An Empirical Study of the Impact of Experimental Settings on Defect Classification Models

by

Baljinder Ghotra

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
May 2017

Abstract

Software quality plays a vital role in the success of a software project. The probability of having defective modules in large software systems remains high. A disproportionate amount of the cost of developing software is spent on maintenance. The maintenance of large and complex software systems is a big challenge for the software industry. Fixing defects is a central software maintenance activity to continuously improve software quality. Software Quality Assurance (SQA) teams are dedicated to this task (e.g., software testing and code review) of defect detection during the software development process. Since testing or review an entire software system are time and resource-intensive. Knowing which software modules are likely to be defect-prone before a system has been deployed help in effectively allocating SQA effort.

Defect classification models help SQA teams to identify defect-prone modules in a software system before it is released to users. Defect classification models can be divided into two categories: (1) classification models that classify a software module is defective or not defective; and (2) regression models that count the number of defects in a software module. Our work is focused on training defect classification models such classification models are trained using software metrics (e.g., size and complexity metrics) to predict whether software modules will be defective or not in the future. However, defect classification models may yield different results when

the experimental settings (e.g., choice of classification technique, features, dataset preprocessing) are changed.

In this thesis, we investigate the impact of different experimental settings on the performance of defect classification models. More specifically, we study the impact of three experimental settings (i.e., choice of classification technique, dataset processing using feature selection techniques, and applying meta-learners to the classification techniques) on the performance of defect classification models through analysis of software systems from both proprietary and open-source domains. Our study results show that: (1) the choice of classification technique has an impact on the performance of defect classification models — recommending that software engineering researchers experiment with the various available techniques instead of relying on specific techniques, assuming that other techniques are not likely to lead to statistically significant improvements in their reported results; (2) applying feature selection techniques do have a significant impact on the performance of defect classification models — a Correlation-based filter-subset feature selection technique with a BestFirst search method outperforms other feature selection techniques across the studied datasets and across the studied classification techniques. Hence, recommending the application of such a feature selection technique when training defect classification models; and (3)meta-learners help in improving the performance of defect classification models, however, future studies employ concrete meta-learners (e.g., Random Forest), which train classifiers that perform statistically similar to classifiers that are trained using abstract meta-learners (e.g., Bagging and Boosting); however, produce a less complex model.

Co-authorship

Earlier versions of the work in the thesis was published as listed below:

- **Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models.**
  Baljinder Ghotra, Shane McIntosh, Ahmed E. Hassan, In Proceedings of the International Conference on Software Engineering (ICSE 2015), pp. 789-800 (Acceptance rate: 84/452 (19%))

- **A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models.**
  Baljinder Ghotra, Shane McIntosh, Ahmed E. Hassan, In Proceedings of the International Conference on Mining Software Repositories (MSR 2017), pp. To appear (Acceptance rate: 37/121 (31%))

Acknowledgments

I would like to thank all the people who contributed in some way to the work described in this thesis. First and foremost, I thank my supervisor Dr. Ahmed E. Hassan for his continuous guidance and support throughout my graduate study. Ahmed, you are the best supervisor one can have. I feel very lucky to work with you, thank you for the useful comments, remarks, motivation, and engagement through the learning process of my research journey. I would like to give special thanks to Dr. Shane McIntosh for being such a great mentor. Shane, thank you so much for all your guidance and endless effort to make my research journey memorable. I would always remember the process you teach me to conduct research and writing papers.

Many thanks to my examiners, Dr. Hossam Hassanein, Dr. Chunfang Devon Lin, and Dr. Kai T. Salomaa for their valuable and insightful feedback on my work. I am grateful to my collaborators Surafel Lemma Abebe and Nasir Ali for their guidance and ideas. I thank my colleagues, Mohamed Sami, Daniel Alencar, Chakkrit (Kla) Tantithamthavorn, and Ravjot Singh at the Software Analysis and Intelligence Lab (SAIL) for their valuable discussions.

Throughout my research journey, I received support from many dear friends in Canada as well as from back in India. I would like to thank Ankit Kaushik, Deepanshu Bhatla, Pardeep Janagal, Dollar Bansal, Rocky Singla, Aman Mehrotra, Sid Walia,

Harman Singh, and Nikhil Dhawan for being such wonderful friends. I thank Bakht Arif for always being supportive and encouraging. I would like to thank Tanvi Sethi back in India for her continuous support and encouragement throughout my years of study.

I thank my loved ones, Rupinder Ghotra (my brother), Aparna Tomar (my sister-in-law), and Rajinder Singh (my uncle), who have supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together. Thank you so much for your support during my hard times, without your support this thesis would not have been possible.

Finally, I dedicate this thesis to my beloved parents Jarnail Ghotra and Balwinder Ghotra. Without their love and support, I would not be able to complete my research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The development of large and complex software systems is a formidable challenge. With the increasing complexity of software systems, the likelihood that these systems will have defective modules is increasing. A defect is an error that leads software system to produce unexpected or incorrect results. The high quality of software systems is one of the biggest concern for an organization before releasing it to the customers. However, software systems generally contain defects [31, 59, 82, 97, 98, 108]. It is estimated that around $3.7 trillion[1] had been spending worldwide on the software development and approximately 23%[2] of the total amount was spent on the quality assurance and testing of software systems. Identifying defects after the software system is deployed usually incurs 100 times more cost than fixing them during the development phase [113]. However, the cost of fixing defects can be reduced significantly if defects are detected and fixed in the early phases of software development [6, 22, 38, 97, 98, 113, 142].

Software development life cycle includes five different phases (i.e., analysis, design,

[1] http://www.gartner.com/newsroom/id/2643919

[2] https://www.capgemini.com/resources/world-quality-report-2013-14

implementation, test, and release). The testing phase is one of the important phases in order to release defect-free software to customers. Software Quality Assurance (SQA) teams are dedicated to ensuring defect-free software systems. However, due to growing size of software systems, testing or reviewing the code of an entire software system is time and resource-intensive [4, 149]. SQA teams typically focus on a subset of system modules hence knowing which software modules are likely to be defect-prone before a system has been deployed allows practitioners to more efficiently and effectively allocate SQA efforts. This focus on a subset of modules is practical because the majority of software system defects are present in a small proportion of system modules [5, 41]. Therefore, to avoid high maintenance cost of software systems after release defect classification models are proposed to help SQA teams to prioritize their limited resources and improve the software quality.

Defect classification models are trained are trained using software features (e.g., size and complexity metrics) to predict whether software modules will be defective or not in the future [3, 30, 63, 74, 96, 97, 107, 124, 164]. Recent studies have proposed different approaches to train classification models [8, 31, 59, 82, 123] using a variety of classification techniques (from simple techniques like logistic regression [9, 19, 112] to more advanced techniques like Multivariate Adaptive Regression Splines (MARS) [12, 67], Personalized Change Classification (PCC) [67], and Meta-learners [35, 153]). Many large software makers have adopted defect classification models, for example Bell Labs [96], Microsoft Research [105–107, 163], Google [83], Blackberry [139], and IBM [21].

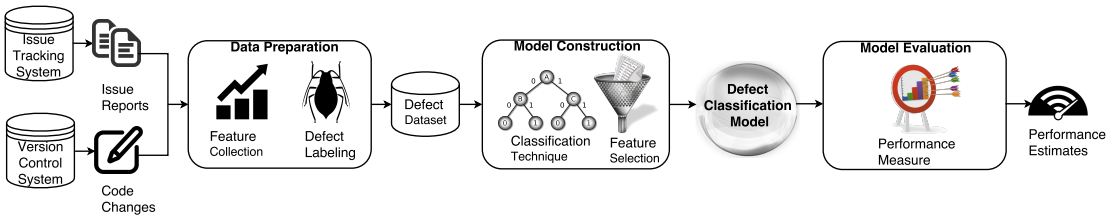

Figure 1.1: An overview of a typical defect classification model and an evaluation approach.

An overview of a baseline defect classification modeling process is shown in Figure 1.1. The training of a classification model consists of three stages: data preparation, model construction, and model evaluation. The first stage involves data preparation, software features (e.g., size, complexity) are collected from the version control system (CVS) that are used to train a classification model while issue reports are used to label defective modules. The second stage involves data preprocessing (i.e., removing outliers, handling missing values or feature selection). Since defect classification models are trained using software features, the quality of a defect classification model is heavily dependent on the value that can be gleaned from those features. Indeed, feature selection is an important part of the training of classification models [117]. However, in some cases where all the features are required to train the classification model, feature selection preprocessing is not applied. After the data preprocessing, a classification technique is used to train defect classification model. In the last stage, the trained classification model performance is evaluated using different performance measure (e.g., Accuracy, Precision, Recall, and AUC).

Recent studies conducted in the context of defect classification models have arrived at contradictory or inconsistent results. Hall *et al.* [59] conducted a literature survey of defect classification studies and their results show that several studies use

top-performing classification techniques in order to train defect classification models to arrive at different conclusions. Therefore, it is hard to derive practical guidelines about the most appropriate defect classification modeling approaches. However, (Menzies and Shepperd) [94] and Tantithamthavorn [147] indicate that conclusions of defect classification studies may differ due to variances in experimental settings while training defect classification models.

In this thesis, we examine the impact of three experimental settings on the performance of defect classification models. First, we study whether the performance of a classification model is impacted by the choice of classification technique that is used to train it. Second, we study the impact of different feature selection techniques (i.e., features that are used to train classification models) on the performance of defect classification models. Lastly, we study the impact of two types of meta-learners: (1) abstract meta-learners that accept the base classification technique as a parameter when producing a meta-classifier (e.g., Bagging and Boosting); and (2) concrete meta-learners which incorporate a particular base classification technique (e.g., Random Forest) on the performance of defect classification models.

1.1 Thesis Statement

Several studies in the past raise concerns about the impact of experimental settings on the performance of defect classification models. For example:

- Studies suggest that the classification technique that is used to train defect classification models has little impact on its performance [82, 93, 135]
- Recent studies have compared the impact of feature selection techniques on the performance of defect classification models [49, 57, 100, 125, 129, 159]. Some

studies conclude that some feature selection techniques are better than others [49, 57, 100, 129, 159], while other studies conclude that there is no significant difference between the performance of feature selection techniques [125].

- Some studies [53, 115, 153, 161] have shown that meta-learners when applied to the classification techniques produce classification models that tends to outperform the classification models that are trained using the base classification techniques, other studies [40, 101, 102, 136] indicate that the performance may vary under different experimental settings.

Therefore, due to contradictory conclusions of the past studies we set out to revisit the impact of three experimental settings: (1) choice of classification technique; (2) impact of feature selection and; (3) impact of meta-learners on the performance of classification models.

1.2 Thesis Overview and Organization

We organize the thesis into two parts:

Part I divides in to three chapters to examine the impact three different experimental settings on the performance of defect classification models.

- **Chapter 2: Revisiting the Impact of Classification Techniques on the Performance of Defect Classification Models.**

  A variety of classification techniques have been used to build defect classification models ranging from simple (e.g., logistic regression) to advanced techniques (e.g., Multivariate Adaptive Regression Splines (MARS)). Surprisingly, recent research on the NASA dataset suggests that the performance of a defect

classification model is not significantly impacted by the classification technique that is used to train it. However, the dataset that is used in the prior study is both: (a) noisy, i.e., contains erroneous entries and (b) biased, i.e., only contains software developed in one setting. Hence, we set out to replicate this prior study in two experimental settings. First, we apply the replicated procedure to the same (known-to-be noisy) NASA dataset. Next, we apply the replicated procedure to two new datasets: (a) the cleaned version of the NASA dataset and (b) the PROMISE dataset, which contains open source software developed in a variety of settings (e.g., Apache, GNU).

- **Chapter 3: A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models**

  The performance of a defect classification model depends on the features that are used to train it. Overuse of correlated or redundant features can introduce multicollinearity, which can hinder the performance of a classification model. To mitigate the risk of multicollinearity, researchers have used feature selection techniques, which transform or select a subset of software features in order to improve the performance of a defect classification model. Recent studies compared the impact of different feature selection techniques on the performance of defect classification models. However, these studies have compared a limited number of classification techniques. Hence, it is not clear how feature selection techniques impact the performance of defect classification models. To address this limitation, we study 29 feature selection techniques (11 filter-based ranking techniques, six filter-based subset techniques, and 12 wrapper-based subset techniques) and 21 classification techniques when applied to 18 datasets from

the NASA and PROMISE corpora. We use a hierarchical clustering approach (i.e., Scott-Knott) to identify statistically distinct ranks of different feature selection techniques.

- **Chapter 4: The Costs and Benefits of using Meta-Learners to Train Defect Classification Models**

  Defect classification models are classifiers that are trained to identify defect-prone software modules. To train these classifiers, a variety of classification techniques have been explored. Recent studies focus on meta-learning algorithms that integrate multiple classifiers to construct a meta-classifier that combines the decisions of the individual classifiers using an aggregation mechanism (e.g., voting). In this chapter, we investigate the costs and benefits of using meta-learners to train of defect classifiers. More specifically, we study abstract meta-learners, which vary the base classification technique that is used to produce the defect classifiers (e.g., Bagging), as well as concrete meta-learners, which incorporate a particular base classification technique to train the defect classifiers (e.g., Random Forest).

Part II draws the thesis summary and discusses future work.

1.3 Thesis Contribution

The main contributions of this thesis are as follows:

- To the best of our knowledge, we present the first work to replicate prior analysis of the impact of classification techniques on the performance of defect classification models using the cleaned NASA corpus and show that some classification

techniques tend to produce defect classification models that outperform others — recommending that software engineering researchers experiment with the various available techniques instead of relying on specific techniques, assuming that other techniques are not likely to lead to statistically significant improvements in their reported results.

- We highlight the impact of feature selection techniques on the performance of defect classification models — Correlation-based feature subset feature selection techniques with a BestFirst search method outperform other feature selection techniques across different datasets and classification techniques.

- To the best of our knowledge, we present the first work to analyze the impact of abstract meta-learners (e.g., Bagging and boosting) when applied to a variety of commonly used classification techniques and concrete meta-learners (e.g., Random Forest) on defect classification models in terms of performance (i.e., AUC) and training cost.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents an empirical study of the impact of classification techniques choice on defect classification models. Chapter 3 presents an empirical study of the impact of feature selection technique on defect classification models. Chapter 4 presents an empirical study of the impact of meta-learners on defect classification models. Chapter 5 draws conclusion and discusses avenues for future research.

Part I

Impact of Experimental Settings on Defect Classification Models

Chapter 2

Revisiting the Impact of Classification Techniques on the Performance of Defect Classification Models

Surprisingly, recent research on the NASA dataset suggests that the performance of a defect classification model is not significantly impacted by the classification technique that is used to train it. However, the dataset that is used in the prior study is both: (a) noisy, i.e., contains erroneous entries and (b) biased, i.e., only contains software developed in one setting. Hence, in this chapter, we set out to replicate this prior study along two experimental settings. First, we apply the replicated procedure to the same (known-to-be noisy) NASA dataset, where we derive similar results to the prior study, i.e., the impact that classification techniques appear to be minimal. Next, we apply the replicated procedure to two new datasets: (a) the cleaned version of the NASA dataset and (b) the PROMISE dataset, which contains open source software developed in a variety of settings (e.g., Apache, GNU). The results in these new datasets show a clear, statistically distinct separation of groups of techniques, i.e., the choice of classification technique has an impact on the performance of defect classification models.

2.1 Introduction

A disproportionate amount of the cost of developing software is spent on maintenance [37]. Fixing defects is a central software maintenance activity. However, before defects can be fixed, they must be detected. Software Quality Assurance (SQA) teams are dedicated to this task of defect detection.

Defect classification models can be used to assist SQA teams with defect detection. Broadly speaking, defect classification models are trained using software features (e.g., size and complexity features) to predict whether software modules will be defective or not in the future. Knowing which software modules are likely to be defect-prone before a system has been deployed allows practitioners to more efficiently and effectively allocate SQA effort.

To perform defect classification studies [60, 138], researchers have explored the use of various classification techniques to train defect classification models [31, 82]. For example, in early studies, researchers used simple techniques like logistic regression [9, 19, 112] and linear regression [63, 164] to train defect classification models.

In more recent work, researchers have used more advanced techniques like Multivariate Adaptive Regression Splines (MARS) [12, 67], Personalized Change Classification (PCC) [67], and Logistic Model Trees (LMT) [126, 131]. Ensemble methods that combine different machine learning techniques have also been explored [35, 153]. Moreover, researchers have started to develop context-sensitive techniques that are aware of the peculiarities of software defects [12, 90].

Despite recent advances in machine learning, studies suggest that the classification technique that is used to train defect classification models has little impact on their

performance [82, 93, 135]. For example, Lessmann *et al.* [82] conducted a study comparing the performance of 22 different classification techniques on the NASA corpus. Their results show that the performance of 17 of the 22 classification techniques are statistically indistinguishable from each other.

However, the NASA corpus that was used in the prior work is noisy and biased. Indeed, Shepperd *et al.* [137] found that the original NASA corpus contains several erroneous and implausible entries. This noise in the studied corpus may have led prior studies to draw incorrect conclusions. Furthermore, the studied corpus only contains proprietary software that is developed within one organization. Software developed in a different setting (e.g., open source) may lead to different conclusions.

Therefore, we set out to revisit the findings of Lessmann *et al.* [82] both in the original, known-to-be noisy setting (Section 2.4), and in two additional settings (Section 2.5). We find that we can indeed replicate the initial results of Lessmann *et al.* [82] when the procedure is reapplied to the known-to-be noisy NASA corpus. On the other hand, the results of our experiment in two additional settings seem to contradict Lessmann *et al.*'s early results. We structure our extended replication by addressing the following two research questions:

RQ1 Do the defect classification models of different classification techniques still perform similarly when trained using the cleaned NASA corpus?

No, we find a clear separation of classification techniques when models are trained using the cleaned NASA corpus [137]. The Scott-Knott statistical test [66] groups the classification techniques into four statistically distinct

ranks. Our results show that models trained using LMT and statistical techniques like Simple Logistic tend to outperform models that are trained using clustering, rule-based, Support Vector Machines (SVM), nearest neighbour, and neural network techniques.

RQ2 Do the defect classification models of different classification techniques still perform similarly when trained using a corpus of open source systems?

No, Scott-Knott test results show that, similar to RQ1, classification techniques like LMT, Simple Logistic (and their combination with ensemble methods) tend to produce defect classification models that outperform models trained using clustering, rule-based, SVM, nearest neighbour, and neural network techniques.

In summary, contrary to prior results, our study shows that there are statistically significant differences in the performance of defect classification models that are trained using different classification techniques. We observe that classification techniques are grouped into largely consistent ranks in both the cleaned NASA and PROMISE corpora. Indeed, some techniques tend to produce defect classification models that outperform models that are trained using other techniques. Given the ease of access to such advanced techniques nowadays in the toolboxes of researchers (e.g., `R` and `Weka`), we encourage researchers to explore the various techniques that are available in such toolboxes instead of relying on prior findings to guide their choice of classification technique.

Chapter Organization: The remainder of the chapter is organized as follows. Section 2.2 describes the classification techniques that we used in our study. Section 2.3 discusses the design of our case study. Section 2.4 discusses the results of our

replication study on the original, known-to-be noisy NASA corpus, while Section 2.5 presents the results of our two research questions. Section 2.6 discloses the threats to the validity of our study. Section 2.7 surveys related work. Finally, Section 2.8 draws chapter summary and describes promising directions for future work.

2.2 Classification Techniques

In this section, we briefly explain the eight families of classification techniques that are used in our study. Table 2.1 provides an overview of each technique.

2.2.1 Statistical Techniques

Statistical techniques are based on a probability model [78]. These techniques are used to find patterns in datasets and build diverse predictive models [11]. Instead of simple classification, statistical techniques report the probability of an instance belonging to each individual class (i.e., defective or not) [78].

In this chapter, we study the Naive Bayes and Simple Logistic statistical techniques. *Naive Bayes* is a probability-based technique that assumes that all of the predictors are independent of each other. *Simple Logistic* is a generalized linear regression model that uses a logit link function.

2.2.2 Clustering Techniques

Clustering techniques divide the training data into small groups such that the similarity within groups is more than across the groups [62]. Clustering techniques use distance and similarity measures to find the similarity between two objects to group them together.

In this chapter, we study the K-means and Expectation Maximization clustering techniques. *K-means* divides the data into k clusters and centroids are chosen randomly in an iterative manner [61]. The value of k impacts the performance of the technique [81]. We experiment with four different k values (i.e., 2, 3, 4, and 5), and found that $k = 2$ tends to perform the best. We also study the *Expectation Maximization* [44] (EM) technique, which automatically splits a dataset into an (approximately) optimal number of clusters [12].

2.2.3 Rule-Based Techniques

Rule-based techniques transcribe decision trees using a set of rules for classification. The transcription is performed by creating a separate rule for each individual path starting from the root and ending at each leaf of the decision tree [78].

In this chapter, we study the Repeated Incremental Pruning to Produce Error Reduction (Ripper) and RIpple DOwn Rules (Ridor) rule-based techniques. *Ripper* [27] is an inductive rule-based technique that creates series of rules with pruning to remove rules that lead to lower classification performance [81]. *Ridor* [47] is a rule-based decision tree technique where the decision tree consists of four nodes: a classification, a predicate function, a true branch, and a false branch. Each instance of the testing data is pushed down the tree, following the true and false branches at each node using predicate functions. The final outcome is given by the majority class of the leaf node [81].

2.2.4 Neural Networks

Neural networks are systems that change their structure according to the flow of information through the network during training [143]. Neural network techniques are repeatedly run on training instances to find a classification vector that is correct for each training set [78].

In this chapter, we study the Radial Basis Functions neural network technique. *Radial Basis Functions* [20] consists of three different layers: an input layer (which consists of independent variables), output layer (which consists of the dependent variable) and the layer which connects the input and output layer to build a model [112].

2.2.5 Nearest Neighbour

Nearest neighbour (a.k.a., lazy-learning) techniques are another category of statistical techniques. Nearest neighbour learners take more time in the testing phase, while taking less time than techniques like decision trees, neural networks, and Bayesian networks during the training phase [78].

In this chapter, we study the KNN nearest neighbour technique. *KNN* [28] considers the K most similar training examples to classify an instance. KNN computes the Euclidean distance to measure the distance between instances [82]. We find $K = 8$ to be the best-performing K value of the five tested options (i.e., 2, 4, 6, 8, and 16).

2.2.6 Support Vector Machines

Support Vector Machines (SVMs) use a hyperplane to separate two classes (i.e., defective or not). The number of features in the training data does not affect the complexity of an SVM model, which makes SVM a good fit for experiments where

Table 2.1: An overview of the studied classification techniques.

Family	Technique	Abbreviation
Statistical Technique	Naive Bayes	NB
	Simple Logistic	SL
Clustering Technique	K-means	K-means
	Expectation Maximization	EM
RuleBased Techniques	Repeated Incremental Pruning to Produce Error Reduction	Ripper
	Ripple Down Rules	Ridor
Neural Network	Radial Basis Function	RBFs
Nearest Neighbour	K-Nearest Neighbour	KNN
Support Vector Machines	Sequential Minimal Optimization	SMO
Decision Trees	J48	J48
	Logistic Model Tree	LMT
Ensemble Method using LMT, NB, SL, SMO, and J48	Bagging	Bag+LMT, Bag+NB, Bag+SL, Bag+SMO and Bag+J48
	Adaboost	Ad+LMT, Ad+NB, Ad+SL, Ad+SMO and Ad+J48
	Rotation Forest	RF+LMT, RF+NB, RF+SL, RF+SMO, and RF+J48
	Random Subspace	Rsub+LMT, Rsub+NB, Rsub+SL, Rsub+SMO, and Rsub+J48

there are fewer training instances than features [78].

In this chapter, we study the Sequential Minimal Optimization (SMO) SVM technique. *SMO* analytically solves the large Quadratic Programming (QP) optimization problem which occurs in SVM training by dividing the problem into a series of possible QP problems [119].

2.2.7 Decision Trees

Decision trees use feature values for the classification of instances. A feature in an instance that has to be classified is represented by each node of the decision tree, while the assumption values taken by each node is represented by each branch. The classification of instances is performed by following a path through the tree from root to leaf nodes by checking feature values against rules at each node. The root node is the node that best divides the training data [78].

In this chapter, we study the Logistic Model Tree (LMT) and J48 decision tree techniques. Similar to model trees (i.e., regression trees with regression functions), *LMT* [80] is a decision tree like structure with logistic regression functions at the leaves [80]. *J48* [121] is a C4.5-based technique that uses information entropy to build the decision tree. At each node of the decision tree, a rule is chosen by C4.5 such that it effectively divides the set of training samples into subsets [132].

2.2.8 Ensemble Methods

Ensemble methods combine different base learners together to solve one problem. Models trained using ensemble methods typically generalize better than those trained using the stand-alone techniques [162].

In this chapter, we study the Bagging, Adaboost, Rotation Forest, and Random Subspace ensemble methods. *Bagging* (Bootstrap Aggregating) [17] is designed to improve the stability and accuracy of machine learning algorithms. Bagging predicts an outcome multiple times from different training sets that are combined together either by uniform averaging or with voting [153]. *Adaboost* [45] performs multiple iterations

each time with different example weights, and gives a final classification through combined voting of techniques [153]. *Rotation Forest* [130] applies a feature extraction method to subsets of instances to reconstruct a full feature set for each technique in the ensemble. *Random Subspace* [64] creates a random forest of multiple decision trees using a random selection attribute approach. A subset of instances is chosen randomly from the selected attributes and assigned to the learning technique [153].

2.3 Study Design

In this section, we discuss the studied corpora, and our approach to constructing and evaluating defect classification models to address our research questions. Figure 2.1 provides an overview of the steps in our approach.

2.3.1 Studied Corpora

In order to replicate the case study of Lessmann *et al.* [82], we use the original, known-to-be noisy NASA corpus. Moreover, in order to perform our extended study, we use the cleaned version (D") of the NASA corpus as provided by Shepperd *et al.* [137] (RQ1), and the PROMISE corpus [1] (RQ2). An overview of the studied corpora is shown in Table 2.2. The studied corpora that we use in our study are available online (`http://openscience.us/repo/`), enabling further replication (and extension) of our results.

2.3.2 Fold Generation

The defect classification models are trained using the 10-fold cross-validation approach, which divides an input dataset into 10 folds of equal size. Of the 10 folds, 9 are

Table 2.2: An overview of the studied corpora.

Corpus	Project	No. of Modules	Defective	Defective (%)
Noisy NASA (Section IV)	CM1	505	48	9.5
	JM1	10,878	2,102	19.3
	KC1	2,107	325	15.4
	KC3	458	43	9.3
	KC4	125	61	48.8
	MW1	403	31	7.6
	PC1	1,107	76	6.8
	PC2	5,589	23	0.4
	PC3	1,563	160	10.2
	PC4	1,458	178	12.2
Clean NASA (RQ1)	CM1	327	42	12.8
	JM1	7,720	1,612	20.8
	KC1	1,162	294	25.3
	KC3	194	36	18.5
	MW1	250	25	10
	PC1	679	55	8.1
	PC2	722	16	2.2
	PC3	1,053	130	12.3
	PC4	1,270	176	13.8
PROMISE (RQ2)	Ant 1.7	745	166	22.3
	Camel 1.6	965	188	19.5
	Ivy 1.4	241	16	6.6
	Jedit 4	306	75	24.5
	Log4j 1	135	34	25.2
	Lucene 2.4	340	203	59.7
	Poi 3	442	281	63.6
	Tomcat 6	858	77	8.9
	Xalan 2.6	885	441	46.4
	Xerces 1.3	453	69	15.2

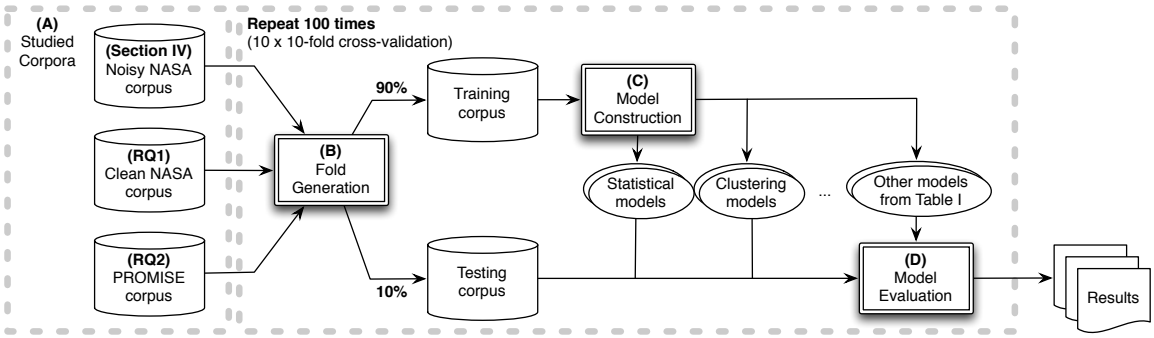

Figure 2.1: An overview of our model construction and evaluation approach.

allocated to the training corpus (90%), and 1 fold is set aside for testing (10%). The training corpus is used to train the models using different classification techniques, whereas the testing corpus is used to analyze model performance. This process is repeated ten times, using each fold as the testing corpus once. To further validate our results, we repeat the entire 10-fold process ten times (100 total iterations).

2.3.3 Model Construction

In order to train our defect classification models, we use implementations of the classification techniques of Section 2.2 provided by the WEKA [156] machine learning toolkit.

The process of training defect classification models using clustering techniques is different than for the other techniques. Specifically, we adopt the approach of Bettenburg *et al.* in order to train models using clustering techniques [12]. The training corpus is divided into clusters and a classification model is trained within each cluster using the Naive Bayes technique. To determine which model to use in order to classify any particular row in the testing corpus, we use the Euclidean distance between each testing data point and the various clusters. The model of the

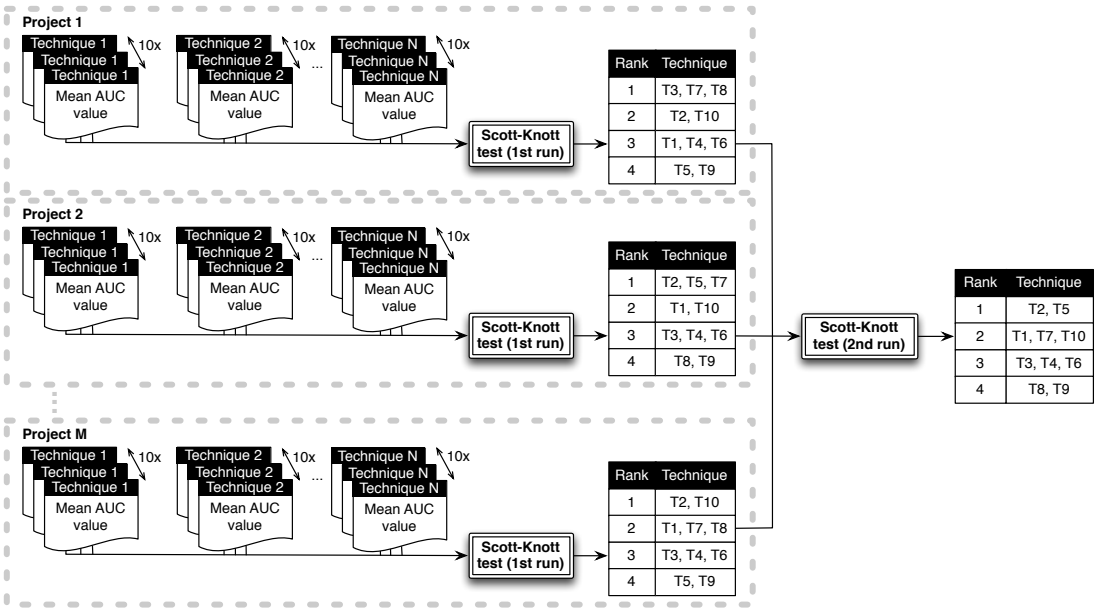

Figure 2.2: Our approach for ranking classification techniques using a double Scott-Knott test.

cluster that is the minimum distance away is used to predict the final outcome for that row.

2.3.4 Model Evaluation

To compare the performance of defect classification models, we use the Area Under the receiver operating characteristic Curve (AUC) [158], which plots the false positive rate (i.e., $\frac{FP}{FP+TN}$) against the true positive rate (i.e., $\frac{TP}{FN+TP}$). Larger AUC values indicate better performance. AUC values above 0.5 indicate that the model outperforms random guessing.

Scott-Knott test: We use the Scott-Knott test [66] to group classification techniques into statistically distinct ranks ($\alpha = 0.05$). The Scott-Knott test uses hierarchical cluster analysis to partition the classification techniques into ranks. The

Table 2.3: Features used to train defect classification models in the NASA corpus.

Category	Metric	Definition	Rationale
McCabe Software Features	cyclomatic_complexity, cyclomatic_density, design_complexity essential complexity and pathological_complexity	Measures of the branching complexity of the software module.	Complex software modules may be more prone to defects [91].
Halstead attributes	content, difficulty, effort, error_est, length, level, prog_time, volume, num_operands, num_operators, num_unique_operands, num_unique_operators	Estimates of the complexity of reading a software module based on the vocabulary used (e.g., number of operators and operands).	Software modules that are complex to understand may increase the likelihood of incorrect maintenance, and thus, increase defect proneness [69].
LOC counts	LOC_total, LOC_blank, LOC_comment, LOC_code_and_comment, LOC_executable and Number_of_lines	Measures of the size of a software module.	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [76].
Miscellaneous	branch_count, call_pairs, condition_count, decision_count, decision_density, design_density, edge_count, essential_density, parameter_count, maintenance_severity, modified_condition_count, multiple_condition_count, global_data_density, global_data_complexity, percent_comments, normalized_cyclomatic_complexity and node_count	Features that are not well defined features in the MDP database [92].	N/A

Scott-Knott test starts by dividing the classification techniques into two ranks on the basis of mean AUC values (i.e., mean AUC of ten 10-fold runs for each classification technique). If the divided ranks are statistically significantly different, then Scott-Knott recursively executes again within each rank to further divide the ranks. The test terminates when ranks can no longer be divided into statistically distinct

ranks [70, 95].

We used the Scott-Knott test to overcome the confounding issue of overlapping groups that are produced by several other post hoc tests, such as Nemenyi's test [34], which was used by the original study. Nemenyi's test produces overlapping groups of classification techniques, implying that there exists no statistically significant difference among the defect classification models trained using many different classification techniques.

To address our research questions, Figure 2.2 provides an overview of our Scott-Knott test approach. To find statistically distinct ranks of classification techniques, we performed a double Scott-Knott test. The first run of the Scott-Knott test is run over each individual project. We input the 10 mean AUC values of the 10 different 10-fold cross-validation runs of each classification technique to the Scott-Knott test to find the statistical distinct ranks of classification techniques within each project. In the second run, for each classification technique, we have ten different Scott-Knott ranks (i.e., one from each project), which we input into another Scott-Knott test to find the final statistically distinct ranks of techniques.

Our approach of using a double Scott-Knott test ensures that our analysis recognizes techniques that perform well across projects, independent of their actual AUC value. For example, it might be the case that one classification technique achieves an AUC of 0.5 for project A and an AUC of 0.9 for project B. At first glance, it might seem that such a technique is not a strong performing technique. However, if an AUC of 0.5 is the best AUC value of the studied techniques for project A, and AUC of 0.9 is the best AUC for project B, then that particular classification technique is the top-performing technique. The first Scott-Knott test creates the project-level ranking

Table 2.4: The studied techniques ranked according to our double Scott-Knott test on the known-to-be noisy NASA corpus.

Overall Rank	Classification Technique	Median Rank	Avg. Rank	Standard Deviation
1	Ad+NB, EM, RBFs, Ad+SL, RF+NB, Rsub+NB, Ad+SMO, K-means, KNN, NB, SL, Bag+NB, Rsub+SL, Ad+J48, Rsub+J48, Ad+LMT, LMT, RF+SL, Bag+SL, RF+J48, Rsub+LMT, Bag+J48, RF+LMT and Bag+LMT	3.2	3.03	0.97
2	Rsub+SMO, SMO, RF+SMO, Ridor, Bag+SMO, Ripper, and J48	8	7.91	1.04

for each technique using the AUC values, then the second Scott-Knott test creates a global ranking of techniques using their project-level rankings.

2.4 Replication Study

In this section, we discuss the results of our replication study on the known-to-be noisy NASA corpus. The NASA corpus provides several software features that can be used to predict defect-prone modules. Table 2.3 describes the features present in the NASA corpus.

Results

Similar to prior work, our replication using the known-to-be noisy NASA data yields a pattern of two statistically distinct groups of classification techniques. Table 2.4 shows that 24 of the 31 studied classification techniques appear in the first Scott-Knott group. Figure 2.3 shows the AUC values

for each of the studied techniques.

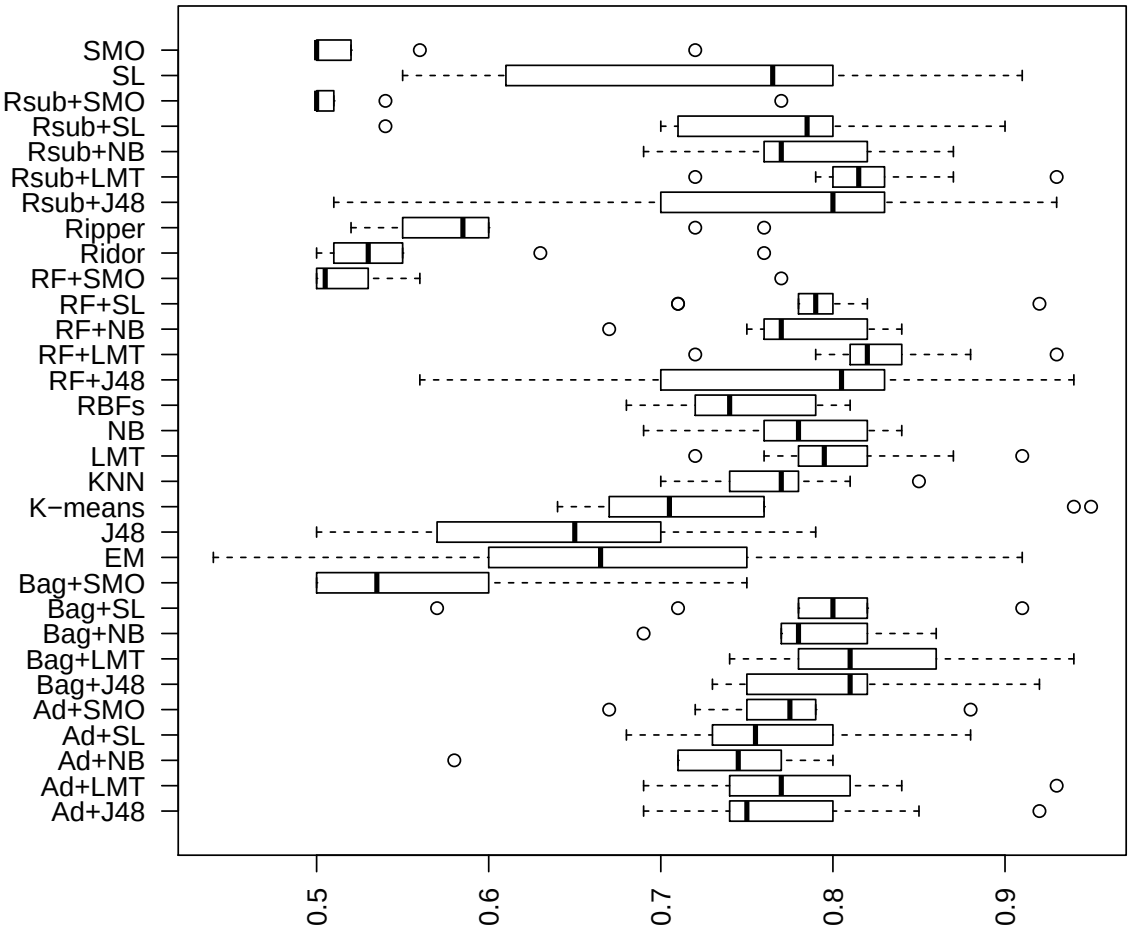

Figure 2.3: AUC values for the studied classification techniques on the known-to-be noisy NASA corpus.

Figure 2.4 highlights the importance of our double Scott-Knott approach. Looking at project `JM1` and project `PC4` (or `KC4`), many of the best techniques for project `JM1` perform worse than some of the worst-performing techniques on the `PC4` (or `KC4`)

project. Our double Scott-Knott approach is able to account for such peculiarities by considering the rank in the second Scott-Knott iteration.

Our replication yields a similar pattern of classification techniques as was found in the previous NASA study, i.e., 24 of the 31 studied techniques end up in the same statistical group. Most classification techniques produce defect classification models that have statistically indistinguishable performance from one another.

2.5 Extended Case Study

In this section, we present the results of the extended Lessmann *et al.* [82] replication with respect to our two research questions. Generally speaking, our goal is to study whether the previously reported results of Lessmann *et al.* [82] (and confirmed by us in Section 2.4) still hold in two new experimental settings, which we formulate as our two research questions. Below, we discuss our approach for addressing each research question and the results.

Table 2.5: Differences in the noisy and clean NASA corpora.

Dataset	Instances before cleaning	Instances after cleaning	Instances removed by cleaning
CM1	505	327	178
JM1	10,878	7,720	3,158
KC1	2,107	1,162	945
KC3	458	194	264
MW1	403	250	153
PC1	1,107	679	428
PC2	5,589	722	4,867
PC3	1,563	1,053	510
PC4	1,458	1,270	188

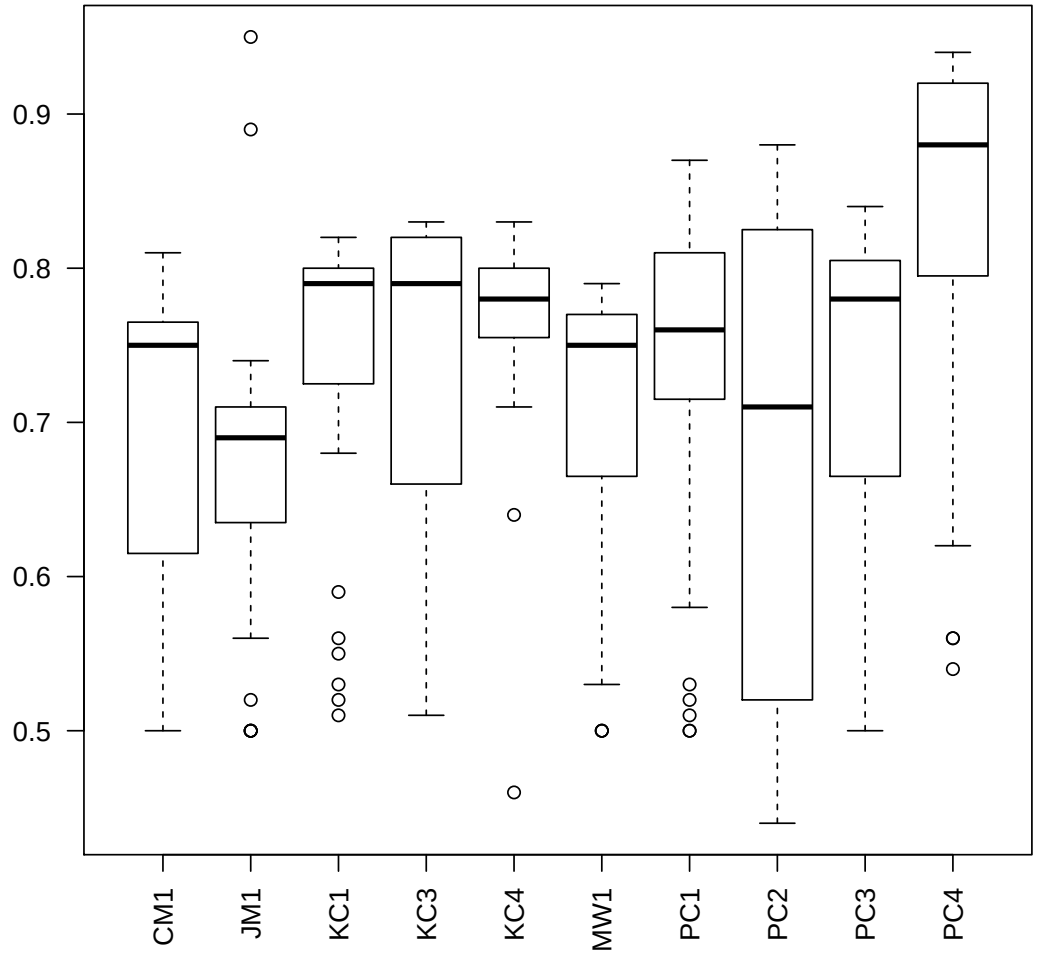

Figure 2.4: AUC values of all classification techniques that are applied to the different projects in the known-to-be noisy NASA corpus.

(RQ1) Do the defect classification models of different classification techniques still perform similarly when trained using the cleaned NASA corpus?

Approach

Our extended study starts with using the cleaned version [137] of nine of the NASA projects that were used by Lessmann *et al.* [82]. The purpose of using the cleaned version is to find whether the removal of noisy instances has an impact on the performance of defect classification models produced using the studied classification techniques. An overview of the cleaned version of the NASA dataset is shown in Table 2.2. We are only able to use 9 of the 10 NASA projects because the KC4 dataset does not contain any instances after cleaning. The differences in the noisy and clean NASA corpora are highlighted in Table 2.5. The preprocessing criteria that Shepperd *et al.* [137] applied to the noisy NASA corpus are presented in Table 2.6.

Results

We find four statistically distinct ranks of classification techniques when models are trained using datasets from the cleaned NASA corpus. Table 2.7 shows that there is a clear separation of classification techniques into four Scott-Knott ranks in the cleaned NASA corpus. This is in stark contrast with the two groups of classification techniques that Lessmann *et al.* [82] found (and that Section 2.4 confirmed).

The techniques are divided into four distinct ranks in which ensemble techniques (i.e., RF+a base learner), decision trees (i.e., LMT), statistical techniques (i.e., Simple Logistic and Naive Bayes), neural networks (i.e., RBFs), and nearest neighbour (i.e., KNN) outperformed models trained using rule-based techniques (i.e., Ripper

Table 2.6: Cleaning criteria applied to the noisy NASA corpus by Shepperd *et al.* [137].

Criterion	Data Quality Category	Explanation
1	Identical cases	Instances that have identical values for all features including class label.
2	Inconsistent cases	Instances that satisfy all conditions of Case 1, but where class labels differ.
3	Cases with missing values	Instances that contain one or more missing observations.
4	Cases with conflicting feature values	Instances that have 2 or more metric values that violate some referential integrity constraint. For example, LOC_TOTAL is less than Commented LOC. However, Commented LOC is a subset of LOC_TOTAL.
5	Cases with implausible values	Instances that violate some integrity constraint. For example, value of LOC=1.1.

Table 2.7: The studied techniques ranked according to the double Scott-Knott test on the clean NASA corpus.

Overall Rank	Classification Technique	Median Rank	Avg. Rank	Standard Deviation
1	Bag+J48, LMT, RF+J48, SL, Bag+SL, RF+SL, Rsub+SL, RF+LMT, Rsub+LMT, and Bag+LMT	1.22	1.24	0.15
2	RBFs, Ad+NB, KNN, Ad+SL, RF+NB, Rsub+NB, NB, Bag+NB, Ad+SMO, Ad+J48, Ad+LMT, and Rsub+J48	2.61	2.62	0.51
3	Ripper, EM, J48, and K-means	5.72	5.58	0.67
4	Rsub+SMO, RF+SMO, SMO, Ridor, and Bag+SMO	7.66	7.4	0.42

and Ridor), clustering techniques (i.e., K-means and EM) and SVM (i.e., SMO). Our results in Table 2.7 show that the classification models trained using ensemble techniques (i.e., RF+a base learner), decision trees (i.e., LMT), and statistical techniques (i.e., simple logistic) also outperformed models trained using nearest neighbour (i.e., KNN) and neural networks (i.e., RBFs). Furthermore, we find that classification models trained using the combination of LMT and simple logistic with ensemble methods like bagging, rotation forest, and random subspace produce defect classification models that perform the best (i.e., achieve the highest average Scott-Knott rank) when compared to other combinations, i.e, Naive Bayes, J48, and SMO with ensemble methods.

Table 2.8: Features used to train defect classification models in the PROMISE corpus.

Category	**Metric**	**Definition**	**Rationale**
Size	loc	Measures of the size of a software module.	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [76].
CK	wmc, dit, cbo, noc, lcom, rfc, ic, cbm, amc, lcom3	Measures of the complexity of a class within an object-oriented system design.	More complex classes may be more prone to defects [51].
McCabe's Complexity	max_cc and avg_cc	Measures of the branching complexity of the software module.	Complex software modules may be more prone to defects [91].
QMOOD	dam, moa, mfa, cam, npm	Measures of the behavioural and structural design properties of classes, objects, and the relations between them [133].	Higher QMOOD features imply higher class quality. On the other hand, lower QMOOD features imply lower class quality [71], which may lead to defects.
Martin's features	ca, ce	Measures of the relationship between software components, including calls as well as number of instances [110].	Higher values indicate low encapsulation, reusability, and maintainability [110], which may lead to defects.

Since four statistically distinct ranks of classification techniques emerge, we conclude that the used classification technique has a statistically significant impact on the performance of defect classification models trained using the datasets from the cleaned NASA corpus.

(RQ2) Do the defect classification models of different classification techniques still perform similarly when trained using a corpus of open source systems?

Approach

In an effort to further generalize our extended study results, we replicate the Lessmann *et al.* study [82] using another ten datasets from the PROMISE corpus. The selected PROMISE datasets were used in several previous studies [112, 116]. The PROMISE datasets provide different features than the NASA corpus. Table 2.2 provides an overview of the PROMISE corpus. Table 2.8 describes the features provided by the PROMISE corpus.

Results

We find that, similar to our results of RQ1, a pattern of four statistically distinct ranks of classification techniques emerges in the PROMISE corpus as well. Table 2.9 shows the four statistically distinct ranks generated by applying the double Scott-Knott test.

The results show that classification models trained using decision trees, statistical techniques, K-nearest neighbour, and neural networks outperform models trained using clustering techniques, rule-based techniques, and SVM. Furthermore, we find that when ensemble methods are used to combine LMT and simple logistic, the

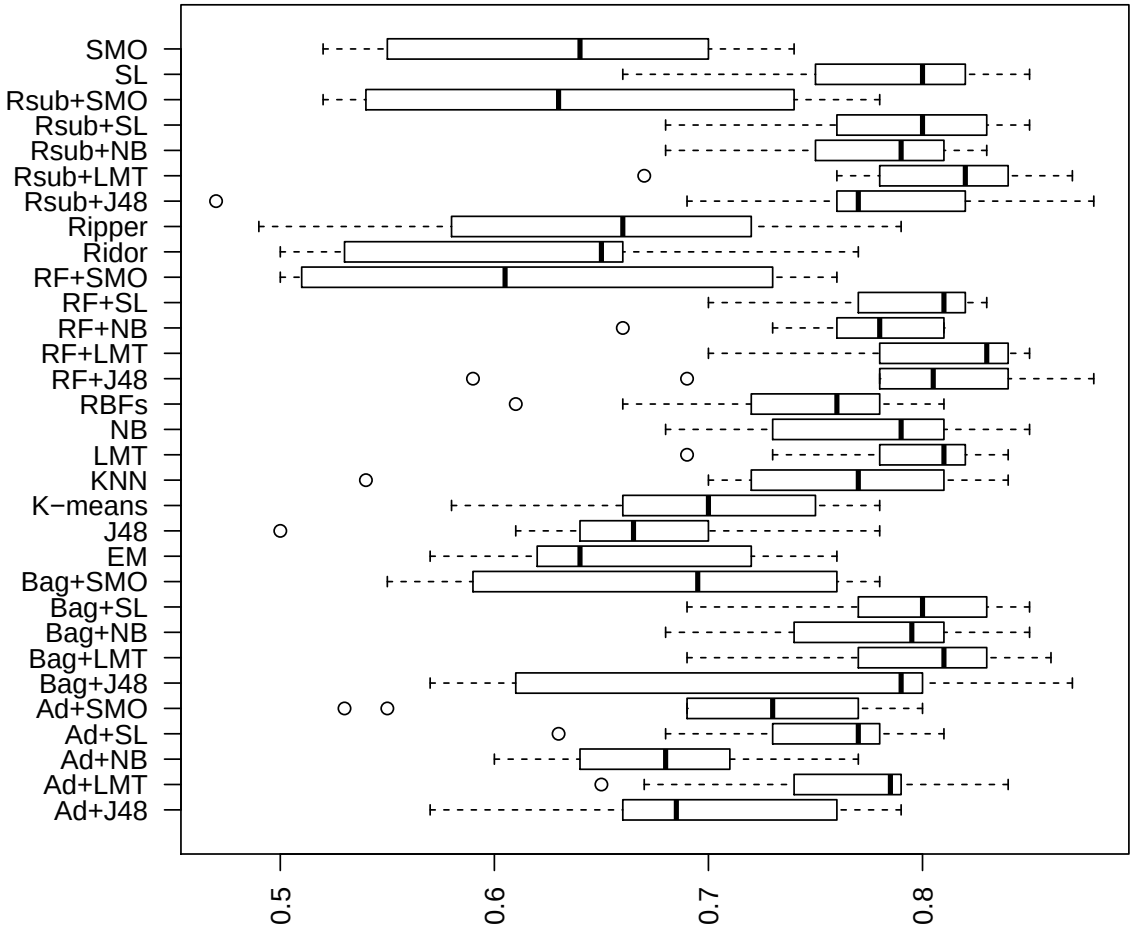

Figure 2.5: AUC values of the classification techniques on the PROMISE corpus.

best performance is achieved when compared to the other classification techniques. Figure 2.5 shows the mean AUC values of techniques on the PROMISE dataset. We observe that the mean AUC value of the best (RF+LMT) and the worst (RF+SMO) classification techniques in the PROMISE corpus have higher values when compared

to best (Bag+LMT) and worst (Rsub+SMO) classification techniques in the cleaned NASA corpus.

Figure 2.6 again emphasizes the importance of using the double Scott-Knott approach in our analysis.The best performing technique for the `Camel` project is well below the worst-performing technique for other projects, such as `Poi` and `Xalan`.

We observed some common patterns in the ranks of both the NASA and PROMISE corpora. Our results show that LMT when combined with ensemble methods (i.e., bagging, random subspace, and rotation forest) achieve top-rank performance in both corpora. Also, ensembles of simple logistic along with stand-alone LMT and simple logistic appear in the top Scott-Knott rank in both corpora. Furthermore, clustering techniques (i.e., EM and K-means), rule-based techniques (Ripper and Ridor), and SVM (i.e., SMO) appear in the lowest Scott-Knott rank in both corpora.

Table 2.9: The studied techniques ranked according to the double Scott-Knott test on the PROMISE corpus.

Overall Rank	**Classification Technique**	**Median Rank**	**Avg. Rank**	**Standard Deviation**
1	Rsub+J48, SL, Rsub+SL, LMT, Bag+SL, RF+SL, RF+J48, Bag+LMT, Rsub+LMT, and RF+LMT	1.7	1.63	0.33
2	RBFs, Bag+J48, Ad+SL, KNN, RF+NB, Ad+LMT, NB, Rsub+NB, and Bag+NB	2.8	2.84	0.41
3	Ripper, EM, J48, Ad+NB, Bag+SMO, Ad+J48, Ad+SMO, and K-means	5.1	5.13	0.46
4	RF+SMO, Ridor, SMO, and Rsub+SMO	6.5	6.45	0.25

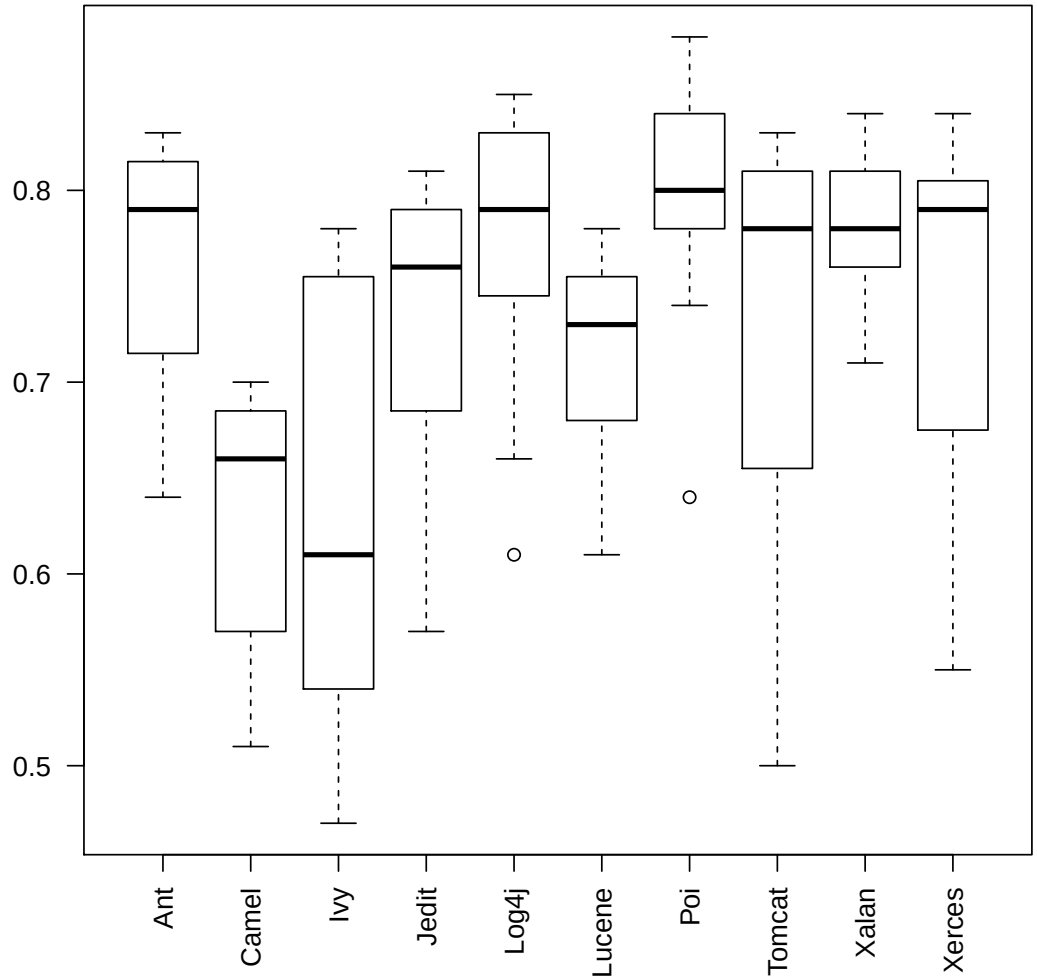

Figure 2.6: AUC values of all classification techniques that are applied to the different projects in the PROMISE corpus.

Complementing our results of the cleaned NASA study, we again find four statistically distinct ranks of classification techniques, which leads us to conclude that the used classification technique has a statistically significant impact on the performance of defect classification models trained using the datasets of the PROMISE corpus as well.

2.6 Threats to Validity

In this section, we discuss the threats to the validity of our case study.

2.6.1 Construct Validity

The datasets that we analyze are part of two corpora (i.e., NASA and PROMISE), which each provide a different set of predictor features. Since the features vary, this is another point of variation between the studied corpora, which may explain some of the differences that we observe when drawing comparisons across the corpora. Nevertheless, our main findings still hold, i.e., there are statistically significant differences between the performance of defect classification models trained using various classification techniques within the cleaned NASA dataset and the PROMISE one as well.

2.6.2 Internal Validity

The tuning of the parameters of the K-means ($k = 2$) and KNN ($K = 8$) techniques that we used may influence our results. For example, this tuning may impact the performance of models trained using these classification techniques when compared to default parameters. To combat this bias, we experimented with several configurations

(k = 2, 3, 4, and 5; and K = 2, 4, 6, 8, and 16) that achieved similar results. Nonetheless, a more carefully controlled experiment of the influence of configuration parameters is needed.

2.6.3 External Validity

We performed our experiments on 29 datasets, which may threaten the generalization of the results of our study. However, these datasets are part of the popular NASA and PROMISE corpora, which have been used by several studies that benchmark defect classification models in the past [82, 112, 116].

The consistency of our results are based on two studied data corpora. The results may vary in other corpora that we have not analyzed. On the other hand, the two corpora contain software systems that are developed in both proprietary and open source paradigms, which helps to counter potential bias. Nonetheless, additional replication studies may prove fruitful.

2.7 Related Work

In order to train defect classification models, defective software modules are predicted using software features, such as object-oriented features (e.g., Chidamber and Kemerer (CK) features [26]), process features [10, 63, 97, 104] (e.g., number of changes, recent activity), product features [56, 103, 106, 109, 145] (e.g., lines of code and complexity), historical features [56, 111] (e.g., code change), and structural features [7] (e.g., cyclomatic complexity or coupling). In this chapter, we used several types of features based on their availability in the studied corpora. A promising direction for future research would be to explore the impact that different features have on

our findings (e.g., perhaps our findings are sensitive to the type of features that are available for training our models).

There exists several evaluation features to measure the performance of classification models, such as accuracy [153], AUC [82, 112], error sum, median error, error variance, and correlation [12]. In this chapter, we use the AUC because it is threshold-independent.

In addition to Lessmann *et al.* [82], there have been several studies that compare the performance of classification techniques across projects in search of the top-performing classification techniques. Panichella *et al.* [112] conducted a study to compare the performance of statistical, neural networks, and decision trees classification techniques using the PROMISE corpus. Bettenburg *et al.* [12] conducted a study using the PROMISE corpus in order to compare the performance of classification models that are trained using clustering and statistical techniques, reporting that clustering techniques tend to perform better than models that are trained using statistical techniques. However, Lemon [81] found contradictory results to Bettenburg *et al.* [12] when building classification models on NASA datasets using clustering and statistical techniques.

The key differences between such prior work and this chapter of thesis are the following:

1. We are the first to replicate a prior analysis using the cleaned NASA corpus.

2. Most prior studies use either the NASA or the PROMISE corpus — they rarely combine both corpora together for a single analysis, hence the generalization of prior results is hindered.

3. Most prior studies use a post hoc statistical tests to compare techniques. However, the results of such tests are not as clear cut as the Scott-Knott tests, which produces non-overlapping ranks.

4. We are the first to extrapolate project-specific analyses to a corpus-level by ranking classification techniques at the project-level and lifting it using a double Scott-Knott test.

5. We are the first study to use several recent classification techniques (e.g., LMT), while ensuring that we capture techniques from a variety of families.

2.8 Chapter Summary

A previous study on the NASA corpus by Lessmann *et al.* [82] showed that the performance of defect classification models that are trained using various classification techniques do not vary much, implying that the use of any technique should be sufficient when predicting defects. In this chapter, we revisit this very impactful finding, since it has influenced a large amount of follow-up literature after its publication — many researchers opt not to investigate the performance of various classification techniques on their datasets. Our replication of Lessmann *et al.* [82] yields the same results even when using a more advanced statistical ranking technique (i.e., the Scott-Knott test). However, when using a cleaned version of the NASA corpus and when using another corpus (the PROMISE dataset, which provides different features than the NASA corpus does), we found that the results differ from those of Lessmann *et al.* [82], i.e., classification techniques produce defect classification models with significantly different performance.

Like any empirical study, our study has several limitations (*cf.* Section 2.6). However, we would like to emphasize that we do not seek to claim the generalization of our results. Instead, the key message of our study is that there are datasets where there are statistically significant differences between the performance of models that are trained using various classification techniques. Hence, we recommend that software engineering researchers experiment with the various available techniques instead of relying on specific techniques, assuming that other techniques are not likely to lead to statistically significant improvements in their reported results. Given the ubiquity of advanced techniques in commonly-used analysis toolkits (e.g., `R` and `Weka`), we believe that our suggestion is a rather simple and low-cost suggestion to follow.

Chapter 3

A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models

The performance of a defect classification model depends on the features that are used to train it. Feature redundancy, correlation, and irrelevance can hinder the performance of a classification model. To mitigate this risk, researchers often use feature selection techniques, which transform or select a subset of the features in order to improve the performance of a classification model. Recent studies compare the impact of different feature selection techniques on the performance of defect classification models. However, these studies compare a limited number of classification techniques and have arrived at contradictory conclusions about the impact of feature selection techniques. To address this limitation, we study 30 feature selection techniques (11 filter-based ranking techniques, six filter-based subset techniques, 12 wrapper-based subset techniques, and a no feature selection configuration) and 21 classification techniques when applied to 18 datasets from the NASA and PROMISE corpora. Our results show that a correlation-based filter-subset feature selection technique with a BestFirst search method outperforms other feature selection techniques across the studied datasets (it outperforms in 70%–87% of the PROMISE–NASA data sets) and across the studied

classification techniques (it outperforms for 90% of the techniques). Hence, we recommend the application of such a selection technique when building defect classification models.

3.1 Introduction

Defect classification models, i.e., statistical regression or machine learning classifiers that are trained to classify software modules to be defective or not help to identify defect-prone modules in a software system before it is released to users. The classifications are used by Software Quality Assurance (SQA) teams to prioritize their limited resources to system areas that are most likely to be defect-prone.

Data preprocessing techniques like feature selection are an important step when building classification models [85, 117, 120]. Prior studies find that defect classification models often exhibit high misclassification rates due to feature redundancy, correlation, and irrelevance [48, 54]. Indeed, prior work suggests that high misclassification rates and low classification accuracy are often attributed to the number of features [125]. Hence, it is advisable to preprocess features before using them to train defect classification models [40].

When selecting features, one must be mindful of irrelevant and redundant features that result in high dimensionality datasets. High dimensionality datasets can lead to a high computational (training) cost and performance degradation of certain classification models [88, 125, 129, 141, 159]. To combat these risks, recent studies apply feature selection techniques prior to training classification models [14, 85]. These feature selection techniques include: (1) filter-based ranking techniques that select features by estimating their capacity for contributing to the fitness of the classification model; (2) filter-based subset techniques that select features that collectively have good predictive capability [49]; and (3) wrapper-based techniques that use a learning algorithm to evaluate the importance of a feature in a predictive model [49].

Recent studies have compared the impact of feature selection techniques on the

performance of defect classification models [49, 100, 125, 129, 159]. Some studies conclude that some feature selection techniques are better than others [49, 100, 129, 159], while other studies conclude that there is no significant difference between the performance of feature selection techniques [125]. We suspect that past studies may have arrived at contradictory results because they compared a limited number of classification techniques across a limited number of datasets.

In this chapter, we conduct a large scale study to analyze the impact of 30 feature selection techniques (including a no feature selection configuration) on the performance of 21 commonly used classification techniques. The 21 studied classification techniques belong to seven families (i.e., Statistical, Nearest Neighbour, Neural Networks, Rule-Based, Support Vector Machines, Decision Tree, and Ensemble Method) and the 29 studied feature selection techniques belong to three families (i.e., filter-based ranking, filter-based subset, and wrapper-based techniques). We train our classification models using datasets from the NASA and PROMISE corpora, which are of different sizes and domains.

A preliminary analysis indicates that features in the datasets of the NASA corpus contains more data redundant features than in the datasets of the PROMISE corpus, suggesting that the corpora should be analyzed separately. Hence, we conduct separate case studies of each corpus.

We train our defect classification models using a combination of feature selection and classification techniques, in order to address the following research question:

RQ: Do feature selection techniques impact the performance of defect classification models?

Our results indicate that a correlation-based filter-subset feature selection technique with a BestFirst search method, which produces feature subsets using greedy hillclimbing with backtracking, outperforms other feature selection techniques ending up in the top statistical rank for

- 87% of the studied NASA datasets and 90% of the studied classification techniques.
- 70% of the studied PROMISE datasets and all of the 21 studied classification techniques.

Implications: Hence, we recommend that future defect classification studies apply a correlation-based filter-subset feature selection technique with a BestFirst search method when building classification models.

Chapter Organization: The rest of the chapter is organized as follows. Section 3.2 describes the studied feature selection techniques. Section 3.3 discusses the design of our case study. Section 3.4 discusses a preliminary comparison of the NASA & PROMISE corpora. Section 3.5 discusses the results of our case study. Section 3.6 discloses the threats to the validity of our study. Section 3.7 surveys related work. Finally, Section 3.8 draws chapter summary.

3.2 Feature Selection Techniques

This section gives a brief of overview of the studied feature selection techniques (*see* Table 3.1). To enable comparison, we select feature selection techniques that have been used in recent studies [49, 100, 125, 129, 159].

Table 3.1: Overview of the Studied Feature Selection Techniques

Family	Type	Methods	Abbreviation
Filter-based Feature Ranking Techniques	Statistics-based Techniques	Chi-Square	FR1
		Correlation	FR2
		Clustering Variation	FR3
	Probability-based Techniques	Probabilistic Significance	FR4
		InfoGain	FR5
		GainRatio	FR6
		Symmetrical	FR7
	Instance-based Techniques	ReliefF	FR8
		ReliefF-Weight	FR9
	Classifier-based Techniques	OneR	FR10
		SVM	FR11
Filter-based Subset Selection Techniques	Correlation-based Feature Subset Selection	BestFirst	FS1
		Genetic	FS2
		RankSearch	FS3
	Consistency-based Feature Subset Selection	BestFirst	FS4
		Genetic	FS5
		RankSearch	FS6
Wrapper-based Subset Selection Techniques	k-Nearest Neighbor	Area Under the ROC Curve (AUC)	WS1
		Area Under the Precision-Recall Curve (PRAUC)	WS2
		Root Mean Squared Error (RMSE)	WS3
	Logistic Regression	Area Under the ROC Curve (AUC)	WS4
		Area Under the Precision-Recall Curve (PRAUC)	WS5
		Root Mean Squared Error (RMSE)	WS6
	Naïve Bayes	Area Under the ROC Curve (AUC)	WS7
		Area Under the Precision-Recall Curve (PRAUC)	WS8
		Root Mean Squared Error (RMSE)	WS9
	Repeated Incremental Pruning to Produce Error Reduction	Area Under the ROC Curve (AUC)	WS10
		Area Under the Precision-Recall Curve (PRAUC)	WS11
		Root Mean Squared Error (RMSE)	WS12
None	None	No-filter	None

3.2.1 Filter-Based Feature Ranking Techniques

Filter-based feature ranking-based techniques order features according to importance scores. In our study, we use the Ranker search method with the following importance scores:

Statistics-Based Scores: When using *ChiSquared* [86], feature importance is evaluated by measuring the chi-squared statistics with respect to the class label (i.e., buggy or non-buggy) [118].

Correlation [54] estimates importance using the Pearson Correlation coefficient between the class label and features.

Clustering Variation [42] evaluates the importance of features by measuring their variation coefficients. High variation coefficients for a feature indicate that its values vary across a wide range, which helps in building a more effective classification

Table 3.2: An overview of the studied classification techniques.

Family	Classification Technique	Description
Statistical	Naïve Bayes (NB)	*NB* trains probability-based models assuming that all the features are independent of each other.
	Bayesian network (BN)	*BN* produces a directed graph that is composed of V vertexes and E edges, where each feature is represented by a vertex and each edge represents a successor relationship within the network [112]. *BN* produces a conditional probability of a vertex using the values that are assigned to other vertices [25].
	Logistic Regression (Log)	*Log* is a regression model that is used to study the relationship between a categorical outcome variable and one or more features [114].
Neural Network	Radial Basis Function (RBF)	*RBF* [20] consists of three layers: input (the features), output (the dependent variable) and a middle layer, which connects the input and output layers [112].
	MultiLayer Perceptron (MLP)	*MLP* uses the back-propagation algorithm to train the model. *MLP* produces a directed graph consisting of multiple layers of nodes: an input layer, one or more hidden layers, and an output layer. In each subsequent layer, the output of the previous layer is used as input [112].
Decision Tree	Logistic Model Tree (LMT)	*LMT* [80] constructs a decision tree with logistic regression models to make the final decision at the leaves [80].
	Classification and Regression Tree (CART)	*CART* is a combination of both classification and regression trees. To obtain models, *CART* divides the dataset recursively to fit a simple predictive model within each partition [87].
	J48 (J48)	*J48* [121] is based on the C4.5 technique, which builds decision trees using information entropy. To divide the set of training samples into subsets, a rule is chosen by C4.5 at each node of the decision tree [132].
	Alternating Decision Trees (ADT)	*ADT* produces a tree structure with leaf and decision nodes. The outcomes are represented by leaf nodes, while the other nodes contain decision rules [112].
	Decision Stump (DS)	*DS* trains simple binary decision stumps (i.e., 1-level decision trees) for the classification of both nominal and numeric problems [157].
	Naive Bayes/Decision-Tree (NBT)	*NBT* is a combination of both naïve bayes and decision trees. NBTree produces a non-cost sensitive model, which consists of a naïve bayes model for each leaf [134].
	Random Tree (RT)	*RT* is also known as randomized C4.5 and uses a modified version of the C4.5 algorithm. To introduce the split at each internal node of the tree, the decision is randomized [153].
Rule Based	Repeated Incremental Pruning to Produce Error Reduction (Ripper)	*Ripper* [27] uses induction to remove the rules that result in lower classification performance by creating a series of rules with pruning.
	One Rule (OneR)	*OneR* uses one or more values from a single feature to build classification rules [92].
	Decision Table (DT)	*DT* is an extension of one-valued decision trees, which forms a rectangular table T, where features are represented in the columns and a set of decision rules are represented in the rows. To find a good subset of features, *DT*uses an inductive reduction algorithm to reduce the likelihood of over fitting the dataset and to eliminate equivalent rules [112].
	Partial Decision Trees (PART)	*PART* produces a decision list using the divide-and-conquer approach, building partial decision trees in each iteration and making the "best" leaf into a rule.
	Ripple Down Rules (Ridor)	*Ridor* [47] is a rule-based decision tree technique where the decision tree consists of four nodes: a classification, a predicate function, a true branch, and a false branch. Each instance of the testing dataset is pushed down the tree, following the true and false branches at each node using predicate functions. The final outcome is given by the majority class of the leaf nodes [81]
K-NN	K-Nearest Neighbour (KNN)	*KNN* classifies an instance on the basis of the K most similar training examples. To calculate the distance between instances, KNN uses Euclidean distance [82]. We use K = 10 as recommended by a previous study [159].
	Kstar (Kstar)	*Kstar* works similar to KNN, however Kstar uses an entropy-based function to calculate the distance between instances [82].
Support Vector Machine	Voted Perceptron (VP)	*VP* uses the information of all the classification vectors that are stored during the training phase to generate a better classification on the test dataset [79].
Ensemble Method	Random Forest (RandF)	*RandF* [18] trains an ensemble of decision trees using a randomly selected subspaces of the dataset [13].

model [43].

Probability-Based Scores: *Probabilistic Significance* [2] is a conditional probability-based technique where a significance score is assigned to each feature according to its ability to discriminate between different class labels. A high two-way association score between a feature and a class label indicates more a significant feature.

InfoGain [29] is an entropy-based technique. Information gain measures the reduction in uncertainty of a class label after observing a feature. InfoGain is notably biased toward features with more values [159].

GainRatio [122] penalizes multivalued features to mitigate the bias of InfoGain [159].

Symmetrical Uncertainty [160] evaluates the value of a set of features by measuring their symmetrical uncertainty with respect to another set of features.

Instance-Based Techniques: *ReliefF* [75] is an instance-based ranking technique. An instance from the dataset is randomly sampled and its nearest neighbour is located from the same or the opposite class. The relevance score of each feature is updated by comparing the values of the nearest neighbours features to the sampled instance. This process is repeated for a specified number of instances [57].

ReliefFW is a parameter tuning of *ReliefF*. The parameter 'WeightByDistance' (i.e., weigh the nearest neighbours by their distance) is set to *true* in WEKA.

Classifier-based Techniques: *One Rule* [65] generates a one-level decision rule for each individual feature and features are ranked according to their classification error rate.

SVM [55] uses a Support Vector Machine to estimate feature importance. SVM assigns a weight to each feature and uses the square of the assigned weight to rank the features [16].

3.2.2 Filter-Based Feature Subset Techniques

Subset-based techniques evaluate the importance of each feature and select a subset of relevant features. A brief description of the subset-based techniques follows:

Correlation-based Feature Subset Selection Technique: Correlation [58] subset-based techniques result in subsets of features instead of evaluating individual features. The best subset consists of features with low inter-correlation to each other, but high correlation with the class label.

Consistency-based Feature Subset Selection Technique: Consistency subset-based technique use consistency as an indicator to measure the importance of a feature subset. This technique results in a minimal feature subset whose consistency is equal to that of all the features [32]. We use three different search methods with Correlation and Consistency subset-based techniques:

1. The *BestFirst* search method finds the feature subsets using greedy hillclimbing with backtracking. The method may (a) start with an empty set of features and search forward, (b) start with the full set of features and search backward, or (c) start at any point and search in both directions by considering all possible single attribute additions and deletions at a given point. We use the forward BestFirst search method.

2. *Genetic Search* [52] finds a subset of features using a genetic algorithm.

3. *Rank Search* [57] uses a feature-subset evaluator to rank all features using a selection search method (e.g., forward selection). Then, subsets of increasing size are evaluated from the ranked list of features i.e., the best attribute, the best attribute plus the next best attribute, and so on. The best attribute set is

reported.

3.2.3 Wrapper-Based Feature Subset Techniques

Wrapper-based feature subset techniques use predetermined classification techniques and evaluation measures to evaluate the importance of a feature subset. Cross validation is used to estimate the accuracy of the predetermined classification technique for a set of features.

In this chapter, we employ four classification techniques and three evaluation measures to construct 12 different wrapper-based subset techniques. Naïve Bayes (NB), Logistic Regression (Log), Repeated Incremental Pruning to Produce Error Reduction (Ripper), and K-Nearest Neighbour (KNN) are described in Table 3.2. For KNN, we select $K = 10$ as recommended by a previous study [159].

Evaluation measures:

1. *Area Under the ROC Curve* (AUC) [39] is a trade-off between the true positive rate (TPR) and false positive rate (FPR).

2. *Area Under the Precision-Recall Curve* (AUPRC) [33] is a trade-off between the precision and recall.

3. *Root Mean Squared Error* (RMSE) [23] measures the deviations of the predicted values and the corresponding true values.

3.3 Study Design

In this section, we discuss the corpora that we use in our study, and our approach to construct and evaluate defect classification models. Figure 3.1 provides an overview

of the steps in our approach.

A. Studied Datasets: We study publicly available datasets from two different corpora (i.e., NASA and PROMISE). The datasets consist of different types of features. We use the cleaned version (D") of the NASA corpus as provided by Shepperd *et al.* [137] and the PROMISE corpus [1]. An overview of the number of features, modules and defective modules (i.e., modules having defects > 0) for each dataset in the studied corpora are shown in Table 3.3. To aid in comparability, we select the studied corpora used in previous defect classification studies [50, 82, 125, 159]. The studied corpora are available online (`http://openscience.us/repo/`), enabling further replication (and extension) of our results.

B. Fold Generation: In order to train our classification models, we use the 10×10-fold cross-validation approach. Cross-validation divides the dataset into 10 folds of equal size (i.e., 90% training and 10% testing dataset). The divided training dataset is used to train a model and the testing dataset is used to evaluate the performance of the model. Our 10-fold cross-validation approach is repeated 10 times (100 iterations in total) to validate our results.

C. Preprocessing: In our study, we use three types of feature selection techniques (i.e., filter ranking-based techniques, filter subset-based techniques, and wrapper subset-based techniques) to preprocess the datasets. To apply the feature selection techniques to the datasets, we first divide the dataset into training and testing datasets. We preprocess *only the training dataset* using the feature selection technique, since the label of each instance would not be available in the testing dataset [144]. We do note that some prior studies have mistakenly applied feature selection techniques on the whole data set, which others have warned is a modeling

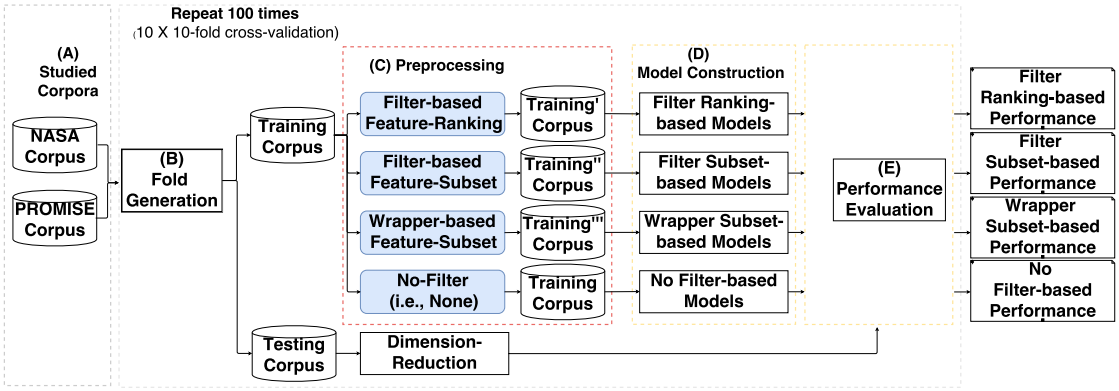

Figure 3.1: An overview of our model construction and evaluation approach.

violation [92, 159]. Incorrect application of feature selection techniques is one possible reason for the contradictory results across prior studies.

While subset-based techniques produce a subset of the original features, ranking-based techniques rank all features. To select the number of features from the list of ranked features, as recommended by prior work [72], we use $log_2 n$, where n is the total number of features. We train our models with the selected features using different types of classification techniques and evaluate the performance of trained classification models.

D. Model Construction: In order to measure the impact of feature selection techniques on the performance of defect classification models, we train our models using 21 classification techniques as described in Table 3.2 on the preprocessed and original datasets.

We selected the most commonly used classification techniques in prior defect classification studies [50, 82, 148]. We use the implementations of the studied classification techniques, as provided by the WEKA [156] machine learning toolkit. In this chapter, we combine 21 classification techniques with 30 feature selection techniques

Table 3.3: An overview of the studied corpora.

Corpus	Dataset	No. of Features	No. of Modules	Defective	Defective (%)
Clean NASA	CM1	37	327	42	12.8
	KC1	21	1,162	294	25.3
	KC3	39	194	36	18.5
	MW1	37	250	25	10
	PC1	37	679	55	8.1
	PC2	37	722	16	2.2
	PC3	37	1,053	130	12.3
	PC4	37	1,270	176	13.8
PROMISE	Ant 1.7	20	745	166	22.3
	Camel 1.6	20	965	188	19.5
	Ivy 1.4	20	241	16	6.6
	Jedit 4.0	20	306	75	24.5
	Log4j 1.0	20	135	34	25.2
	Lucene 2.4	20	340	203	59.7
	Poi 3.0	20	442	281	63.6
	Tomcat 6.0	20	858	77	8.9
	Xalan 2.6	20	885	441	46.4
	Xerces 1.3	20	453	69	15.2

(including a no feature selection configuration) to train our models on 18 datasets ($21 \times 30 \times 18 \times 100$ folds = 1,134,000 folds).

E. Model Evaluation: To compare the performance of defect classification models, we use the Area Under the receiver operating characteristic Curve (AUC) because it is threshold-independent. AUC plots the false positive rate (i.e., $\frac{FP}{FP+TN}$) against the true positive rate (i.e., $\frac{TP}{FN+TP}$). Larger AUC values indicate better performance. AUC values above 0.5 indicate that a model outperforms random guessing.

Statistical Comparison Tests: The Scott-Knott [66] test produces statistically distinct ranks ($\alpha = 0.05$) of classification techniques. The Scott-Knott test uses hierarchical cluster analysis to partition the classification techniques into ranks. The Scott-Knott test starts by dividing the classification techniques into two ranks on the basis of mean AUC values (i.e., mean AUC of the 10×10-fold cross-validation runs for

each classification technique per pre-processor). If the divided ranks are statistically significantly different, then Scott-Knott recursively executes again within each rank to further divide the ranks. The test terminates when ranks can no longer be divided into statistically distinct ranks [70, 95].

In this chapter, instead of using the traditional Scott-Knott test, we use the extended Scott-Knott with effect size awareness (Scott-KnottESD) as suggested by Tantithamthavorn [146]. The extended Scott-Knott test post-processes the statistically distinct ranks that are produced by the traditional Scott-Knott test, while merging any pair of ranks that have a negligible Cohen's d effect size between them [146].

3.4 A Preliminary Comparison of NASA & PROMISE Datasets

In this section, we present a preliminary comparison of the dataset characteristics of the NASA and PROMISE corpora. An overview of the features in the studied corpora is shown in Table 3.4.

3.4.1 Dataset Analysis

We conduct a dataset redundancy characteristics analysis prior to studying the impact of feature selection techniques on defect classification models since:

1. The datasets within the NASA and PROMISE corpora consist of different types of features. For example, the NASA corpus has measurements of size (LOC) and complexity (i.e., McCabe and Halstead), whereas in addition to size and complexity measures, the PROMISE corpus contains OO metrics like coupling and cohesion.

Table 3.4: Features in the studied corpora.

Corpus	Category	Feature	Definition	Rationale
NASA	McCabe	cyclomatic_complexity, cyclomatic_density, design_complexity, essential complexity and pathological_complexity	Measures of the branching complexity of the software module.	Complex software modules may be more prone to defects [91].
	Halstead	content, difficulty, effort, error_est, length, level, prog_time, volume, num_operands, num_operators, num_unique_operands, and num_unique_operators	Estimates of the complexity of reading a software module based on the used vocabulary (e.g., number of operators and operands).	Software modules that are complex to understand may increase the likelihood of incorrect maintenance, and thus, increase defect proneness [69].
	Size	LOC_total, LOC_blank, LOC_comment, LOC_code_and_comment, LOC_executable, and Number_of_lines	Measures of the size of a software module.	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [76].
	Miscellaneous	branch_count, call_pairs, condition_count, decision_count, decision_density, design_density, edge_count, essential_density, parameter_count, maintenance_severity, modified_condition_count, multiple_condition_count, global_data_density, global_data_complexity, percent_comments, normalized_cyclomatic_complexity and node_count	Metrics that are not well defined metrics in the MDP database [92].	N/A
PROMISE	CK	amc(average method complexity), cbm(coupling between methods), cbo(coupling between objects), dit(depth of inheritance tree), ic(inheritance coupling), lcom(lack of cohesion in methods), lcom3(another lack of cohesion measure), noc(number of children), rfc(response for a class), and wmc(weighted methods per class)	Measures of the complexity of a class within an object-oriented system design.	More complex classes may be more prone to defects [51].
	McCabe	avg_cc(average mccabe) and max_cc(maximum mccabe)	Measures of the branching complexity of the software module	Complex software modules may be more prone to defects [91].
	QMOOD	cam(cohesion among classes), dam(data access), mfa(functional abstraction), moa(aggregation), and npm(number of public methods)	Measures of the behavioural and structural design properties of classes, objects, and the relations between them [133].	Higher QMOOD metrics imply higher class quality.On the other hand, lower QMOOD metrics imply lower class quality, which may lead to defects [71].
	Size	loc(lines of code)	Measures of the size of a software module	Larger software modules may be difficult to understand entirely, and thus, may be more prone to defects [76].
	Martin	ca(afferent couplings) and ce(efferent couplings)	Measures of the relationship between software components, including calls as well as number of instances [110].	Higher values indicate low encapsulation,reusability, and maintainability, which may lead to defects [110].

2. The datasets within the NASA corpus consist of features at the module-level, while the PROMISE corpus datasets consist of features at the class-level.

3. The datasets within NASA (i.e., proprietary) and PROMISE (i.e., open source) corpora developed in different settings.

The datasets of both corpora have data redundancy. We use Principal Component Analysis (PCA) to find how many orthogonal components are needed to account for 95% of the data variance in the datasets of each corpus.

The NASA and PROMISE datasets have different data redundancy characteristics. To account for 95% of the data variance, the NASA datasets need

Table 3.5: Number of components that are needed to account for 95% of the data variance.

Corpus	Dataset	Total Components	95% Variance (No. of Components)	95% Variance (No. of Components %)
NASA	CM1	37	11	30
	KC1	21	7	33
	KC3	39	10	26
	MW1	37	11	30
	PC1	37	12	32
	PC2	36	10	28
	PC3	37	12	32
	PC4	37	14	38
PROMISE	Ant	20	12	60
	Camel	20	12	60
	Ivy	20	10	50
	Jedit	20	12	60
	Log4j	20	12	60
	Lucene	20	12	60
	Poi	20	12	60
	Tomcat	20	12	60
	Xalan	20	12	60
	Xerces	20	12	60

an average of 31% of the components while the PROMISE datasets need an average of 59% of the components to account 95% of the data variance. Indeed, the PCA results are very consistent among datasets within each corpus, but very different when comparing datasets across the corpora. There is consistency in the percentage of components that are needed to account for the different levels of data variance among the datasets of each corpus (i.e., NASA and PROMISE) as shown in Table 3.5.

Our PCA analysis shows that the NASA and PROMISE datasets are quite different from each other in the context of data variance information in their feature sets. Figure 3.2 shows the PCA analysis of the CM1 NASA-dataset and the Ant PROMISE-dataset. For CM1, the majority of the data variance information is contained in the first principal component while for Ant, the data variance information is more proportionally divided across the different principal components. Hence, future

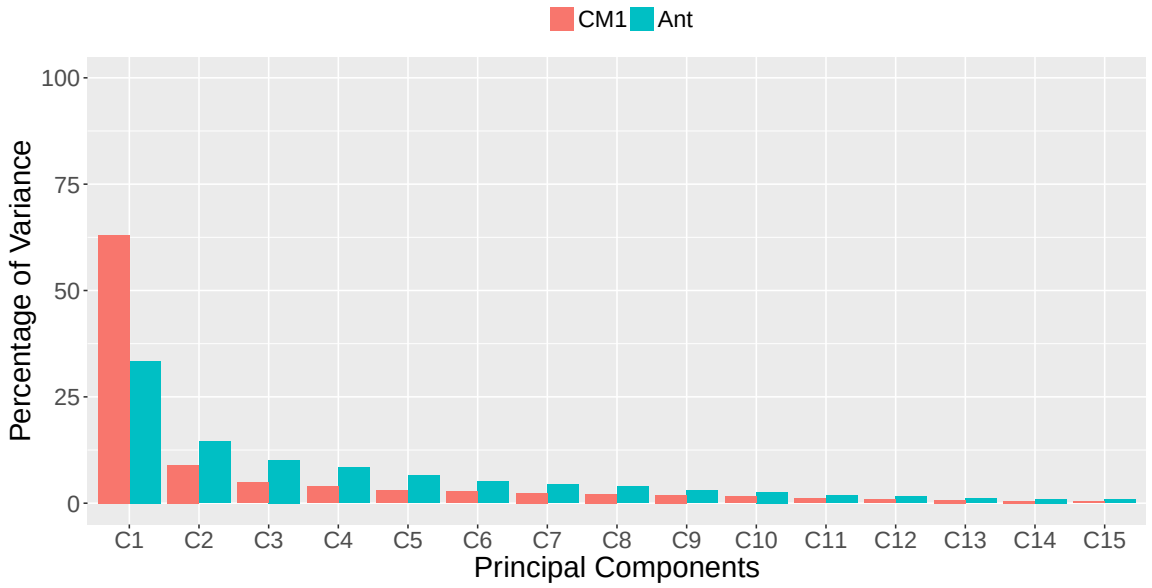

Figure 3.2: PCA for the CM1 NASA dataset and Ant PROMISE dataset.

studies should consider the much richer nature of the PROMISE features versus the NASA features.

The features in the PROMISE corpus are much richer than the NASA features which are very redundant, highlighting the simplistic nature of the NASA corpus and the importance of examining the NASA corpus separately.

3.5 Case Study Results

In this section, we present the results of our case study, which addresses our research question:

RQ: Do feature selection techniques impact the performance of defect classification models?

3.5.1 Approach

To address our research question, we start with the AUC values of the 21 studied classification techniques that are trained using the preprocessed and non-preprocessed datasets. We use the Scott-Knott test to produce statistically distinct ranks of the 30 feature selection techniques.

3.5.1.1 Dataset level

In order to study the impact of feature selection techniques on the performance of defect classification models in each dataset, we calculate the Scott-Knott rank of each feature selection technique across all of the classification techniques. For each dataset, we feed the AUC values of the 10×10 fold cross-validation iterations of 21 classification techniques after applying and without applying the feature selection technique to the Scott-Knott test. The Scott-Knott test results in statistically distinct ranks of 30 feature selection techniques for each dataset i.e., each feature selection technique ends up with eight (i.e., eight datasets studied) and 10 (i.e., 10 datasets studied) Scott-Knott ranks in the NASA and PROMISE corpora, respectively. We then calculate the number of datasets in which each feature selection technique appears in the top-rank.

3.5.1.2 Classification-Technique level

In order to find the impact of feature selection techniques, we feed the AUC values of the 10×10 fold cross-validation iterations of each classification technique after applying and without applying the feature selection technique across all of the datasets to the Scott-Knott test. After the Scott-Knott test run, each feature selection technique

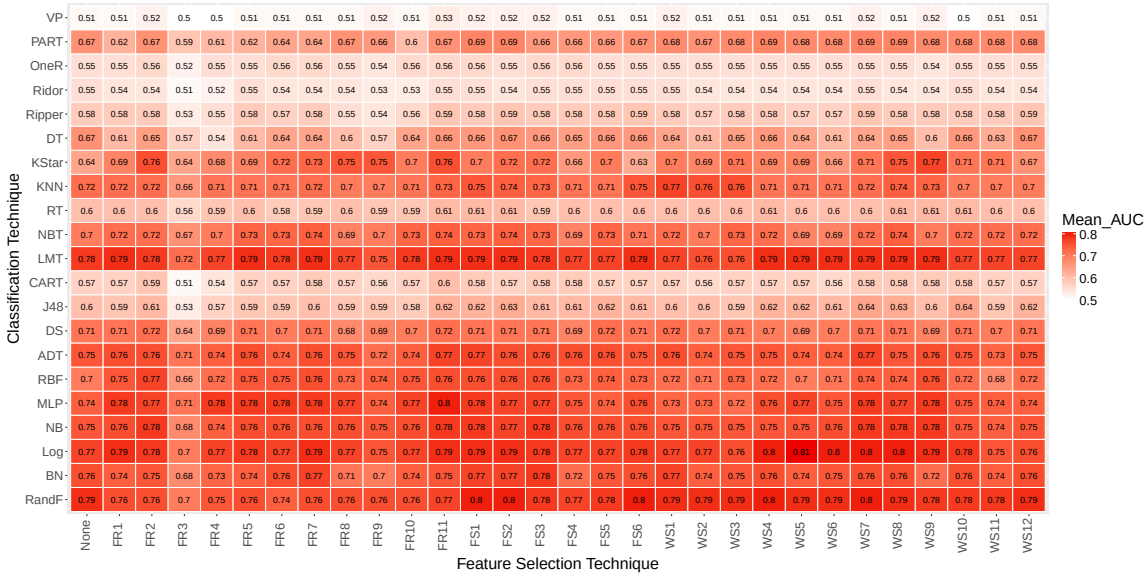

Figure 3.3: Mean AUC value of each feature selection technique for each classification technique across all datasets in the NASA corpus.

has 21 ranks (i.e., one for each of the studied classification technique). We calculate the number of classification techniques in which each feature selection technique appears in the top-rank.

3.5.2 Results

NASA: Classification-Technique level

FS1 outperforms other feature selection techniques, appearing in top Scott-Knott rank for 90% of the studied classification techniques. Figure 3.3 shows the mean AUC values of the 30 feature selection techniques for each classification technique across the NASA corpus. Figure 3.4 shows the Scott-Knott rank of each feature selection technique across the 21 studied classification techniques, FS1 appears in top Scott-Knott rank for 19 of the 21 classification techniques. In addition to FS1, ranking-based techniques (i.e., FR2, FR11), filter subset-based techniques

(i.e., FS2 and FS3), and wrapper-based technique WS8 outperform other feature selection techniques, appearing in the top Scott-Knott rank for >80% of the classification techniques. FR3 is the worst performing feature selection technique, appearing in the top Scott-Knott rank for only the CART classification technique.

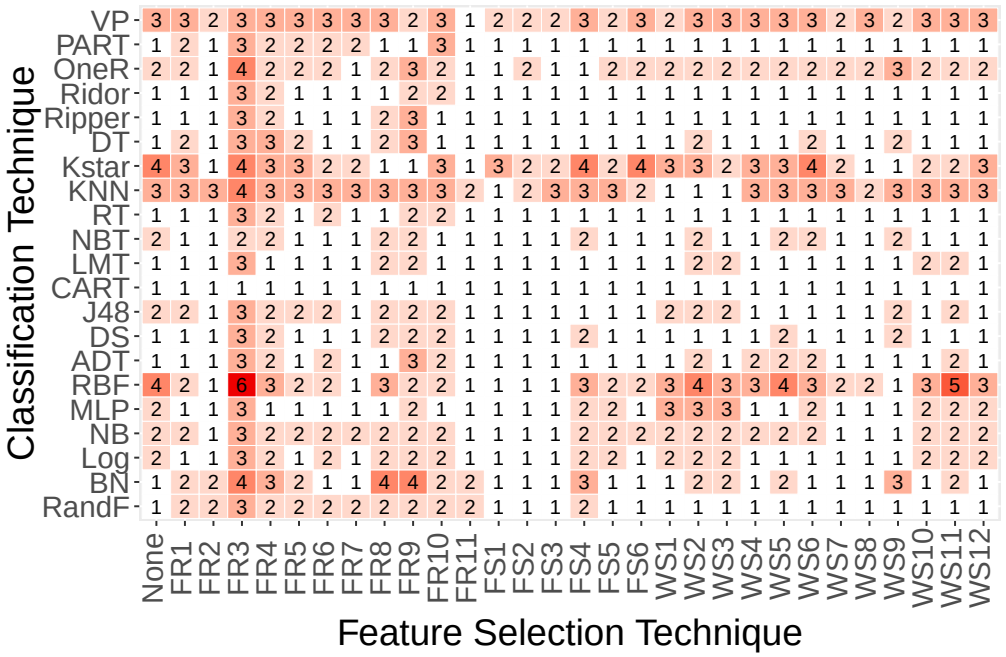

Figure 3.4: Scott-Knott rank of each feature selection technique for each classification technique across all the eight NASA datasets. Darker colors indicate lower Scott-Knott rank.

Our results show that the impact of feature selection techniques also varies across different classification techniques. For CART, all of the 30 feature selection techniques (including a no feature selection configuration) appear in the same statistical rank i.e., CART is not sensitive to feature selection techniques. On the other hand, there is only one feature selection technique (i.e., FR11) that appears in the top Scott-Knott rank for VP.

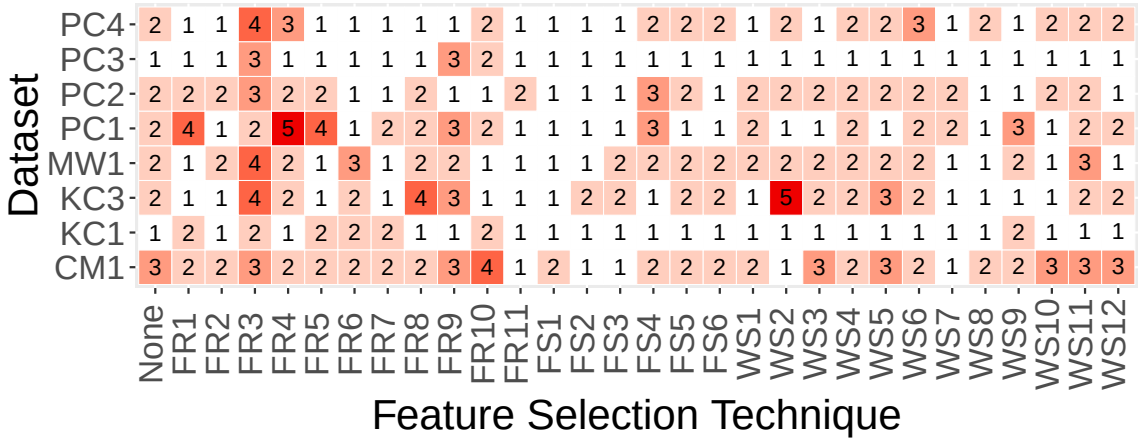

Figure 3.5: Scott-Knott rank of each feature selection technique for each NASA dataset across all the 21 classification techniques. Darker colors indicate lower Scott-Knott rank.

Furthermore, we find that for RandF, filter-based and wrapper-based subset feature selection techniques outperform filter-based ranking ones. This agrees with the findings of Xu *et al.* [159].

NASA: Dataset level

FR11, FS1, and FS2 appear in the top Scott-Knott rank for 87% of the studied datasets. Figure 3.5 shows the Scott-Knott rank of each feature selection technique for each dataset. We find that the filter-based feature subset techniques (i.e., FS1 and FS2) and filter-based feature ranking technique FR11 appears in the top Scott-Knott rank in seven of the eight studied datasets. On the other hand, None, FR4, WS4, WS6, and WS11 are performing worse appearing in the top Scott-Knott rank for only two datasets.

Furthermore, the impact of feature selection techniques varies across the datasets. For dataset CM1, there are only five feature selection techniques that appear in the top Scott-Knott rank, while for dataset PC3, 27 feature selection techniques appear

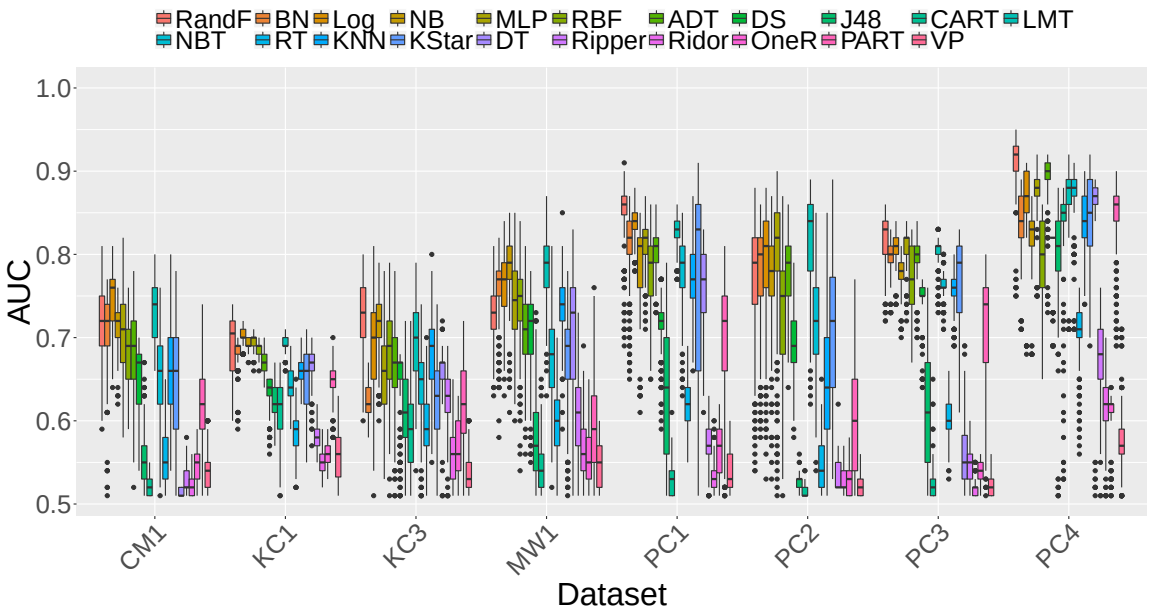

Figure 3.6: The AUC values of each dataset for each classification technique across all the feature selection techniques in the NASA corpus.

in the top Scott-Knott rank. Figure 3.6 highlights the AUC values distribution of the 10×10 fold cross-validation iterations of each classification technique after applying feature selection techniques for each dataset.

The filter-based feature subset technique FS1 outperforms other feature selection techniques, appearing in the top Scott-Knott rank for 87% of the studied datasets and 90% of the studied classification techniques.

PROMISE: Classification-Technique level

FS1 and FS2 techniques outperform other feature selection techniques, appearing in the top Scott-Knott rank for 100% of the studied classification techniques. The mean AUC values of the 30 feature selection techniques for each classification technique across the PROMISE corpus is shown in Figure 3.7. Furthermore, Figure 3.8 shows the Scott-Knott rank of each feature selection techniques

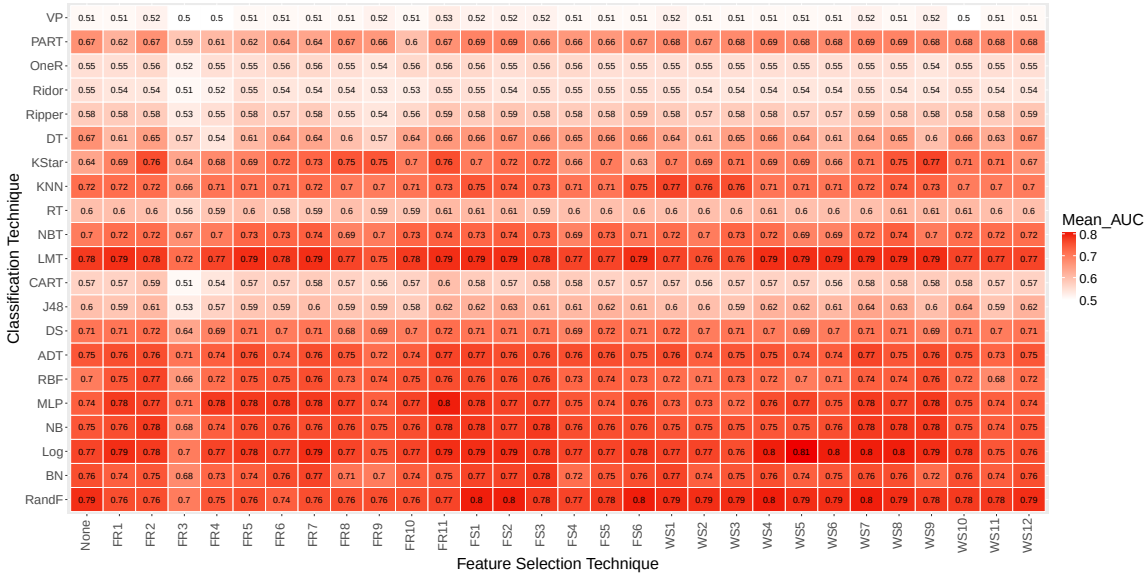

Figure 3.7: Mean AUC value of each feature selection technique for each classification technique across all datasets in the PROMISE corpus.

across all the classification techniques. For all of the 21 studied classification techniques, FS1 and FS2 appear in the top Scott-Knott rank. Similar to the NASA corpus, FR3 appears in the top Scott-Knott rank for least number of classification techniques (i.e., eight only). Furthermore, similar to the NASA corpus, our results support the findings of recent work by Xu *et al.* [159], i.e., for RandF, filter-based and wrapper-based subset feature selection techniques outperform ranking-based ones. However, Xu *et al.* [159] study is only limited to the impact of feature selection techniques on the performance of *Random Forest*.

Our results show a similar pattern to the NASA corpus, the impact of feature selection techniques varies across different classification techniques. First, we find that all of the feature selection techniques appear in the same Scott-Knott rank for CART, confirming that CART is not sensitive to the feature selection configuration across both NASA and PROMISE corpora. In addition to CART, BN, ADT, NBT,

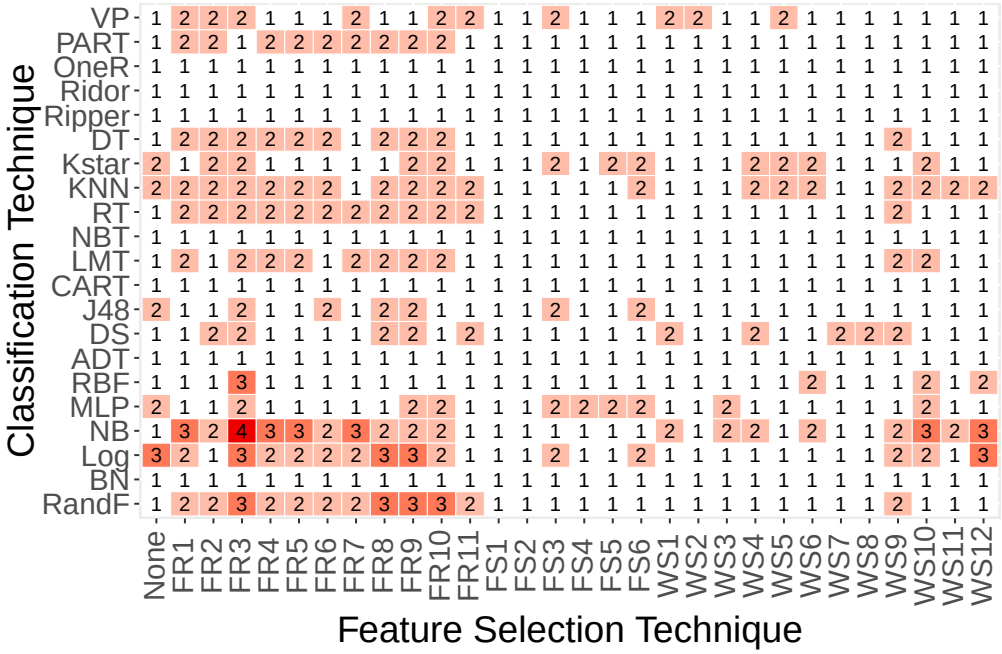

Figure 3.8: Scott-Knott rank of each feature selection technique for each classification technique across all the 10 PROMISE datasets. Darker colors indicate lower Scott-Knott rank.

Ripper, Ridor, and OneR are not sensitive to the feature selection techniques.

In summary, the FS1 feature selection technique consistently outperforms other feature selection techniques across different classification techniques and across different corpora.

PROMISE: Dataset level

FS1, FS2, FS4, WS1, WS4, and WS6 feature selection techniques appear in the top Scott-Knott rank for 70% of the studied datasets. Figure 3.9 shows the Scott-Knott rank of each feature selection technique across the datasets. Filter-based subset techniques (FS1, FS2, and FS4) and Wrapper-based subset techniques (WS1, WS4, and WS6) outperform other feature selection techniques, appearing in the top Scott-Knott rank for seven of the 10 studied datasets. Filter-based

ranking technique FR9 appears in the top Scott-Knott rank for only one dataset.

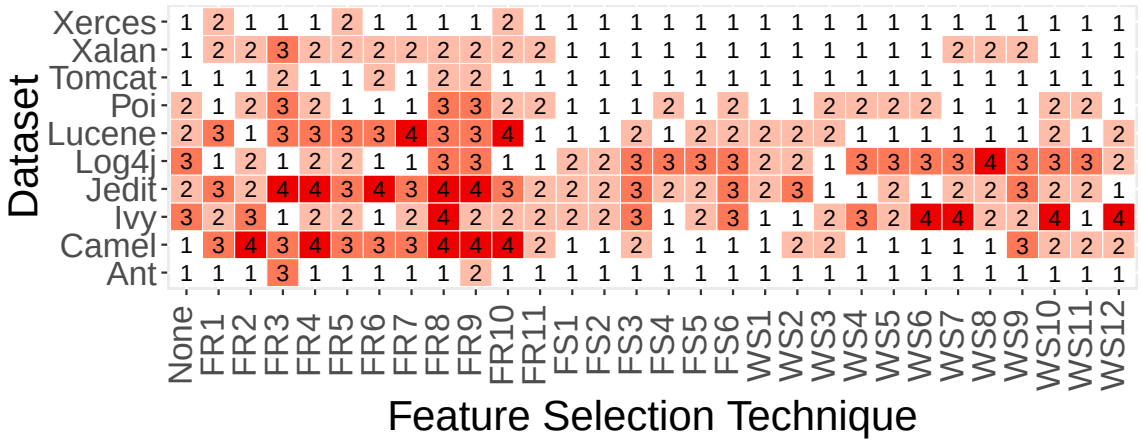

Figure 3.9: Scott-Knott rank of each feature selection technique for each PROMISE dataset across all the 21 classification techniques. Darker colors indicate lower Scott-Knott rank.

Similar to the NASA corpus, our results show that the impact of feature selection techniques varies across the datasets. For Ant and Xerces datasets, 28 feature selection techniques appear in the same statistical rank (i.e., top Scott-Knott rank) while for Jedit there are only four feature selection techniques that appear in the top Scott-Knott rank. The AUC values distribution of the 10×10 fold cross-validation iterations of each classification technique after applying feature selection techniques for each dataset are shown in Figure 3.10.

Similar to our results of the NASA corpus, FS1 appears in the top Scott-rank for 70% of the studied datasets and 100% of the studied classification techniques in the PROMISE corpus.

3.6 Threats to Validity

We now discuss the threats to the validity of our case study.

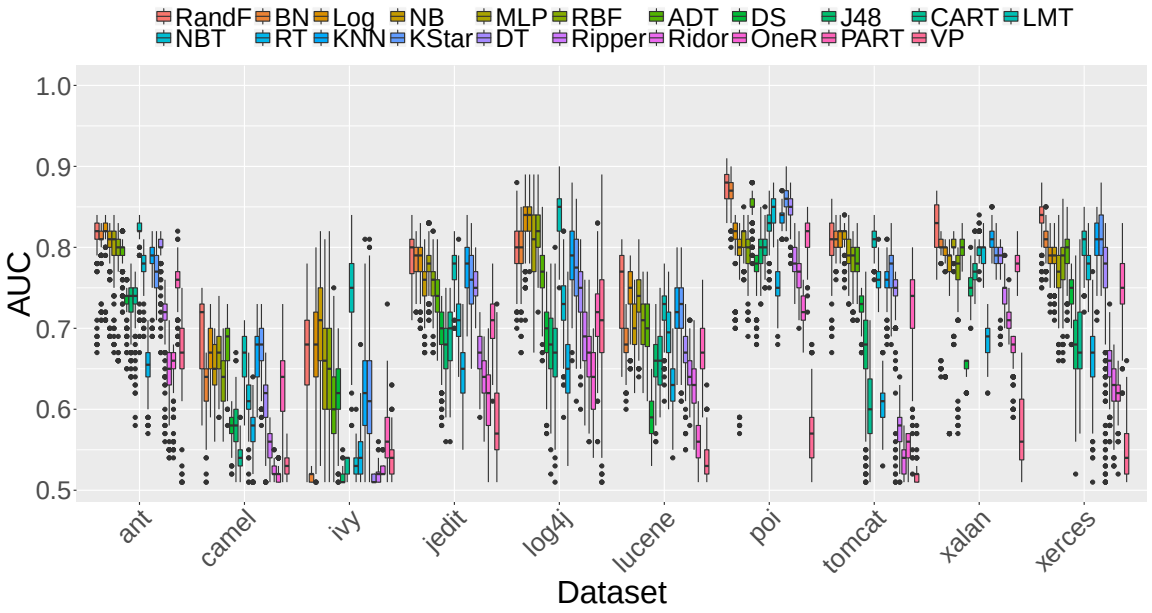

Figure 3.10: The AUC values of each dataset for each classification technique across all the feature selection techniques in the PROMISE corpus.

Construct Validity: We used datasets from two different corpora (NASA and PROMISE) to train our classification models. The corpora that are used consist of different sets of features, which may explain the differences that we observe when drawing conclusions across the corpora. Nevertheless, our main findings still hold on both corpora, i.e., filter-based subset feature selection technique FS1 outperforming other feature selection techniques ending in the top Scott-Knott rank for the majority of the datasets (70% and 87% of the studied datasets for PROMISE and NASA corpora, respectively) and classification techniques (90% and 100% of the studied classification techniques for NASA and PROMISE corpora, respectively) used.

Internal Validity: We apply the feature selection techniques to the studied classification techniques with default parameter settings except for KNN. We use $K = 10$ for KNN for parameter tuning as recommended by a previous study [159].

However, only two (i.e., CART and MLP) out of the 21 used classification techniques have a large impact of parameter tuning as shown in a previous study [148], i.e., the majority of the classification techniques used in our study are not sensitive to parameter tuning. Furthermore, we use the AUC to compare the performance of our models. Other performance measures may yield different results. Future studies should examine classification techniques after applying parameter tuning.

External Validity: We performed our experiments on 18 datasets, which may threaten the generalizability of our study results. On the other hand, these datasets are part of the two most commonly studied corpora (NASA and PROMISE) and have been analyzed by several defect classification studies in the past [82, 112, 116]. We use 30 feature selection techniques (11 filter-based ranking techniques, six filter-based subset techniques, 12 wrapper-based subset techniques, and a no feature selection configuration) and applied them to 21 different classification techniques to train our classification models. Other feature selection techniques may yield different results. However, we include the feature selection techniques that are commonly used in prior studies [49, 125, 129, 159] and applied them to a variety of classification techniques that have also frequently used in past defect classification studies [49, 50, 57, 82, 125, 129, 148, 159]. Furthermore, we use different types of feature selection techniques within the selected three categories — filter ranking-based (i.e., statistics-based, probability-based, instance-based, and classifier-based), filter subset-based (i.e., Correlation-based and Consistency-based), and wrapper-based (i.e., use different four classifiers). Nonetheless, additional replication studies may prove fruitful.

Table 3.6: Comparison of our study to the previous studies

Study	Corpus used	Classification Techniques	Feature Selection Techniques	Feature Selection type	Main Findings
Hall *et al.* [57]	Corpus: UCI repository [84] No. of datasets: 19	No. of techniques: 2 Techniques: C4.5 and NB	No. of techniques: 6	Filter feature ranking: 3 Filter feature subset: 2 Wrapper Subset: 1	Yes, there is an impact. Wrapper-based subset techniques outperform others.
Rodr{i}guez *et al.* [128]	Corpus: PROMISE [140] No. of datasets: 5	No. of techniques: 2 Techniques: C4.5 and NB	No. of techniques: 5	Filter feature ranking: 0 Filter feature subset: 3 Wrapper Subset: 2	Yes, there is an impact. Wrapper-based subset techniques outperform others but it is more computationally expensive.
Muthukumaran *et al.* [100]	Corpus: NASA [137] No. of datasets: 11 Corpus: AEEEM [30] No. of datasets: 5	No. of techniques: 3 Techniques: NB, RandF, and Log	No. of techniques: 10	Filter feature ranking: 7 Filter feature subset: 2 Wrapper Subset: 1	Yes, there is an impact. Filter feature subset outperforms other techniques.
Rathore *et al.* [125]	Corpus: PROMISE [1] No. of datasets: 4	No. of Techniques: 2 Techniques: NB and RandF	No. of techniques:15	Filter feature ranking: 7 Filter feature subset: 6 Wrapper Subset: 1	Yes, there is an impact. Feature ranking techniques outperform other techniques.
Xu *et al.* [159]	Corpus: NASA [137] No. of datasets: 11 Corpus: AEEEM [30] No. of datasets: 5	No. of Techniques: 1 Techniques: RandF	No. of techniques:32	Filter feature ranking: 14 Filter feature subset: 2 Wrapper Subset: 12 Other (Clustering-based and PCA): 4	Yes, there is an impact. Filter-based and wrapper-based feature subset evaluation methods can achieve the best performance.
Menzies *et al.* [92]	Corpus: NASA [137] No. of datasets: 8	No. of Techniques: 1 Techniques: OneR, J48, and NB	No. of techniques: 1	Filter feature ranking: 1 Filter feature subset: 0 Wrapper Subset: 0	Yes, there is an impact. Naïve Bayes produces the best classification model.
Our study	Corpus: NASA No. of datasets: 8 Corpus: PROMISE No. of datasets: 10	No. of techniques: 21 Techniques: RandF, BN, Log, NB, MLP, RBF, ADT, DS, J48, CART, LMT, NBT, RT, KNN, Kstar, DT, Ripper, Ridor, OneR, PART, and VP	No. of techniques: 29	Filter feature ranking: 11 Filter feature subset: 6 Wrapper Subset: 12	Yes, there is an impact. Filter-based subset techniques outperform other feature selection techniques across the projects and across the classification techniques.

3.7 Related Work

Plenty of recent research has examined the high dimensionality problem of datasets using feature selection techniques.

Hall *et al.* [57] find that wrapper-based techniques outperform other techniques, but with a high computational cost. Rodrìguez *et al.* [128] apply three filter and two wrapper-base feature selection techniques. They conclude that the smaller datasets maintain the prediction capability with a lower number of features than the original datasets. In addition, wrapper-based feature selection techniques outperform others. Menzies *et al.* [92] apply the InfoGain feature ranking technique in order to rank features and compare the performance of classification models that are trained using three classification techniques (i.e., NB, J48, and OneR). They conclude that Naïve Bayes yields the best classification model.

Muthukumaran *et al.* [100] studied ten feature selection techniques (seven ranking-based techniques and three subset-based techniques) and find that feature selection techniques tend to improve classification accuracy. The results of their study show that subset-based feature selection techniques outperform other techniques [100].

Rathore *et al.* [125] conducted a study with fifteen different feature selection techniques that consist of ranking-based and subset-based techniques. They find that InfoGain and PCA ranking-based techniques tend to outperform other ranking-based techniques, whereas ClassifierSubsetEval and Logistic Regression outperform other subset-based techniques.

Xu *et al.* [159] conducted a study to compare the impact of 32 feature selection techniques on the performance of defect classification models that are trained using the Random Forest classifier on the three corpora (i.e., noisy NASA, clean NASA, and open source AEEEM). They found a significant difference in the impact of feature selection techniques. Furthermore, their results show that filter and wrapper subset-based techniques outperform other techniques. We select 24 of 32 feature selection techniques used in Xu *et al.* [159] study. However, the scope of Xu *et al.* [159] study limits to a single classification technique (i.e., Random Forest). However, in order to generalize the impact of feature selection techniques, we select the 21 most commonly used classification techniques.

Table 3.6 provides an overview of the prior work. The limitations of the studies discussed above are:

1. The impact of feature selection techniques is studied using a limited number of classification techniques. Hall *et al.* [57], Rodrìguez *et al.* [128], and Rathore *et al.* [125] use only two classification techniques. On the other hand, Muthukumaran *et al.* [100] use three classification techniques to study the impact of feature selection techniques, while Xu *et al.* [159] only study the impact on Random Forest. Since there are some classification techniques that outperform others [50], it is interesting to examine the impact of feature selection techniques

on the performance of different classification techniques. Therefore, in this paper, we study the impact of feature selection techniques on the performance of most commonly used classification techniques in prior defect classification studies [50, 82, 148]

2. Studies conducted by Hall *et al.* [57], Rodrìguez *et al.* [128], and Rathore *et al.* [125] use only datasets from one domain. Since the characteristics of dataset in some specific domains influence the impact feature selection techniques as discussed in Section 3.4, it is important to evaluate the impact on different types of datasets.

Table 3.6 shows that recent studies have concluded that feature selection techniques have an impact on the performance of defect classification models. However, the results of these studies are inconsistent with each other, since the studies were conducted under different conditions, e.g., datasets of different domains, using different classification and feature selection techniques. Hall *et al.* [57] and Rodrìguez *et al.* [128] show that wrapper-based feature subset techniques outperform filter-based feature ranking and feature subset techniques, whereas Muthukumaran *et al.* [100] show that filter-based feature subset techniques outperform filter-based feature ranking and wrapper-based techniques. On the other hand, Xu *et al.* [159] show that filter-based subset techniques and wrapper-based techniques outperform filter-based feature ranking techniques. Furthermore, Rathore *et al.* [125] show that filter-based feature ranking techniques outperform filter-based feature subset and wrapper-based techniques.

To address the inconsistency of the previous studies result, we performed a larger scale study to analyze the impact of 30 feature selection techniques (including a no

feature configuration) on the performance of 21 classification techniques using 18 datasets from the NASA and PROMISE corpora.

3.8 Chapter Summary

In this chapter, we apply 30 feature selection techniques (11 filter-based ranking techniques, six filter-based subset techniques, 12 wrapper-based subset techniques, and a no feature selection configuration) to analyze their impact on the performance of defect classification models that are trained using 21 classification techniques on the NASA and PROMISE corpora. Our results show that:

1. FS1(correlation-based filter-subset technique with the BestFirst search method) feature selection technique outperforms other feature selection techniques, appearing in the top Scott-Knott rank for 70%-87% of the studied datasets in the PROMISE-NASA corpus.

2. FS1 outperforms other feature selection techniques, appearing in the top Scott-Knott rank for 90%-100% of the studied classification techniques in the NASA-PROMISE.

3. The impact of feature selection techniques varies across the datasets, i.e., few feature selection techniques appear in the top Scott-Knott rank of one dataset, whereas for other datasets, there is no statistical difference among the studied feature selection techniques.

4. The impact of feature selection techniques also varies across the classification techniques. There is a clear statistical difference in the impact of feature selection techniques when applied to some of the studied classification techniques,

whereas we find that classification techniques like CART are not sensitive to feature selection techniques, i.e., all of the feature selection techniques appear in the same statistical rank across the NASA and PROMISE corpora.

We would like to emphasize that we do not seek to claim the generalization of our results on other datasets. However, we conclude that the choice of feature selection technique (i.e., ranking-based or subset-based) has an impact on the performance classification models. FS1 outperforms other feature selection techniques across different datasets, corpora, and classification techniques. Hence, we recommend that future defect classification studies should consider applying the FS1 (correlation-based) feature selection technique.

Chapter 4

The Costs and Benefits of using Meta-Learners to Train Defect Classification Models

To train defect classifiers, a variety of classification techniques have been explored. Recent studies focus on meta-learning algorithms that integrate multiple classifiers to construct a meta-classifier that combines the decisions of individual classifiers using an aggregation mechanism (e.g., voting). In this chapter, we investigate the costs and benefits of using meta-learners to train defect classifiers. More specifically, we study abstract meta-learners, which allow the base classification technique to vary (e.g., Bagging), as well as concrete meta-learners, which incorporate a particular base classification technique (e.g., Random Forest). Through analysis of two concrete meta-learners, eight abstract meta-learners, and 15 base classification techniques, we observe that: (1) abstract or concrete meta-learners produce top performing classifiers – outperforming base classification techniques; and (2) The performance of abstract and concrete meta-learners is statistically identical. Furthermore, both abstract and concrete meta-learners consume equal training time (except when abstract meta-learners combined with the base classification technique MLP) to train classifiers. However, concrete meta-learners produce less complex models than the abstract meta-learners. Therefore, we recommend that future studies employ concrete meta-learners, which

train classifiers that perform statistically similar to classifiers that are trained using abstract meta-learners; however, produce a less complex model.

4.1 Introduction

The maintenance of large and complex software systems is a big challenge for the software industry. The probability of having defective modules in large software systems remains high. Since testing an entire software system is time and resource-intensive, Software Quality Assurance (SQA) teams typically focus on a subset of system modules. This is practical because the majority of faults are present in a small proportion of system modules [5, 41].

Defect classifiers help SQA teams to identify software modules that are likely to be defective in the future. SQA teams can use such classifiers to allocate their limited resources to areas of the system that are most likely to be defect-prone. The classifiers are usually trained using software process (e.g., churn) and product (e.g., complexity) metrics. To train defect classifiers, researchers have explored different types of base classification techniques [36, 50, 73, 82, 99].

Prior work explores meta-learners, which combine base classification techniques using different aggregation mechanisms (e.g., voting) to enhance the accuracy of the model fit. Recent studies [45, 151, 155] show that meta-learners like Bagging or Boosting are often more accurate and robust to the effects of noisy datasets, and often yield more accurate and robust fits when compared to the base classification techniques. Meta-learners can be divided into two categories: (1) abstract meta-learners that accept the base classification technique as a parameter when producing a meta-classifier (e.g., Bagging and Boosting); and (2) concrete meta-learners which incorporate a particular base classification technique (e.g., Random Forest and Logistic Model Tree).

Studies [53, 115, 153, 161] show that meta-learners tend to produce classifiers that outperform classifiers that are trained using the base classification techniques. On

the other hand, other studies [40, 101, 102, 136] indicate that the performance may vary under different experimental settings. Little is known about how different types of meta-learners impact the defect classifiers in terms of the performance and the training cost.

In this chapter, we investigate the costs and benefits of meta-learners. We compare eight abstract meta-learners, two concrete meta-learners, and 15 base classification techniques. Through analysis of the NASA and PROMISE corpora, we address the following research questions:

RQ1 Do meta-learners impact the performance of defect classifiers?

Our results show that classifiers that are trained using abstract or concrete meta-learners appear in the top rank for both the NASA and PROMISE corpora. Furthermore, no classifiers that are trained using base classification techniques appear in the top rank. These results suggest that meta-learners provide a statistically significant boost in the performance of defect classifiers and should be always used.

RQ2 How much is the cost of training defect classifiers using meta-learners?

The total training time that is consumed by abstract meta-learners in the top statistical rank to train classifiers is almost equivalent to concrete meta-learners (except when abstract meta-learners combined with base classification technique MLP) across the NASA and PROMISE datasets.

The results of our analysis lead us to conclude that defect classifier that is trained using meta-learners significantly outperform classifiers that are trained using the base classification techniques. Although, the performance of classifiers that are produced

by abstract and concrete meta-learners is statistically indistinguishable. Furthermore, both abstract and concrete meta-learners consume almost equal training time, however, concrete meta-learners produce less complex models to train classifiers than the abstract meta-learners. Therefore, we recommend that future defect classification studies use concrete meta-learners to train defect classifiers.

Chapter Organization: The remainder of the chapter is organized as follows. Section 4.2 surveys related work. Section 4.3 describes the studied meta-learners. Section 4.4 presents the design of our case study, while Section 4.5 discusses the results. Section 4.6 discloses the threats to the validity. Finally, Section 4.7 draws conclusions.

4.2 Related Work

Prior studies show that the performance of classifiers that are trained using meta-learners is superior to classifiers that are trained using base classification techniques [50, 68, 82, 115, 152, 153, 161].

Tosun *et al.* [152] proposed an ensemble of three base classification techniques (i.e., Naive Bayes, Neural Networks, and Voting Feature Intervals) to train defect classifiers. Zheng [161] compared the performance of classifiers that are trained using three cost-sensitive boosting meta-learners when applied to a neural network classification technique. However, both Tosun *et al.* [152] and Zheng's [161] work focused on a single ensemble method.

Peng *et al.* [115] use three meta-learners (Bagging, Boosting, and Stacking) to train the defect classifiers and compared the performance to classifiers that are trained using 12 base classification techniques. In their work, they only compare the performance

of abstract meta-learners with the base classification techniques, however there is no comparison of classifiers that are trained using concrete meta-learners (e.g., Random Forest). Furthermore, there is no training cost analysis of the different meta-learners.

Wang *et al.* [153] applied abstract (i.e., Bagging, Boosting, Random Subspace, Stacking, and Voting) and concrete (i.e., Random Forest) meta-learners to train defect classifiers. The performance of classifiers that are trained using meta-learners is compared to the classifiers that are trained using the base classification technique (i.e., Naive Bayes). In their, study they only applied abstract meta-learners (Bagging, Boosting, RandomSubspace) to a single base classification technique (i.e., Naive Bayes), however the impact of meta-learners on other base classification techniques is unknown. Lessmann *et al.* [82] compare classifiers that are trained using 19 base classification techniques to the classifiers that are trained using two concrete meta-learners. However, in their study the impact of abstract meta-learners is not studied and there is no cost analysis of meta-learner to train the classifiers.

Table 4.1: Difference in previous studies and our study.

Comparisons	**Zheng**	**Peng**	**Wang**	**Tosun**	**Lessmann**	**Our Study**
Ensemble of base classifiers (Simple Voting)				✓		
Abstract ML only	✓					✓
Base vs Concrete ML			✓		✓	✓
Base vs Abstract ML		✓	✓			✓
Concrete ML vs Abstract ML			✓			✓
Performance based comparison	✓	✓	✓		✓	✓
Cost Benefit Analysis	✓					✓

Table 4.1 shows that there are still meta-learners whose impact on the defect classifiers has not been fully explored so far in terms of the performance and the training cost in the above mentioned studies.

4.3 Meta-Learners

In this section, we briefly explain the meta-learners used in our study.

4.3.1 Abstract Meta-learner

An abstract meta-learner takes a base classification technique as a parameter i.e., different type of classification techniques can be supplied to an abstract meta-learner. In this chapter, we apply eight abstract meta-learners to 15 studied classification techniques to train our classifiers. A brief description of the studied abstract meta-learners follows:

Adaboost *(ML1)*: *ML1* [45] trains individual base classifiers sequentially to construct a composite meta classifier. *ML1* constructs an ensemble by performing multiple iterations each time using different example weights. The weights of training examples are adjusted (i.e., correctly classified examples weights are decreased while the weights of the misclassified examples are increased) in each training round which makes the classifier concentrate on different examples and results in diverse classifiers. Finally, a simple voting of classifications by individual classifiers is used to form a composite classifier [161].

Bagging *(ML2)*: *ML2* [17] is one of the simplest meta-learners that is used to improve the performance of a base classifier. Bagging creates multiple datasets of equal size from the original dataset using bootstrapping. The created datasets is then supplied to base classification techniques and the final outcome is measured using voting of the individual classifiers [127].

Dagging *(ML3)*: *ML3* [150] creates a number of disjoint, stratified folds out of the dataset and feeds each chunk of the dataset to a copy of the base classifier. *ML3* gets

the final outcome using a majority voting scheme using disjoint samples instead of bootstrapping as used in Bagging [150].

DECORATE *(ML4)*: *ML4* (Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples) [89] creates an ensemble iteratively by training a classifier at each iteration and adding it to the current ensemble. The base classifier is trained initially using the given training dataset. In each successive iteration, the classifier is trained using the original training dataset and an artificial dataset. The artificial training dataset is generated from the dataset distribution; where the number of examples to be generated is specified as a fraction of the training dataset. The new classifier is trained on the union of the original training dataset and the artificially generated dataset. The newly trained classifier is not added to the current ensemble if it increased the training error of the current ensemble.

MultiBoost *(ML5)*: *ML5* [154] is a combination of AdaBoost with wagging (i.e., a variant of Bagging which uses re-weighting for each training dataset instead of using re-sampling of the datasets for training). *ML5* takes advantage of the variance reduction characteristic of wagging to control the high bias and variance that is produced by AdaBoost [77].

Random Subspace *(ML6)*: *ML6* [64] uses a random selection feature approach to generate a random forest that consists of multiple decision trees. From the selected features, a subset of instances is randomly chosen to assign to the learning algorithm.

RealAdaBoost *(ML7)*: *ML7* [46] is a generalized form of AdaBoost in which a weak classifier returns a class probability estimate (buggy or non-buggy). The contribution to the final classifier is half of the logit transform of the probability estimate.

Rotation Forest *(ML8)*: *ML8* [130] constructs an ensemble of decision trees by combining randomly chosen feature subsets and bagging approaches with principal feature generation. *ML8* creates diversity for each base classifier using Principal Component Analysis (PCA) for feature selection.

4.3.2 Concrete Meta-learner

Concrete meta-learners combine different base classifiers together to solve one problem. Classifiers that are trained using concrete meta-learners typically generalize better than those trained using the base classification techniques. However, alike abstract meta-learners, concrete meta-learners incorporate a particular base classification technique to train the defect classifiers. A brief description of the used concrete meta-learners follows:

Random Forest *(RF)*: *RF* [18] trains an ensemble of decision trees using a randomly selected subspaces of the dataset [13].

Logistic Model Tree *(LMT)*: *LMT* [80] is a decision tree like structure with logistic regression functions at the leaves [80].

4.4 Study Design

In this section, we discuss the used corpora in our study, our approach to construct and evaluate the performance of defect classifiers. Figure 4.1 provides an overview of steps in our approach.

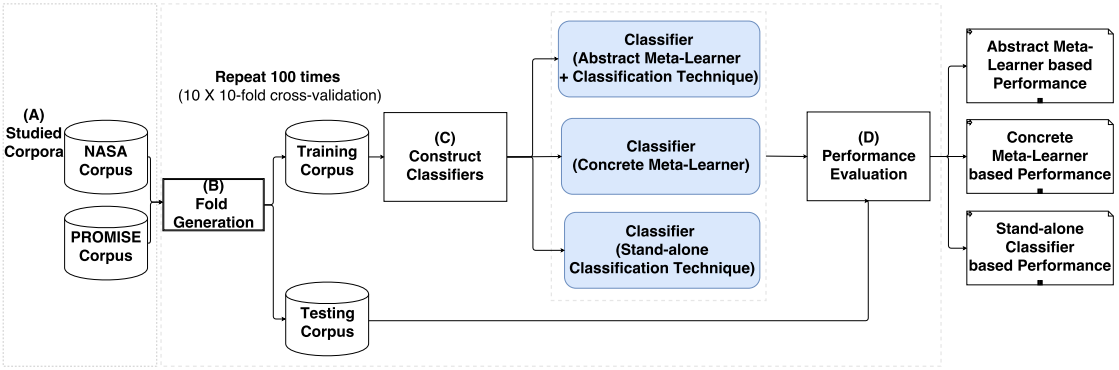

Figure 4.1: An overview of our classifier construction and performance evaluation approach.

4.4.1 Studied Datasets

Our study consists of datasets from two different corpora; cleaned version (D") of the NASA corpus as provided by Shepperd *et al.* [137] and the PROMISE corpus [1]. An overview of the studied corpora is shown in Table 4.2. The studied corpora that we use in our study are available online(`http://openscience.us/repo/`), enabling further replication (and extension) of our results.

4.4.2 Fold Generation

We use a 10 × 10-fold cross-validation approach to train our classifiers. Cross-validation divides the dataset into 10 folds of equal size. Of the 10 folds, 9 are allocated to the training dataset (90%) that is used to train the classifier, while 1 fold is set aside for the testing dataset (10%) that is used to evaluate the performance of the classifier. In order to validate our results, we repeat the 10-fold cross-validation approach ten times (100 total iterations).

Table 4.2: An overview of the studied corpora.

Corpus	Dataset	No. of Modules	Defective	Defective (%)
Clean NASA	CM1	327	42	12.8
	KC1	1,162	294	25.3
	KC3	194	36	18.5
	MW1	250	25	10
	PC1	679	55	8.1
	PC2	722	16	2.2
	PC3	1,053	130	12.3
	PC4	1,270	176	13.8
PROMISE	Ant 1.7	745	166	22.3
	Camel 1.6	965	188	19.5
	Ivy 1.4	241	16	6.6
	Jedit 4	306	75	24.5
	Log4j 1	135	34	25.2
	Lucene 2.4	340	203	59.7
	Poi 3	442	281	63.6
	Tomcat 6	858	77	8.9
	Xalan 2.6	885	441	46.4
	Xerces 1.3	453	69	15.2

4.4.3 Classifier Construction

In order to measure the impact of meta-learners have on the performance of defect classifiers, we use 15 base classification techniques. Table 4.3 provides an overview of the classification techniques (*see* Table 3.2 in Chapter 3 for description) use to train our classifiers. The base classification techniques used in our study are part of previous defect classification studies [50, 82, 148, 153]. We use the implementations of studied classification techniques as provided by the WEKA machine learning toolkit [156]. In this chapter, we train our classifiers using 15 base classification techniques, two concrete meta-learners, and applied eight abstract meta-learners described in Section 4.3 to the 15 studied base classification techniques on 18 datasets

$((15 \times 18 + 2 \times 18 + 8 \times 15 \times 18) \times 100$ iterations $= 246,600$ folds).

4.4.4 Classifier Performance Evaluation

To compare the performance of defect classifiers, we use the Area Under the receiver operating characteristic Curve (AUC) [15] because it is threshold-independent. AUC plots the false positive rate (i.e., $\frac{FP}{FP+TN}$) against the true positive rate (i.e., $\frac{TP}{FN+TP}$). Larger AUC values indicate better performance. AUC values above 0.5 indicate that a classifier performs better than random guessing.

Table 4.3: An overview of the studied classification techniques.

Family	**Classification Technique**	**Abbreviation**
Statistical	Naïve Bayes	NB
	Logistic Regression	Log
	Bayesian Network	BN
Neural Network	Radial Basis Function	RBF
	MultiLayer Perceptron	MLP
Decision Tree	Classification and Regression Tree	CART
	J48	J48
	Alternating Decision Tree	ADT
	Decision Stump	DS
	Random Tree	RT
Rule Based	Repeated Incremental Pruning to Produce Error Reduction	Ripper
	One Rule	OneR
	Partial Decision Tree	PART
K-NN	K-Nearest Neighbour	KNN
Support Vector Machine	Voted Perceptron	VP

4.5 Case Study Results

In this section, we present our case study results with respect to our two research questions:

(RQ1) Do meta-learners impact the performance of defect classifiers?

Approach

We use the Scott-Knott test [66] to group classification techniques into statistically distinct ranks of all the classifiers used in our study. The Scott-Knott test [66] produces statistically distinct groups ($\alpha = 0.05$) of classification techniques. The Scott-Knott test uses a hierarchical cluster analysis to partition classification techniques into ranks. The Scott-Knott test starts by dividing classification techniques into two ranks on the basis of mean AUC values. If the divided ranks are statistically significantly different, then Scott-Knott recursively executes again within each rank to further divide the ranks. The test terminates when ranks can no longer be divided into statistically distinct ranks [70, 95].

In this chapter, instead of using the traditional Scott-Knott test, we use the extended Scott-Knott with effect size awareness (Scott-KnottESD) suggested by Tantithamthavorn *et al.* [146]. The extended Scott-Knott test post-processes the statistically distinct ranks that are produced by the traditional Scott-Knott test, while merging any pair of ranks that have a negligible Cohen's d effect size between them [146].

In order to find the statistically distinct ranks of all the techniques used in our study, we use the double Scott-Knott approach as proposed in our previous study [50]. Our approach of double Scott-Knott is shown in Figure 4.2. For each base classification technique, we input AUC values of 10×10-fold cross-validation iterations after

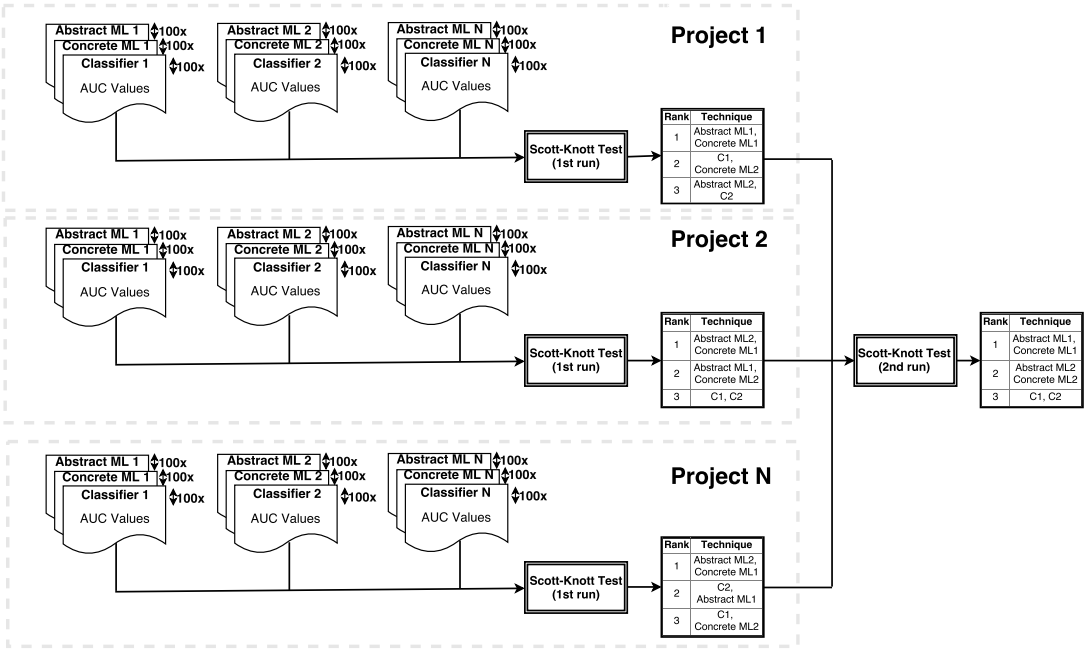

Figure 4.2: Double Scott-Knott approach to produce statistically distinct ranks of classifiers.

applying and without applying abstract meta-learners, and AUC values (10×10-fold cross-validation iterations) of concrete meta-learners to the first run of the Scott-Knott test for each dataset. Each classifier has one rank for each dataset. For the second run of the Scott-Knott test, we input the ranks of each classifier from all the datasets which produces statistically distinct ranks of all the classifiers that are trained using base classification techniques, abstract and concrete meta-learners.

Result

NASA: In the NASA corpus, we observe that classifiers that are trained using abstract and concrete meta-learners outperform classifiers that are trained using base classification techniques. Figure 4.3 shows the mean AUC values and statistically distinct ranks that are produced by the double Scott-Knott

test of all the classifiers. Classifiers that appear in the top Scott-Knott rank are shown in red color while classifiers that appear in the second to last Scott-Knott rank are shown in the light red to white color. *None* indicates classifiers where abstract meta-learners are not applied (i.e., the base classification techniques, and the concrete meta-learners).

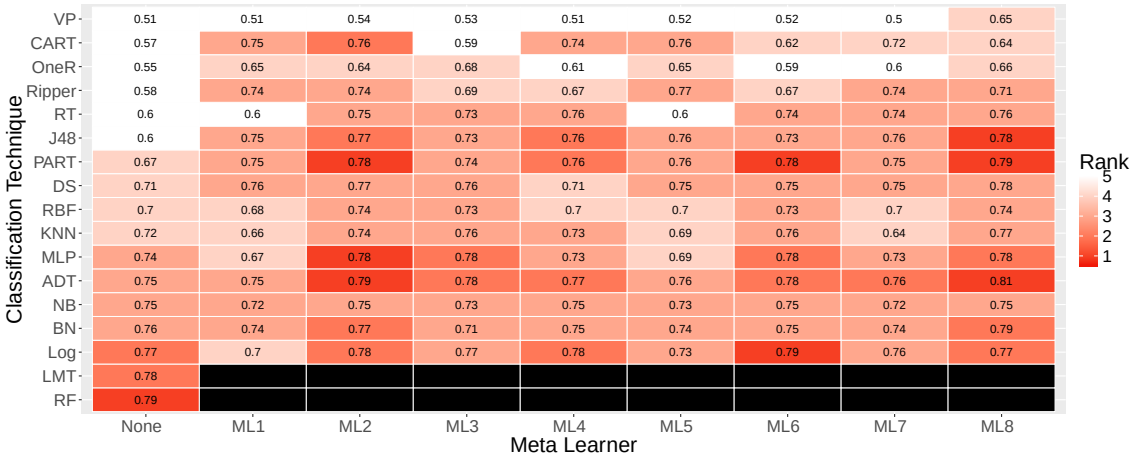

Figure 4.3: Mean AUC and the Scott-Knott rank of the base classifiers, concrete and abstract meta-learner classifiers in the NASA corpus. Lighter colors indicate lower statistical ranks.

Figure 4.3 shows that the abstract meta-classifiers (Log+ML6, ADT+ML2, ADT+ML8, MLP+ML2, PART+ML2, PART+ML6, PART+ML8, and J48+ML8), and the concrete meta-classifier RF outperforms other classifiers. Furthermore, we find that there is no base classification technique which tends to appear in the top Scott-Knott rank. The black color is used to indicate that the abstract meta-learners are not applied to the concrete meta-learners (RF and LMT).

In terms of the performance of defect classifiers, the LMT and RF concrete meta-classifiers achieve AUC values of 0.78 and 0.79 respectively, across all the datasets in the NASA corpus. We calculate the performance difference between the abstract

meta-classifiers and classifiers that are trained using the base classification techniques, our results show that the abstract meta-learners are associated with an AUC boost of the trained classifiers by up to 19 percentage over classifiers that are trained using the base classification techniques. The maximum AUC boost is for CART (with ML2, and ML5) and Ripper (with ML5). Figure 4.4 shows the AUC distribution of the meta-learners and the studied base classification techniques (shown as None).

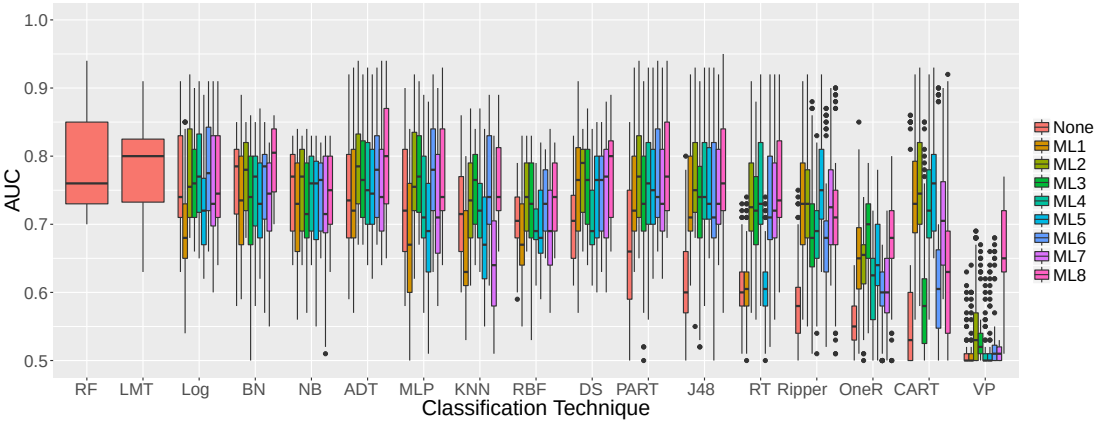

Figure 4.4: AUC distribution of concrete meta-learners, abstract meta-learners, and the base classification techniques in the NASA corpus.

PROMISE: Similar to the NASA corpus, we observe that the abstract or concrete meta-classifiers appear in the top Scott-Knott rank. On the other hand, no classifiers that are trained using the base classification techniques appear in the top Scott-Knott rank. Figure 4.5 shows the mean AUC of all the classifiers. The double Scott-Knott test produces four statistically distinct ranks of classifiers. The classifiers appearing in the top Scott-Knott rank are shown in the red color while classifiers appearing in the second to last Scott-Knott rank is shown in the light red to white color.

Figure 4.5 shows that the abstract meta-classifiers (Log+ML4, Log+ML6, BN+ML2,

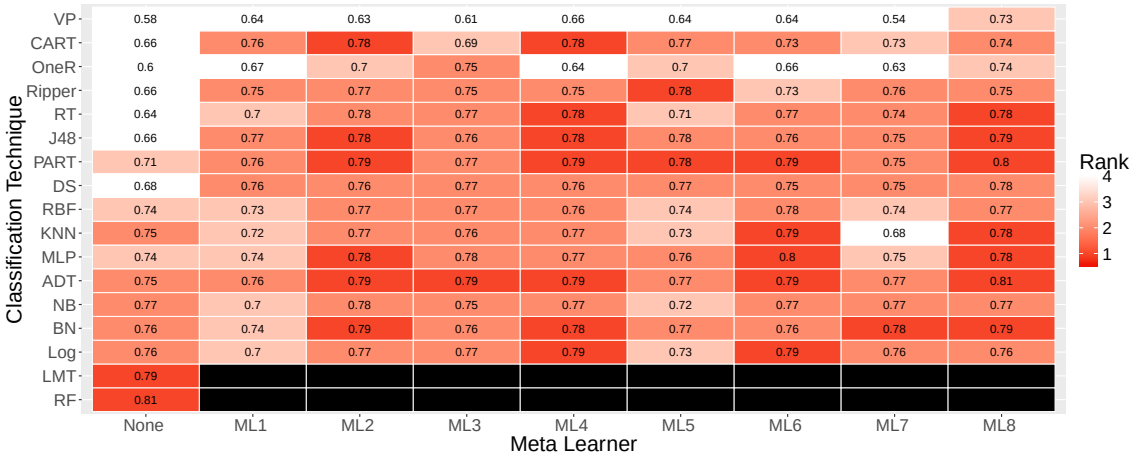

Figure 4.5: Mean AUC and the Scott-Knott rank of the base classifiers, concrete and abstract meta-learner classifiers in the PROMISE corpus. Lighter colors indicate lower statistical ranks.

BN+ML4, BN+ML7, BN+ML8, ADT+ML2, ADT+ML3, ADT+ML4, ADT+ML6, ADT+ML8, MLP+ML2, MLP+ML6, MLP+ML8, KNN+ML6, KNN+ML8, PART+ML2, PART+ML4, PART+ML5, PART+ML6, PART+ML8, J48+ML2, J48+ML4, J48+ML8, RT+ML4, RT+ML8, Ripper+ML5, CART+ML2, and CART+ML4), and both concrete meta-classifiers (RF and LMT) appear in the top Scott-Knott rank.

In the PROMISE corpus, LMT and RF achieved an AUC values of 0.79 and 0.81, respectively while the abstract meta-classifiers yield a maximum AUC boost of up to 15 percentage points over classifiers that are trained using the base classification techniques. The maximum boost is for OneR (with ML3), and VP (with ML8). Figure 4.6 shows the AUC distribution of the studied meta-learners and the base classification techniques (shown as None).

Classifiers that are trained using abstract or concrete meta-learner outperform classifiers that are trained using the base classification techniques in both the NASA and PROMISE corpora.

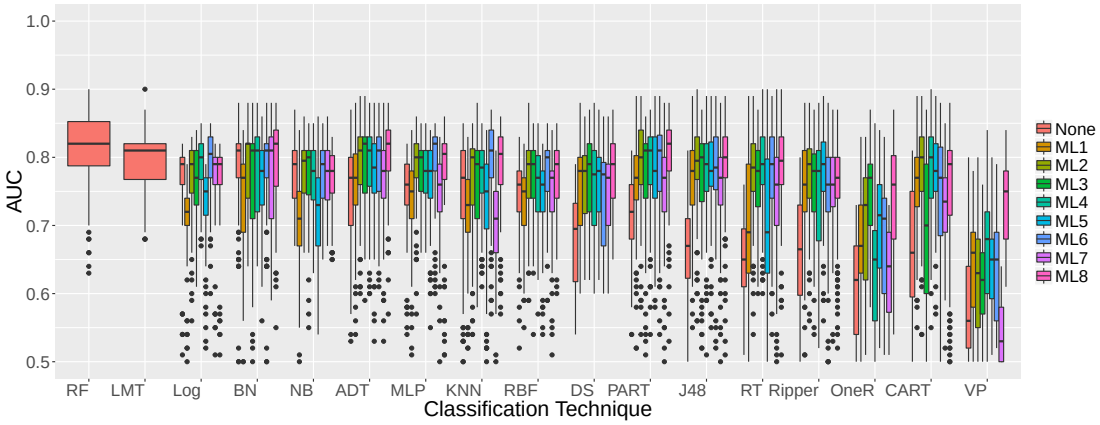

Figure 4.6: AUC distribution of concrete meta-learners, abstract meta-learners, and the base classification techniques in the PROMISE corpus.

(RQ2) How much is the cost of training defect classifiers using meta-learners?

Approach

In order to find the effectiveness of each meta-learner in terms of the training cost, we calculate the total execution time that is consumed by abstract and concrete meta-learners appearing in the top Scott-Knott rank (i.e., from RQ1) to train a classifier (since such classifiers have statistically similar performance).

Result

NASA: Our results show that abstract and concrete meta-learners are consuming almost equal training time to train classifiers (except when the abstract meta-learners are combined with the classification technique MLP). The training time is shown in Table 4.4 is the total time that is consumed by each meta-learner to train a classifier across all the datasets in the NASA corpus. Except the abstract meta-learners that are combined with MLP, the training time of

Table 4.4: Training time of abstract and concrete meta-learners appearing in the top Scott-Knott rank in the NASA corpus.

Category	Meta-learner	Base Classification Technique	Training Time (sec)
Concrete	None	RF	200.85
Abstract	ML2	ADT	725.015
		MLP	33,230.735
		PART	383.66
	ML6	Log	305.123
		PART	263.493
	ML8	ADT	1,216.208
		J48	306.519
		PART	437.818

top-performing abstract meta-learners shown in Table 4.4 is 4 to 20 minutes while concrete meta-learner (i.e., RF) consumes three minutes to train the classifier. The abstract meta-learner (i.e., ML2) is consuming the highest time (i.e., 553 minutes) to train a classifier when combined with the classification technique MLP. However, we find that ML2 is consuming only six minutes when combined with the classification technique PART. Therefore, the base classification technique MLP results in the high training time of the abstract meta-learner ML2.

PROMISE: Similar to the NASA corpus, we observe that the total training time that is consumed by the abstract and concrete meta-learners to train classifiers is almost equivalent (except for meta-learners that are combined with MLP and the abstract meta-learner ML4 that is consuming higher training time than the other abstract meta-learners when combined with different classification techniques). The training time in the Table 4.5 is the total time that each meta-learner is consuming to train a classifier across all the datasets of the PROMISE corpus. Except the abstract meta-learners that are combined with the base classification technique MLP and classifiers that are trained

Table 4.5: Training time of abstract and concrete meta-learners appearing in the top Scott-Knott rank in the PROMISE corpus.

Category	Meta-learner	Base Classification Technique	Training Time (sec)
Concrete	None	LMT	223.655
		RF	179.959
Abstract	ML2	ADT	283.798
		BN	202.554
		CART	427.333
		J48	235.122
		MLP	13,607.781
		PART	225.8
	ML3	ADT	240.502
	ML4	ADT	3,440.327
		BN	1,551.301
		CART	7,816.239
		J48	1,046.895
		Log	1,711.114
		PART	1,814.32
		RT	244.853
	ML5	PART	302.037
		Ripper	241.262
	ML6	ADT	260.146
		KNN	228.984
		Log	281.856
		MLP	5,313.506
		PART	303.678
	ML7	BN	220.692
	ML8	ADT	485.591
		BN	255.406
		J48	277.809
		KNN	213.074
		MLP	14,715.646
		PART	282.708
		RT	287.382

using abstract meta-learner ML4, the training time of top-performing abstract meta-learners is 3 to 8 minutes while the training time of concrete meta-learners (i.e, RF and LMT) are three minutes and 3 minutes 43 seconds, respectively.

Furthermore, we find that the abstract meta-learner (i.e., ML8) is consuming the

highest time (i.e., 245 minutes) when combined with the base classification technique MLP to train a classifier. However, ML8 consume only three minutes when combined with the base classification technique KNN to train a classifier. In addition, the abstract meta-learner ML4 is consuming higher training time when combined with the base classification technique CART (i.e., 130 minutes).

Across both the NASA and PROMISE corpora, we find that the base classification technique MLP results in the high training time (i.e., 553 minutes and 245 minutes across NASA and PROMISE corpora, respectively) when applied to the abstract meta-learners.

Abstract and concrete meta-learners are consuming almost equal training time to train defect classifiers (except when the abstract meta-learners combined with the base classification technique MLP across both NASA and PROMISE corpora). However, concrete meta-learners produce less complex models than the abstract meta-learners yet the performance of classifiers that are trained using the concrete meta-learners and abstract meta-learners is indistinguishable.

4.6 Threats to Validity

In this section, we disclose the threats to the validity of our case study.

4.6.1 Construct Validity

We used datasets from two different corpora (NASA and PROMISE) to train our defect classifiers. The corpora that are used consist of different sets of features, which may explain the differences that we observe when drawing conclusions across the corpora. Nevertheless, our main findings still hold on both corpora, i.e., (1) classifiers

that are trained using abstract or concrete meta-learners outperform classifiers that are trained using base classification techniques and (2) classifiers that are trained using abstract and concrete meta-learners are statistically similar in performance and consume almost equal training time, however concrete meta-learners produce less complex models to train classifiers than the abstract meta-learners.

4.6.2 Internal Validity

We apply meta-learners to the studied base classification techniques to train defect classifiers with default parameter settings except for KNN. We use $K = 10$ for KNN for parameter tuning as recommended by a previous study [159]. However, only two (i.e., CART and MLP) out of the 15 used base classification techniques have a large impact of parameter tuning as shown in a previous study [148], i.e., the majority of the base classification techniques used in our study are not sensitive to parameter tuning. Furthermore, we use the AUC to compare the performance of our models. Other performance measures may yield different results. Future studies should include base classification techniques that are sensitive to the parameter tuning and other performance measures.

4.6.3 External Validity

We performed our experiments on limited number of datasets, which may threaten the generalizability of our study results. However, the 18 studied datasets are part of two most commonly used corpora (NASA and PROMISE) and have been analyzed by several defect classification studies in the past [50, 82, 112, 116, 148, 153]. We train our

classifiers using 15 base classification techniques, two concrete meta-learners, and applied eight abstract meta-learners to 15 studied base classification techniques. Other classification techniques and meta-learners may yield different results, however, we choose classification techniques belonging to six different families and that are part of previous defect classification studies [24, 50, 82, 148]. Furthermore, the studied meta-learners have been used in several previous studies [68, 82, 115, 153]. Nonetheless, additional replication studies may prove fruitful.

4.7 Chapter Summary

In this chapter, we study the impact of eight abstract meta-learners, and two concrete meta-learners on the performance of defect classifiers that are trained using 18 datasets from the NASA and PROMISE corpora. Our results show that:

- Classifiers that are trained using abstract or concrete meta-learners appear in the top Scott-Knott rank. On the other hand, no classifiers that are trained using base classification techniques appear in the top rank for either the NASA and PROMISE corpora.

- The performance of classifiers that are trained using abstract and concrete meta-learners is statistically indistinguishable. Furthermore, the training time of abstract meta-learners (except the abstract meta-learners that are combined with the base classification technique MLP) appear in the top statistical rank is equivalent to concrete meta-learners across NASA and PROMISE corpora, yet concrete meta-learners produce less complex models than the abstract meta-learners.

We conclude that the abstract and concrete meta-learners train significantly better-performing classifiers as compared to classifiers that are produced by most commonly used nowadays base classification techniques. Furthermore, concrete meta-learners train classifiers that are statistically similar in the performance to those that are trained by the abstract meta-learners, however, produces less complex models. Hence, we recommend that future defect classification studies that are performance and training cost focused should use concrete meta-learners to train defect classifiers.

Part II

Epilogue

Chapter 5

Summary, Conclusion, and Future Work

5.1 Summary

Software Quality Assurance (SQA) teams must use their limited resources effectively and efficiently to areas of software systems that are most likely to be defective in future. To that end, defect classification models can help SQA teams to identify defect-prone software modules. However, the classification accuracy of a software defect classification model may be influenced by various experimental settings. Therefore, in this thesis, we study three different experimental settings: (1) choice of classification technique to train a defect classification model; (2) impact of applying feature selection techniques on defect classification models and (3) impact of applying meta-learners on defect classification models.

5.2 Conclusion and Key Findings

5.2.1 Findings

The key findings of this thesis show that:

Choice of Classification Techniques: Defect classification model performance is impacted by the choice of the classification technique that is used to train it — some classification techniques tend to produce defect classification models that outperform others.

Impact of Feature Selection: Defect classification model performance is impacted by feature selection techniques — a Correlation-based filter-subset feature selection technique with a BestFirst search method outperforms other feature selection techniques across the studied datasets (it outperforms in 70–87% of the PROMISE–NASA data sets) and across the studied classification techniques (it outperforms for 90% of

the techniques).

Impact of Meta-Learners: Defect classification model performance is impacted by meta-learners — abstract or concrete meta-learners produce top performing classifiers – outperforming base classification techniques; and The performance of abstract and concrete meta-learners is statistically identical and consume equal training time (except when abstract meta-learners combined with the base classification technique MLP) to train classifiers, however, the concrete meta-learners produce less complex models than the abstract meta-learners.

5.3 Future Work

In this thesis, we study three different experimental settings and believe that our findings will help future defect classification studies to train the defect classification models. However, there are various other experimental settings that may impact the performance of defect classification models. In this section we highlight some directions for future work:

1. In the Chapter 2, we study the impact of choice of a classification technique on the performance of defect classification models using datasets of two corpora (i.e., NASA and PROMISE). Furthermore, we train our classification models using default parameter setting of the classification techniques. However, Tantithamthavorn ea [148] shows that parameter tuning of classification technique has an impact on the performance of defect classification models. Therefore, a priority of future work should be to revisit the impact of choice of classification techniques after applying parameter tuning.

2. We examine the impact of feature selection techniques along three different categories (i.e., filter-based ranking techniques, filter-based subset techniques, and wrapper-based subset techniques) on the performance of classification models that are trained using a variety of classification techniques. However, parameter tuning of both feature selection techniques and classification technique should be an extension of Chapter 3.

3. We only study the impact of feature selection techniques on the performance of classification models that are trained using stand-alone and concrete classification techniques, however, the impact of feature selection techniques on meta-learners is unknown.

4. A priority of future work should include datasets that consist of different software features than the ones in the NASA and PROMISE corpora.

Bibliography

[1] Studied corpora repository:. `http://openscience.us/repo/`.

[2] A. Ahmad and L. Dey. A feature selection technique for classificatory analysis. *Pattern Recognition Letters*, 26(1):43–56, 2005.

[3] F. Akiyama. An example of software system debugging. In *IFIP Congress (1)*, volume 71, pages 353–359, 1971.

[4] D. S. Alberts. The economics of software quality assurance. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 433–442. ACM, 1976.

[5] C. Andersson. A replicated empirical study of a selection method for software reliability growth models. *Empirical Software Engineering*, 12(2):161–182, 2007.

[6] Ö. F. Arar and K. Ayan. Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33:263–277, 2015.

[7] E. Arisholm and L. C. Briand. Predicting fault-prone components in a java legacy system. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 8–17. ACM, 2006.

[8] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[9] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[10] A. Bernstein, J. Ekanayake, and M. Pinzger. Improving defect prediction using temporal features and non linear models. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 11–18, 2007.

[11] A. Berson, S. Smith, and K. Thearling. An overview of data mining techniques. *Building Data Mining Application for CRM*, 2004.

[12] N. Bettenburg, M. Nagappan, and A. E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 60–69. IEEE Press, 2012.

[13] G. Biau. Analysis of a random forests model. *The Journal of Machine Learning Research*, 13(1):1063–1095, 2012.

[14] A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.

[15] A. 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.

[16] J. Brank, M. Grobelnik, N. Milic-Frayling, and D. Mladenic. Feature selection using linear support vector machines. In *Proceedings of the third international conference on data mining methods and databases for engineering, finance and other fields. Bologna, Italy*, 2002.

[17] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[18] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] L. C. Briand, J. Wüst, and H. Lounis. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical software engineering*, 6(1):11–58, 2001.

[20] M. D. Buhmann. Radial basis functions. *Acta Numerica 2000*, 9:1–38, 2000.

[21] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, and E. Cialini. Merits of organizational metrics in defect prediction: an industrial replication. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 89–98. IEEE, 2015.

[22] E. Ceylan, F. O. Kutlubay, and A. B. Bener. Software defect identification using machine learning techniques. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 240–247. IEEE, 2006.

[23] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[24] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. *International Journal on Artificial Intelligence Tools*, 17(02):389–400, 2008.

[25] J. Cheng and R. Greiner. Comparing bayesian network classifiers. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 101–108. Morgan Kaufmann Publishers Inc., 1999.

[26] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[27] W. W. Cohen. Fast Effective Rule Induction. In *Proceedings of the International Conference on Machine Learning*, pages 115–123, 1995.

[28] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[29] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[30] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41. IEEE, 2010.

[31] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.

[32] M. Dash and H. Liu. Consistency-based search in feature selection. *Artificial intelligence*, 151(1):155–176, 2003.

[33] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

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

[35] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[36] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[37] L. Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23, 2000.

[38] M. E. Fagan. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*, pages 301–334. Springer, 2001.

[39] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[40] N. E. Fenton and M. Neil. A critique of software defect prediction models. *Software Engineering, IEEE Transactions on*, 25(5):675–689, 1999.

[41] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software engineering*, 26(8):797–814, 2000.

[42] S. Fong, J. Liang, R. Wong, and M. Ghanavati. A novel feature selection by clustering coefficients of variations. In *Digital Information Management (ICDIM), 2014 Ninth International Conference on*, pages 205–213. IEEE, 2014.

[43] S. Fong, J. Liang, and Y. Zhuang. Improving classification accuracy using fuzzy clustering coefficients of variations (fccv) feature selection algorithm. In *Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on*, pages 147–151. IEEE, 2014.

[44] C. Fraley and A. E. Raftery. Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of Classification*, 24(2):155–181, 2007.

[45] Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156, 1996.

[46] J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[47] B. R. Gaines and P. Compton. Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, 5(3):211–228, 1995.

[48] K. Gao, T. M. Khoshgoftaar, and N. Seliya. Predicting high-risk program modules by selecting the right software measurements. *Software Quality Journal*, 20(1):3–42, 2012.

[49] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya. Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience*, 41(5):579–606, 2011.

[50] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 789–800. IEEE Press, 2015.

[51] B. Goel and Y. Singh. Empirical investigation of metrics for fault prediction on object-oriented software. In *Computer and Information Science*, pages 255–265. Springer, 2008.

[52] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[53] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 417–428. IEEE, 2004.

[54] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[55] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[56] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented

metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[57] M. Hall, G. Holmes, et al. Benchmarking attribute selection techniques for discrete class data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 15(6):1437–1447, 2003.

[58] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[59] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[60] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[61] G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, pages 281–288, 2004.

[62] K. Hammouda and F. Karray. A comparative study of data clustering techniques. *Fakhreddine Karray University of Waterloo, Ontario, Canada*, 2000.

[63] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[64] T. K. Ho. The random subspace method for constructing decision forests.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, 1998.

[65] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[66] E. Jelihovschi, J. C. Faria, and I. B. Allaman. Scottknott: A package for performing the scott-knott clustering algorithm in r. *Trends in Applied and Computational Mathematics*, 15(1):003–017, 2014.

[67] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013*, pages 279–289. IEEE, 2013.

[68] A. Kaur and K. Kaur. Performance analysis of ensemble learning for predicting defects in open source software. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 219–225. IEEE, 2014.

[69] K. Kaur, K. Minhas, N. Mehan, and N. Kakkar. Static and dynamic complexity analysis of software metrics. *World Academy of Science, Engineering and Technology*, 56:2009, 2009.

[70] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan. Prioritizing the devices to test your app on: a case study of android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 610–620. ACM, 2014.

[71] Y. Khan and O. Khararah. A systematic review on the relationships between mood/qmood metrics and external software quality attributes. Technical report, Technical report, Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, 2014.

[72] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse. An empirical study of learning from imbalanced data using random forest. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, volume 2, pages 310–317. IEEE, 2007.

[73] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning. Application of neural networks for predicting program faults. *Annals of Software Engineering*, 1(1):141–154, 1995.

[74] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, pages 489–498. IEEE Computer Society, 2007.

[75] I. Kononenko. Estimating attributes: analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer, 1994.

[76] A. G. Koru, D. Zhang, K. El Emam, and H. Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2009.

[77] S. Kotsianti and D. Kanellopoulos. Combining bagging, boosting and dagging for classification problems. In *International Conference on Knowledge-Based*

and Intelligent Information and Engineering Systems, pages 493–500. Springer, 2007.

[78] S. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[79] S. Kotsiantis, K. Patriarcheas, and M. Xenos. A combinational incremental ensemble of classifiers as a technique for predicting students′ performance in distance education. *Knowledge-Based Systems*, 23(6):529–535, 2010.

[80] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[81] B. Lemon. *The Effect of Locality Based Learning on Software Defect Prediction*. PhD thesis, West Virginia University, 2010.

[82] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[83] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead Jr. Does bug prediction support human developers? findings from a google case study. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 372–381. IEEE Press, 2013.

[84] M. Lichman. UCI machine learning repository, 2013.

[85] H. Liu and H. Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

[86] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *ICTAI*, pages 388–391, 1995.

[87] W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[88] H. Lu, B. Cukic, and M. Culp. Software defect prediction using semi-supervised learning with dimension reduction. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 314–317. IEEE, 2012.

[89] P. Melville and R. J. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *IJCAI*, volume 3, pages 505–510. Citeseer, 2003.

[90] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 343–351. IEEE, 2011.

[91] T. Menzies, J. S. Di Stefano, M. Chapman, and K. McGill. Metrics that matter. In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, page 51. IEEE Computer Society, 2002.

[92] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[93] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[94] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1):1–17, 2012.

[95] N. Mittas and L. Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on Software Engineering*, 39(4):537–551, 2013.

[96] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[97] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190. ACM, 2008.

[98] R. E. Mullen and S. S. Gokhale. Software defect rediscoveries: a discrete lognormal model. In *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, pages 10–pp. IEEE, 2005.

[99] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.

[100] K. Muthukumaran, A. Rallapalli, and N. Murthy. Impact of feature selection techniques on bug prediction models. In *Proceedings of the 8th India Software Engineering Conference*, pages 120–129. ACM, 2015.

[101] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE transactions on software engineering*, 25(4):510–525, 1999.

[102] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(5):380–391, 2005.

[103] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 580–586. ACM, 2005.

[104] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 284–292. ACM, 2005.

[105] N. Nagappan and T. Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 364–373. IEEE, 2007.

[106] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[107] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pages 309–318. IEEE, 2010.

[108] T. T. Nguyen, T. N. Nguyen, and T. M. Phuong. Topic-based defect prediction (nier track). In *Proceedings of the 33rd international conference on software engineering*, pages 932–935. ACM, 2011.

[109] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[110] M. F. Oliveira, R. M. Redin, L. Carro, L. d. C. Lamb, and F. R. Wagner. Software quality metrics and their impact on embedded software. In *Proceedings of the 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, 2008*, pages 68–77. IEEE Computer Society, 2008.

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

[112] A. Panichella, R. Oliveto, and A. De Lucia. Cross-project defect prediction models: L'union fait la force. In *Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 164–173. IEEE, 2014.

[113] L. Pelayo and S. Dick. Applying novel resampling strategies to software defect prediction. In *NAFIPS 2007-2007 Annual Meeting of the North American Fuzzy Information Processing Society*, pages 69–72. IEEE, 2007.

[114] C.-Y. J. Peng and T.-S. H. So. Logistic regression analysis and reporting: A primer. *Understanding Statistics: Statistical Issues in Psychology, Education, and the Social Sciences*, 1(1):31–70, 2002.

[115] Y. Peng, G. Kou, G. Wang, W. Wu, and Y. Shi. Ensemble of software defect predictors: an ahp-based evaluation method. *International Journal of Information Technology & Decision Making*, 10(01):187–206, 2011.

[116] F. Peters, T. Menzies, and A. Marcus. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 409–418. IEEE Press, 2013.

[117] S. L. Pfleeger. Software metrics: progress after 25 years? *IEEE Software*, (6):32–34, 2008.

[118] R. L. Plackett. Karl pearson and the chi-squared test. *International Statistical Review/Revue Internationale de Statistique*, pages 59–72, 1983.

[119] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods*, pages 185–208. MIT Press, 1999.

[120] D. Pyle. *Data preparation for data mining*, volume 1. Morgan Kaufmann, 1999.

[121] J. R. Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

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

[123] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

[124] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu. Bugcache for inspections: hit or miss? In *Proceedings of the 19th ACM SIGSOFT symposium*

and the 13th European conference on Foundations of software engineering, pages 322–331. ACM, 2011.

[125] S. S. Rathore and A. Gupta. A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. In *Proceedings of the 7th India Software Engineering Conference*, page 7. ACM, 2014.

[126] J. Ratzinger, T. Sigmund, and H. C. Gall. On the relation of refactorings and software defect prediction. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 35–38. ACM, 2008.

[127] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 43. ACM, 2014.

[128] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz. Detecting fault modules applying feature selection to classifiers. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, pages 667–672. IEEE, 2007.

[129] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, J. Aguilar-Ruiz, and M. Garre. Attribute selection in software engineering datasets for detecting fault modules. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pages 418–423. IEEE, 2007.

[130] J. Rodríguez, L. Kuncheva, and C. Alonso. Rotation forest: A new classifier

ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[131] P. S. Sandhu, S. Kumar, and H. Singh. Intelligence system for software maintenance severity prediction. *Journal of Computer Science*, 3(5):281, 2007.

[132] L. Sehgal, N. Mohan, and P. S. Sandhu. Quality prediction of function based software using decision tree approach. In *Proceedings of the International Conference on Computer Engineering and Multimedia Technologies (ICCEMT)*, pages 43–47, 2012.

[133] A. Shaik, C. Reddy, B. Manda, C. Prakashini, and K. Deepthi. Metrics for object oriented design software systems: a survey. *Journal of Emerging Trends in Engineering and Applied Sciences*, 1(2):190–198, 2010.

[134] S. Sheng and C. X. Ling. Hybrid cost-sensitive decision tree. In *Knowledge Discovery in Databases: PKDD 2005*, pages 274–284. Springer, 2005.

[135] M. Shepperd, D. Bowes, and T. Hall. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616, 2014.

[136] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[137] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, (9):1208–1215, 2013.

[138] E. Shihab. *An exploration of challenges limiting pragmatic software defect prediction*. PhD thesis, Queen's University, 2012.

[139] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 62. ACM, 2012.

[140] J. S. Shirabad and T. J. Menzies. The promise repository of software engineering databases. *School of Information Technology and Engineering, University of Ottawa, Canada*, 24, 2005.

[141] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim. Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, 39(4):552–569, 2013.

[142] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 249–258. IEEE, 2002.

[143] Y. Singh and A. S. Chauhan. Neural networks in data mining. *Journal of Theoretical and Applied Information Technology*, 5(6):36–42, 2009.

[144] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, 37(3):356–370, 2011.

[145] R. Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297–310, 2003.

[146] C. Tantithamthavorn. ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test. `https://cran.r-project.org/web/packages/ScottKnottESD/index.html`, 2016.

[147] C. Tantithamthavorn. Towards a better understanding of the impact of experimental components on defect prediction modelling. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 867–870. ACM, 2016.

[148] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*, pages 321–332. ACM, 2016.

[149] G. Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, 7007(011), 2002.

[150] K. M. Ting and I. H. Witten. Stacking bagged and dagged models. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 367–375. Morgan Kaufmann Publishers Inc., 1997.

[151] K. M. Ting and Z. Zheng. A study of adaboost with naive bayesian classifiers: Weakness and improvement. *Computational Intelligence*, 19(2):186–200, 2003.

[152] A. Tosun, B. Turhan, and A. Bener. Ensemble of software defect predictors: a case study. In *Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 318–320. ACM, 2008.

[153] T. Wang, W. Li, H. Shi, and Z. Liu. Software defect prediction based on classifiers ensemble. *Journal of Information & Computational Science*, 8(16):4241–4254, 2011.

[154] G. I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine learning*, 40(2):159–196, 2000.

[155] T. Wilson, J. Wiebe, and R. Hwa. Recognizing strong and weak opinion clauses. *Computational Intelligence*, 22(2):73–99, 2006.

[156] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[157] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. *Proceedings of ANNES'99 International Workshop on emerging Eng*, 1999.

[158] S. Wu and P. Flach. A scored auc metric for classifier evaluation and selection. 2005.

[159] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia. The impact of feature selection on defect prediction performance: An empirical comparison. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 309–320. IEEE, 2016.

[160] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*, volume 3, pages 856–863, 2003.

[161] J. Zheng. Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, 37(6):4537–4543, 2010.

[162] Z.-H. Zhou. Ensemble learning. In *Encyclopedia of Biometrics*, pages 270–273. Springer, 2009.

[163] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, pages 531–540. ACM, 2008.

[164] T. Zimmermann, R. Premraj, and A. Zeller. Predicting Defects for Eclipse. In *Proceedings of the International Workshop on Predictor Models in Software Engineering*, 2007.