, 2009.

[30] Shang Zheng, Jinjing Gai, Hualong Yu, Haitao Zou, and Shang Gao. Training data selection for imbalanced cross-project defect prediction. *Computers & Electrical Engineering*, 94:107370, 2021.

[31] Nayeem Ahmad Bhat and Sheikh Umar Farooq. An improved method for training data selection for cross-project defect prediction. *Arabian Journal for Science and Engineering*, 47(2):1939–1954, 2022.

[32] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100, 2009.

[33] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19:167–199, 2012.

[34] Marian Jureczko and Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th international conference on predictive models in software engineering*, pages 1–10, 2010.

[35] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. Local vs. global models for effort estimation and defect prediction. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 343–351, 2011.

[36] Steffen Herbold. Training data selection for cross-project defect prediction. In *Proceedings of the 9th international conference on predictive models in software engineering*, pages 1–10, 2013.

[37] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology*, 107:125–136, 2019.

[38] Zhongbin Sun, Junqi Li, Heli Sun, and Liang He. Cfps: Collaborative filtering based source projects selection for cross-project defect prediction. *Applied Soft Computing*, 99:106940, 2021.

[39] Eunseob Kim, Jongmoon Baik, and Duksan Ryu. Heterogeneous defect prediction through correlation-based selection of multiple source projects and ensemble learning. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 503–513, 2021.

[40] Shailza Kanwar, Lalit Kumar Awasthi, and Vivek Shrivastava. Candidate project selection in cross project defect prediction using hybrid method. *Expert Systems with Applications*, 218:119625, 2023.

[41] Haonan Tong, Bin Liu, Shihai Wang, and Qiuying Li. Transfer-learning oriented class imbalance learning for cross-project defect prediction. *arXiv preprint arXiv:1901.08429*, 2019.

[42] S Kaliraj, AM Kishoore, and V Sivakumar. Software fault prediction using cross-project analysis: A study on class imbalance and model generalization. *IEEE Access*, 2024.

[43] Seyedrebvar Hosseini, Burak Turhan, and Mika Mäntylä. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Information and Software Technology*, 95:296–312, 2018.

[44] Yogita Khatri and Sandeep Kumar Singh. An effective feature selection based cross-project defect prediction model for software quality improvement. *International Journal of System Assurance Engineering and Management*, 14(Suppl 1):154–172, 2023.

[45] Saka Nanda, Benfano Soewito, et al. Modeling software effort estimation using hybrid pso-anfis. In *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 219–224. IEEE, 2016.

[46] Seyyed Hamid Samareh Moosavi and Vahid Khatibi Bardsiri. Satin bowerbird optimizer: A new optimization algorithm to optimize anfis for software development effort estimation. *Engineering Applications of Artificial Intelligence*, 60:1–15, 2017.

[47] Muhammad Arif Shah, Dayang Norhayati Abang Jawawi, Mohd Adham Isa, Muhammad Younas, Abdelzahir Abdelmaboud, and Fauzi Sholichin. Ensembling artificial bee colony with analogy-based estimation to improve software development effort prediction. *IEEE Access*, 8:58402–58415, 2020.

[48] Tirimula Rao Benala and Rajib Mall. Dabe: Differential evolution in analogy-based software development effort estimation. *Swarm and Evolutionary Computation*, 38:158–172, 2018.

[49] Aditi Sharma and Ravi Ranjan. Software effort estimation using neuro fuzzy inference system: Past and present. *arXiv preprint arXiv:1912.11855*, 2019.

[50] Amir Karimi and Taghi Javdani Gandomani. Software development effort estimation modeling using a combination of fuzzy-neural network and differential evolution algorithm. *International Journal of Electrical and Computer Engineering*, 11(1):707, 2021.

[51] Pravali Manchala and Manjubala Bisi. Ensembling teaching-learning-based optimization algorithm with analogy-based estimation to predict software development effort. In *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2022.

[52] Shaima Hameed, Yousef Elsheikh, and Mohammad Azzeh. An optimized case-based software project effort estimation using genetic algorithm. *Information and Software Technology*, 153:107088, 2023.

[53] Zygmunt Jelinski and P Moranda. Software reliability research. In *Statistical computer performance evaluation*, pages 465–484. Elsevier, 1972.

[54] Amrit L Goel and Kazu Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE transactions on Reliability*, 28(3):206–211, 1979.

[55] John D Musa and Kazuhira Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *Proceedings of the 7th international conference on Software engineering*, pages 230–238, 1984.

[56] Chin-Yu Huang, Michael R. Lyu, and Sy-Yen Kuo. A unified scheme of some nonhomogenous poisson process models for software reliability estimation. *IEEE transactions on software engineering*, 29(3):261–269, 2003.

[57] Hoang Pham. Software reliability and cost models: Perspectives, comparison, and practice. *European Journal of operational research*, 149(3):475–489, 2003.

[58] Kapil Sharma, Rakesh Garg, Chander Kumar Nagpal, and Ramesh Kumar Garg. Selection of optimal software reliability growth models using a distance based approach. *IEEE Transactions on Reliability*, 59(2):266–276, 2010.

[59] Xuemei Zhang, Xiaolin Teng, and Hoang Pham. Considering fault removal efficiency in software reliability assessment. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(1):114–120, 2003.

[60] Taghi M Khoshgoftaar and Timothy G Woodcock. Software reliability model selection: a cast study. In *Proceedings. 1991 International Symposium on Software Reliability Engineering*, pages 183–184. IEEE Computer Society, 1991.

[61] Carol Smidts, Martin Stutzke, and Robert W Stoddard. Software reliability modeling: an approach to early reliability prediction. *IEEE Transactions on Reliability*, 47(3):268–278, 1998.

[62] J Dobbins. Ieee guide for the use of ieee standard dictionary of measures to produce reliable software. *Inst. Electr. Electron. Eng. New York, NY, USA, Tech. Rep. IEEE Std 982.2–1988*, 1989.

[63] James A McCall, William Randell, Janet Dunham, and Linda Lauterbach. Software reliability, measurement, and testing software reliability and test integration. *Rome Lab., Rome, Italy, Tech. rep. RL-TR-92-52*, 1992.

[64] Vittorio Cortellessa, Harshinder Singh, and Bojan Cukic. Early reliability assessment of uml based software models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 302–309, 2002.

[65] Ralf H Reussner, Heinz W Schmidt, and Iman H Poernomo. Reliability prediction for component-based software architectures. *Journal of systems and software*, 66(3):241–252, 2003.

[66] N Raj Kiran and Vadlamani Ravi. Software reliability prediction by soft computing techniques. *Journal of Systems and Software*, 81(4):576–583, 2008.

[67] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[68] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[69] Maurice H Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[70] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, 2013.

[71] Barry W Boehm. Software engineering economics. *IEEE transactions on Software Engineering*, (1):4–21, 1984.

[72] Allan J. Albrecht and John E Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6):639–648, 1983.

[73] JM Desharnais. Analyse statistique de la productivitie des projets informatique a partie de la technique des point des function. *Masters Thesis, University of Montreal*, 1989.

[74] Yadong Dong, Huaping Guo, Weimei Zhi, and Ming Fan. Class imbalance oriented logistic regression. In *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 187–192. IEEE, 2014.

[75] David Muchlinski, David Siroky, Jingrui He, and Matthew Kocher. Comparing random forest with logistic regression for predicting class-imbalanced civil war onset data. *Political Analysis*, 24(1):87–103, 2016.

[76] David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. Using the support vector machine as a classification method for software defect prediction with static code metrics. In *International Conference on Engineering Applications of Neural Networks*, pages 223–234. Springer, 2009.

[77] Marcelo Beckmann, Nelson FF Ebecken, Beatriz SL Pires de Lima, et al. A knn undersampling approach for data balancing. *Journal of Intelligent Learning Systems and Applications*, 7(04):104, 2015.

[78] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, 2006.

[79] Burak Turhan and Ayse Bener. Analysis of naive bayes' assumptions on software fault data: An empirical study. *Data & Knowledge Engineering*, 68(2):278–290, 2009.

[80] C Manjula and Lilly Florence. Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*, 22(4):9847–9863, 2019.

[81] Gang Zhao and Jeff Huang. Deepsim: deep learning code functional similarity. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 141–151, 2018.

[82] Lei Qiao, Xuesong Li, Qasim Umer, and Ping Guo. Deep learning based software defect prediction. *Neurocomputing*, 385:100–110, 2020.

[83] Osama Al Qasem, Mohammed Akour, and Mamdouh Alenezi. The influence of deep learning algorithms factors in software fault prediction. *IEEE Access*, 8:63945–63960, 2020.

[84] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[85] Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering*, 46(11):1200–1219, 2018.

[86] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

[87] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008.

[88] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, Kwabena Ebo Bennin, Md Alamgir Kabir, and Miao Zhang. Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology*, 129:106432, 2021.

[89] C Arun and C Lakshmi. Diversity based multi-cluster over sampling approach to alleviate the class imbalance problem in software defect prediction. *International Journal of System Assurance Engineering and Management*, pages 1–13, 2023.

[90] Nasraldeen Alnor Adam Khleel and Károly Nehéz. Software defect prediction using a bidirectional lstm network combined with oversampling techniques. *Cluster Computing*, pages 1–24, 2023.

[91] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2008.

[92] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Third IEEE international conference on data mining*, pages 435–442. IEEE, 2003.

[93] Mingxia Liu, Linsong Miao, and Daoqiang Zhang. Two-stage cost-sensitive learning for software defect prediction. *IEEE Transactions on Reliability*, 63(2):676–686, 2014.

[94] Michael J Siers and Md Zahidul Islam. Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. *Information Systems*, 51:62–71, 2015.

[95] Kyung Hye Kim and So Young Sohn. Hybrid neural network with cost-sensitive support vector machine for class-imbalanced multimodal data. *Neural Networks*, 130:176–184, 2020.

[96] Wentao Wu, Shihai Wang, Bin Liu, Yuanxun Shao, and Wandong Xie. A novel software defect prediction approach via weighted classification based on association rule mining. *Engineering Applications of Artificial Intelligence*, 129:107622, 2024.

[97] Zhongbin Sun, Qinbao Song, and Xiaoyan Zhu. Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1806–1817, 2012.

[98] Haonan Tong, Bin Liu, and Shihai Wang. Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Information and Software Technology*, 96:94–111, 2018.

[99] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. Bpdet: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications*, 144:113085, 2020.

[100] Zhidan Yuan, Xiang Chen, Zhanqi Cui, and Yanzhou Mu. Altra: Cross-project software defect prediction via active learning and tradaboost. *IEEE Access*, 8:30037–30049, 2020.

[101] Nitesh V Chawla, Aleksandar Lazarevic, Lawrence O Hall, and Kevin W Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *European conference on principles of data mining and knowledge discovery*, pages 107–119. Springer, 2003.

[102] Sheng Chen, Haibo He, and Edwardo A Garcia. Ramoboost: Ranked minority oversampling in boosting. *IEEE Transactions on Neural Networks*, 21(10):1624–1642, 2010.

[103] Jinfu Chen, Jiaping Xu, Saihua Cai, Xiaoli Wang, Haibo Chen, and Zhehao Li. Software defect prediction approach based on a diversity ensemble combined with neural network. *IEEE Transactions on Reliability*, 2024.

[104] Taghi M Khoshgoftaar, Edward B Allen, Wendell D Jones, and John P Hudepohl. Data mining for predictors of software quality. *International Journal of Software Engineering and Knowledge Engineering*, 9(05):547–563, 1999.

[105] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, 36(6):1291–1302, 2003.

[106] Georgios Douzas and Fernando Bacao. Geometric smote a geometrically enhanced drop-in replacement for smote. *Information sciences*, 501:118–135, 2019.

[107] José A Sáez, Julián Luengo, Jerzy Stefanowski, and Francisco Herrera. Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, 291:184–203, 2015.

[108] Lina Gong, Shujuan Jiang, and Li Jiang. Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access*, 7:145725–145737, 2019.

[109] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461, 2006.

[110] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751–761, 1996.

[111] Lionel C Briand, Walcelio L. Melo, and Jurgen Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE transactions on Software Engineering*, 28(7):706–720, 2002.

[112] Sunjae Kwon, Duksan Ryu, and Jongmoon Baik. An effective approach to improve the performance of ecpdp (early cross-project defect prediction) via data-transformation and parameter optimization. *Software Quality Journal*, 31(4):1009–1044, 2023.

[113] Yi Bin, Kai Zhou, Hongmin Lu, Yuming Zhou, and Baowen Xu. Training data selection for cross-project defection prediction: which approach is better? In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 354–363. IEEE, 2017.

[114] Qiao Yu, Junyan Qian, Shujuan Jiang, Zhenhua Wu, and Gongjie Zhang. An empirical study on the effectiveness of feature selection for cross-project defect prediction. *IEEE Access*, 7:35710–35718, 2019.

[115] Shinya Watanabe, Haruhiko Kaiya, and Kenji Kaijiri. Adapting a fault prediction model to allow inter languagereuse. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 19–24, 2008.

[116] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54(3):248–256, 2012.

[117] Yahaya Zakariyau Bala, Pathiah Abdul Samat, Khaironi Yatim Sharif, and Noridayu Manshor. Improving cross-project software defect prediction method through transformation and feature selection approach. *IEEE Access*, 2022.

[118] Nachai Limsettho, Kwabena Ebo Bennin, Jacky W Keung, Hideaki Hata, and Kenichi Matsumoto. Cross project defect prediction using class distribution estimation and oversampling. *Information and Software Technology*, 100:87–102, 2018.

[119] Sousuke Amasaki. Cross-version defect prediction: use historical data, cross-project data, or both? *Empirical Software Engineering*, 25:1573–1595, 2020.

[120] Kun Zhu, Nana Zhang, Shi Ying, and Dandan Zhu. Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network. *IET Software*, 14(3):185–195, 2020.

[121] Chao Ni, Wang-Shu Liu, Xiang Chen, Qing Gu, Dao-Xu Chen, and Qi-Guo Huang. A cluster based feature selection method for cross-project software defect prediction. *Journal of Computer Science and Technology*, 32:1090–1107, 2017.

[122] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on software Engineering*, 42(10):977–998, 2016.

[123] Haonan Tong, Wei Lu, Weiwei Xing, and Shihai Wang. Array: adaptive triple feature-weighted transfer naive bayes for cross-project defect prediction. *Journal of Systems and Software*, 202:111721, 2023.

[124] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches—a survey. *Annals of software engineering*, 10(1-4):177–205, 2000.

[125] Boehm Barry et al. Software engineering economics. *New York*, 197:40, 1981.

[126] Lawrence H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE transactions on Software Engineering*, (4):345–361, 1978.

[127] Allan J. Albrecht and John E Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6):639–648, 1983.

[128] Roger J Best. An experiment in delphi estimation in marketing decision making. *Journal of Marketing Research*, 11(4):447–452, 1974.

[129] Walter E Riggs. The delphi technique: An experimental evaluation. *Technological forecasting and social change*, 23(1):89–94, 1983.

[130] Wen-Tin Lee, Kuo-Hsun Hsu, Jonathan Lee, and Jong Yih Kuo. Applying software effort estimation model based on work breakdown structure. In *2012 Sixth International Conference on Genetic and Evolutionary Computing*, pages 192–195. IEEE, 2012.

[131] Krishnamoorthy Srinivasan and Douglas Fisher. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, 21(2):126–137, 1995.

[132] Mohammad Azzeh, Ali Bou Nassif, and Leandro L Minku. An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. *Journal of Systems and Software*, 103:36–52, 2015.

[133] Vahid Khatibi Bardsiri, Dayang Norhayati Abang Jawawi, Siti Zaiton Mohd Hashim, and Elham Khatibi. A pso-based model to increase the accuracy of software development effort estimation. *Software Quality Journal*, 21:501–526, 2013.

[134] Pravali Manchala, Manjubala Bisi, and Sonali Agrawal. Bafs: binary artificial bee colony based feature selection approach to estimate software development effort. *International Journal of Information Technology*, 15:2975–2986, 2023.

[135] Priya Varshini AG, Anitha Kumari K, and Vijayakumar Varadarajan. Estimating software development efforts using a random forest-based stacked ensemble approach. *Electronics*, 10(10):1195, 2021.

[136] Halcyon Davys Pereira De Carvalho, Roberta Fagundes, and Wylliams Santos. Extreme learning machine applied to software development effort estimation. *IEEE Access*, 9:92676–92687, 2021.

[137] Sofian Kassaymeh, Mohammed Alweshah, Mohammed Azmi Al-Betar, Abdelaziz I Hammouri, and Mohammad Atwah Al-Ma'aitah. Software effort estimation modeling and fully connected artificial neural network optimization using soft computing techniques. *Cluster Computing*, 27(1):737–760, 2024.

[138] Muhammad Sufyan Khan, Farhana Jabeen, Sanaa Ghouzali, Zobia Rehman, Sheneela Naz, and Wadood Abdul. Metaheuristic algorithms in optimizing deep neural network model for software effort estimation. *Ieee Access*, 9:60309–60327, 2021.

[139] Mohammad Azzeh, Daniel Neagu, and Peter I Cowling. Analogy-based software effort estimation using fuzzy numbers. *Journal of Systems and Software*, 84(2):270–284, 2011.

[140] Ali Idri, Mohamed Hosni, and Alain Abran. Improved estimation of software development effort using classical and fuzzy analogy ensembles. *Applied Soft Computing*, 49:990–1019, 2016.

[141] Vidisha Agrawal and Vishal Shrivastava. Performance evaluation of software development effort estimation using neurofuzzy model. *vol*, 4:193–199, 2015.

[142] Sudhir Sharma and Shripal Vijayvargiya. An optimized neuro-fuzzy network for software project effort estimation. *IETE Journal of Research*, pages 1–12, 2022.

[143] Praynlin Edinson and Latha Muthuraj. Performance analysis of fcm based anfis and elman neural network in software effort estimation. *Int. Arab J. Inf. Technol.*, 15(1):94–102, 2018.

[144] Asad Ali and Carmine Gravino. Improving software effort estimation using bio-inspired algorithms to select relevant features: An empirical study. *Science of Computer Programming*, 205:102621, 2021.

[145] Kamal Z Zamli, Hussam S Alhadawi, and Fakhrud Din. Utilizing the roulette wheel based social network search algorithm for substitution box construction and optimization. *Neural Computing and Applications*, 35(5):4051–4071, 2023.

[146] Sachin P Patel and Sanjay H Upadhyay. Euclidean distance based feature ranking and subset selection for bearing fault diagnosis. *Expert Systems with Applications*, 154:113400, 2020.

[147] Dimitrios Pappas, Konstantinos Kiriakopoulos, and George Kaimakamis. Optimal portfolio selection with singular covariance matrix. In *International Mathematical Forum*, volume 5, pages 2305–2318, 2010.

[148] G Boetticher. The promise repository of empirical software engineering data. *http://promisedata. org/repository*, 2007.

[149] Tim Menzies, Burak Turhan, Ayse Bener, Gregory Gay, Bojan Cukic, and Yue Jiang. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 47–54, 2008.

[150] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 11–18, 2008.

[151] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, pages 179–186. Citeseer, 1997.

[152] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38, 2004.

[153] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[154] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[155] Wei Liu, Sanjay Chawla, David A Cieslak, and Nitesh V Chawla. A robust decision tree algorithm for imbalanced data sets. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 766–777. SIAM, 2010.

[156] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. *Int. J. Intell. Technol. Appl. Stat*, 11(2):105–111, 2018.

[157] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[158] Dave S Kerby. The simple difference formula: An approach to teaching nonparametric correlation. *Comprehensive Psychology*, 3:11–IT, 2014.

[159] Maciej Tomczak and Ewa Tomczak. The need to report effect size estimates revisited. an overview of some recommended measures of effect size. *Trends in sport sciences*, 1(21):19–25, 2014.

[160] Mohammad Azzeh, Yousef Elsheikh, and Marwan Alseid. An optimized analogy-based project effort estimation. *arXiv preprint arXiv:1703.04563*, 2017.

[161] Haowen Chen, Xiao-Yuan Jing, Zhiqiang Li, Di Wu, Yi Peng, and Zhiguo Huang. An empirical study on heterogeneous defect prediction approaches. *IEEE Transactions on Software Engineering*, 2020.

[162] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering*, 43(4):321–339, 2016.

[163] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[164] R Rao. Rao algorithms: Three metaphor-less simple algorithms for solving optimization problems. *International Journal of Industrial Engineering Computations*, 11(1):107–130, 2020.

[165] Karpagalingam Thirumoorthy et al. A feature selection model for software defect prediction using binary rao optimization algorithm. *Applied Soft Computing*, 131:109737, 2022.

[166] Manchala Pravali and Bisi Manjubala. Diversity based imbalance learning approach for software fault prediction using machine learning models. *Applied Soft Computing*, page 109069, 2022.

[167] Qiao Yu, Shujuan Jiang, and Junyan Qian. Which is more important for cross-project defect prediction: instance or feature? In *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, pages 90–95. IEEE, 2016.

[168] Yuming Zhou, Yibiao Yang, Hongmin Lu, Lin Chen, Yanhui Li, Yangyang Zhao, Junyan Qian, and Baowen Xu. How far we have progressed in the journey? an examination of cross-project defect prediction. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(1):1–51, 2018.

[169] Kwabena Ebo Bennin, Jacky W Keung, and Akito Monden. On the relative value of data resampling approaches for software defect prediction. *Empirical Software Engineering*, 24:602–636, 2019.

[170] Masanari Kondo, Cor-Paul Bezemer, Yasutaka Kamei, Ahmed E Hassan, and Osamu Mizuno. The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, 24:1925–1963, 2019.

[171] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, 21:2107–2145, 2016.

[172] J-SR Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3):665–685, 1993.

[173] Siamak Talatahari, Hadi Bayzidi, and Meysam Saraee. Social network search for global optimization. *IEEE Access*, 9:92815–92863, 2021.

[174] Mohd Najib Mohd Salleh, Noureen Talpur, and Kashif Hussain. Adaptive neuro-fuzzy inference system: Overview, strengths, limitations, and solutions. In *Data Mining and Big Data: Second International Conference, DMBD 2017, Fukuoka, Japan, July 27–August 1, 2017, Proceedings 2*, pages 527–535. Springer, 2017.

[175] R Timothy Marler and Jasbir S Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41:853–862, 2010.

[176] Solomon Mensah, Jacky Keung, Michael Franklin Bosu, and Kwabena Ebo Bennin. Duplex output software effort estimation model with self-guided interpretation. *Information and Software Technology*, 94:1–13, 2018.

[177] Mizanur Rahman, Teresa Goncalves, and Hasan Sarwar. Review of existing datasets used for software effort estimation. *International Journal of Advanced Computer Science and Applications*, 14(7), 2023.

[178] Ingunn Myrtveit and Erik Stensrud. Validity and reliability of evaluation procedures in comparative studies of effort prediction models. *Empirical Software Engineering*, 17:23–33, 2012.

[179] Martin Shepperd and Steve MacDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8):820–827, 2012.

[180] Geeta Nagpal, Moin Uddin, and Arvinder Kaur. Analyzing software effort estimation using k means clustered regression approach. *ACM SIGSOFT Software Engineering Notes*, 38(1):1–9, 2013.

[181] Mohammad Azzeh, Daniel Neagu, and Peter I Cowling. Analogy-based software effort estimation using fuzzy numbers. *Journal of Systems and Software*, 84(2):270–284, 2011.

[182] Pravali Manchala and Manjubala Bisi. Diversity based imbalance learning approach for software fault prediction using machine learning models. *Applied Soft Computing*, 124:109069, 2022.

[183] Magne Jørgensen. Regression models of software development effort estimation accuracy and bias. *Empirical Software Engineering*, 9(4):297–314, 2004.

[184] Martina Mittlböck. Calculating adjusted r2 measures for poisson regression models. *Computer Methods and Programs in Biomedicine*, 68(3):205–214, 2002.

[185] Ingunn Myrtveit and Erik Stensrud. Validity and reliability of evaluation procedures in comparative studies of effort prediction models. *Empirical Software Engineering*, 17:23–33, 2012.

List of Publications

Publications from the Thesis

1. **Manchala Pravali**, and Manjubala Bisi. "Diversity based imbalance learning approach for software fault prediction using machine learning models." Applied Soft Computing 124 (2022): 109069. (**Published**, Indexing: SCIE, IF: 7.2, Publisher: Elsevier)

2. **Manchala Pravali**, and Manjubala Bisi. "TSoptEE: two-stage optimization technique for software development effort estimation" Cluster Computing (2024): https://doi.org/10.1007/s10586-024-04418-2. (**Published**, Indexing: SCIE, IF: 3.6, Publisher: Springer)

3. **Manchala Pravali**, and Manjubala Bisi. "A study on cross-project fault prediction through resampling and feature reduction along with source projects selection." Automated Software Engineering 31.2 (2024): 67.(**Published**, Indexing: SCIE, IF: 2.0, Publisher: Springer)

4. **Manchala Pravali**, and Manjubala Bisi. " A source project selection based cross-project fault prediction with imbalance learning and feature selection technique" The Journal of Supercomputing. (**Revision Submitted**, Indexing: SCIE, IF: 2.5, Publisher: Springer)

Other related publications

1. **Manchala Pravali**, Manjubala Bisi, and Sonali Agrawal. "BAFS: binary artificial bee colony based feature selection approach to estimate software development effort." International Journal of Information Technology 15.6 (2023): 2975-2986. (**Published**, Indexing: Scopus, Publisher: Springer)

2. **Manchala Pravali**, Ankur Tiwari, and Manjubala Bisi. " Multi Objective Binary Rao Feature Optimization for Software Defect Prediction using Machine Learning Models" Soft Computing. (**Accepted**, Indexing: SCIE, IF: 3.1, Publisher: Springer)

3. **Manchala Pravali**, and Manjubala Bisi. "Ensembling Teaching-Learning-Based Optimization Algorithm with Analogy-Based Estimation to Predict Software Development Effort." 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), IEEE, 2022.

4. Patil Mohit, Manjubala Bisi, and **Pravali Manchala**. "Source Project Selection for Cross-Project Software Defect Prediction using Clustering Approach." IEEE 20th India Council International Conference (INDICON), IEEE, 2023.

5. Diksha Jain, **Manchala Pravali**, Sarika Mustyala and Manjubala Bisi. "Software Development Effort Estimation using SNS Algorithm" 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), IEEE, 2024.