SOFTWARE DEFECT PREDICTION USING MAXIMAL INFORMATION COEFFICIENT AND FAST CORRELATION-BASED FILTER FEATURE SELECTION

by

BONGEKA MPOFU

submitted in accordance with the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: Prof E Mnkandla

November 2017

I can't change the direction of the wind, but I can adjust my sails to always reach my destination.

~ Jimmy Dean

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

JDT	Java Development Tools
PDE	Plug-in Development Environment
FCBF	Fast Correlation Based Filter
LOC	Lines of Code
VCS	Version Control System
OSS	Open source software
VCS	Version Control System
SCCS	Source Code Control System
RCS	Revision Control System
MIC	Maximal Information Coefficient
DIT	Depth of Inheritance Tree
NOC	Number of children
WMC	Weighted Methods per Class
CBO	Coupling between objects
CFS	Correlation Based Feature Selection
AUC	Area Under the Curve
ROC	Receiver Operating Characteristic
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
NASA	National Aeronautics and Space Administration
FCM	Fuzzy C Means
CART	Classification And Regression Tree
SU	Symmetric Uncertainty
MI	Mutual Information
IG	Information Gain
DBN	Deep Belief Network

RBM	Restricted Boltzmann’s Machines
PCA	Principal Component Analysis PCA
CD	Critical Distance CD

LIST OF PUBLISHED PAPERS

The study contained in this thesis has yielded a number of publications. The following are the publications related to this thesis.

1. Mpofu, B & Mnkandla, E. 2017. *Defect Prediction based on Maximal Information Coefficient and Fast Correlation-Based Filter Feature Selection.* Second International Conference on the Internet, Cyber Security and Information Systems (ICICIS 2017). Johannesburg, South Africa. ISBN: 978-0-86970-802-6, pp. 139-145.
2. Mpofu, B & Mnkandla, E. 2016. *Software Defect Prediction on a Search Engine Software using Process Metrics*. IEEE International Conference on Advances in Computing, Communication & Engineering (IEEE ICACCE 2016). Durban, South Africa. ISBN: 987-1-5090-2576-6, pp 254-260.

ACKNOWLEDGEMENTS

I would like to acknowledge the assistance of my supervisor Professor E. Mnkandla for his support and critical assistance towards the drafting and editing of the thesis. I owe my gratitude to Nonhlanhla Ngcobo for availing resources for some areas of this research. Charity Makakaba deserves a tremendous thank you for keeping herself occupied until the odd hours, so as to sit with me while I wrote the chapters of my thesis. I thank my sisters for providing assistance in times of need, Edson and Lindiwe Malinga for their unlimited support.

DEDICATION

I dedicate this project to my children Sindiso (aka Londiwe) and Nkosikhona.

DECLARATION

Student Number: 49131699

Name: Bongeka Mpofu

I declare that **Software Defect Prediction Using Maximal Information Coefficient and Fast Correlation-Based Filter Feature Selection,** is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

_____ 31/01/2018_____

SIGNATURE DATE

Bongeka Mpofu

ABSTRACT

Software quality ensures that applications that are developed are failure free. Some modern systems are intricate, due to the complexity of their information processes. Software fault prediction is an important quality assurance activity, since it is a mechanism that correctly predicts the defect proneness of modules and classifies modules that saves resources, time and developers' efforts. In this study, a model that selects relevant features that can be used in defect prediction was proposed. The literature was reviewed and it revealed that process metrics are better predictors of defects in version systems and are based on historic source code over time. These metrics are extracted from the source-code module and include, for example, the number of additions and deletions from the source code, the number of distinct committers and the number of modified lines. In this research, defect prediction was conducted using open source software (OSS) of software product line(s) (SPL), hence process metrics were chosen. Data sets that are used in defect prediction may contain non-significant and redundant attributes that may affect the accuracy of machine-learning algorithms. In order to improve the prediction accuracy of classification models, features that are significant in the defect prediction process are utilised. In machine learning, feature selection techniques are applied in the identification of the relevant data. Feature selection is a pre-processing step that helps to reduce the dimensionality of data in machine learning. Feature selection techniques include information theoretic methods that are based on the entropy concept. This study experimented the efficiency of the feature selection techniques. It was realised that software defect prediction using significant attributes improves the prediction accuracy. A novel MICFastCR model, which is based on the Maximal Information Coefficient (MIC) was developed to select significant attributes and Fast Correlation Based Filter (FCBF) to eliminate redundant attributes. Machine learning algorithms were then run to predict software defects. The MICFastCR achieved the highest prediction accuracy as reported by various performance measures.

***Key Terms**:* defect prediction; feature selection; software metrics; relevant metrics; redundancy; machine learning algorithms; filter; wrapper; embedded; information theory

CHAPTER 1

INTRODUCTION

1.1 Background

Software-based systems are a fundamental element of modern life and include innovative applications used in human activity, from financial, social, engineering to safety critical systems (Rahman & Devanbu 2013: 432-441;Ricky, Purnomo & Yulianto 2016:307-313). Many companies rely on software systems to support their day-to-day operations and deliver products or services to customers. Today's software applications are complex and organisations are faced with growing competitive pressure to deliver high quality solutions, short development and deployment schedules using limited resources. Companies that develop applications (e.g. Windows, Google Apps, Internet Explorer Firefox) have changed their development processes to rapid releases (Mäntylä, Adams, Khomh, Engström & Petersen 2015:1384-1425). The organisations have limited the development and subsequent release times for a major release to weeks, days, or sometimes in hours to quicker deliver the latest features to customers (Mäntylä et al. 2015:1384-1425; HP 2011:1-8). As a result, some testing teams now focus on modules that are error prone to save time.

Software engineering originally focused on achieving system functional requirements. Due to the increase in business and industrial considerations, this gradually included quality as well (Hneif & Lee 2011: 72). Quality software, (discussed in Section 1.3) fulfils its requirements efficiently and effectively, while providing customer satisfaction (Duarte 2014:31; Kapur & Shrivastava 2015:1). It is important to conduct intensive software testing to achieve quality. The interests of software engineering in quality assurance are activities such as testing, verification and validation, fault tolerance and fault prediction (Abaei & Selamat 2013: 79-95).

Too often in the world of software development, quality is not considered until the programming is almost completed. This approach is inadequate, due to short delivery cycles. Consequently, the place of software testing has begun to change. It is recommended that solution testing starts as soon as

the program commences, in parallel with solution development. Errors must be located and eradicated early at the preliminary phases of software development.

The majority of the tools and resources of software development and maintenance are linked to software testing (Taipale, Kasurinen, Karhu & Smolander 2011:120; Misirli, Bener & Turhan 2011:532). According to Capgemini Group (2017: 17), software testing costs have increased in the recent years, despite the fact that software testing is a maturing discipline. The distribution of the test budget (see Figure 1.1), shows that hardware and infrastructure costs remain the biggest area in budget allocation. The costs increased from 37% in 2015 to 40% in 2016, due to challenges faced by numerous companies in mastering their test environments.

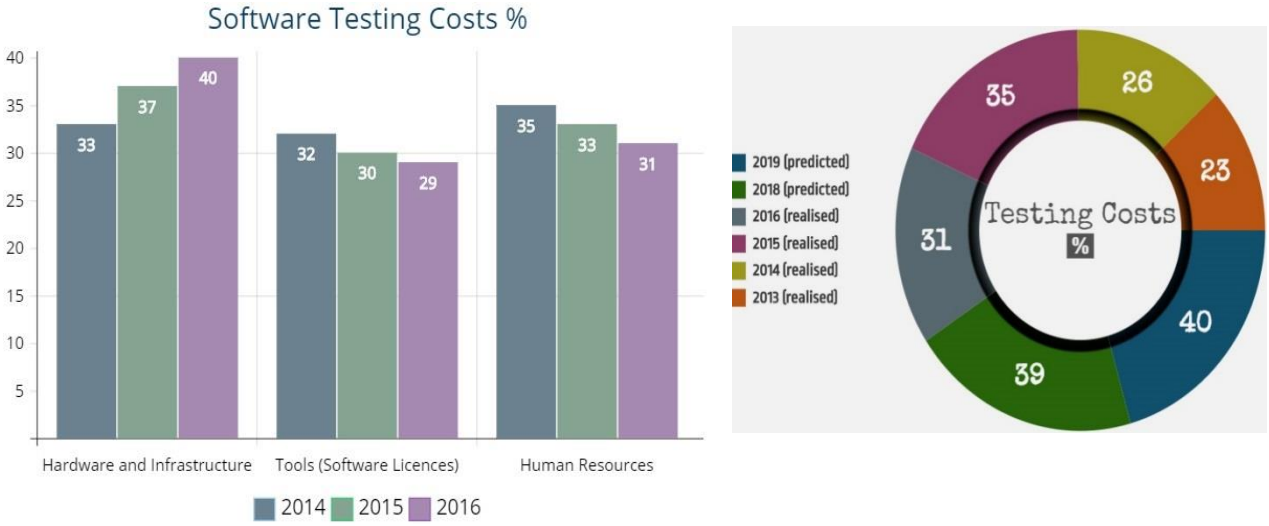


Figure 1.1: Allocation of the testing budget (Capgemini Group 2017:53)

Spending on human resources dropped from 33% in 2015 to 31% in 2016, despite the need for new skill sets. This was achieved with increased automation, by leveraging offshore resources, use of flexible service contracts and greater adoption of open source tools. The research data also shows an increase of 3% (to 33%) in 2016 for tools and software testing costs. OSS has become better and

offers more complete solutions, since it is now being accepted as part of development and testing. Software testing costs are estimated to increase to 39% in 2018 and to 40% in 2019. The year-on-year growth of Quality Assurance and Testing budgets indicates that testing is not as efficient as it should be.

Research has also focused on software defect prediction (Bell, Ostrand & Weyuker 2013:479; Caglayan, Tosun, Bener & Miranskyy 2015:206). Defect prediction is a proactive approach that predicts the fault-proneness of modules and allows software developers to assign limited resources to the defect-prone modules, so that reliable applications can be developed on time and within budget (Zhang & Shang 2011: 138; Wang, Shen & Chen, 2012:13). Software defect prediction is essential with the emergence of rapid release software that is aimed at quick functionality (Li, Zhang, Wu & Zhou 2012:203). Understanding the current and previous benefits and limitations of the applications could be used to predict, if the tool will be effective to detect new types of errors in the next software version. Samples of the current version could also be used to create a model to be applied in predicting the effectiveness of the next version. Daniel and Boshernitsan (2008:364) affirm that test tool effectiveness could also be measured in terms of test coverage, using metrics extracted from the program structure.

This research predicts the future reliability of Equinox, Mylyn, Plug-in Development Environment (PDE), Lucene and Eclipse Java Development Tools (JDT) product releases. The main challenge is to predict and eliminate defects of the software, thereby enabling short release cycles of the applications to be developed. Software defect prediction methods can depend on the source code and defect data of current and previous applications.

1.2 Software defects

Software development teams conduct software testing to ensure that the quality of applications meets the users' expectations. Software defect prediction is one of the testing activities, hence there has been ongoing research to identify and eliminate software defects. As defined by IEEE - SA Standards Board (2010:1-15), a defect is an error, failure or fault in any application that has been

created, which does not meet specifications and needs to be repaired. The common terms used in the software context, as defined in the Classification of Software Anomalies are (IEEE - SA Standards Board 2010:5);

Defect: *An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.*

Error: *A human action that produces an incorrect result.*

Failure*: (A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. (B) An event in which a system or system component does not perform a required function within specified limits.*

Steps must be taken to prevent system failure. Quality software must be free of defects and errors when delivered to the customer.

1.3 Software quality management

The aim of software quality management is to create good software and enhance the effectiveness of software testing in order to improve software quality and reliability, thereby providing a product that satisfies the user within budget and scheduled time (Gill 2005:14). In the opinion of Jin and Zeng (2011: 639), quality is an abstract measurement and can reveal the grade of a product or service and its level is related to the satisfaction of customers. Software quality encompasses application features that include business concerns such as application correctness and accuracy, functionality and integrity, legacy policies, time to market and robustness to non-functional factors, which include security, portability, usability, flexibility and maintainability.

Software Quality Assurance (SQA) consists of processes and methods that ensure conformance to explicitly and implicitly defined organisational requirements. These pre-specified standards are vital in the development of high assurance systems. SQA utilises resources and includes tasks such as manual code inspections, metrics and measurement procedures, review meetings, intensive software testing (to improve software quality), reporting and quality control mechanisms. Modules that are likely to contain defects are inspected and fixed, thereby reducing the cost of locating faults at a later

stage. Locating and correcting defects is costly and time consuming. Defect prediction models guarantee better efficiency by prioritising quality assurance activities. Since, the distribution of defects is usually skewed, (i.e., the distribution of defects in modules is not uniform), such models can locate the most defective bits of code and enable developers to eliminate the defects without spending too much resources and time in the quality assurance activities.
Defects are caused by errors in logic or coding, which result in failure or unpredicted results; hence they have an unfavourable effect on software quality. Defects may cause the delay of a product release, loss of reputation and increased development costs.

According to Harter, Kemerer and Slaughter (2012:810-827), higher standards of quality control greatly minimise the possibility of serious defects. This is beneficial when the requirements are clear, complete and unambiguous. Incessant verification, validation and testing must be a goal in application development. Defects found in the earlier stages of software development can be corrected with minimal expense.

Problem: *(A) Difficulty or uncertainty experienced by one or more persons, resulting from an unsatisfactory encounter with a system in use. (B) A negative situation to overcome.*

1.4 Software testing

Software testing is the evaluation of a system or its element(s) with the aim of inspecting whether it fulfils the specified requirements or not. The purpose of software testing is to locate defects and analyse the software quality (Lee 2007:191-216).

Software testing is the most common SQA activity and is an activity that must be conducted throughout the software development life cycle (Lee 2007:195). Software testing techniques are influenced by design models, development process, programming languages and other software development technologies. Therefore, test methods are not applicable to all the software and test requirements. Designed test cases can be re-used to increase efficiency and reduce time for writing test methods. A test case is a set of constraints or variables, which indicate if an application meets

the requirements or not. The basic types of testing are the execution and non-execution based testing.

1.4.1 Execution-based testing

The method is also called dynamic testing. Test cases which are prepared beforehand are used in program testing. The application is considered faulty if the output is wrong. The application is tested for its utility, correctness, reliability and performance. The absence of errors in the output does not imply the software is fault free, the software may be running correctly on that particular test data.

1.4.2 Non execution-based testing

This is also called static testing. Software is tested without running test cases. Review methods such as walkthroughs and inspections are used to discuss and evaluate a software. A group of knowledgeable people, other than the author, with a wide range of skills, discuss the software as a group. Walkthroughs have fewer actions and are less formal than inspections. Inspections include planning, measurement and control in managing processes.

Walkthroughs

A walkthrough team may consist of the team responsible for the current development, manager, the next project development team, clients and the SQA representative. The software is checked for later correction. Walkthroughs locate errors in the software specification, design, plan and source code.

Code inspections

The reason for conducting code inspections is to locate defects and spot any process improvements, if any.

This process may include metrics that can be used to correct defects for the document under review and aid improvements in coding standards. Attendees may benefit from the cross pollination of ideas during inspection. Preparations before the meeting are important, since they include reading of source documents to ensure readiness and uniformity.

1.4.3 Testing in a box

Software testers may have internal or external access to a system.

1.4.3.1 Functional testing

This technique is also known as black box testing. The tester does not have access to internal structures and checks for errors, using the functionality of the system. Test cases are designed to test if the system is according to the customer's specifications and requirements. Black box testing is usually done for validation.

1.4.3.2 Structural or white box testing

In white box testing, testers have access to the internal structures of a program and are capable of detecting and fixing all logical errors, using designed test cases. Programming proficiency is required to identify all logic paths through the application. Not all paths may be tested mainly if there are several loop statements in the application, therefore, instead of absolute paths, logic paths are considered. White box testing is commonly conducted during the unit testing stage and it is typically applied for verification.

1.4.3.3 Gray box testing

Gray box testing is when the software program or device's internal workings are partly understood. The testing can be described in two ways;

(i) Gray box testing is the integration of black box and white box testing

(ii) Gray box testing method tests software with partial knowledge of its underlying source code or logic. Testers have more knowledge of the code, but do not focus on exploiting the code.

1.4.4 Performance testing

Software performance testing is conducted to identify performance limitations of the system and make adjustments if necessary. The execution speed of some components is calculated. Some of the types of performance testing are load and stress testing.

1.4.5 Other types of testing

(i) Load testing. The handling of numerous simultaneous requests to the system is tested. This is to test if the system can manage a sudden increase in traffic. Volume testing is conducted on web systems to check if there is performance degradation when a system is processing numerous requests. System components (e.g. the central processing unit and graphics card) are expected not to crash during load testing.

(ii) Stress testing. The system is deliberately made to process heavy chores to the point of complete failure to test its stability. A stress test may test the simultaneous management of users and the further test the overload of resources on the system. This may result in possibility of system failure and the system is also tested if it can recover from that failure. The stress testing process must test for potential security loopholes, data corruption issues and slowness at peak user periods.

(iii) Reliability testing. This is the probability of an application functioning and not having any failure over certain duration in a specific environment.

(iv) Compatibility testing. This is tests if the system is compatible with other objects in the environment without any discrepancies. Objects include peripherals, operating systems, database and other system software. The purpose is to test if the system performs in the environment.

(v) Regression testing. Re-testing is conducted to ensure that software modules that were not part of the modification process have not been affected because of the new changes.

1.4.6 Levels of testing

Software applications are normally designed using top-down or bottom-up hierarchy strategies. The smallest part of program design is a module. Units of source code, methods, procedures, or functions are tested separately.

The functionality and performance of the whole software system greatly rely on the characteristics of each unit. **Unit testing** locates code level faults in the methods and classes of separate components. Hence, units must be tested first.

Integration testing, associated system modules are integrated and tested as a group or subsystem to ascertain functionality of the system. This test exposes errors that result from component integration. It is crucial to locate and fix faults at each level to minimise testing costs.

System testing is the last test; it is conducted to determine if the application's operation is per specified requirements. Unit testing emphasises on structural testing, whereas system testing is based on functional testing.

Acceptance testing is when the tester and stakeholders test the system to determine if the system meets user requirements. User requirements may change during the software development. This test is conducted to check if the system is ready for release.

Critical systems must be incessantly operational. These systems must promptly and automatically recover from failures, should they happen.

1.5 Software fault tolerance

System failure despite faults prevailing in the software, can be prevented using fault tolerance techniques (Kienzle 2003: 45-67). A **crash failure** arises when the system entirely ceases to function. A **fail–silent** and **fail–stop** behaviour is when a unit stops functioning and produces a failure. **Omission failures** transpire when the system does not respond to a request when it is anticipated to do so.

Timing failures can occur in real–time systems if the system fails to respond within the specified time slice. Both early and late responses are regarded as timing failures; late timing failures are occasionally known as performance failures.

1.5.1 Redundancy

The main supporting concept for fault tolerance is redundancy. In software development, redundancy can be of different types: **functional redundancy**, **data redundancy** and **temporal redundancy**. The purpose of **functional redundancy** is to tolerate design errors. Unlike hardware fault tolerance, software design and implementation faults cannot be identified by merely duplicating identical

software units, since the same fault will exist and manifest itself in all copies. The plan is to bring variance into the software imitations, producing dissimilar versions, variants or alternates. These versions functionally match, (i.e. use the same specification), but internally use dissimilar designs, algorithms and implementation methods. **Information or data redundancy** comprises the use of added information that permits one to check for integrity of vital data, for instance error-detecting or error-correcting codes. Varied data, namely identical data characterised in dissimilar formats, also fall into this group. Lastly, **temporal redundancy** includes the use of extra time to bring about fault tolerance. Temporal redundancy is an effective method of permitting transient errors. If the temporary conditions causing the fault are excluded later, simple re-execution of the failed operation will be successful. Overall, most software fault tolerance methods add execution overhead to an application and therefore use additional time in contrast to a fault-tolerant application.

1.5.2 Error processing

Forward error recovery necessitates a more or less accurate damage assessment. The error must be diagnosed so as to repair it in a logical way. This diagnosis for forward error recovery relies on the specific system. Exemptions are provided in programming languages to indicate and recognise the type of a fault. **Forward error** recovery can be accomplished through exception handling.

Backward error recovery demands that a prior accurate state occurs: such systems occasionally stock a copy of a clear state (sometimes called recovery point, check point, save point or recovery line, based on the recovery method), which can be rolled back in the event of an error.

Backward error recovery is a general method: since it re–installs a preceding accurate system state, it does not rely on the nature of the fault nor on the application’s semantics. Its main disadvantage is that it experiences an overhead, even in fault–free executions, because recovery points have to be established occasionally.

An increasing number of companies are working on software projects. Developers may sometimes work on a common set of files over a period of time. Changes made to these files must be monitored and controlled.

1.6 Software product line and versioning

A Software Product Line (SPL) reduces the development costs of software systems that are members of a product family. Identical product features between products are captured. Program developers of SPLs focus on certain product issues, instead of aspects that are common to all products (Botterweck & Pleuss 2014:266).

Software versioning is the development of software with the same name and with some features or functions introduced. Codes are allocated to successive and unique states of computer software. These are determined by the basis of the apparent significance of modifications between versions without any clear criterion. A new software version aids in outlining a threshold, which makes a change from a present state to a new state.
Version control is crucial in the software sector in managing software development and modifications, where developers frequently alter source files to implement technical specifications. New versions must be better than previous ones and not introduce new bugs.

Successive versions of software defect test tools that match the advancement of mobile devices and applications on the market have been developed. The high level of similarity and low degree of dissimilarities among versions of a software product enable us to learn about the product trends and predict files that are likely to contain errors in the product line, from information about modifications, bug fixes and failures in the previous software of the product line. In the current fast-paced business environment, most businesses are reducing the time between successive product releases.
Predicting the effectiveness of the software will enable developers to focus on the fault-prone code and allocate scarce resources to the problem areas, thereby improving test efficiency and reducing development cost and time.

1.7 Testing software product lines

Companies can develop SPLs using reduced resources and produce better quality than they can for single systems. This however requires the reusable objects' quality to be high. Quality assurance and

especially testing, which is still the most common quality assurance activity, is critical to product line efforts.

1.8 Problem statement

Previous research on software defect prediction has been conducted. It has focused on the testing and quality of applications using static and change metrics. Successive versions of the application have been developed. Since the versions of the application are released rapidly, the prediction of defects may assist in locating the defect-prone parts, which the developers may focus on, thereby saving financial resources and saving valuable time.

This research was conducted to develop a novel feature selection method to select relevant attributes for software defect prediction. The Mylyn, Equinox, Apache Lucene, Eclipse PDE and Eclipse JDT software applications were used in the experiments. Process metrics were applied in quantifying the defects. Software test effectiveness compares the number of software defects found to the quantity of test cases that have been executed.

1.9 Research questions

This study was conducted in a sequence of linear stages, each of which had its own research question. The most effective process metrics will be selected e code to predict defects. Experiments involving various feature selection methods and machine learning techniques were conducted. A novel feature selection method was used to choose the best set of attributes for the defect prediction process.

The main goal of the research is:

> *To develop a novel feature selection method to identify the most relevant attributes for defect prediction*

Feature selection has been used in software defect prediction to identify relevant and non-redundant attributes. It aims to counter balance these two factors. In this study, an optimal feature selection

criterion will be derived from an information theoretic method. An algorithm that will select suitable attributes will be designed.

The research questions derived from the main goal are as follows:

1.9.1 Primary research question

How can a novel feature selection method that will choose suitable attributes for predicting defects be designed?

The new feature selection method has been tested using various machine learning models and prediction results have been compared with those of other feature selection methods. The primary question is subdivided into secondary questions.

1.9.2 Secondary research questions

RQ1: Which metrics are suitable for predicting defects in the versions of a software product line?

This study predicts software defects in revolving software. A literature review was conducted on software defect prediction for software product lines. Previous studies suggest that process metrics are suitable for predicting post-release defects (Xu, Xuan, Liu & Cui 2016:370-381; Liu, Chen, Liu, Chen, Gu & Chen 2014. This study applies process metrics, feature selection and classification techniques for defect prediction. Bug process metrics for Mylyn, Equinox, Lucene, JDT and PDE applications were obtained.

RQ2*:* Which information theoretic methods have been used in previous research?

The techniques that are based on the entropy concept include the Information Gain, Mutual Information, Symmetric Uncertainty and the Maximal Information Coefficient (Chapter 4).

RQ3*:* How is the performance of the information theory-based methods compared to other algorithms?

Previous studies have used feature selection techniques for defect prediction. These include statistical, information theoretic, instance-based and probabilistic methods. The results from the tests

indicate that the information-based theories are more accurate. The Maximal Information Coefficient selected significant features that resulted in high performance of the algorithms (Chapters 2, 4 and 5).

RQ4*:* Are the data mining techniques consistently effective in predicting defects?

In this research, the Naïve Bayes, PART and J48 algorithms are applied in the prediction process. Algorithms are assessed and performance measures evaluated (Chapter 6).

RQ5*:* How can a data redundancy removal technique be derived from the concept of predominant correlation?

Non-redundant attributes are selected from the list of relevant ones using the predominant correlation (Section 3.3)

RQ6: How can a model that will predict defects in the next versions of the software applications be derived?

An optimal information theoretic feature selection measure that selects relevant attributes and maximises prediction accuracy is derived (Chapter 3).

1.10 Research objectives

The main goal of the research is to develop a novel feature selection method to be used in predicting defects in rapidly evolving software.

The objectives of the study are to:

- Identify the process metrics that can be used to quantify the software defects.
- Develop a unique feature selection method to select features to be used to predict the effectiveness of the next version of the applications.
- Study the performance of statistic and information theory based feature selection algorithms in previous research
- Compare the performance of information theoretic feature selection algorithms with other algorithms in previous studies

- Evaluate if the developed novel Maximal Information Coefficient based algorithm and existing Linear Correlation, Information Gain and ReliefF feature selection algorithms can be used by machine learning algorithms to predict defects in rapidly evolving software in this study.
- Conduct internal and external validity tests to check if the machine learning methods are consistently effective in predicting defects

1.11 Research methodology

There are various types of research.

1.11.1 Research types

The basic types of research are:

Descriptive vs analytical

Descriptive research employs fact finding and surveys. Its goal is to acquire the description of the current situation. The researcher has no control over the variables and can only narrate and explain what transpired. In contrast, in analytical research, the researcher uses facts or information on hand to analyse and to make analytical assessment of the material.

Applied vs fundamental

The goal of applied research is to find a resolution to a high priority issue affecting a community or business organisation, while fundamental research is pertained to generalisations and with the inventions of a theory. Pure or basic research is gathering knowledge for knowledge's sake. Studies that make generalisations of human behaviour are examples of fundamental research.

Research may also be conducted to determine social, political or economic trends affecting society or a business organisation. These are examples of applied research and intend to resolve issues at hand. Basic research focuses on obtaining information to be added to the current organised body of scientific knowledge.

Qualitative vs quantitative

Quantitative research focuses on measurement of data and it is suitable to observations that can be illustrated in terms of quantity. Conversely, qualitative research is exploratory and uses non-numerical data. The outcome of qualitative research is descriptive instead of predictive.

Conceptual vs empirical

Conceptual research is focused on some abstract ideas or theories. It is normally employed by theorists to create innovative concepts or to explain current ones and it (conceptual research) is also commonly used in social sciences. Conversely, empirical research depends on experience or experimentation and usually does not take system and theory into consideration. Data is utilised to attain conclusions which can be verified using observations or experiments. It is an experimental type of research and is commonly used by scientists.

The design of novel methods that provide effective problem solutions has been recognised by the research community as a methodology.

1.11.2 Design Science

Design science research (DSR) approach aims to create and evaluate artefacts (Adikari, McDonald & Campbell, 2009; Peffers, Tuunanen, Rothenberger & Chatterjee, 2007). An artefact refers to an object that has, or can be transformed into, a material existence as an artificially made object (e.g., model, instantiation) or process (e.g., method, software) (Gregor & Hevner 2013). Concepts that outline a contribution in a Ph.D. thesis, which are also applicable to research articles were defined by Davis (2005). One of the contributions is to consider if the thesis

> *develops and demonstrates new or improved design of a conceptual or physical artefact. This is often termed "design science." The contribution may be demonstrated by reasoning, proof of concept, proof of value added, or proof of acceptance and use*

An effective DSR should provide clear contributions to the real-world environment from which the research problem or opportunity is drawn (Gregor & Hevner 2013). The DSR generally consists of the following steps:

i. identify problem;

ii. define solution objectives;
iii. design and development;
iv. demonstration;
v. evaluation
vi. communication

The design and development stage may involve techniques that will lead to the production of an artefact.

1.11.3 Techniques used in software defect prediction

Defect prediction techniques aim at identifying error-prone parts of a module of a software application as early as possible. These techniques vary in the types of data they require. Some of the techniques are discussed below.

1.11.3.1 LOC, Halstead, McCabe complexity metrics

The Lines of Code (LOC) method uses the code from historical software applications to predict defects. The LOC, object oriented metrics or the combination of the metrics are derived from the structure of the code. The standard equation for the LOC is expressed as (Erfanian & Darav 2012: 69-78);

$$Defect\ (D) = 4.86 + 0.018\ Lines\ of\ Code(L) \quad (1.1)$$

The Halstead metrics and McCabe complexity metrics are common attributes of source code complexity (Ogasawara, Yamada & Kojo 1996: 179-188). McCabe's metric reflects the software application's control structure and measures the decision statements in the application's code. Halstead's metric evaluates the new addition of operators and operands into an application, which may be caused by the growth of program length. This raises the value of the Halstead's effort measure.

1.11.3.2 Software reliability growth model

The Reliability Growth Model predicts defects using the data from the current software application. They use statistical models to predict the reliability of the application and are usually applied during the final testing phase. Failure is modelled using a Non-homogeneous Poisson Process. These are the cumulative failures which are likely to arise after the program has run for time (*t)*. The mean value function *m (t)* is defined as (Ullah 2015:62);

$$P\{N(t) = n\} = \frac{m(t)^n}{n!} e^{-m(t)} \tag{1.2}$$

where *N (t)* represents a counting procedure over time *t.*

1.11.3.3 Machine learning techniques

Machine learning techniques are a division of artificial intelligence regarding computer programs learning from data (Alshayeb, Eisa & Ahmed 2014:7865-7876), Figure 1.2.

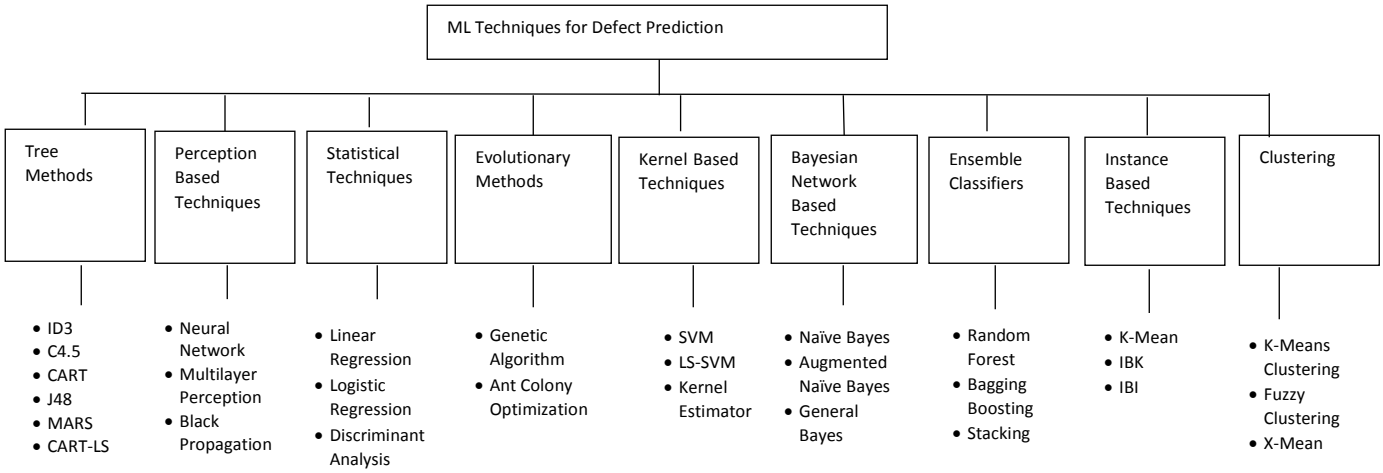


Figure 1.2 Techniques for Defect Prediction (Rathore & Kumar 2016: 5)

Machine learning algorithms include Decision Trees, Bayesian Networks, Probabilistic Classifiers and Evolutionary Based Classifiers.

1.11.3.4 Transfer learning/cross project

Cross-project defect prediction models are used if there is inadequate or unavailable historical source data (He, Peters, Menzies & Yang 2013: 45-54). Researchers filter, reduce differences and cluster data from diverse projects. The data is then trained using algorithms for each cluster separately.

1.11.3.5 Capture/recapture analysis

These models rely on expert inspectors to identify and quantify defects in software releases. Duplicates are identified by comparing the newly-located defects to defects in the preceding files. The equation below calculates expected defects as (Zubrow & Clark 2001:1:7);

$$Defects\ Expected = \frac{n(expert1) * n(expert2)}{m(no\ of\ faults\ found\ by\ both\ experts} \quad (1.3)$$

1.11.3.6 Topic based

This is a bug prediction approach that is involved on technical issues of a system. Algorithms use these as input for software fault prediction. This is founded on the belief that names of methods, classes, comments or embedded documentation expose the uneasiness they implement (Nguyen, Nguyen & Phuong 2011: 932-935). Examples of functions with issues are *Connector.Abort* and *faultCode=PARSER_ERROR.*

1.11.3.7 Test coverage

Test coverage is used to predict defects using structural testing strategy. It relies on the assumption that a correlation between code coverage and software reliability exists. An adequate number of objects must be tested or covered using test cases. Test cases are used as input in testing software

applications. The objects tested may be statements, branches, decisions, functions or loops of the application. The derived metric is called Test Effectiveness Ratio (TER). The condition of test data competence *C* is a function *C*:*PX S X T → {true, false}. C (p,s,t)*=true implies *t* is suitable for testing program *p* against specifications as specified by criterion *C*, else *t* is unsuitable.

1.11.3.8 Expert opinion

The defects are quantified by various field experts. They provide informed judgement on the least, most, best and worst occurrences of the most likely defects. Experts may use the rule-based approaches to predict defects. These human-based opinions may also be captured and reused in upcoming projects by recognising the effect of expected influential features in the specific context, without creating big data sources (Erturk & Sezer 2015:757:766).

1.11.3.9 Exception handling

Software applications regularly use exception handling to respond to unforeseen exceptions during program execution. Complicated exception handling may be the source of defects. This technique applies complicated exception handling as a defect prediction factor. It requires exception-based software metrics related to exception handling. These include the number of exception handlers *(nHandler),* the number of thrown exception types *(nThrown)* and the number of exception handling subgraphs a class belongs to *(nSubgraph)* (Sawadpong & Allen 2016: 55-62).

1.11.4 Data analysis

Features were ranked using the feature selection algorithms. Scores assigned to each feature were compared and the most important features were selected. The algorithms predicted defects using the selected relevant features. Performance evaluation measures were applied to analyse the quality of the prediction models used in the experiments.

1.12 Limitations of the study

The study had the following limitations:

- Only defects from OSS applications were used in this study.
- Only process metrics and machine-learning techniques were utilised for defect prediction.

1.13 Thesis outline

Chapter 1 – Introduction

The chapter included the objectives of the study, the common terms used for software anomalies, software defect prediction techniques and software versioning. Discussion of the research questions, research design and data analysis were conducted. The significance of the study is covered.

Chapter 2 – Theoretical background

The chapter focuses on studies that have been conducted on software defect prediction. Machine-learning algorithms that have been designed to improve defect prediction are discussed.

Chapter 3 – Research methodology

Different approaches used for research are discussed. The research instrument for this study has been presented.

Chapter 4 – Information theory

Measures that are based on the information theoretical concept of entropy are discussed. These methods are used in feature weighting, ranking and selection.

Chapter 5 – Feature selection

This chapter discusses feature relevancy and redundancy. Types of feature weighting techniques are covered. Attribute weighting methods are compared. Feature selection processes and methods are analysed.

Chapter 6 –Prediction model evaluation

The analysis of data recorded during software defect testing has been undertaken. The metrics and outputs from the defect models are analysed. The chapter presents the model developed to predict the effectiveness of the software versions. The chapter presents model validation.

Chapter 7 – Conclusion and future work

An outline of the effectiveness of different versions of the software applications is included. The final recommendation for the predicting the effectiveness of mobile devices testing tools is presented. The significance of the study is discussed and suggestions for future research are discussed.

1.14 Chapter summary

Software Quality Management (SQA) in general is the management of activities that ensure the delivery of high quality software. SQA activities include software testing. Software defect prediction is one of the most supporting activities of the testing phase of System Development Life Cycle. The next chapter presents a survey of related work.

CHAPTER 2

BACKGROUND

2.1 Introduction

This chapter discusses the literature review and some of the topics that are fundamental to software defect prediction. Software defect prediction has received substantial attention in the software industry. Software defect data sources, metrics and types of machine learning are explored. Previous research has been conducted to analyse the effect that metrics has on fault proneness. Some of the defect prediction papers that have been published since 2007 have been reviewed. The defect data that has been used in previous research emanates from different sources (Madeyski & Jureczko 2015: 393-422; Muthukumaran, Choudhary & Murthy 2015: 15-20;Bowes, Hall, Harman, Jia, Sarro, Wu 2016:330-341;Fukushima, Kamei, McIntosh, Yamashita & Ubayashi 2014: 172-181), which include open source and industrial projects.

2.2 Data sources

In general, most of the data used in software defect prediction is obtained from the freely available open source repositories, which include the source code management systems and bug tracking systems. Other data is sourced from industrial projects.

2.2.1 Company/Industrial data

Software defect data is sourced from company software development operations. The difference between the development processes of industrial and OSS may affect the defect prediction results (Madeyski & Jureczko 2015: 393-422). In company environments, formal, centralised methods are applied in software development. These processes include formal software verification techniques. Co-located, well-structured teams develop data. Responsibilities may be divided between members of a team. Functional teams are generally used in software development organisations. Developers with similar skills are grouped together. One team in a company may design the interface; another may be focused on database design, while the other team may do implementation and testing.

Product teams working on industrial projects are organised, unlike the open source ones. In a study conducted by (Madeyski & Jureczko 2015: 393-422), at least one process metric in all versions of industrial software improved the prediction models, but this was not true for nearly half of the analysed versions of open source projects. This could be due to the organised manner in the development processes of industrial software.

2.2.2 Open source code repository

Task allocations and relationships between users and developers in open source development are less formal. Development processes are more decentralised in open source environments (Madeyski & Jureczko 2015: 393-422). Open source projects are developed as global collaborations of skilled developers. The developers apply different skills than in the industrial projects. The authors who commit software modifications in open source development are active in development, while their counterparts are less active. Therefore, less training, support and technical skills are required to develop OSS.

2.2.3 Bug life cycle participants

According to Ullah & Khan (2011:98-108) there are many contributors in the bug life cycle. They have responsibilities and roles, some of which are as follows:

- Bug reporter

  This is the participant who reveals the bug and creates a report for it, by entering the bug data in the bug tracking tool. The reporter inputs the bug details that include the title, bug priority, severity, dependencies and the component where the bug is located.

- Bug group

  This consists of people who regularly receive updates concerning the bug in a bug report. They include the bug reporter, the developer, tester and the quality assurance manager.

- Bug owner

  The bug owner ensures that information about the bug in the bug tracking system is adequate. The owner manages bugs and guarantees that, for example, high priority bugs in the system are fixed within the shortest possible time.

2.2.4 Bug tracking system

This is a software system that tracks the progress of a bug. A reported bug is analysed, allocated to a developer, fixed and resolved (Babar, Vierimaa &Oivo 2010: 1-407). The bug tracking system records the characteristics of the bugs, such as, defect reported date, the section in which the bug was located, commit date and other properties concerning the bug (Shihab, Ihara, Kamei, Ibrahim, Ohira, Adams, Hassan & Matsumoto 2013: 1005-1042), (see Table 2.1).

Table 2.1 Software bug attributes

Attribute	**Description**
Bug ID	A distinct identification number of a bug
Severity	It indicates the impact of the bug, i.e. critical, trivial
Priority	This describes the importance of the bug when contrasted with other bugs. P1 regarded as the leading priority, while P5 is the last
Resolution	This specifies how the bug was corrected, such as, fixed
Status	This is the present condition of a bug, (e.g. new, resolved)
Comments	Users add comments to the bugs. These are the number of comments that have been added to the report
Create date	This is the reported date of the bug

Dependencies	These are the bugs that depend on other bugs to be fixed for them to be also fixed
Summary	The summary of the problem is written in a single sentence
Date of close	This is the date a bug was closed
Keywords	These are the keywords that are used to tag and define the bugs
Version	This is the version of the software where the bug was located
Platform and OS	This defines the environment where the bug was found

The Bugzilla Life Cycle is displayed in Figure 2.1, according to the manner Bugzilla users check and modify the bug status in the database (Sunindyo, Moser, Winkler & Dhungana 2012: 84-102).This life cycle is regarded as the model for developing software projects, particularly for OSS projects. The stages in the cycle demonstrate the procedures followed by OSS developers when modifying bugs. The processes employed when modifying the bug status are regarded as engineering processes and the phases are similar to those in the software development life cycle.

Initially, a bug is presented by users or contributors as an unconfirmed bug. The existence of the bug is then verified and the bug state is changed to ‘new’.

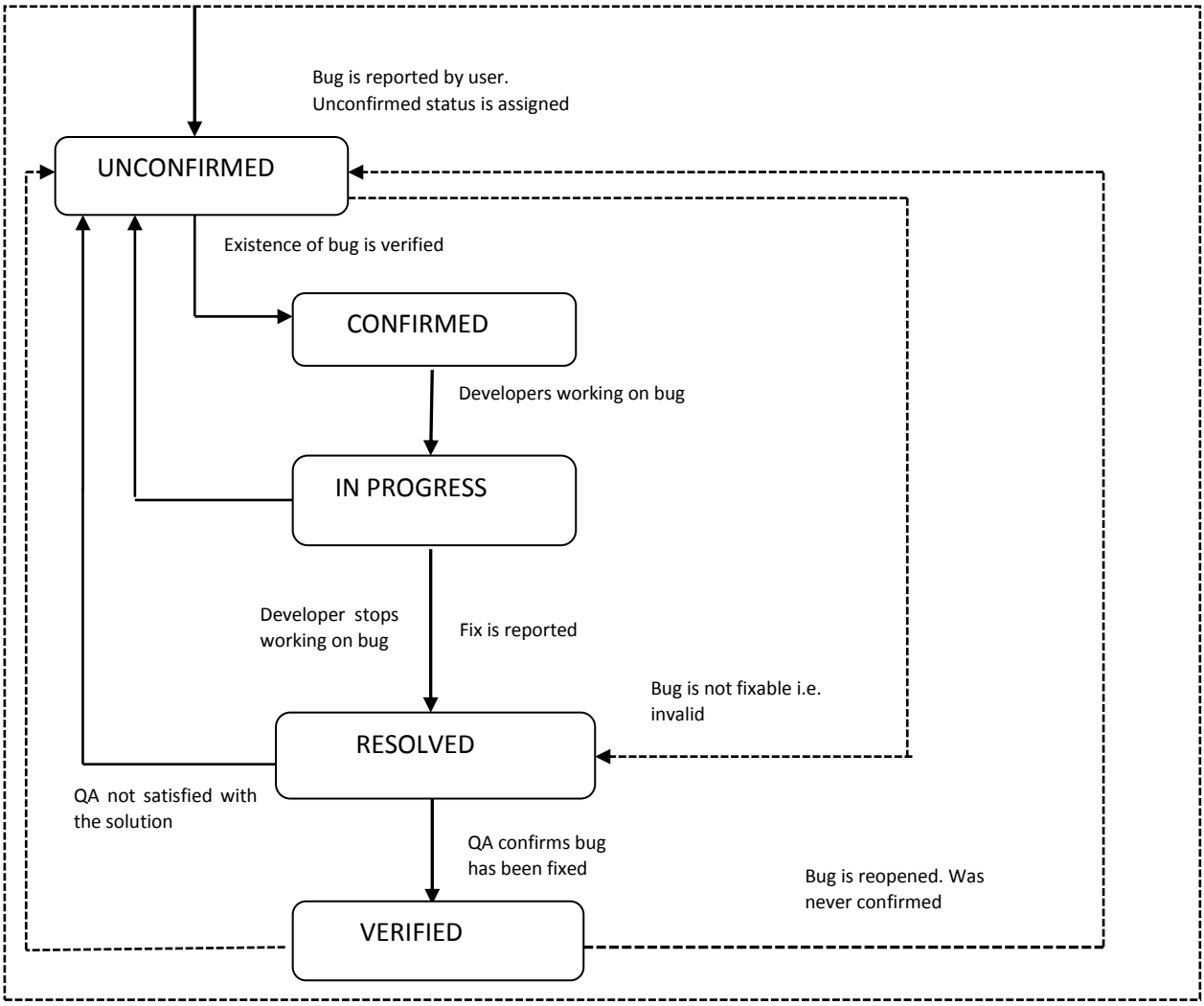


Figure 2.1 Bugzilla Life Cycle (Sunindyo, Moser, Winkler & Dhungana 2012: 89)

The new bug is allocated to other contributors or fixed instantly. Bugs that are verified as fixed are closed. Some bugs may be wrongly labelled, 'RESOLVED' and may need to be reopened. The Bugzilla bug states are meant to assist the contributors to specify bug status. Contributors may also devise their personal state names.

The defect data used in this research was extracted from Bugzilla and Jira repositories (Ambros, Lanza & Robbes 2010: 31-41). Bug fixes made to the Mylyn, JDT, Lucene, Equinox and PDE open source projects were saved and used to create the defect files. In general, some of the open source

systems are used in business to save time and costs. Contributors write modules and correct reported bugs. The contributors' updates are saved in defect files that are used in software defect prediction.

2.3 Defect prediction approaches

Software quality is an extensively researched area in the software engineering domain (Seliya, Khoshgoftaar & Van Hulse 2010: 26-34). Techniques such as unit testing, code inspections and defect prediction are applied to reduce defects in quality assurance activities (Seliya et al. 2010:26 ;Tan, Peng, Pan & Zhao 2011:244-248 ;Ahmed, Mahmood & Aslam 2014:65-69). Software developers may predict and remove defects in new versions of software (Kastro & Bener 2008: 543-562).

2.3.1 Single version software

One version of software is developed. There is an assumption that the present piece of code determines the existence of future defects. The single version approaches do not depend on the software's historical data, but examine more its present structure, using different metrics.

2.3.2 Versioning systems

Process metrics are derived from the versioning software. These approaches consider that the newly or regularly modified files are the most possible origin of imminent bugs. Hassan presented the entropy concept to evaluate the code modifications (Hassan 2009:78-88). The FreeBSD, NetBSD, OpenBSD, KDE, KOffice, and PostgreSQL applications were used to assess the entropy metrics. The results proved that the amount of preceding bugs is a better predictor than the number of previous file modifications.

A version control system (VCS) is a repository of files that supports the revision of software and the management of application changes. Revisions are a result of software modifications. VCSs facilitate

distributed and collaborated software development. Modifications to software are tracked and references are generated for commits that alter the application (Thongtanunam, Mcintosh, Hassan & Iida 2016: 1039-1050). Sites such as GitHub, SourceForge and Google Code support version control (Yu, Mishra & Mishra 2014:457:466). The services that are provided by the sites include archiving, online code browsing, bug trackers, version downloads and web hosting. Companies that do not have resources to manage their own servers utilise the version control services provided by the web hosting sites. Source Code Control System (SCCS) and Revision Control Systems were created in the 1970s and 1980s respectively. The SCCS and RCS software tools store file versions, while subsequent systems also permitted for remote and mainly centralised repository of the file releases (Cochez, Isomottonen, Tirronen & Itkonen 2013:210).

A multi-sited version control system is a distributed VCS that is administered at different locations to align the development work of numerous people that team up to build a single piece of software. The CVS used to be the most popular open source version control system, but has been surpassed by GitHub and Subversion. Concurrent Versions System (CVS) and Subversion (SVN) are common centralised systems. In distributed version control systems (DVCS), individual users have local copies of the storage, which can be synchronised with other storages. Git and Mercurial use this kind of decentralised system.

The process metrics used in this study were five open source systems obtained by (Ambros, Lanza & Robbes 2010: 31-41),who created models to depict how the systems changed, since they were created by analysing the versioning system log files. The history of the systems was modelled, using the transactions extracted from the systems' SCM repositories. The systems were developed using different versioning systems (CVS and SVN) and different bug tracking systems, Bugzilla and Jira. Metrics, also known as software features or attributes, are used to predict defects.

2.4 Software metrics

Software metrics are indicators of defects and therefore essential in the efficient allocation of resources (Madeyski & Jureczko 2015: 393-422).Researchers have used software measures to predict defects and evaluate software. The categories of defect prediction metrics are static and process metrics.

Data in the data sets is in the form of data tables. In the defect data tables, rows represent data from a single file or module. These are also known as “functions”, “methods”, or “procedures”, depending on the application. Columns in the software defect data describe one of the defect features or attributes (Menzies, Milton, Turhan, Cukic, Jiang & Bener 2010: 375-407).Common metrics contained in datasets are LOC, Halstead, McCabe and Chidamber-Kemerer metrics suites, see Table 2.2.

Table 2.2 Software Metrics (Ghotra, Mcintosh & Hassan 2015: 789-800)

Metrics suite	**Metric**	**Description**	**Justification**
McCabe Software Metrics	cyclomatic complexity, cyclomatic density, design complexity essential complexity and pathological complexity	The number of branches in an application is quantified. This determines the complexity of a software element.	Complex software elements might be more susceptible to defects
Halstead attributes	content, difficulty, effort, length, level, prog time, volume, num operands, num operators, num	The complexity of a software component is approximated. The quantity of operands may determine the difficulty in the way a software component is read depending on the language used (e.g., number of operators and operands)	Software components which are complicated to learn may intensify the chances of improper maintenance, and as a result, increase the likelihood of defects
LOC Counts	LOC total, LOC blank, LOC comment, LOC code and comment, LOC executable and	This measure calculates number of lines on a file	Software elements with many lines of code may contain more defects.

	Number of lines		
Miscalleneous	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	Metrics which are not distinct	N/A
Chidamber Kemerer	wmc, dit, cbo, noc, lcom,	Evaluate a class complexity rfc, ic, cbm, amc, lcom within an object-oriented system design.	Classes which are complicated are likely to contain errors

Software metrics can be categorised as static or historical.

2.4.1 Static code metrics

These metrics quantify properties of code that involve the size and complexity of an application. Defect prediction can be conducted using data that represents static code metrics, whose class label is defective and has values that are true or false (Menzies et al. 2010: 375-407). These are product metrics that do not contain process or developer details. Static metrics, such as LOC, are acquired from a single snapshot of an application.

2.4.1.1 Lines of code

LOC were first used in the 1960s to measure programming productivity, effort and quality. LOC is a common defect prediction approach that associates defects to the application itself (Syer, Nagappan, Adams & Hassan 2015:176-197; Barb, Neill, Sangwan & Piovoso 2014:243-260; Caglayan, Tosun, Miranskyy, Bener & Ruffolo 2010; Alemerien & Magel 2014:1-9). The code metrics are extracted from software development records, there is no other project feature is measured in order to derive direct metrics and therefore LOC is regarded as a direct metric. Quality features that include complexity, effort and defect density are influenced by other measures.

These are indirect metrics and evaluating them directly is impossible. They must be obtained from the validated alongside other metrics (Barb, Neill, Sangwan & Piovoso 2014: 243-260).

Numerous concerns have been conveyed concerning the collection of LOC measures. These entail vague criteria for the count of lines such as physical versus logical LOC, how non-executable and comment lines are handled, including techniques for code reusability. The use of LOC as a productivity measure is subject to the Hawthorne effect that mentions the fact that other developers' behavioural aspects may influence their coding practices, which may determine the software development quality (Barb et al. 2014: 243-260).

2.4.1.2 Halstead metrics

In the 1977, the Halstead metrics were proposed (Menzies et al. 2010: 375-407) to measure program size and complexity. The Halstead metrics assess reading complexity, depending on the number of operators and operands in which a function that is hard to read is presumed to contain faults (Y. Yang, Zhou, Lu, Chen , Chen, Xu, Leung & Zhang 2015:331-357).

The Halstead complexity metrics measure:

a. Volume

   The quantity of program's content that must be read to be understood by the reader.

b. Difficulty

   How much mental effort must be exerted in developing program's code or understanding what it means (McIntosh, Adams & Hassan 2012: 578-608).

c. Effort

   How much mental effort would be needed to reconstruct an application?

Table 2.3 Halstead Metrics (Abaei & Selamat 2013: 79-95)

Metric	**Description**
N	Halstead total operators + operands
V	Halstead "volume"
L	Halstead "program length"
D	Halstead "difficulty"
I	Halstead "intelligence"
E	Halstead "effort"
B	Halstead "delivered bugs"
T	Halstead's time estimator
IOCode	Halstead's line count
IOComment	Halstead's count of lines of comments
IOBlank	Halstead's count of blank lines

Modules that are more complicated are usually defective. Halstead metrics are regarded as method level metrics and applied as independent variables. (Yu & Jiang 2016: 90-95; Catal & Diri 2008: 244-257).

2.4.1.3 McCabe Cyclomatic Complexity

The cyclomatic complexity metrics are method level metrics that were presented in 1976 by Thomas McCabe. Unlike the Halstead metrics, McCabe (1976: 308-320) maintained that the program control structure is more discerning than counting the symbols. Conditional statements in the code are measured (Ogasawara et al. 1996: 179-188). McCabe metrics deal with the programming effort (Paramshetti & Phalke 2014:1394-1397). Once the value of McCabe's metric surpasses a specific threshold for a certain module, (i.e. if a value is greater than 12), the modules must be subdivided to decrease their sizes.

The number of program control flows in a file is counted. McCabe cyclomatic number v (G) records the complexity of a piece of code. If a branch is encountered (if…, for…, while…., do, case, as well as the **&&** and **||** conditional logic operators), the v (G) is increased by one.

***McCabe* cyclomatic complexity measures**

cyclomatic_complexity

decision_density

cyclomatic_density

essential_complexity

pathological_complexity

Cyclimatic complexity is defined as;

$$CC = \bar{E} - \bar{V} + 2 \qquad (2.1)$$

2.4.1.4 Object-oriented product metrics

Although metrics already existed, the introduction of the Object Oriented (OO) approach in software development led to the development of new metrics. The Chidamber and Kemerer (CK) and Metrics for Object-Oriented Design (MOOD) are the class level metrics suite and were designed for object-oriented design (Chidamber & Kemerer 1991: 197-211). The OO metrics are used to predict defective classes (Kpodjedo, Ricca, Galinier, Guéhéneuc & Antoniol 2010: 141-175).

The CK metrics are discussed below (Singh & Verma 2012:323-327).

(i) Depth of Inheritance Tree (DIT)

DIT is also known as generalisation and is the distance from a class to the root class in the inheritance tree. Parent classes can have an influence on a class. Classes with more multiple inheritances have more complex behaviour. Conversely, a large DIT designates that a lot of methods can be reused.

(ii) Number of children (NOC)

The NOC counts the number of proximate sub-classes of a class structure. The number of children in a class indicates the degree of reusability of the class. Reusability increases as NOC grows. In contrast, as NOC increases, the testing effort will accumulate, since more sub-classes in a class signify many responsibilities and increase in testing duration.

(iii) Weighted Methods per Class (WMC)

The WMC is an object-oriented metric that calculates the complexity of a class. It is total complexities of all methods defined in a class. It indicates the amount of work needed for the development and maintenance of a specific class. The complexity of a class can be computed using the cyclomatic complexities of its methods. A class *C1* with methods *M1, . . . ,Mn* that are stated in the class. Let $k_{1,}k_{2,}$ ... $k_{n,}$ be complexities of the individual methods. WMC is described as

$$WMC = \sum_{i=1}^{k} c_i \qquad (2.2)$$

(iv) Coupling between Objects (CBO)

Coupling is when a method or instance declared in one class is directly linked to a method of another class. The CBO counts the distinct number of reference types that take place through method calls, instance variables, return types and thrown exceptions. This increases the fan out of the class to other objects (Aloysius & Arockiam 2012: 29-35). Arise of CBO values implies that the reusability of a class will decrease. Therefore, the CBO inter-coupling or interclass dependencies should be minimal.

$$CBO = \sum_{i=1}^{k} c_i \qquad (2.3)$$

(v) Response for a Class (RFC)

This is the count of unique methods that can possibly be executed as a result of a message being sent to an object of the class or by some methods in the class.

The MOOD metrics are:

Method Inheritance Factor (MIF)

It represents the proportion of inherited methods to the total number of available methods in all classes. The MIF is described by:

$$MIF = \frac{\sum M_i C_i}{\sum M_a(C_i)} \qquad (2.4)$$

Given that $i = 1$ to the total number of classes

Attribute Inheritance Factor (AIF)

It represents the proportion of inherited attributes to the sum of all available attributes in all classes. The AIF is defined by Chawla & Nath (2013:2903-2908) as:

$$AIF = \frac{\sum A_i(C_i)}{\sum A_a(C_i)} \quad (2.5)$$

Static code metrics are specific to a given version of software. On the other hand, process metrics are related to module changes throughout various versions of a system.

2.4.2 Process metrics

Historical metrics are based on previous information concerning the software and contain pre-release defects and code churn (Tse-Hsun, Thomas, Nagappan & Hassan 2012: 189-198). The metrics are extracted from the source-code data and contain, for example, the number of additions and deletions from the source code, distinct committers and the number of modified lines. Process attributes provide an insight into the competence of an existing development process.

A compilation of process metrics is conducted across all projects and over a long duration. These attributes are intended to modify the software development process(Singh & Sangwan 2014: 831-836).

Table 2.4 Process metrics (Bernstein, Ekanayake & Pinzger 2007: 1-8)

Metrics	Metrics definitions
Lines added until	Total number of lines of code added to a file
Age with respect to	Age of a file counting backwards from a specific release
Avg lines removed until	Average lines fo code removed removed from the file
Code churn until	Total number of lines of code added, deleted and modified
Number of authors until	Number of authors who modified the file
Major bugs	Total number of major bugs located in the file

Non-trivial bugs	Number of significant bugs
Number of versions	Number of released versions of a file
Software metrics Nnumber of fixes	Number of times a file was involved in bug-fixing
Number of refactorin	Number of times a file has been refactored

Software metrics are inputs in machine learning algorithms for the defect prediction process. The classification models are trained to predict change-prone classes. Some metrics are significant to the class, while others are not. Insignificant metrics may be removed before the prediction process begins. Software metrics are used to train machine learning algorithms in defect prediction.

2.5 Machine learning

Machine-learning techniques are labelled as supervised or unsupervised learning (Aleem, Capretz & Ahmed 2015: 11-23).

2.5.1 Supervised learning

Supervised learning is also known as classification or inductive learning. Models are trained using data from past experiences. The training data includes the class labels of the class attribute. Larger training sets produce better prediction accuracy, while small training sets reduce the prediction accuracy. Thus, the limitation of a supervised learning is that a lot of training data is required (Lu, Cukic & Culp 2014: 416-425).

Another disadvantage is that the learned examples might encompass inconsistent information, which may result in noise, unless some form of generalisation handles the noise. A learner must apply a set of rules to produce valid generalisations from various training examples that can execute unknown situations, with some level of confidence.

Supervised learning is dissimilar from clustering and other unsupervised learning tasks that require a learner to create its own class from the training data. Supervised learning methods include ensemble algorithms like Bagging and Stacking, Bike, Naive Bayes, Support vector machine, Random Forest and Decision Trees (Aleem et al. 2015: 11-23).

2.5.2 Unsupervised learning

This algorithm is trained on the unlabelled data and creates its own class for defect prediction (Antony & Singh 2016: 67-73). The training data is split into test data and training data (Liu 2011: 63-128). In an unsupervised learning method, class labels are not created. Unsupervised learning can be accomplished using clustering or association, whilst similar classes or clusters are grouped together.

Clustering technique may be used to separate data into various clusters based on some criteria, (e.g. separate into two clusters according to whether they contain defects or not). The applicable algorithms are applied on the data to create clusters. Groups that have similar data points are placed together in clusters (Aleem et al. 2015: 11-23). A function is required to define and calculate the distance between variables and the distance between clusters.

Nearest neighbour clustering

The number of clusters is pre-defined and k observations that are similar to a new record are identified. The algorithm classifies the new record into the correct class. In general, the Euclidean distance is applied.

Agglomerative clustering

This is a bottom-up technique that begins with empty clusters. Variables are consecutively added. In hierarchical clustering, all the observations are initially considered as individual clusters. Two samples that are similar are put closer to each other and in the later stages, the clusters can be combined.

Association

It applies an association-rule learning algorithm that discovers relations between variables, (i.e. customers that buy X also buy Y).

2.5.3 Semi-supervised learning

This is a self-training method. Rather than using unlabelled data to train a specific model, active learning creates an active learner that creates queries, normally unlabelled data instances to be classified by a human annotator. The objective of active learning is that an algorithm can predict more accurately with less training labels, if it is permitted to select the data from which it learns. However, most active learning techniques assume that there is a budget for the active learner to pose queries in the domain of interest. In real systems, there may be a restricted budget, which implies that the labelled data queried by active learning may not be adequate for learning (Pan 2014:537-570). Active learning is one of the methods of learning where a large number of unlabelled data exists. Its goal is to attain acceptable performance by learning with a small possible quantity of labelled data (Li, Zhang, Wu & Zhou 2012:201-230). E-mail spam detection is one of the examples of active learning.

Previous studies, for projects of varying sizes have been conducted on software defect prediction. Different types of software metrics have been used as input in these projects. Various types of predictors that include statistical, machine learning have been utilised as predictors.

2.6 Literature review

Statistical methods devise and apply formula to establish the correlation between software module properties and the likelihood of defects. These methods include the logistic regression, linear regression and the discriminant analysis.

2.6.1 Minimising defects in software

Boehm and Basili (2005: 426-431) provided useful insight about issues in software development using data. They state that locating and correcting defects software problems (after the system has

been delivered), costs 100 times more than correcting the error during development. They suggested that developers must correct errors early. The researchers also point out that 40% to 50% of existing project work is spent on errors that are avoidable. They maintain that rework can be avoided by improving software productivity.

There are methods that can be used to detect errors in the early stages of the software development life-cycle. Peer reviews can help to identify an average of 60% of the errors. Boehm and Basili (2005: 426-431) recommend the use of Harlan Mills' Clean room software development process and Watts Humphrey's Personal Software Process for enforcing personal discipline in creating highly-structured software, during the software development process.

Fenton and Neil (1999: 675-689) provided a critique review of models created for software defect prediction. They believe that size and complexity metrics cannot predict defects in software, as they only assume that defects are caused by the internal structure of a module. However, defects may be caused by modules that are difficult to write, programs specifications that are inconsistent and solving of a program incorrectly. According to the researchers, modules that consist of 200-400 LOC may contain human made errors that cause defects. Smaller systems may link modules, thus also causing more defects. It was declared that bigger modules have more reliability than smaller modules. However, this contradicts the theory of program decomposition, which is so central to software engineering. The researchers argue that averaged data in analysis prejudices the original data. They advise that an average of grouped data may suggest a trend that is not supported by the raw data. In this study, process metrics will be utilised in defect prediction.

2.6.2 Metrics and classifiers

Previous studies have been conducted to study the effectiveness of code metrics, process metrics and object-oriented metrics in software defect prediction.

2.6.2.1 Code, object-oriented and process metrics

Rahman and Devanbu (2013:432-441) compared the performance, consistency, flexibility and stasis of various metrics. The results revealed that code metrics, despite their extensive use, are generally

less beneficial than process metrics. They found out that code metrics have high stagnancy, (i.e., there is little transformation between the software versions) and that results in dormant defect prediction models. This leads to the same modules reported as defect prone, since the files that are persistently predicted as being defect prone may turn out to contain fewer defects.

Micro interaction metrics (MIMs) that capture developer interaction data were extracted from Mylyn for use in defect prediction. The behavioural interaction patterns of programmers can influence the software quality. The developer activities, such as modifying files and browsing tasks were recorded. The Correlation-Based Feature Selection (CFS) was used in the selection of features. The F-Measure evaluated the effectiveness of the algorithm, while the F-Measure of the MIM-based defect prediction (Lee, Nam, Han, Kim & Peter 2016:1015-1035). In the study, the MIMS metrics when combined with the source code and change history metrics improved the defect classification performance. About 59% of defects were detected from 21% of source code selected by the MIM-based defect prediction model.

Bug predictors of one project were used to predict defects in another project using open source static code metrics (Ferzund, Ahsan, Wotawa, 2008: 331-343). A Decision Tree Algorithm was applied to determine if the files were defect free or not. The algorithm was trained on the Firefox defect data and the testing of data was conducted on Apache HTTP Server and vice versa. The output varied depending on the version of the projects. The prediction accuracy ranged from 68 to 92%.

Xia, Yan and Zhang (2014:77-81) analysed the performance of combined code metrics, life-cycle process metrics and history change metrics in defect prediction. Defects were predicted in the Aerospace Tracking Telemetry and Control (TT&C) application using the Support Vector Machine optimised by Particle Swarm Optimisation. The results revealed that the combination set of code metrics, life-cycle process metrics and history change metrics can enhance the software fault prediction accuracy and that the history change metrics have an effect in producing better prediction accuracy for the TT&C software.

A data set based on Chidamber and Kemerer's object-oriented metrics was employed in locating software defects. The data was tested using the Levenberg-Marquardt-based neural network

algorithm (Singh 2013:22-28). The model had higher accuracy of 80.3% when compared to the polynomial function-based neural network predictors that had 78.8% prediction accuracy.

A defect prediction study was conducted by Malhotra and Khanna (2013: 273-286) to compare object-oriented metrics and the likelihood of change. The data from open source Java applications, Frinika, FreeMind and OrDrumBox were used in the study. The ROC was used in assessing algorithms' performances. The study revealed that the Response for Class (RFC) was a relevant metric of the likelihood of change in all the three data sets. The Random Forest and Bagging methods had better results than the Logistic Regression model, although all techniques produced good Area under the Curve (AUC), using Receiver Operating Characteristic (ROC) evaluation results.

Kaur, Kaur and Kaur (2015:1-5) studied code and process metrics for predicting faults in open source mobile applications. The process metrics results were better than the code metrics results. The performance measures that were compared were the correlation coefficient, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The process and code metrics hybrid model outperformed the code metrics model. The results showed that models that applied process metrics produced better accuracy in defect prediction of mobile applications in all seven ML methods.

Many prediction models are created during the initial stages of the projects and may no longer be appropriate to associate metrics values and defect proneness. Cavezza, Pietrantuono and Russo, (2015:8) used a continuously evolving approach for the defect prediction of a rapidly evolving software, Eclipse. Their dynamic approach refined software defect prediction models through the use of newly-obtained commit data to determine if the commit introduced a bug. Some of the process metrics used were associated with complexity, (e.g. statements added). The theory was that complicated commits are inclined to introduce defects; hence other metrics measured the knowledge of the developer who made a commit. A developer with experience was likely to create more defect free commits than the one with less experience. Since they used a dynamic approach, they had to retrain their predictor. They concluded that the dynamic approach has a better performance than the static one, for predicting the defectiveness of changes in software.

Statistical and machine learning techniques are some of the different methods of defect prediction.

2.6.2.2 Statistical linear regression

A regression model is created using N observed data and it symbolises the correlation between a variable, Y (dependent or output variable), (i.e.software defects), and a group of independent variables (also called input or predictor variables, (i.e. LOC, Authors, LinesAdded), $x\,j(j = 1,2,\dots,n)$(Valles-Barajas 2015: 277-287).

The correlation between the variable *Y* and each variable x_i can be defined by the equation (2.6):

$$\hat{Y}_i = \widehat{\beta_0} + \hat{\beta}_1 x_{i1} + \cdots + \widehat{\beta_n} x_{in}, \qquad (2.6)$$

where $\widehat{Y}$ is an approximation of the i_{th}value of the dependent variable,
n is the number of independent variables and N is the number of observed data.
$\beta_j(j = 0,1,\dots,n)$ are regression parameters representing the correlation between the dependent variable and the independent variables.

Valles-Barajas (2015:279-280) compared the fuzzy regression and statistical regression techniques. Statistical linear regression represents uncertainty as randomness, while fuzzy linear regression represents uncertainty as fuzziness. The results indicated that statistical regression model had better results than the fuzzy regression model. The RMSE and MAE values for the fuzzy regression model were greater than the values of RMSE and MAE for the statistical regression model. It was argued that the uncertainty in prediction models is due to randomness; therefore it is logical to create a prediction model using statistical linear regression technique, rather than the fuzzy linear regression method.

2.6.2.3 Logistic regression

Logistic regression (LR) statistical method formulates relationships among variables. Multivariate LR is applied in the creation of a model that predicts the change proneness of classes. Logistic regression is a suitable regression analysis to apply if the dependent variable is dichotomous (binary). The multivariate (LR) method can be described as (Malhotra & Khanna 2013: 274-286);

$$prob\ (Xn, X2 \ldots . Xn)^n = \frac{e(A0 + A1X1 + \cdots + AnXn)}{1 + e(A0 + A1X1 + \cdots + AnXn)} \quad (2.7)$$

where $X_i\ i = 1, 2, \ldots, n$ are the independent variables

$prob$ is the probability of detecting whether the class has changed.

A study was conducted by Malhotra and Khanna (2013:274-286) on defect prediction, using three selected open source, Java based software. The proficiency of the predicted models was assessed using the ROC analysis. The Random Forest (RF) had the best ROC results. Bagging and Rf had good AUC, specificity and sensitivity results. The study indicated that ML methods are equivalent to regression techniques. It was suggested that testing of change proneness of an application improves quality by predicting defects on the highly change prone modules.

2.6.2.4 Naïve Bayes

A comparative analysis of code and a set of change metrics in defect prediction was conducted (Moser, Pedrycz & Succi. 2008: 181-190). The Logistic Regression, Naïve Bayes, and Decision Trees were used to classify the Eclipse Java files as faulty or not. The Naïve Bayes is defined as (Ladha & Deepa 2011:1787-1797):

$$fi(X) = \prod_{j=1}^{n} P(xj \backslash cj) P(ci) \quad (2.8)$$

where $X = (x_1, x_2, \dots x_n)$ represents the vector of an attribute, i.e. x_1 is the value of feature X and $c_j, j = 1,2, \dots, N$ are the potential labels of the class, $P(x_j \setminus c_j)$ are conditional probabilities and $nP(c_i)$ are prior probabilities.

The results proved that process metrics provided better prediction accuracy for the Eclipse data, than code metrics. The code model had better TP, FP and accuracy results than the static code model. Decision Trees had the best Percentage Accuracy results compared to the J28 and Naïve Bayes. The cost sensitive classification produced more than 75% of accurately categorised files, a Recall greater than 80%, and a False Positive rate less than 30% on change metrics.

2.6.2.5 Rule-based techniques

A rule reduction technique was proposed by Monden, Keung, Morisaki & Matsumoto (2012: 838-847) to remove complex or identical rules without reducing the prediction performance. The experiment was conducted using Mylyn and Eclipse PDE datasets. In the experiment using Mylyn dataset, the reduction technique decreased the quantity of rules from 1347 to 13, whereas the change of the prediction outcome was .015 (from .757 down to .742) according to the F1 prediction condition. In tests conducted using the PDE dataset, the new association rule mining method minimised the quantity of rules from 398 to 12, whereas the prediction performance produced better results (from .426 to .441).

The rule-based prediction was compared with algorithms such as Logistic Regression, RF, CART and Naïve Bayes algorithm. Consequently, the recommended association rule mining approach produced accuracy that was comparative to the normally used machine learning algorithms.

2.6.2.5 Distance and clustering

Some of the clustering techniques that have been employed in previous defect prediction studies include:

2.6.2.5.1 K-Means clustering

Clustering is a method that divides an unlabelled dataset into groups, where the separate groups comprise of objects that are identical to each other, according to a specific similarity degree (Coelho, Guimarães & Esmin 2014: 356). The objective of the clustering method is to locate groups of firmly connected classes, which have the possibility of containing a set of identical attributes.

This common, prototype-based method, partial-clustering method, endeavours to locate a designated quantity of clusters (*C*), which are characterised by their centroids (Tan, Steinbach & Kumar 2006: 488-567).

Basic K-Means Algorithm (Tan, Steinbach & Kumar 2006: 488-567):

1. **Select C** points as initial centroids
2. **repeat**
3. **Create C** clusters by allocating each point to its nearest centroid
4. R**ecalculate** the centroid of each cluster
5. **Until** the centroids do not change

The mean is regarded as a centroid. The points are allocated to a centroid, then the centroid is revised. A proximity measure is used to quantify the closest centroid. Euclidian (L_2) distance is one of the proximity measures that can be used the distance to the closest centroid (Pandeeswari & Rajeswari 2015:179-185).

A scatter, which is called the *Sum of Squared Error* (SSE), calculates the quality of clustering. In a case of two separate clusters which are created by two different runs of *K-means*, the one with the minimum squared error is preferred, since it implies that the prototypes (centroids) of the clustering are an improved depiction of points in their cluster.

Table 2.5 Sum of Squared Error

Symbol	Description

X	An object
C_i	The i^{th} cluster
c^i	The centroid of cluster C_i
m_i	The number of objects in the i^{th} cluster
M	The number of objects in the data set
K	The number of clusters

The SSE is defined by the equation (2.9):

$$\text{SEE} = \sum_{i=1}^{K} \sum_{x \in Ci} dist(ci, x)^2 \quad (2.9)$$

The centroid that minimises the SSE of the cluster is the mean. The centroid (mean) of the i^{th} cluster is described by the equation (2.10);

$$ci = \frac{1}{mi} \sum_{x \in ci}^{\infty} (\text{x}) \quad (2.10)$$

In a research conducted by Ghotra, McIntosh and Hassan (2015: 279-280), the *k-means* and Expectation Maximisation clustering methods were analysed on the National Aeronautics and Space Administration (NASA) and PROMISE datasets. The results showed that prediction algorithms tested using Decision Trees, statistical techniques, K-nearest neighbour, and Neural Networks perform better than the algorithms trained using clustering methods, rule-based techniques and SVM. The main findings proved that there were statistically significant differences between the performances of

defect prediction models, trained using numerous algorithms within the cleaned NASA dataset and the PROMISE one as well.

2.6.2.5.2 Package-based clustering

A new clustering technique known as Package-Based Clustering (PBC) was used in defect prediction. The technique was based on linked objected-oriented classes, which create packages in Java. The method applied textual analysis on source codes to locate object-oriented classes from a software application. To create clusters, the method obtained the package information from each class and searched for the package name. If the quantity of classes of a cluster was lesser than the quantity of explanatory variables used in the prediction model, the method combined small clusters to qualify them to create a prediction model. Lastly, the linear regression model using PBC was analysed on JEdit 3.2. The results proved that software defect prediction using the proposed PBC performed better than the prediction models using Border Flow, K-means and the Entire system, since PBC uses source code similarities and relationships to group the software into clusters. The prediction model using PBC was 54%, 71%, 90% better than the prediction models created on Border Flow, K-means and the whole system respectively (Islam & Sakib 2014: 81-86).

2.6.2.5.3 Fuzzy C Means clustering

A hybrid Fuzzy C Means (FCM) clustering and RF software prediction model was proposed by (Pushphavathi, Suma & Ramaswamy 2014:1-5). The FCM algorithm ranked the features according to their importance. A new subset was created from the ranked list and input in a RF algorithm for defect prediction.

The aim of the FCM clustering is to have different degrees of membership to each of the clusters. An object can belong to more than one cluster on the basis of fuzzy membership value $([0,1])$ rather than on the ground of crisp value $(\{0,1\}])$ as in *k-means* algorithm (Gupta & Kumar 2017:135-145).

The FCM clustering technique is founded on a target function. So as to let the target function meet specific circumstances, a dynamic iteration which changes the clustering centroids is run.

The quantity of samples p, the quantity of clusters c $(1 < c < p)$, the samples $X_1, X_2, \dots, X_p$, the fuzzy factor $m(m > 1)$, and the initial clustering centroids $C_1, C_2, \dots, C_c$ should be initialised. A target function is set to achieve the clustering;

$$J_m(U,V) = \sum_{j=1}^{p}\sum_{i=1}^{c} u_{ij}^{m}\, d_{ij}^{2} \qquad (2.11)$$

subject to:

1 $$\sum_{i=1}^{c} u_{ij} = 1, 1 \le j \le p;$$

2 $$u_{ij} \ge 0, 1 \le i \le c, i \le j \le p$$

3 $$\sum_{j=1}^{p} uj > 0, 1 \le i \le c$$

A membership degree is computed by;

$$u_{ij}(t) = \frac{1}{\sum_{r=1}^{c}\left(\frac{d_{ij}(t)}{d_{rj}(t)}\right)^{\frac{2}{m-1}}} \qquad (2.12)$$

where $1 \le i \le c$ and $1 \le j \le p$

Once the target function satisfies the conditions, the appropriate clusters are formed. Samples from the same cluster would have more resemblance, whereas samples from different clusters would have little resemblance (Li, Zhao & Xu 2017:1-10).

During the iteration, the centroids are modified to keep the centroids and cluster positions accurate. In a research conducted by Pushphavathi, Suma and Ramaswamy (2014: 1-5) the accuracy, sensitivity and specificity were applied in the performance evaluation of the prediction models, RF, FCM and the hybrid FCM and RF. The accuracy measurements was 81.7% of web applications, 91% of business,89% of retail, 98% of medical and 87.9% of ERP applications. The accuracy of the model indicated that both RF and FCM were in adequate accuracy level, but hybrid model displayed more accuracy as compared to individual of RF and FCM models.

2.6.2.5.4 Mahalanobis-Taguchi

Liparas, Angelis and Feldt (2012:141-165) used the Mahalanobis-Taguchi (MT) strategy to detect and evaluate defective modules. The datasets for this study were ten defect-prone modules from the NASA Metrics Data Program repository. The MT method combines mathematical and statistical concepts like Mahalanobis distance, Gram-Schmidt orthogonalisation and experimental designs to support diagnosis and decision-making based on multivariate data.

The Mahalanobis distance (MD)

1. It considers relationships between the features.
2. It can be affected by changes in the reference data.
3. The quantity of dimensions in a system has no influence.

In the MT, the MD has been applied in two different ways: The Mahalanobis
Taguchi System (MTS) and Mahalanobis Taguchi Gram-Schmidt process (MTGS). In a dataset that contains k variables and n cases (the size of the sample). Let x_{ij} be the value of the i_{th}variable ($i = 1, \dots k$),on thej_{th}case($j = 1, \dots, n$).

The variables are standardised by;

$$z_{ij} = (x_{ij} - m_i)/s_i \qquad (2.13)$$

where m_i and $s_i$represent the sample mean and standard deviation respectively of the ith variable. The computation of the MD in MTS is;

$$MD_j = (1/k)Z'_jC^{-1}Z_j) \quad (2.14)$$

where $MD_j$is the Mahalanobis distance calculated for the j_{th} case $(j = 1, \dots, n)$, and Z_j is the column vector comprising the standardized values of the j_{th}case. The C^{-1} denotes the inverse of the sample correlation matrix.In MTGS, the MD is calculated in a different way than MTS.

In the study, two thirds of the defect free and defect observations were used as training set, while the rest (one third) observations were used as test set, to evaluate the predictive capability of MT (Liparas et al. 2012:141-165).

To assess the capability of the method, the ROC curves were plotted and the AUC metric was computed together with its significance. As a result of the application of the two-step cluster analysis on the training sets and the definition of the appropriate thresholds, MT produced either very high or in some cases, perfect training classification accuracy in all data sets.

2.6.2.6 Tree-based techniques

The tree-based techniques produce a model of decisions created on real values of attributes in the data. Decisions split the tree structures until a prediction result is attained for a specific record. The trees are trained to resolve classification and regression problems. Decision trees often have faster processing speed and better accuracy and are preferred in machine learning. The input and output variables can be both categorical and continuous. The sample is separated into two or more similar sets founded on the most relevant divider in input variables.

Types of decision trees

These are split into:

a. **Categorical variable decision tree**: It has categorical output variable, (e.g. if one is a cricket player), where the target variable was “The member is a cricket player or not” (i.e. YES or NO.)
b. **Continuous variable decision tree**: It has a continuous target variable.

Decision trees are not influenced by missing values or outliers. They can cater for both numerical and categorical variables. They are non-parametric, (i.e. they have no assumptions about the space distribution and the classifier structure).

Gini Index

It is one of the algorithms for splitting a decision tree. Given *c* classes of the target attribute, with the probability of the ith class being *Pi*, the Gini Index is (2.15);

$$Gini(S) = 1 - \sum_{c=1}^{c} \binom{n}{c} \mathrm{p}i^2 \qquad (2.15)$$

The attribute that is used to split is the one with the maximum decrease in the value of the Gini Index. The common decision tree methods are the Classification and Regression Tree CART, C48 and ID3.

2.6.2.6.1 CART

The Classification and Regression Tree, (CART), is a method for analysing data. CART demonstrates the prediction of data using a sequence of decisions at each node of the tree. The input data set is split into root nodes by a progression of repeated binary splits. The binary divisions are created by CART, based on the significant independent variables. At each binary split, two homogeneous subsets are produced with respect to the dependent variable, (can be the number of defects in a software file). The CART algorithms first create a large tree and then prunes it back to avoid over fitting (Khoshgoftaar & Seliya 2003:259).The different types of CART are the CART-LS (Least Squares) and CART-LAD (Least Absolute Deviation).

In a software defect prediction study conducted by Muthukumaran, Choudhary and Murthy (2015:15-20) the classification algorithms, Gaussian Naïve Bayes, CART Decision Tree, Logistic Regression and Naïve Bayes Tree were used to create defect prediction models for all versions of Eclipse JDT project. Precision, Recall and F-Measure were calculated. The Naïve Bayes Tree algorithm had better accuracy than all other three separate algorithms for each separate version. The average values for Precision, Recall, F-Measure of Naïve Bayes Tree were 75.02, 76.44, 74.62. The results were better than those of the Decision Tree, Logistic Regression and the G. Naïve Bayes algorithms.

2.6.2.6.2 C4.5 and J48

The J48 is a variation of C4.5, which is a standard decision tree classification algorithm. It was introduced by Quinlan (1986:81-106) and is used to create a decision tree based on a training data set. The C4.5 is built using entropy that stems from the concept of information entropy. It was derived from the original ID3 algorithm (Seliya, Khoshgoftaar & van Hulse 2010:26-34).

The basic concepts of the ID3 are that each node in the tree links to a non-categorical attribute (decision node) and each branch to a potential value of the attribute. The terminal node of the tree specifies the projected value of the categorical attribute for the records described by the path from the parent node to the terminal node.

Information Gain is applied to select the most informative non-categorical attribute among the attributes that have not yet been examined in the path from the root. Information Gain is based on entropy, a notion that was presented by Claude Shannon in Information Theory (Ellerman 2009:119-149). The ID3 was originally developed by J. Ross Quinlan (Quinlan 1986:81-106).
The C4.5 uses Gain Ratio to select features at the parent node of a sub-tree when the decision tree is created. The Gain Ratio is (Wang, Khoshgoftaar, Wald & Napolitano 2012:301-307):

$$Gain\ Ratio(S,T) = \frac{Gain(S,T)}{SplitInfo(S,T)} \qquad (2.16)$$

where the split information is:

$$SplitInfo(S,T) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|} \quad (2.17)$$

where S_i is c sample sub-sets by dividing, S using c values of attribute T. Split information is the entropy of S on all values of attribute T.

A research that compared three compressed C4.5 models and the original C4.5 model was conducted in a study (Wang et al. 2012:301-307). The Compressed C4.5 Model I, Compressed C4.5 Model II and Compressed C4.5 Model III apply the Spearman's rank correlation coefficient as the foundation of selecting the parent node of the decision tree, which influences the models to be better defect predictors. An experiment was conducted to test the effectiveness of the Compressed C4.5 Model I, Compressed C4.5 Model II and Compressed C4.5 Model III models. The enhanced models minimised the decision tree's size by 49.91% on average and improved the prediction accuracy by 4.58% and 4.87%.

Seliya, Khoshgoftaar & van Hulse (2010:26-34) investigated the Roughly-Balanced Bagging formula for predicting software defects using imbalanced data. The method combined bagging and data sampling for solving the class imbalance issue. Software defect prediction models were created using the RB-Bag algorithm and contrasted with the models that were constructed without bagging or data sampling. The contrast was meant to highlight the need to address class imbalance during defect prediction modelling. Two normally used classification algorithms for defect prediction, C4.5 and Naıve Bayes, were applied in the study. A case study involving fifteen software metrics and defect data sets acquired from numerous actual high assurance systems was conducted.

Six thousand defect prediction models were created. The main assumptions made in the research were the following:

(a) The RB-Bag formula efficiently addressed the class imbalance issue when creating defect prediction models.

(b) The software quality models that applied the RB-Bag algorithm achieved a more significant performance than the models that did not utilise the bagging or data sampling techniques, particularly when the C4.5 learner was used.

(c) Generally, the Naïve Bayes algorithm had a superior significant performance than the C4.5 classification algorithm, on the other hand the combination of the R-B Bag and the C4.5 outperformed the Naïve Bayes classification algorithm.

2.6.1.7 Ensemble techniques in machine learning

Ensemble methods combine and build models using different or similar classifiers that yield better results than a using single classifier (Liu, Wu & Zhang 2011:979-984). Common examples of Ensemble methods are *Bagging* and *Boosting*.

Bagging involves building multiple *k* models (e.g. decision trees, neural networks from various *N* samples of the training dataset). The prediction result averages the *k* models. Boosting builds or adds models each of which learns to correct prediction errors of previous models in the chain. Stacking is a technique that creates several models (naturally of different types) and a supervisor model that learns how to best consolidate the predictions of the primary models to create a higher level prediction model.

Random forest

This is a tree-based method that applies the Bagging technique. Each model is built independently with an aim to reduce variance. Random forests are a means of averaging several deep decision trees that are trained on various parts of the same training set, with the aim of solving the over-fitting problem of a separate decision tree. A Random forest is an ensemble-learning technique for classification and regression that creates many decision trees during training and yields a class that is the mode of the classes output by individual trees. Feature selection or multi-dimensional scaling, (MDS) are used to identify similar or dissimilar nodes.

Three successive versions of Eclipse were studied in software defect prediction study using active learning. The Random Forest was chosen as the base algorithm in the active learning tests. It was observed that dimensionality reduction methods, mainly multi-dimensional scaling with Random Forest similarity, produce superior results compared with other active-learning methods and feature selection techniques due to their capacity to recognise and consolidate essential information in data set attributes (Lu, Kocaguneli & Cukic 2014:315). Multi-dimensional scaling with Random Forest had the best performance in terms of the Precision, Recall and Accuracy measures.

Random Forest was one of the machine-learning models used in an exploratory study that examined if test execution metrics can be utilised in assessing software quality and to create pre- and post-release fault prediction models. The study demonstrated that test metrics acquired in Windows 8 development could be used to build pre- and post-release defect prediction models in the initial development phases of a system. The test metrics outperformed pre-release defect counts when predicting post-release defects (Herzig 2014: 309). In the experiment that predicted post-release defects, the Random Forest model had the best scores compared to other models. It scored 0.81 for the Precision and 0.70 for the Recall in the binary level test. The Random Forest Precision and Recall values for the file level were 0.65 and 0.24 respectively.

A feature-level bug prediction by method that was based on test cases traversal path was proposed by Anand (2015:1111-1117). For every change or addition to a function, an Impact Score was calculated per feature based on the test case traversal path to a function. The prediction was conducted at feature level instead of class, file, package or binary level, since certain features in software systems are more critical than others and faulty ones have an impact on the functioning of the entire system. Metrics were used to calculate the Impact Score for functions added, deleted or modified (Anand 2015:1112). Prediction accuracy was measured using the discounted cumulative gain. The approach scored the gain value of 0.684 for predicting defective attributes.

2.6.3 Feature selection

Software defect prediction research that is based on feature selection has been conducted. In previous research, dimension reduction and the selection of the most significant attributes results in improved prediction accuracy.

Wang, Khoshgoftaar & Seliya (2011:69-74) suggested that an average of three software metrics is capable of predicting defects. The researchers developed a feature ranking method called, Threshold-Based Feature Selection technique (TBFS), a feature ranking technique to select important attributes. The five different types of feature ranking algorithms that were utilised included the Mutual Information, Kolmogorov-Smirnov, Deviance, Area Under the ROC Curve, and Area Under the Precision-Recall Curve. The subsets that were chosen by the five selection algorithms were of different sizes. More than 98.5% of the attributes were removed. The AUC performed best than the other rankers in all 12 cases where the Multilayer Perceptrons algorithm was used. The AUC had better performances in 9 out of 12 cases where the *k* nearest neighbour algorithm was used and 7 out of 12 where the logistic regression was used.

A hybrid search method that comprised of seven (7) feature ranking methods and three (3) feature subset selection techniques was presented by (Gao, Khoshgoftaar, Wang & Seliya 2011:579-606).The chi-square feature ranking technique had consistently poorest performance. Five common classifiers were used in the prediction process. The Naive Bayes, multilayer perceptron, and logistic regression had better performances than the support vector machine and *k* nearest neighbour. Even though the feature ranking methods had similar performances, the hybrid method produced the best results. The classifiers' performances improved or remained constant after 85% of the features were removed.

Weyuker and Ostrand (2008:1-11) developed a prediction model for systems that have regular releases. When the model was tested using a system with no releases, its accuracy dropped. However, the top 20% of the files still had about 75% defects. The accuracy of the model increased

slightly after the developer's access information was added. The model was able to identify 81.3%, 94.8%, and 76.4% of the faults in three subsystems, compared to 81.1%, 93.8%, and 76.4% of faults before the developer data was added.

In a feature selection study, Jose and Reeba, (2014) introduced a novel Fast clustering-based feature Selection algoriThm (FAST) algorithm that eliminates both insignificant and redundant features. Symmetric uncertainty (SU) was applied to select relevant features. SU measures the linear association or correlation between two features and between a feature and a class value. The minimum spanning tree grouped identical features in respective clusters. The features which were most relevant to the target classes were selected and redundant ones were removed from each cluster.

Feature ranking by means of the wrapper method was employed to select the best variables for predicting software defects in a very large legacy telecommunications software system (LLTS) and in NASA software. Data sampling techniques proved to offset the negative effects of class imbalance. The experiments were run on the Naïve Bayes, Multilayer Perceptron, Logistic Regression, KNN and SVM algorithms in collaboration with nine performance metrics. The results prove that feature selection is effective after data sampling except for the Wilson's editing sampling method (Gao, Khoshgoftaar & Seliya 2012: 3-42).
Stratification techniques were demonstrated to improve defect prediction accuracy (Pelayo & Dick 2012:516-525). These techniques solve dataset imbalances, (i.e. defects that are not uniformly distributed). The interactions between oversampling and undersampling were analysed using the ANOVA and blocked factorial design methods, they (interactions) were shown to influence prediction accuracy. Oversampling on its own had no effect.

Fast Correlation Based Filter

The FCBF method for assessing feature relevance and redundancy was proposed by (Yu & Liu 2003:1-8). The method is based on the **predominant correlation** concept.

Attributes that are predominant in predicting a class concept are regarded as good. The predominant or main features are selected while the remaining ones are eliminated. Tests were done using Waikato Environment for Knowledge Analysis (WEKA) implementation of the classification algorithms which include the FCBF. A total of ten data sets were chosen from the UCI Machine Learning Repository (Blake & Merz 1998) and the UCI KDD Archive.

Four feature selection algorithms, FCBF, ReliefF, CorrSF, ConsSF, respectively were executed per data set. The running time was recorded and attributes were chosen for each algorithm. The C4.5 and NBC were applied on the original data set and each newly-acquired data set comprising of only the selected features from each algorithm. The 10-fold cross-validation was used to obtain the train and test sets. FCBF achieved the highest level of dimensionality reduction, since it selected the least number of features (with only one exception in US Census 90), which is consistent with the theoretical analysis about FCBF's ability to locate redundant features (Yu & Liu 2003:1-8).

A method was designed to predict student success in admission in an engineering stream. Data encompassing students' academic in addition to socio-demographic variables was investigated. The features such as family pressure, interest, gender, XII marks and CET rank in entrance examinations and historical data of previous batch of students was covered. The FCBF was implemented in Netbeans in selecting relevant and non-redundant features. The features were run on the NBtree, MLP, Naïve Bayes and IBk (Doshi & Chaturvedi 2014: 197-206).

In a similar study, a novel feature selection algorithm, MICHAC, was designed. The algorithm uses MIC to eliminate irrelevant features and Hierarchical Agglomerative Clustering to select non-redundant features and optimise the performance of classifiers in defect prediction. The experiment was conducted on 11 NASA and 4 AEEEM projects. The results were compared with those of other machine learning algorithms that were used to select features. The evaluation measures indicated that the MICHAC algorithm produced better results than the other methods in defect prediction (Xu, Xuan, Liu & Cui 2016:370-381).

2.6.4 Machine learning techniques

Various approaches have been applied in developing software fault prediction models. These comprise methods such as testing metrics, complexity metrics, multivariate approaches and machine learning. Machine learning techniques include Decision Trees, Clustering, Neural Network and Support Vector Machines.

Machine learning is a branch of artificial intelligence regarding computer programs learning from data (Alshayeb, Eisa & Ahmed 2014: 7866). It aims at imitating human learning process with computers and is about observing a phenomenon and generalising from the observations. Machine learning can be categorised as supervised or unsupervised learning. Supervised learning is learning from examples with known outcome, while unsupervised learning is learning from data with unknown outcome (Shepperd, Bowes & Hall 2014:604).

Supervised learning, also known as classification is learning from examples with known outcome, it classifies instances into two or more classes. Previous software defect prediction studies have used different types of machine learning algorithms (also called classifiers) for supervised learning. These include Decision Trees, classification rules, Neural Networks and probabilistic classifiers.

In a defect prediction study, a Mutual Information (MI) and fuzzy integral-based algorithm was used to analyse the interaction among attributes. The algorithm used the fuzzy measure set function to obtain information about the attributes. The best attributes which were deemed to improve the prediction performance were selected (Liu, Lu, Shao & Liu 2015: 93-96).

In a feature selection study, Jose and Reeba (2014) introduced a novel FAST algorithm that eliminates both insignificant and redundant features. Symmetric Uncertainty (SU) was applied to select relevant features. The SU measures linear association or correlation between two features and between a feature and a class value. The minimum spanning tree grouped identical features in respective clusters. The features which were most relevant to the target classes were selected and redundant ones were removed from each cluster.

In a defect prediction research for the NASA open-source system, a novel non-negative sparse graph semi-supervised learning method (that employed the Laplacian score sampling strategy) was created. The graph was designed to enhance the prediction ability. The Laplacian score sampling as used to train the data and resolve the class imbalance problem. The label propagation method predicted the labels of software modules for software defect prediction. The algorithm had better results than other prediction methods (Zhang, Jing & Wang 2016:1-15).

In a defect prediction study, a comparison of the Principal Component Analysis and Information Gain (IG) in the identification of irrelevant features was conducted. Random data was used for training and testing. The PCA and IG methods are based on entropy uncertainty. The PCA transforms a larger input space and represents all variables in a smaller input space. The influence of the PCA and IG was studied on the Classification Tree and Fuzzy Inference System prediction models. In the research, the IG approach enhanced the classifiers' prediction accuracy better than the PCA method, except in small datasets with many independent variables (Rana, Awais & Shamail 2014:637-648).

A hybrid algorithm of the Random Forest (RF) and FCM clustering was designed (Pushphavathi et al. 2014: 1-5). Random Forests are powerful techniques for high dimensional classification and skewed problems that can be used in pattern recognition and machine learning. The FCM ranked attributes in order of importance. A total of 19 predictor-sets created the new dataset. Afterwards, the data was loaded into the FCM method, which created models for predicting defects. The performance of the models was assessed using accuracy, sensitivity and specificity. The output showed that the hybrid technique was more efficient and noncomplex, allowing better prediction of software defects.

A novel algorithm for selecting features using FEature Clustering and feature Ranking (FECAR) was employed, they select highly important attributes to be used in locating defects. The method first grouped attributes into clusters using the FF-Correlation and then selected relevant attributes from each cluster based on the FC-Relevance measure. Clustering causes the inner-cluster attributes to strongly correlate with each other. The datasets used were the derived from three releases of Eclipse

and all NASA software. The Eclipse datasets comprised of code complexity metrics and abstract syntax tree metrics. The Naïve Bayes and C4.5 classification algorithms were used predict defects. The results revealed that removing 85% of attributes did not affect results. The SU was used as the correlation measure and Information Gain, Relief F and Chi-Square were used to select the relevant attributes (Liu, Chen, Liu, Chen, Gu & Chen 2014: 426-435) .

This research uses machine learning algorithms to predict if classes of the systems that are tested are error prone.

2.6.5 Deep learning

A method called Deeper, which influences deep learning methods in predicting defect-prone modifications was presented (Yang, Li, Xia, Zhang & Sun 2015: 17-26). Deeper comprises of the attribute selection and classification phases. The Deeper leverages a Deep Belief Network to attain superior achievement. The Deep Belief Network (DBN) contains of many Restricted Boltzmann's Machines (RBM). The DBN is employed to produce and incorporate advanced attributes from the initial attributes. A classification algorithm is linked to the last RBM, in which the hidden layer of the last RBM is the input layer of the classification algorithm. The strong point of DBN compared to Logistic Regression is that the DBN can create an unambiguous set of features from the initial set. The created feature set, which may contain $x + y, x^y$ and more complex non-linear combination of the initial features, is more influential in expressing the complexity of problems.

In the algorithm, the DBN is used and it encompasses three piled RBMs and a Logistic Regression classification algorithm. The dimensions of input consisted of 14 basic features and output was made up of 2 labels. These were fixed, what was changed was the numbers of hidden layers and units.
The whole network structure chosen had layers of size 14-20-12-12-2, which implied that the first RBM had 14 visible units and 20 hidden units, the second RBM had 20 visible units and 12 hidden units, the third RBM had 12 visible units and 12 hidden units and the classification algorithm had 12 input units and 2 output units.

Data sets from six open source projects, (i.e., Bugzilla, Columba, JDT, Platform, Mozilla and PostgreSQL), comprising a total of 137,417 changes were utilised for just-in-time defect prediction (Yang, Lo, Xia, Zhang & Sun 2015:17-26). Defects were identified and fixed in time using the Deeper approach. This was compared with the method presented by (Kamei, Shihab, Adams, Hassan, Mockus, Sinha & Ubayashi 2013:757-773)

In a new change x the confidence scores for x are calculated to determine if defective or defect free. This is described as $Conf_{defective}(x)$ and $Conf_{\mathrm{dfree}}(x)$in the following formula:

$$Conf_{defective}(x) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i \times x_{f_i})} \quad (2.18)$$

$$Conf_{dfree}(x) = \frac{\exp(w_0 + \sum_{i=1}^{m} w_i \times x_{f_i})}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i \times x_{f_i})} \quad (2.19)$$

The score $Out(x)$ is calculated as:

$$Out(x) = \frac{Conf_{defective} - Conf_{dfree}}{LOC(x)} \quad (2.20)$$

The results from the experiments display that across the six projects, Deeper located 32.22% more defects on average, compared with Kamei et al. (2013:757-773) technique (51.04% versus 18.82% on average). In addition, Deeper can accomplish F1- scores of 0.22 to 0.63, which are statistically and significantly higher than those of Kamei et al. (2013:757-773) approach on four out of the six projects. The defect data used in this study is from versioning systems and therefore process metrics were chosen. Previous software defect prediction research has been conducted based on information theoretic feature selection.

2.7 Chapter summary

The development of information technologies has given rise to large amounts of data. Defect data in software repositories is useful in evaluating software quality. This data contains information about the changes applied on the source code, due to defects. Predictors mine the data and assist project managers to identify modules that are error prone and prioritise them. Resources are then allocated to the critical modules, thus saving time and costs.

This chapter discussed the literature review concerning sources of defect data and software defect prediction techniques. The data used in previous studies was from company and open source. Earlier studies used source code metrics for predicting defects. In the recent years, process metrics and the hybrid of process and code metrics have been the preferred metrics for defect prediction research. Most of the studies use statistical or machine learning techniques in locating error-prone classes. The next chapter will present the methodology that was used in conducting the experiments.

CHAPTER 3

EXPERIMENTAL DESIGN AND METHODOLOGY

3.1 Introduction

In the preceding chapter, the background study which gave motivation for this research was discussed. The review of software defect prediction studies was conducted to understand the metrics and defect prediction models suitable for revolving software products. This chapter provides an overview of different epistemological approaches, the research methodology that was selected for this research and the philosophical views is supporting this approach. The research experiment part discusses the methods that will be used to provide a solution to the research questions. The details of the data used in the research, tests conducted and processes are elaborated.

3.2 Research

Different approaches are used for research. The methods selected depend on the questions asked pertaining a specific topic that is of interest to a researcher.

3.2.1 Research paradigm

Kuhn (1970: 176) theorises that a scientific community is defined by its members who have shared beliefs, similar education and professional indoctrinations and have learned from the same technical literature. Misunderstandings are quickly eradicated due to the members' shared assumptions, beliefs, models and views. This shared belief system is a 'paradigm'.

In the opinion of Johannesson and Perjons (2014:167), a research paradigm addresses *ontological questions* concerning the nature of reality, entities that exist, their relationship and interaction. A research paradigm also deals with *epistemological questions* on the methods used by people to acquire knowledge, (see Figure 3.1). A research paradigm answers *methodological questions* about valid methods of investigating reality and how to approve that the knowledge obtained is legitimate.

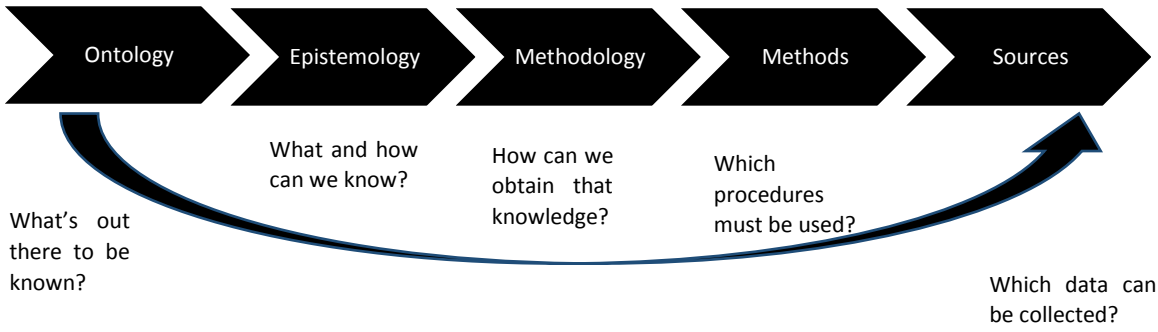


Figure 3.1 The relationship between ontology, epistemology, methodology and sources in conducting research (Zou, Sunindijo &Dainty 2014: 316-326)

The paradigms that can be used in research are the Behavioural Science and Design Science approaches.

3.2.2 Design Science Approach

Design Science is a paradigm that aims to create an original artefact to address business problems (Peffers, Tuunanen, Rothenberger & Chatterjee, 2007:10-53). Design Science is presently recognised as a formal research method. This paradigm has its origins in engineering and the sciences of the artificial. The difference between natural science and design science in that the former relates to how things are and the latter is concerned with how things should be. Behavioural Science research is an origin of natural science and its goals are to develop and defend theories which explain or predict organizational human phenomena surrounding the analysis, design, implementation, management, and use of information systems. On the other hand, Design Science Research (DSR) aims at creating innovations that define ideas, practices, technical capabilities, and product through the analysis, design, implementation and management (Adikari, McDonald &

Campbell, 2009: 549-558). Behavioural Science attempts to “understand” the problem. Design Science attempts to “solve” it.

The goal of the DSR is to design a solution for an environment that is connected to the design activities, see Figure 3.2. The knowledge base provides existing knowledge to the research. This consists of foundations, current experiences and skills, and existing artefacts and processes (Adikari et al. 2009: 549-558).

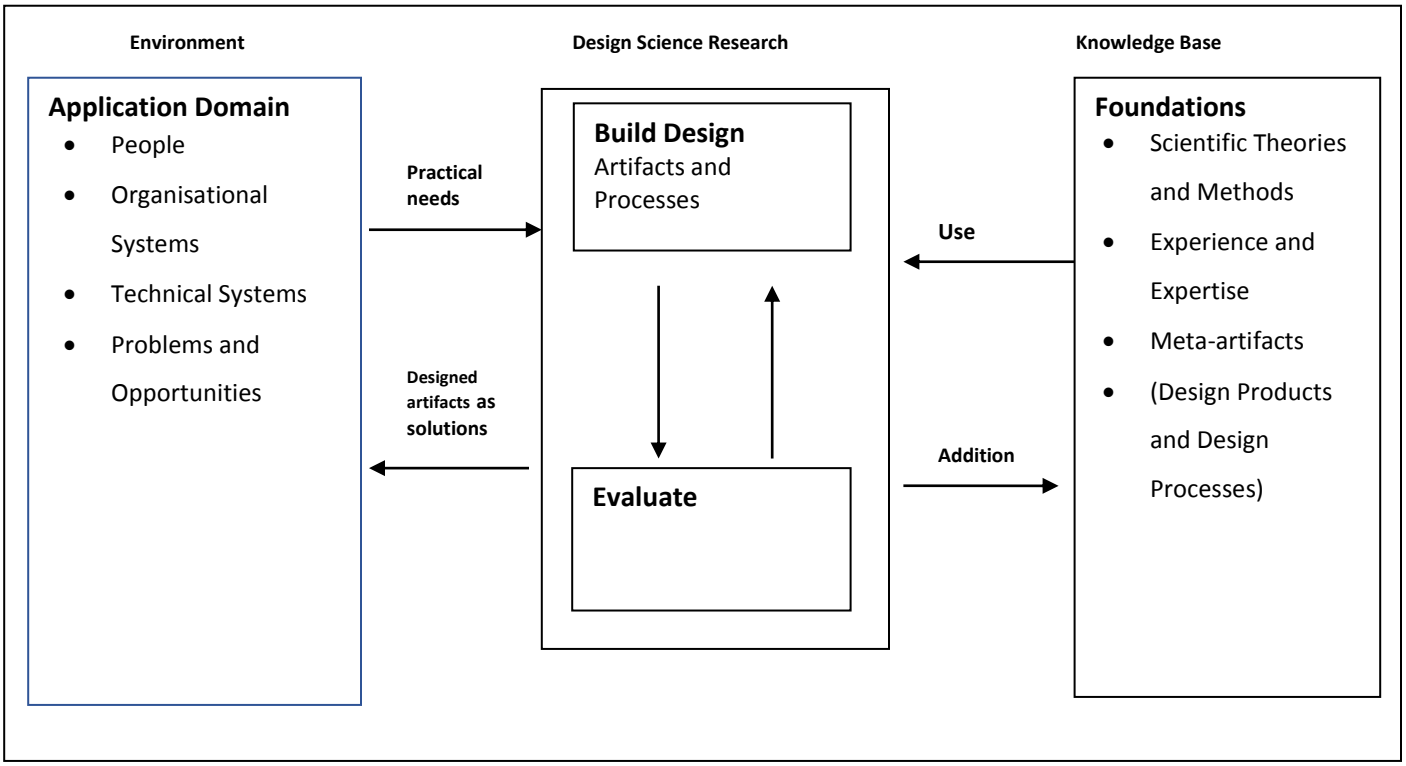


Figure 3.2 Design Science Research Cycles (Adikari et al. 2009: 551)

In this study, the Design Science Approach was used to create a novel feature selection algorithm. The artefact, MICFastCR was evaluated to test its effectiveness.

3.3 Research experiment

A literature review on software defect prediction was conducted. The literature review included software defect prediction studies of short release cycle applications. This study used the positivist research paradigm, which relies on experimental approaches. Research on the type of software metrics and applications suited for them was conducted.

3.3.1 Data

In software defect prediction studies, metrics are extracted from open source or commercial data. In this study, process metrics were used in defect prediction. These metrics contain process indicators that show the evolvement of software. Common pre-processing methods comprise sampling, selecting relevant attributes, techniques for reducing the size of attributes, translating the data and removal of noisy features.

Data sets

The open source datasets Apache Lucene, Mylyn, Equinox, PDE and JDT compiled by Ambros, Lanza & Robbes (2010: 31-41) were used in the experiment, (see Table 3.1). The data is from the Apache and Eclipse systems and the researchers created a website to share data. The data consists of 502 change metrics and their histories.

Table 3.1 Fault Data

Name	Files	Description
Lucene	691	Full-text search engine library
Mylyn	1862	Task and application lifecycle management
Equinox framework	324	OSGI core framework
Eclipse PDE	1497	Development
Eclipse JDT	997	Eclipse Java Development Tools

The datasets consist of classes which are considered not to contain defects if the bug value is 0, or else they are defective.

Software metrics

This section answers a research question mentioned in Section 1.9. The reasons for the selection of metrics and prediction techniques are specified.

RQ1. Which metrics are suitable for predicting defects in the versions of a software product line?

Previous studies indicate that change metrics serve as good predictors of software defects of evolving products.

Process metrics

The metrics assist software developers to analyse the current process by gathering data from all the revisions and over a long duration. Process metrics assess the changes that transpired, while developing a software version. The metrics can be measured in relation to a period of time.

3.3.2 Dimension reduction and feature selection

Attribute selection is a method that is applied in the selection of an ideal subset of attributes to improve a prediction model's accuracy. Dimension reduction determines the least number of dimensions that can build an effective prediction model (Lu, Cukic & Culp 2014: 416-425). It minimises storage requirements and speeds up the processing time of the classifiers and improves the prediction accuracy (Bafna, Metkewar & Shirwaikar 2014: 65-67). Relevant features are selected from an original data file.

3.3.3 Redundancy elimination

This is a pre-processing step in which redundant or highly-correlated features are removed from the data. Redundant features supply information which exists in other attributes and thereby reduce the predictive performance (Xu et al. 2016:370-381). In this study, a hybrid algorithm that is based on the

FCBF was used to eliminate redundant features in all sets selected by the feature selection algorithms.

Symmetric uncertainty (SU)

In information theory, SU is a normalised measure that evaluates the dependencies of features using entropy and conditional entropy. The entropy of X given that X is a random variable and the probability of *x* is *P(x)* is defined as:

$$H(X) = -\sum_{i} P(x_i) log_2(P(x_i)) \quad (3.1)$$

The conditional entropy, also known as the conditional uncertainty of X after given the values of an attribute Y is:

$$H(X\backslash Y) = -\sum_{j} P(y_j) \sum_{i} P(x_i\backslash y_j) log_2(P(x_i\backslash y_j)) \quad (3.2)$$

The SU figure of 0 implies that features are totally independent, while an SU amount of 1 signifies that a feature can totally predict the value of another feature.

According to (Yu & Liu 2003: 1-11), an attribute which has a certain degree of correlation with a concept, for example, a class may also have the same or even higher degree of correlation to other concepts. Thus, the attribute and the target concept are correlated at a level that is greater than a specific threshold δ and therefore causing this attribute to be significant to the class concept. This correlation is by no means predominant or significant in determining the target concept. The concept of predominant correlation is defined as follows (Singh, Kushwaha & Vyas 2014:95-105);

Definition 1 – Predominant correlation

The correlation between a feature $F_i (F_i \in S)$ and the class *C* is predominant

$iff\ SU_{i,c}\ \geq\ \delta$, and $\forall\ F_j\ \in\ S'(j \neq)$, there exists no F_j such that $(SU_{j,i} \geq SU_{i,c})$.

Definition 2 – Predominant Feature

A feature is predominant to the class; iff its correlation to the class is predominant or can become predominant after eliminating other attributes from the class.

As stated by the preceding explanations, an attribute is *good* if it is *predominant* in the prediction of the class concept. Selecting attributes by classifying them is a procedure that recognises all attributes that are predominant to the class and eliminates non-key features.

The following three heuristics can efficiently recognise predominant attributes and eliminate redundancy from all significant or relevant attributes, with no need to test for peer redundancy for each attribute in S', and thereby avoiding investigations of correlations between pairs of all significant attributes. If two redundant attributes are recognised, eliminating one of them that is less significant to the class results in the retainment of weightier information for the class prediction whilst decreasing redundant attributes.

Heuristic 1

SP_i is the set all redundant peers to F_i

$if\ (S_{Pi}^{+} = \emptyset)$ consider F_i as a predominant feature, eliminate all attributes in $Sp_i -$, and skip identifying redundant peers for them(Yu & Liu 2003).

Heuristic 2

$if\ (S_{Pi}^{+} \neq \emptyset)$ action all attributes in S_{Pi}^{+}prior to deciding on F_i. If none of them becomes predominant, follow Heuristic 1, else only eliminate F_i and decide if or not to eliminate any attributes in S_{Pi}^{-} based on other attributes in S'.

Heuristic 3 *(starting point)*.

The attribute with the biggest $SU_{i,c}$ value is always a predominant attribute and can be used to eliminate other attributes.

3.3.3.1 Fast correlation-based filter

The FCBF is an algorithm that selects good features based on **predominant** correlation, and then presents a fast algorithm with less than quadratic time complexity (Yu & Liu 2003). The algorithm applies the predominance concept. The FCBF uses SU as a correlation measure (Wu et al. 2006). It is composed of two sections, and the first section selects relevant attributes. An attribute *p* is significant to the target attribute *C iff SU (p,c)* ≥ δ given that δ is a predefined threshold.

In the second section, redundant features are selected from the relevant ones, according to its redundancy definition: a feature *q* is said to be redundant *iff p* is a predominant feature, *SU(p,c)* > *SU(q,c)* and *SU(p,q)* ≥ *SU(q,c).*The inequalities imply that *p* is a better predictor of class *c* and that *q* is more similar to *p* than to *c* (Yang et al. 2016).

The steps of identifying redundant features consist of: (1) choosing a predominant attribute, (2) removing all attributes for which it forms an approximate Markov blanket, and (3) iterate steps (1) and (2) until no more predominate attributes can be found. An optimal feature subset can therefore be approximated by a set of predominant features without redundancy.

3.3.3.2 FastCR

The proposed FAST Correlation-based Redundancy elimination (FastCR) algorithm is based on the MIC and FCBF method and is implemented in Java. The original FCBF code selects significant features and removes redundant attributes. The FastCR algorithm for this research removes redundant attributes. Relevant attributes are selected using the MIC algorithm, resulting in a hybrid MICFastCR algorithm.

In lines 2-4, the algorithm calculates the MIC values for all the features and saves them in a list. In the second part (line 6-20), redundant features are removed if the SU values of two features are the same, Table 3.2.

As stated in Heuristic 1, a feature $F_{p,}$ which has been ascertained to be a predominant attribute, can constantly be used to eliminate other attributes that are graded lower than F_p and have F_p as one of its redundant peers (Yu & Liu 2003:1-11).

The loop begins from the first element (Heuristic 3) inS'_{list} (line 7) and runs as detailed below:

Considering all the prevailing attributes (from the one right next to F_p to the last one inS'_{list}), if F_p turns out to be a redundant peer to a feature F_q, F_q will be eliminated from S_{list} list (Heuristic 2). After selecting attributes for one cycle subject to F_p,, the algorithm will utilise the feature that currently remains, is beside F_p as the new reference (line 19) to reiterate the selection. The algorithm ends when there are no more attributes that can be eliminated from S_{list}list.

Table 3.2 MIC and FastCR Algorithm (Zhao, Deng & Shi 2013:70-79; Yu & Liu 2003:1-8)

Input: $S(F_1, F_2, \dots, F_N, C)$ // training dataset δ //a predefined threshold **Output:**S_{best} //an optimal subset	
1	**Begin**
2	for all $f_i,\ f_j \in D, i \neq j$ do begin
3	Calculate MIC values and Set $M_{i,j} = MIC(f_i, f_j)$;
4	end for
5	Sort distinct values of $M_{i,j}$ in descending order as S'_{list};
3	calculate $SU_{i,c}\ for\ f_i$;

7	$F_p = getFirstElement(S'_{list});$
8	do begin
9	$F_q = getNextElement(S'_{list}, F_p);$
11	do begin
12	$F'_q = F_q;$
13	$if\ (SU_{p,q} = 1)$ // if features are identical
14	remove $F_q\ from\ S'_{list};$
15	$F_q = getNextElement(S'_{list}, F'_q);$
16	$else\ F_q = getNextElement(S'_{list}, F_q);$
17	$end\ until\ (F_q == NULL);$
18	$F_p = getNextElement(S'_{list}, F_p);$
19	$end\ until\ (F_q == NULL);$
20	$S_{best} = S'_{list};$
21	**end;**

3.3.4 Machine learning algorithms

RQ4: Are the data-mining techniques consistently effective in predicting defects?

In this research, machine learning techniques were applied in WEKA in predicting defects, (see Appendix). These were the PART, Naïve Bayes and J48 algorithms.

3.3.4.1 Rule based algorithms

The rule-based classification algorithms, which include, OneR, JRip, ZeroR and PART approaches could deliver a valued innovative method, improving current methods, when analysing association data. These approaches have the ability to analyse both categorical and continuous values. The

results of the analysis are easy to interpret. The rule-based learner method could create testable hypotheses for further evaluation. Furthermore, as computing using these algorithms is inexpensive, the algorithms may be utilised in the selection of variables to be used in methods that are intricate and involve many computations. Regardless of being used separately or in combination with other methods, rule-based classifiers are vital in the analysis of complicated association data (Lehr, Yuan, Zeumer, Jayadev & 2011: 1-14).

RIPPER algorithm

The Repeated Incremental Pruning to Produce Error Reduction (RIPPER) method directly derives rules from the data. The algorithm is regarded to be more effective compared to decision trees on big data containing noise (Thangaraj & Vijayalakshmi 2013:1-7). A new rule linked to a class value will cover several attributes of that class, (i.e. attribute values are used to create rule conditions).

The algorithm advances over four stages:

(a) rule growing,

(b) rule pruning,

(c) optimisation,

(d) selection.

In the rule growing stage, attributes are added to create a rule until the rule encounters a discontinue after having met a condition. In the pruning stage, each rule is gradually pruned, permitting the pruning of any final order of the variables, until a pruning metric is achieved. In the optimisation phase each rule which is created is further optimised by (a) greedily adding variables to the original rule and (b) by independently growing a new rule undertaking a growth and pruning phase. Lastly, in the selection stage, the best rules are retained, while the rest of the rules are removed.

RIDOR algorithm

The Ripple Down Rule (RIDOR) learner is also a direct method. Exceptions with the least error rate are identified using an incremental reduced error pruning. The “best” exceptions for each exception are created and iterated until pure (Veeralakshmi 2015: 79; Thangaraj & Vijayalakshmi 2013: 1-7). The rules created look like a tree, where each rule has exceptions that successively have exceptions. Thus, an expansion that resembles a tree expansion of exceptions is produced. Exceptions are composed of rules that predict classes other than the default (Veeralakshmi 2015:79-85).

PART algorithm

The Partial Decision Tree (PART) is an indirect method for rule generation. PART generates a pruned decision tree using the C4.5 statistical classifier and the RIPPER. A partial tree is built from a complete training data set (Salih, Salih & Abraham 2014: 41:51).

The sub tree replacement as a pruning strategy is used to build the partial tree. The algorithm expands the nodes in accordance with the minimum entropy until a node whose children are all leaves is located. Then, the pruning process starts.

Sub tree replacement analyses if the node can be replaced by one of its leaf children and perform better. The algorithm then applies the separate-and-conquer strategy.

In this research study, the PART rule-based classifier was one of the ML methods used.

Classification rules

“If…then…” Rules

(Wings=Yes) ∧ (Blind=Yes) →Bat

(Income <R5K) ∧ (Family Size=Medium) → Loan=Yes

Rule: (Condition) → y

where

Condition is a conjunction of attribute tests

y is the class label

LHS: rule antecedent or condition

RHS: rule consequent

Rule-based classifier example

A rule *r* covers an instance *i* if the attributes of the instance satisfy the condition (LHS) of the rule.

Rule One: (Two legs = no) ∧ (Eats grass = yes) →Cow

Rule Two: (Four legs = no) ∧ (Living in water = yes) →Fish

Rule Three: (Two legs = yes) ∧ (Has wings = yes) →Bird

Rule Four: (Eat meat = sometimes) ∧ →Humans

Rule Five: (Four legs = yes) ∧ (Has pouch = no) →Kangaroo

3.3.4.2 Tree-based classifiers

In this study, the J48 tree was one of the three classifiers that were used in defect prediction. The J48 is WEKA variation of the C4.5, which is a standard decision tree learning classifier proposed by (Quinlan 1986:81-106).

3.3.4.2.1 C4.5 Algorithm and J48

The C4.5 is commonly used for inductive learning. It extends (improves) the ID3 by considering continuous and discrete variables, missing attribute values and prunes a tree after its creation (Setsirichok, Piroonratana, Wongseree & Usavanarong 2012: 202-212). The C4.5 decision tree is a supervised learning algorithm and uses training and test examples. It uses the concept of information entropy. Entropy determines how informative a certain input attribute is, concerning an output for a subset of the training data. It was proposed by (Shannon 1948:379-423) as a measure of uncertainty in communication systems. Entropy is vital in modern information theory.

The most informative feature is chosen as the parent node. A child of the parent node is formed for either each probable value of this variable if it is a discrete-valued attribute or each likely discretised

interval of this variable if it is a continuous-valued variable. These training samples are then sorted to the relevant successor node (Setsirichok et al. 2012:202-212).

The procedure iterates and utilises training data linked with each child node in choosing the most suitable attribute to test. The greedy search, in which the classifier does not backtrack to re-examine previous node selections, is used to build a tree (Johansson & Niklasson 2010). Even if there is a possibility to create an additional new node to the tree, until all samples that are allocated to one node are members of the same class, the tree is not permitted to grow to its maximum depth. A node is only added to the tree if there are adequate samples remaining after sorting. After the full tree is created, tree pruning is conducted to prevent data over-fitting. The decision tree approach is most suitable for classification problems. Using this method, a tree is constructed to model the classification procedure (Setsirichok et al. 2012: 202-212).

The Information Gain (IG) ratio, *GainRatio(A, S)* of feature *F* relative to the sample set *S* is described as;

$$Gain\ Ratio(F,S) = \frac{Gain(F,S)}{SplitInformation(F,S)} \tag{3.3}$$

where entropy is

$$Entropy(S) = \sum_{i=1}^{n} - Pr(Ci) * log_2\ Pr(Ci) \tag{3.4}$$

and

$$G(S,F) = E(S) - \sum_{i=1}^{m} \Pr(F_i)\, E(S_{F_i}) \tag{3.5}$$

where *E(S)* is the information entropy of *S, G(S,F)* is the gain of S after a split on variable F, *Pr(Ci)* is the frequency of class *Ci* in *S, n* is the amount of classes in *S,m* is the amount of values of attribute *F* in *S, Pr(Fi)*is the frequency of cases that have *Fi* value in *S, E(Fi)* is the subset of *S* with items that have *Fi value.*

The IG ratio can be computed for discrete-valued variables. On the other hand, continuous-valued attributes must be discretised before the Information GainRatio calculation.

3.3.4.2.2 Tree Pruning

Overfitting

Pruning is conducted to avoid overfitting. The basic approaches of decision tree pruning are pre-pruning and post-pruning.

Pre- pruning

Pruning can be applied during tree creation. During top-down construction, if there is no longer adequate data, the creation of the tree discontinues. Tree creation may also end when the attributes become inappropriate, i.e. wrong values of attributes. The technique is faster, but difficult to perform.

Post-pruning

The full tree is grown and then sub-trees that are not useful are removed. Some branches are removed by either using sub-tree raising or sub-tree replacement.

1. Reduced-error pruning

A sub-tree at each node within the tree is replaced with a leaf, passing on all observations in the new leaf to the majority class (for classification problems) or assigning them the mean (for regression problems). If the replacement of this sub-tree with a leaf does not affect the overall error/cost, it is retained and else it is not added. The iteration continues until the pruning is no longer beneficial.

2. Cost-complexity pruning

The method continuously collapses the node which, creates the least per-node rise in the error/cost, while at the same time weighing the overall complexity of the tree. A decision is taken on the best pruned tree that minimises the cost-complexity function.

A tree T_{max} that overfits data has a misclassification cost $R(T)$ and numerous leaves. Another tree with fewer leaves must be created at the cost of letting $R(T)$ to rise somewhat. There must be a balance between the number of leaves and the misclassification cost. The complexity of a tree T is the quantity of its terminal nodes $|\tilde{T}|$

The *cost-complexity measure* is:

$$R_\alpha(T) = R(T) + |\tilde{T}| \qquad (3.6)$$

where $\alpha > 0$is the *complexity parameter.* We find trees that minimise R_α by pruning T_{max}

3. Pessimistic pruning

A penalty term is added to the error at each node. This penalty term is often referred to as an "error correction," with the motivation that the true error at each node must be conservatively estimated.

3.3.4.3 Rule sets vs decision trees

Rule learning is valuable. Decision trees are commonly quite complicated and difficult to understand. Quinlan (1993) has noted that even pruned decision trees may be too bulky, complicated and unreadable to provide understanding into the domain at hand and has thus invented methods for simplifying decision trees into pruned production rule sets. Supporting confirmation for this comes from Rivest (1987:229-246), who proves that decision lists (ordered rule sets) with at most *k* conditions per rule are more communicative than decision trees of depth *k*.

There is a limit of decision tree classifiers to non-overlapping rules which causes strong controls on learnable rules. This has resulted in the replicated sub-tree problem (Pagallo & Haussler 1990:71-99). Identical sub-trees must be learned at a number of positions in a decision tree, due to the fragmentation of the example space forced by the restriction to non-overlapping rules. Rule learning

does not form such a limit and is therefore less susceptible to this obstacle. An illustration for this problem has been given by Cendrowska (1987:349-370), who revealed that the minimal decision tree for the concept *x* is described as:

IF A = 3 AND B = 3 THEN Class = x

IF C = 3 AND D = 3 THEN Class = x

The tree has 10 interior nodes and 21 leaves supposing that each attribute A...D can be instantiated with three different values. Lastly, propositional rule learning algorithms spread out naturally to the frame work of inductive logic programming framework, where the goal is basically the induction of a rule set in first-order logic, (e.g., in the form of a Prolog program). First-order background knowledge can also be used for decision tree but once more, Watanabe and Rendell (1991:770-776) have noted that first-order decision trees are usually more complicated than first-order rules.

3.3.4.4 Naive Bayes

According to Abraham & Simha (2007:44-49), the Naïve Bayes is a classifier that is founded on the Bayesian networks theory and uses probability for predicting the class an instance is associated with, given the set of features defining the instance (Singh & Verma 2012:323-327). Features are considered to contribute independently to the probability, regardless of correlations between them. The classifier learns from the training data, which parameters are suitable for the classification task. The Bayes rule joins the prior probability of every variable and the likelihood to create a highest posterior probability that is used to predict a class. The classifier constructs the posterior probability for the class c_j among a set of possible classes in *C* (Novakovic, Strbac & Bulatovic 2011: 119-135);

$$f(X_i) = \prod_{j=1}^{N} P(xj \mid cj)P(ci) \qquad (3.7)$$

Where $X = x_1, x_2, \dots, x_n)$ represents a set of feature values, i.e. *x1* is the value of feature *X* and *cj , j = 1, 2, ..., N,* are the potential labels of the class. *P(xj\cj)* are conditional probabilities and *nP(ci)* are prior probabilities.

PART

This is a rule-based method that combines C4.5 and RIPPER algorithms to create ordered set of rules (Shafiullah, Ali, Thompson & Wolfs 2010: 1-5). The method is also known as a partial decision tree algorithm and builds a partial decision trees that converts them into a corresponding decision rules. A rule is created from a leaf with the biggest coverage (Lehr et al. 2011).

RQ3: How can a model that will predict defects in the next versions of the software applications be derived?

The Equinox, Mylyn, PDE, JDT and Lucene are evolving product lines and the study will involve various versions of the software. A literature review of defect prediction of software product lines was conducted. Information about previous versions of the product lines will assist in the prediction of failure-prone files. The selected relevant attributes will improve the performance of the machine learning algorithms. The new feature selection model and the traditional ones were run through Python, R and WEKA.

3.3.5 Applications

3.2.5.1 Feature ranking

Feature selection packages were integrated in R and the code was written in R to weigh and rank features according to their importance.

3.2.5.2 Machine learning

In this study, machine learning algorithms were used to predict software defects. WEKA is a Java-based open source machine learning system that was designed by researchers at the University of Waikato in New Zealand. It holds machine-learning algorithms used in data mining. Routines are implemented as classes and logically arranged as packages. A GUI interface or command line is used. Calculations using the WEKA data-mining classifiers can be used on a dataset or run from a Java application. The WEKA tools are for data pre-processing, classification, regression, clustering, association rules and visualisation. They can also be used to create new ML algorithms.

WEKA Data Format

WEKA uses flat files. The default file type is Attribute Relation File Format (ARFF). Data can be imported from various file types including CSV and ARFF. Data can also be read from a website or from a database.

3.3.6 Defect prediction stages

The stages of defect prediction are data pre-processing, feature extraction, classification and data post-processing, see Figure 3.3.

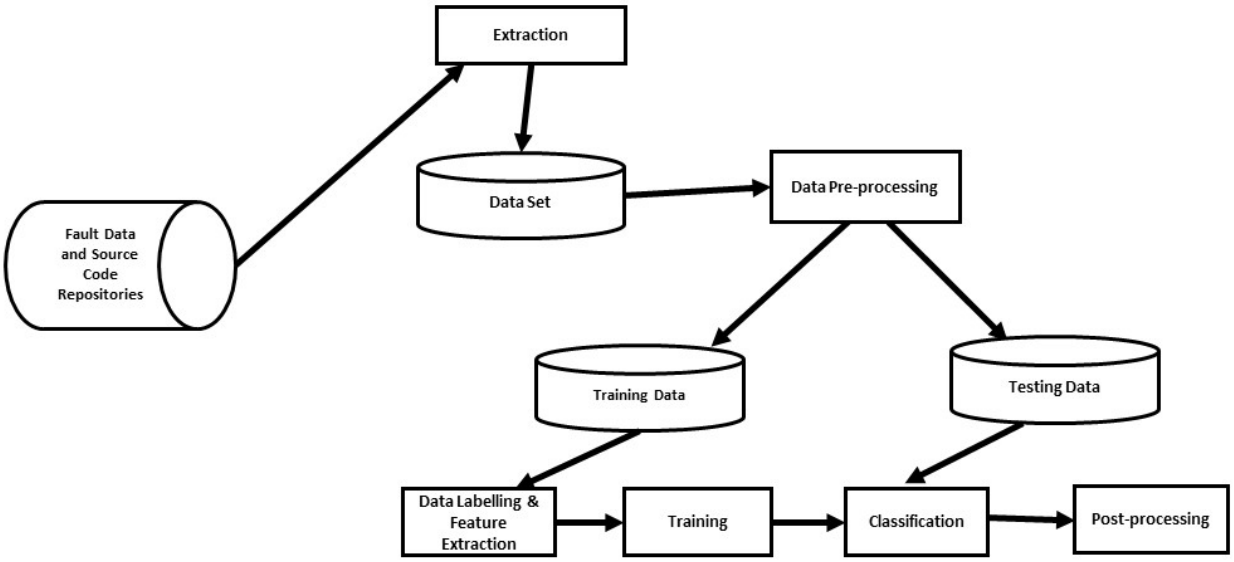


Figure 3.3 Defect Prediction Process

3.3.6.1 Data pre-processing

Pre-processing tools in WEKA are known as filters. The uses of WEKA filters include discretisation, sampling, feature selection, transforming and the joining of attributes, see Figure 3.4. Pre-processing turns data into a form that improves the classification algorithm. Pre-processing may include data normalisation and the filling in of missing values.

Normalisation in machine learning is a data pre-processing method that scales feature values to fall within a specified range. Normalisation is normally applied in the classification procedures that involve distance measures (Tiwari & Singh 2010: 28-34). Normalisation techniques include the Min-Max Normalization, Decimal Scaling and the Standard Deviation Method.

A metric *m* is normalised as follows:

$$m_z(i, c_j, V_k, P) = \frac{m\big(i, c_j, V_k, P\big) - \mu(i, V_k, P)}{\sigma(i, V_k, P)} \quad (3.8)$$

where $m\big(i, c_j, V_k, P\big)$ is the value of the i^{th} metric. This normalisation is applied on data of both the training and testing versions during the software defect prediction process.

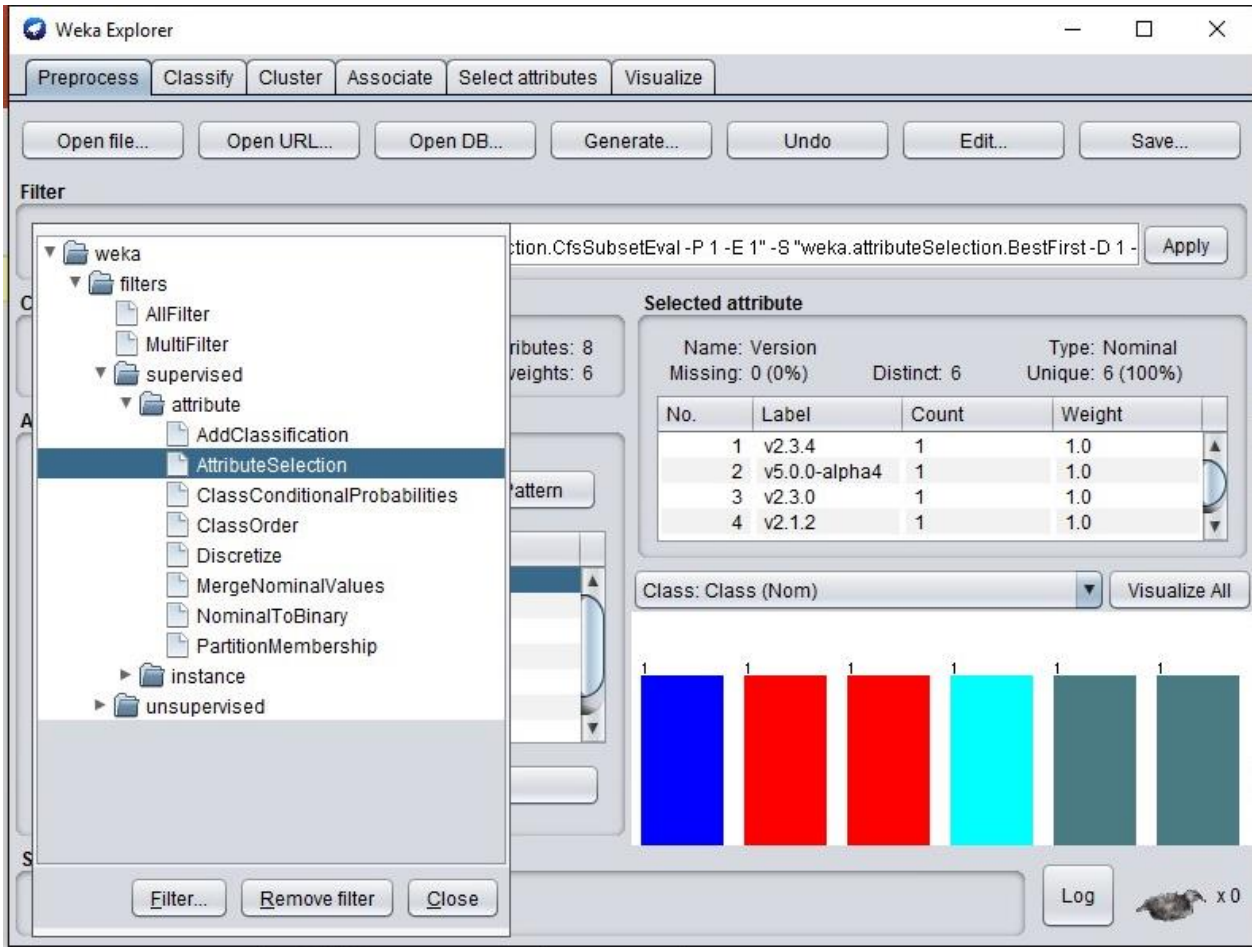


Figure 3.4 Data Pre-processing in WEKA

3.3.6.2 Feature extraction

This is a method that converts pre-processed data into a structure that can be used by the pattern recognition machine. A form is created that is optimised to the machine-learning algorithm that will be used. Numeric attributes may be discretised into nominal attributes, depending on the class information, using MDL methods. Some learning methods can only process nominal data, (e.g. the weka.classifiers.rules). Prism. *Nominal to Binary* encodes all nominal attributes into binary.

Perceptrons, are a form of neural networks and provide binary 0 or 1 as an output and thus require binary as input for training together with real-valued vectors.

3.3.6.3 WEKA Prediction

WEKA classifiers are prediction nominal or numeric quantities. The implemented machine-learning methods include Decision Trees, Support Vector Machines, Regression and Naïve Bayes. Meta classifiers include Bagging, Boosting, Stacking and Weighted Learning. The diverse classifiers have different strengths and weaknesses that may be suitable for specific needs.

A classification algorithm is assessed on its data prediction accuracy. A model that is output is built from all the training data. The classification model is output so that it can be viewed. The statistics for each class is returned. The entropy-evaluation measures are included in the results. The WEKA attributes are used in the prediction of a class variable.

3.3.6.4 Post processing

Developers would expect to view the results of identified particular defects from the source files at a particular line and at a column number. The reason why that is a defect may be reported. The *Percentage of Correctness* of prediction among the test sets measures the accuracy of the classification algorithms.

3.3.6.4.1 Hold-out method

The dataset is separated using boots trap into the training set and the testing set (Untan, Hadihardaja, Cahyono & Soekarno 2014: 228-233; Pushphavathi et al. 2014: 1-5). The proportion between the training and the testing data is not binding, but to ensure that the variant between the models is not too wide, 2/3 of the data is generally used for the training and the other 1/3 is used for testing. The training set is applied in testing the model. The test set measures the error rate of the trained classification algorithm. The drawbacks of the hold-out method are that in cases of sparse data, dataset to be set aside for testing may not be available. Considering that the training and testing are executed only once, the hold-out method will be distorted if the split is poorly executed.

3.3.6.4.2 Random subsampling

This method performs *s* data splits of the dataset. In each data split the classification algorithm is retrained with the training dataset and the error rate is calculated using the testing dataset. The error

rate is computed as the average of the separate error estimates in from the splits (Zhang & Yang 2015: 95-112).

$$E = \frac{1}{s}\sum_{i=1}^{s} E_i \qquad (3.9)$$

3.3.6.4.3 k-Fold cross validation

This validation method is also known as rotation estimation (Untan et al. 2014: 228-233) and is similar to random sampling, except that all its subsets are used for both testing and training. It reduces the bias linked to the random sampling of data samples used in analysing the prediction accuracy of two or more techniques. Part of the data is eliminated before the training starts. After the training is complete, the removed data can be utilised in testing the prediction capability of the learned model on "new" data. The cross-validation procedure randomly splits the dataset into *k* disjoint subsets, with each fold comprising almost the same number of records.

In this study, the experiments were conducted according to the 10-fold cross-validation approach.

3.3.6.4.4 Leave-one-out

This is method is a special type of the k fold-cross validation. The data D of size l is split into l subdivisions of size1.

$$D = Q_1 \, UQ_2 \, U..Q_{l-1}UQ_{l,} \qquad (3.10)$$

and

$$Q_i \cap Q_j = \emptyset \qquad (3.11)$$

where $Q_i = \{(x_i, y_{i)\}}$ and $Q_i = \{\left(x_j, y_j\right)\}$ for *i, j*=1 and $i \neq j$

Each part Q_i is used for testing, while the leftover parts are used for training. The number of folds is the same as the number of instances (Wong 2015: 2839-2846). The average error is calculated and

used to assess the model. The Leave-One-Out cross validation can be computationally expensive, because it generally requires one to construct many models, equal in number to the amount of training data.

3.2.3.4 Bootstrap

A bootstrap is a general resampling plan (Efron 1979:1-26). The sample contains *n* samples randomly drawn with replacement from the original dataset. Certain samples will be drawn numerous times, whereas others will not be sampled at all. A learner is created on the bootstrap sample and tested on samples that were not chosen. The left out samples are known as Out-Of-Bag samples. The representation of each model on its left out samples that are averaged can deliver an approximate accuracy of the bagged models. This projected performance is normally called the OOB estimate of performance.

There are some common variants of the method such as balanced bootstrap or 0.632 bootstrap (Efron & Tibshirani 1993).

Performance evaluation

The accuracy of classifiers is the percentage of correctness of prediction among the test sets.

Sensitivity provides the performance of a binary classification test. The output results may be:

True Positive (TP) rate is the percentage of instances which were categorised as class *k*, out of all instances which actually belong to class *k*. This measure is identical to Recall (WEKA 2016:1-7).

False Positive (FP) rate is the percentage of examples categorised as class *k*, but are members of another class, out of all examples that do not belong to class *k*.

TN = true negatives. The number accurately predicted as negative

FN = false negatives. The number inaccurately predicted as negative

Recall represents the TP rate, (i.e. all the defective modules that the classifier can locate), True positives / Actual positives. It is defined as;

$$Recall = \frac{tp}{tp+fn} \quad (3.12)$$

Precision is TP /positively predicted. It evaluates the number that was predicted to be defect prone and turned out to be defective. It is represented by;

$$Precision = \frac{tp}{tp+fp} \quad (3.13)$$

The minimum and maximum values of Recall and Precision are 0 and 1 respectively and greater values demonstrate improved prediction accuracy. In the ideal scenario, both Recall and Precision are equivalent to 1, which implies the prediction algorithm locates all modules that are susceptible to defects, without False Negative or False Positive. Recall and Precision values are normally mutually exclusive, (i.e., a high Recall value usually has a low Precision value). Attaining both high Recall and Precision simultaneously is unlikely.

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn} \quad (3.14)$$

The F–Measure is the harmonic mean of precision and recall. Precision and recall are equally weighted.

$$F = 2.\frac{precision.recall}{precision+recall} \quad (3.15)$$

Mean absolute error

The Mean Absolute Error (MAE) is a useful tool for model evaluations. This measure calculates the average magnitude of the errors (difference between the estimated and the real value). This calculates the accuracy of variables and measures of the differences between the percentage

prediction and the actual observation. The MAE value nearer to zero is regarded as having the superior prediction ability. MAE is described as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|\ \hat{mi} - mi\ |) \quad (3.16)$$

Given that n is the amount of tests, ^mi is the value from the prediction test and mi is the observation value. The Area Under the ROC Curve (AUC) evaluates the level of discrimination realised by the model.

Root mean-squared error (RMSE)

The RMSE It is a quadratic scoring rule which measures the differences between the prediction values produced by the prediction models and the actual observed values. A lower value of RMSE produces a better goodness of fit.

$$RSME = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (3.17)$$

The error due to bias is the average prediction error. Variation is the standard deviation of prediction error. The Area Under the ROC Curve (AUC) assesses the level of discrimination realised by the model. The value of AUC ranges from 0 to 1 and random prediction has AUC of 0.5. The advantage of AUC is that it is insensitive to decision threshold like precision and recall.

3.4 Chapter Summary

The development of a feature selection-based software defect prediction model was discussed in this chapter. The research experiment and properties of the data used in this empirical study were

presented in detail. The types of data dimension reduction techniques were explained. Machine-learning techniques predict defective classes, using defect data. Performance measures and their applicability were explored. The next chapter will discuss the information theory concept and its measures.

CHAPTER 4

INFORMATION THEORY

4.1 Introduction

Information theory and the entropy concept are presented in this chapter. The entropy concept measures the amount of information in an event or signal. Measures from information theory are discussed. This chapter also lays out the data pre-processing techniques which include replacing missing values, removing redundant data, handling conflicting data and selecting features.

4.2 Shannon's entropy and information theory

The theory was presented by Claude Shannon in 1948 and initially applied in communication systems to obtain in-depth information about data compression and transmission rate. It has been subsequently implemented in other several technology fields, including machine learning (Bettenburg & Hassan 2013: 375-431).

These concepts provide guidance on the efficient compression of a data source before communicating or storing it. The recipient must be able to recover data that is not distorted. Shannon's Communication System is displayed in Figure 4.1 (Shannon 1948: 379-423).

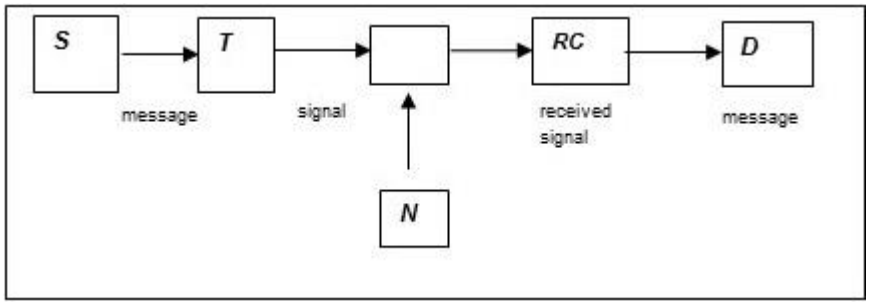


Figure 4.1 Shannon's Communication System

S is the information source and produces the information that is to be received at the destination.

T is the Transmitter that transforms the information at the source into a signal

N is the noise: the average amount of information received at *D* but not generated at *S*

RC is the receiver that recreates the message from the signal

D is the destination

A message is regarded as a series of characters from an alphabet. The *Source coding* is a procedure that captures each character from the source data and links it with a codeword. The mapping between input symbols and codewords is called a code. Quantification of “information” concerning an event should be influenced by the probability of the event. The smaller the probability of an event, the bigger the information associated with knowing that the event has occurred.

Definition of information

This insight was applied by Hartley (1928:535-563), who introduced the following definition of information connected with an event, whose probability of occurrence is p:

$$I \equiv \log\left(\frac{1}{p}\right) = -\log(p) \qquad (4.1)$$

given that p is the probability of an event

Information theory is used to evaluate and describe the quantity of information in a message. The theory measures uncertainty that is associated with information (Hassan 2009:78-88). In 1948 Shannon’s entropy that is based on information theory was proposed to measure the uncertainty of random variables (Liu, Lin, Lin, Wu & Zhang 2017: 11-22). The entropy of a set *Y* is defined as (Rana, Awais & Shamail 2014: 637-648):

$$Entropy\ (Y) = \sum_{i=1}^{n} -p_i log_2 p_i \qquad (4.2)$$

given that n is the quantity of classes and p_i is the percentage of samples of class i. The measurement is in bits of information. The entropy of the destination D is described as the average quantity of information that reaches the destination, see Figure 4.2. (Ellerman 2009:199-149).

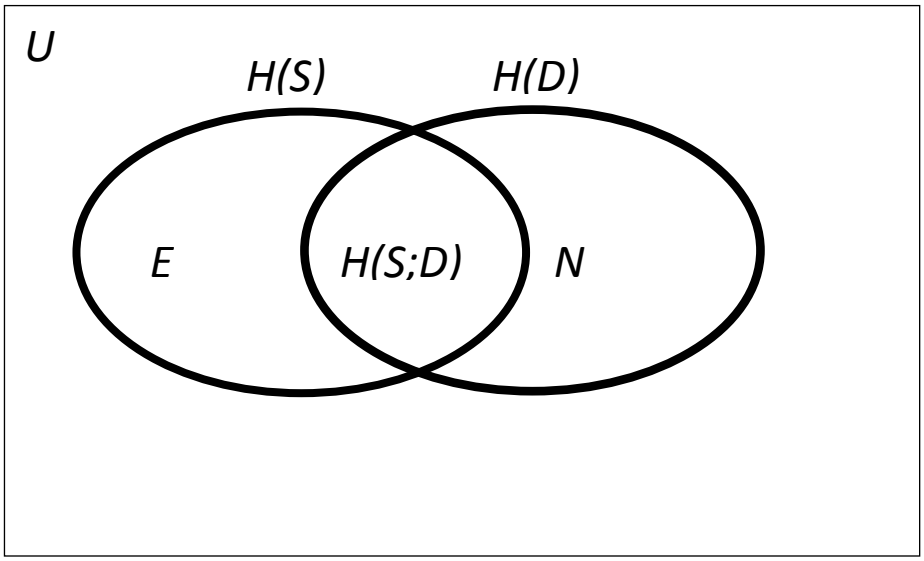


Figure 4.2. Relationship between the Source H(S) entropy and the Destination H(D) entropy.

Mutual Information H(S;D)

This is the average information created by the sender that reaches the receiver.

(a) Equivocation E

Information that is lost during transmission

(b) Noise - N is the noise

The figure evidently shows that mutual information can be computed as:

$$H(S;D) = H(S) - E = H(D) - N \qquad (4.3)$$

4.3 Information theory measures

Information theory is applied in assessing and defining the amount of information in a message. The types of theoretic measures include:

4.3.1 Information gain

A measure that satisfies a constraint is:

$$I(X) = -log_2(p) \quad (4.4)$$

Given that *p* is the probability of an event X. This is measured in bits of information. Information Gain (IG) is the degree of variation between two probability distributions. The entropy (H) of a random attribute is a degree of its uncertainty. An entropy for a random attribute X with N outcomes is described by;

$$H(X) = -\sum_{i=1}^{n} p(xi) log_b p(xi) \quad (4.5)$$

The IG of a variable X in a set $S$is described as (Rana et al. 2014):

$$IG(S, X) = Entropy(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} Entropy S_i \quad (4.6)$$

Where k is quantity of distinct values in attribute X and S_i is a set of examples that contain a specific value from domain of X.

IG is a symmetrical measure:

$$IG = H(X\backslash Y) - H(X) - H(X\backslash Y) \quad (4.7)$$

The IG $IG(X\backslash Y)$computes the extent by which the entropy of $X$is lessened when the values of $Y$are provided. The disadvantage of the algorithm is that it prefers attributes with the most values.

4.3.2 Gain ratio

The formula calculates the value of a variable by evaluating the gain ratio with regard to the class. The Gain Ratio mitigates the bias of the IG (Novakovic, Strbac & Bulatovic 2011: 119-135). The Gain Ratio is (Wang, Khoshgoftaar, Wald & Napolitano 2012: 301-307):

$$Gain\ Ratio(S,X) = \frac{Gain(S,X)}{SplitInfo(S,X)} \tag{4.8}$$

where the split information is:

$$SplitInfo(S,X) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|} \tag{4.9}$$

where $S_i$is c sample sub-sets, c values of variable X are used. Split information is the entropy of S on all values of variable X.

4.3.3 Mutual information

The Mutual Information (MI) measures the decrease of uncertainty about attribute X after observing Y. The MI is beneficial in selecting features since it provides a way to measure the significance of a feature subset regarding the output vector C. The joint entropy $H(X,Y)$ of two attributes X and Y is:

$$H(X,Y) = -\sum\sum p(x,y) log\ p(x,y) \tag{4.10}$$

Conditional entropy evaluates the uncertainty of an attribute, if the other one is known. Given the values of Y, the conditional entropy $H(X|Y)$of X with regard to Y is (Liu, Wu & Zhang 2011:979-984);

$$H(X\backslash Y) = \sum\sum p(x,y) log\ p(x\backslash y) \tag{4.11}$$

The conditional entropy is 0 when X is fully dependent on Y. This implies that no additional data is needed to define X when Y is provided. On the other hand,

$$H(X\backslash Y) = H(X) \qquad (4.12)$$

if they are independent with each other.

The Venn diagram in Figure 4.3 shows the relationships described in Equation 12.

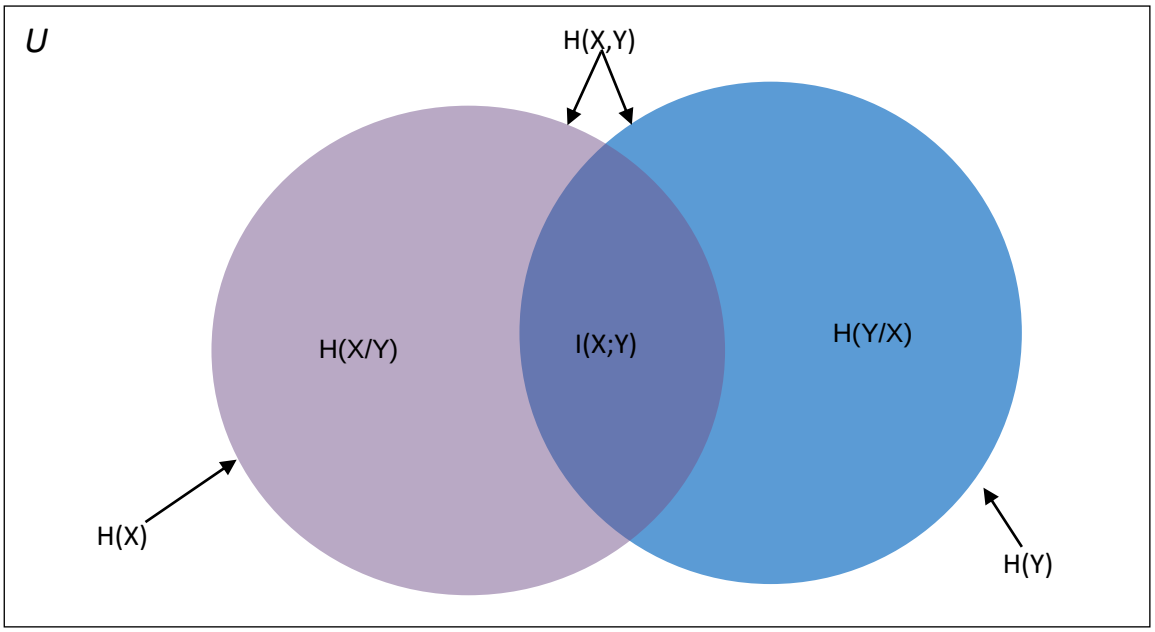


Figure 4.3. Venn diagram depicting relations between MI and entropies

MI is described as the amount of information a random attribute communicates about another. Two relevant attributes have a higher MI and *I(X;Y)=0* implies that the attributes are statistically independent and irrelevant to each other. Since MI is calculated over joint and marginal pdfs of the variables and does not utilise statistics of any grade or order, it can be used to evaluate and quantify any type of association between attributes (Kinney & Atwal 2014:21-26).

4.3.4 Symmetrical uncertainty

The SU evaluates the dependencies of features using entropy and conditional entropy. The method calculates relevancy between a feature and a class. SU is described as (Regha & Rani 2015: 135-140);

$$SU(X,Y) = \frac{2X\ Gain(X|Y)}{H(X) + H(Y)} \quad (4.13)$$

$H(X)$ represents the entropy of a discrete random variable X. $Gain(X|Y)$ is described as the Information Gain concerning$Y$when $X$is given. The SI measure deals with the IG bias by dividing it with the total of X and Y and confining the SU values to fall between 0 and 1 (Novakovic et al. 2011:119-135). The SU figure of 0 implies that features are totally independent while an SU amount of 1 signifies that a feature can totally predict the value of another feature.

4.3.5 Relief

This filter method grades an attribute by its capability to discriminate samples that are derived from different classes, but identical. Relief allocates a relevance score to individual features according to the importance of the feature to the target concept (John, Kohavi & Pfleger 1994: 121-129). In the algorithm below, *m* vectors are randomly selected and attributes are chosen from each vector. Using an arbitrarily chosen attribute $x_i = \{x_{1i}, x_{2i}, \dots, x_{ni},\}$,two closest neighbours are located; the first is derived within the class, the attribute is located and is known as the nearest hit H.

The second is chosen from another class is called the nearest miss, M.(Sanchez-Morono, Alonso-Betanzos & Tombilla-Sanroman 2007: 178-187) and (John, Kohavi & Pfleger 1994: 121-129).

$$W_f \leftarrow W_f - \frac{1}{m}\mathrm{diff}_f(K,A) + \frac{1}{m}\mathrm{diff}_f(K,B) \quad (4.14)$$

where diff_f, for numerical feature f, is the normalised difference between the values of f and for nominal f,it is the truth value of the equality of given values. If a value of an attribute changes and there is a subsequent change in class, the attribute is weighted based on the assumption that the change in attribute value led to the change in the class. On the other hand, if an attribute value changes, but there is no class change, the attribute weight decreases on the assumption that changing the attribute does not affect the class. The estimates of all features are then updated subject to the values of x_i, *A* and *B*.

4.3.6 ReliefF

This was designed to overcome the limitations of the Relief method. The ReliefF filter method is capable of handling multi-classes, noisy and incomplete data. Unlike other filter methods, the ReliefF is less biased.

4.3.7 Minimum redundancy maximum relevancy

The Minimum Redundancy Maximum Relevance (mRMR) feature selection approach introduced by Peng, Long & Ding (2005:1226-1238) identifies the discriminant features of a class (Agarwal & Mittal 2013:13-24). The mRMR technique chooses features that are highly dependent on the class (maximum relevancy) and less dependent among other features (minimum redundancy). Attributes that are greatly significant to the class may be redundant with other attributes. Mutual information measures the dependency between attributes and class attribute and among attributes.

Maximum relevance, represented as $max\, D(X, c)$, is the increase of the significance of an attribute subset X to the class label *c*. Attribute subset relevance is denoted by:

$$D(X, c) = \frac{1}{|X|} \sum_{f_i \in X} \Phi\,(f_i, c) \qquad (4.15)$$

where $\Phi(f_i, c)$ represents the relevance of an attribute f_i to c based on mutual information.If two relevant attributes extremely rely on each other, the class-discriminative power would not be much different if one of them is eliminated. The redundancy of attributes is based on pair-wise attribute

dependence. Minimum redundancy min $R(X)$ is applied in the selection of an attribute subset of mutually-exclusive features. The redundancy of an attribute subset $R(X)$is denoted as:

$$D(X,c) = \frac{1}{|X|^2} \sum_{f_i, f_j \in X} \Phi\,(f_i, f_j) \tag{4.16}$$

mRMR is described as the simple operator max $\Phi(D,R) = D - R$ which optimises D and R simultaneously. Given a feature subset $X_m - 1$of $m - 1$selected features, the task is to select the mth feature that optimises the following criterion:

$$\underset{f_i \in X_{m-1}}{max} \left[\Phi(f_i, c) - \frac{1}{m-1} \sum_{f_i \in X_{m-1}} \Phi\,(f_i, f_j) \right] \tag{4.17}$$

4.3.8 Pearson correlation

Pearson correlation coefficient is utilised in computing the connection among attributes X and Y (Hao, Li, Zhang, Chen & Zhu 2016:635-639).The Pearson formula is described as:

$$\rho\,(X,Y) = \frac{cov(X,Y)}{\sigma X \sigma Y} = \frac{E((X - \mu_x)(Y - \mu_y))}{\sigma X \sigma Y} \tag{4.18}$$

$$\frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E^2(X)}\sqrt{E(Y^2) - E^2(Y)}} \tag{4.19}$$

4.3.9 Maximal information coefficient

Similarity, between features is measured using the relationship coefficient. Pearson coefficient is one of the relationship algorithms, but it can only capture linear relationships and does not possess the superposition property (Zhao, Deng & Shi 2013:70-79). It fails to cater for functional sin or cubic.

The MIC was proposed by Reshef, Reshef, Finucane, Grossman, Mcvean, Turnbaugh, Lander, Mitzenmacher and Sabeti (2011: 1518-1524) and is developed on the basis of mutual information and caters for functional and non-functional associations (Hao, Li, Zhang, Chen & Zhu 2016: 635-639). It is based on the concepts of information theory. It symbolises the non-linear relationship between two variables. Further, it is a real number whose values range from 0 to 1, where 0 represents uncorrelation and 1 represents complete correlated, noiseless functional relationship (Romito 2013:1-6). Mutual information, the dependence between the attributes X and Y is represented by:

$$I\ (X;Y) = H(X) + H(Y) - H(X,Y) \qquad (4.20)$$

Where the joint distribution of variables X and Y is$p = (X,Y)$. The computation of the normalised mutual information value is based on pairs of integers (x,y)on the grid. The highest of the normalised values is known as the MIC.

$$MIC\ (X;Y) = \max_{X,Y_{total}<B} \frac{I(X;Y)}{log_2(\min(X,Y))} \qquad (4.21)$$

The advantages of MIC is that it can explore hidden relationships between variables and reduce noise (Romito 2013: 1-6).

4.4 Chapter summary

Shannon's entropy was discussed in this chapter. Selecting attributes using information theoretic and probabilistic techniques has been conducted in previous research. In this study, attribute selection methods were contrasted with those that are based on information theory. The next chapter will present feature selection algorithms that include the information theoretic algorithms.

CHAPTER 5

FEATURE SELECTION

5.1 Introduction

This chapter discusses the feature selection process, feature relevancy and redundancy. The previous chapter described and compared information theoretic measures. Feature weighting methods and their role in feature selection is explained in the current chapter. This chapter will also discuss feature selection using probabilistic, information theoretic and other methods.

5.2 Feature selection

Feature selection, according to Liu et al. (2011:979-984) is a procedure that is designed to select relevant features for classification and remove redundant ones with respect to the task being learned (Hall 1999:1-178). Novakovic et al. (2011:119-135) alluded that feature space is created by selecting the least set of M features from an initial feature set N, depending on specific assessments which depend on the suitability of the attribute for data reduction (Zhihua & Wenqu 2015:1-17). A feature, also known as an attribute or variable, describes a characteristic of the data (Ladha & Deepa 2011: 1787-1797). The different types of features are discrete, continuous, ordinal or nominal. The attributes can be relevant, irrelevant or redundant. Selecting the best set of relevant attributes improves the accuracy of the classifiers employed in software defect prediction (Khan, Gias, Siddik, Rahman, Khaled & Shoyaib. 2014:1-4). The feature selection process benefits are;

1. The process decreases the dimensionality of the feature space, thus limiting the storage requirements
2. It increases the processing speed of the classification algorithm
3. The machine learning algorithm's performance and prediction accuracy are increased
4. Data quality is improved after the removal of irrelevant and redundant data

5.3 Feature relevance and redundancy

Molina, Belanche & Nebot (2002: 216-227) state that relevant features have an effect on the output and their function cannot be undertaken by any other feature. Relevant features must be associated with the class. They maximise the accuracy of the predictive model (Hewett 2011: 245-257).

5.3.1 Relevant features

In the definitions of relevance, as defined by John et al. (1994:121-129), each instance X is a component of the set $F_1X\ F_2X\ \ldots X\ F_m$, where $F_i$is the domain of the ith attribute. The label or output is Y. The value of feature *X* $_i$is denoted by x_i. A probability measure *p* is on the space $F_1X\ F_2X\ \ldots XF_mX\ Y$.

Definition 1

Xi is relevant iff there exists some x_i, and y for which $p(X_i = x_i) > 0$ such that

$$p(Y = y \setminus Xi = xi) \neq p(Y = y) \qquad (5.1)$$

Definition 2

X_i is relevant iff there exists some $x_i, y$and $s_i$for which $p(X_i = x_i) > 0$ such that

$$p(Y = y, Si = si \setminus Xi = xi) \neq p(Y = y, Si = si) \qquad (5.2)$$

The first definition falls short in defining the significance of attributes on the parity concept and may be amended. Let Si be the set of all features, except Xi and si be the value assigned to all features in Si.

Definition 3

Xi is relevant iff there exists some x_i, y and si for which $p(Xi = xi, Si = si) > 0$ such that

$$p(Y = y \setminus Xi = xi, Si = si) \neq p(Y = y \setminus Si = si) \quad (5.3)$$

This implies that Xi is significant if knowing its value can change Y, therefore Y is conditionally dependent on Xi.

Definition 4 MIC relevancy

$$S = \{F_i \setminus MIC(F_i, C) > t\}, \quad (5.4)$$

where t is a predetermined irrelevancy threshold.

Definition 5 MIC redundancy

F_i is redundant if there exists another feature F_j, such that

$$MIC(F_j, C) > MIC(F_{i,} C) \quad (5.5)$$

and

$$MIC(F_j, F_i) > MIC(F_{i,} C) \quad (5.6)$$

Definition 6 – Weak Relevance

A feature Xi is weakly relevant iff it is not strongly relevant, and there exists a subset of features $Si'$of Si for which there exists some xi, yi and si' *with* $p(Xi = xi, Si' = si') > 0$ such that

$$p(Y = y \setminus Xi = xi, Si' = si') \neq p(Y = y \setminus Si' = si') \quad (5.7)$$

Weak relevance specifies that an attribute can occasionally impact the prediction accuracy.

5.3.2 Irrelevant features

Irrelevant variables do not contribute to the predictive accuracy (Jose & Reeba 2014:380-383). They affect the learning accuracy of the algorithms. In most situations, the learning accuracy declines with the increase of irrelevant features (Wolf & Shashua 2003:1-5). A large amount of irrelevant features increases the training and classification time.

5.3.3 Redundant features

Redundant features do not give a better predictive accuracy in identifying a particular class than the currently selected features. The information they offer already exists in other features (Natarajan, Anand, Shanmukh, Saneen & Darshan 2015: 366-372).

5.4 Feature weighting

Weighting attributes allocate a continuous score to each attribute and is more versatile than selecting features. Feature selection is considered an extra ordinary type of feature weighting, whose weight value is constrained to comprise only zero or one. Feature weighting contains more weight values than feature selection. Weights are assigned according to the feature's importance. The feature weights are calculated depending on the quantity of information about the target concept an observed feature value provides. Defining or selecting a correct method that accurately evaluates the quantity of information is critical. IG is normally applied in measuring the significance of attributes, including decision trees (Lee, Gutierrez & Dou 2011:1146-1151).

The following paragraphs discuss the attribute weighting methods.

5.4.1 Equal weight

Each attribute is allocated an equal weight. The Equal Weight Method (EW) needs the least information concerning the importance of criteria and less input by the decision maker. If information about the actual weights is inaccessible, then the correct weights could be denoted as a uniform distribution on the unit $m$simplex of weights described by the domain of $0 \leq \mathrm{w_i} \leq 1, \mathrm{i} = 1,2, \ldots, \mathrm{m}$, and $\sum_{\mathrm{i}} \mathrm{w_i} = 1$. This is denoted as the simplex of weights (Fischer & Dyer 1998: 85:102).

With regard to two attributes and the information is unavailable, the simplex of weights is the multiple of points on the line segment whose vertices are (1,0) and (0,1). The total points on this line have coordinates that sum to one (e.g.(1/3,2/3). This is the 'unit two simplex'. If no knowledge is available concerning the weights, then the information can be denoted by a uniform probability density function over this line. The anticipated value of this distribution is centroid of the line (point with coordinates are (1/2, 1/2). If knowledge about weights is unavailable, then the projected value of the weights distribution is the equal weights vector described by (Fischer & Dyer 1998:85-102);

$$w_i = 1/m, \;\; i = 1,2, \ldots, m \qquad (5.8)$$

However, the validity of using the equal weight for features to be evaluated can be questioned, as each one of these has its own characteristics and preferences (Pakkar 2016:71-86).

5.4.2 Rank sum weight method

In this technique, the weights are the separate ranks. Normalisation is applied by dividing by the total number of the ranks (Roszkowska 2013:14-30). The equation below calculates the weights:

$$w_j(RS) = \frac{n - r_j + 1}{\sum_{k=1}^{n} n - r_k + 1} = \frac{2(n + 1 - r_j}{n(n+1)} \qquad (5.9)$$

given that r_j is the rank of the jth criterion, $j = 1,2, \ldots, n$.

5.4.3 Rank exponent weight method

This weight technique is the generality of the rank sum method. The next equation computes the weight;

$$w_j(RE) = \frac{(n - r_j + 1)^p}{\sum_{k=1}^{n}(n - r_k + 1)^p} \quad (5.10)$$

given that r_j is the rank of the jthcriterion, $p$the parameter describing the weights,

$j = 1,2, \dots, n.$ The parameter p may be approximated by a decision maker, using the weight of the greatest crucial condition or through interactive scrolling. The *p = 0* results to equal weights, *p=1*rank sum weight. As *p* increases, the weights distribution becomes steeper.

5.4.4 Inverse or reciprocal weights

This technique applies the reciprocal of the ranks that are normalised by dividing each term by the sum of the reciprocals (Stillwell, Seaver & Edwards 1981:62-77). The equation is below:

$$w_j(RR) = \frac{1/r_j}{\sum_{k=1}^{n}(1/r_k)} \quad (5.11)$$

where r_j is the rank of the *j-th* criterion, $j = 1,2, \dots, n$

Rank-order centroid weight method

The rank-order centroid (ROC) weight technique creates an approximation of the weights that reduces the biggest error of each weight by finding the centroid of all likely weights preserving the rank order of objective importance. Weights gained in this manner are very stable. If rank order of the correct weight is known, but no other quantitative information about them is available, then the assumption is that the weights are uniformly distributed on the simplex of rank-order weight;

$$w_{r_1} \geq w_{r_2} \geq \cdots \geq w_{r_n} \quad (5.12)$$

where $w_{r_1} + w_{r_2} + \cdots + w_{r_n} = 1$ and r_i is a rank position of w_{r_i}.

5.5. Feature ranking

Feature ranking methods grade features independently without applying classification algorithms. Given a set of features $F = \{f_1 \ldots, f_n, \}$ arrange the attributes by an individual scoring function S(f). If Sf_1 is bigger than the threshold value *t*, feature f_i is added to the new feature subset F'.

Gupta, Jain and Jain (2014:86-91) explain that feature weighting can be changed to a feature ranking by sorting the weights and a ranking can be changed to a feature subset by choosing a suitable threshold.

5.6 Discretisation of attributes

Continuous variables in a data set are converted to categorical values. Numerous machine-learning algorithms have been designed for discrete attributes. Many real-world classification tasks involve continuous features, which therefore require the discretisation of continuous features. Discretisation splits the continuous variables into intervals, so that each interval is treated as a value. Further, discretisation reduces the learning complexity and improves the classification accuracy.

Discretisation processes contain four procedures as follows:

(i) Order the values of the attribute in a sequence.
(ii) Determine a value that will separate the continuous values into subgroups.
(iii) Divide or combine the intervals of continuous values.
(iv) Select the stopping criteria of the discretisation process.

Unsupervised discretization methods

Two simple unsupervised discretisation techniques are the Equal Width Discretisation (EWD) and Equal Frequency Discretisation.

The EWD algorithm decides the least and highest values of the discretised feature and then splits the array into the amount of equal- width discrete intervals, such that each cut point is *xmin+ m x ((xmax - xmin) / i*); where *i* is the quantity of intervals, and *m* takes on the value from 0..(*i*-1).

The EFD formula regulates the least and highest values of the discretised feature, arranges all values from highest to lowest and splits the array into a quantity of intervals, so that each interval holds the same amount of arranged values. Attribute may be lost due to the pre-determined values of the interval *i.*

Supervised discretization methods

The common methods of supervised discretisation are the chi merge and entropy. The x^2statistic determines if the class is independent from two neighbouring intervals, joining them if they are not independent and permitting them to be isolated otherwise. The formula combines the pair of intervals with the lowest value of x^2 provided that the amount of intervals is more than the predetermined highest amount of intervals.

The entropy discretisation proposed by Fayyad and Irani (1993:1022-1027) assesses candidate cut points through an entropy-based technique to select boundaries for discretisation. Instances are arranged into ascending numerical order and then the entropy for each candidate cut point is measured. Cut points are recursively selected to decrease entropy until a stopping criteria is attained. In this model, the stop criterion attains five intervals of the attribute.

5.7 Feature selection processes

Feature selection can be accomplished using individual assessment or subset assessment. Weights are allocated to features, depending on their level of significance. On the contrary, subset evaluation selects the candidate features based on a specific search strategy. The four basic processes in feature selection are subset creation, subset assessment, ending algorithm execution as per threshold value and validating subsets. Subset creation generates a candidate subset using the

exhaustive, sequential (heuristic) or random search using three strategies, forward, backward and bi-directional.

The forward selection method begins with a null set of attributes. An attribute that reduces an error is incorporated into the set one at a time, until an ideal feature subset is attained. The new attributes improve the performance of the previously selected metrics (Lu, Kocaguneli & Cukic 2014:312-322). The backward selection method begins with all features and repeatedly eliminates the least significant feature based on some evaluation criterion (Guyon & Elisseeff 2003:1157-1182). The deletion stops when a certain criterion is fulfilled. Features are added and removed when using the bi-directional method.

The subset is then assessed depending on conditions for instance, similarity, redundancy and information gained through attribute. The procedure ends once the threshold has been reached. Lastly, the selected subset is validated.

5.8 Feature extraction methods

Feature extraction methods transform original features into a lower dimensional space. On the other hand, feature selection methods select a subset of existing features without transformation. The most common feature extraction methods are the Principal Component Analysis and the Linear Discriminant Analysis (Thawonmas & Abe 1997).

5.8.1 Principal component analysis

The Principal Component Analysis (PCA) is a technique that decreases the dimensionality of a subset. The technique is also known as an orthogonal linear transformation that changes data to a new coordinate known as principal components (Abaei & Selamat 2013: 75-96). PCA extracts feature instead of selecting them. The new features are attained by a linear combination of the initial attributes. Features with the greatest variance are utilised to implement the decrease.

The PCA method maps a vector of attributes *v* vectors$\{x_1, x_2, \dots, x_v\}$ from the *s*-dimensional space to *v* vectors $\{x'_1, x'_2, \dots, x'_v\}$ in a new *s′*-dimensional space.

$$x'_i = \sum_{k=1}^{d'} a_k \, i\text{e}k, \quad s' \leq s \tag{5.13}$$

given that $\text{e}k$ are eigenvectors that correspond to s'biggest eigenvectors for the scatter matrix *S* and a_k,*i* are the projections (principal components original datasets) of the original vectors x_i on the eigenvectors $\text{e}k$.

5.8.2 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) is a feature extraction method that is related to analysis of variance and regression analysis (Thakur & Goel 2016). Unlike the linear regression it can be used to analyse two classes and multi classes. The LDA selects features using the backward selection search using the interclass Euclidean distance as the class separation measure. The LDA and the PCA consider the linear combination of variables that best explain data (Rathi & Palani 2012). The LDA attempts to model differences between classes of data. The PCA does not take into account differences in class.

Combination is built on differences instead of similarities. The LDA searches for vectors in the underlying space that discriminate the classes. Two measures are created (Rathi & Palani 2012);

(i) Within class scatter matrix

$$S_w = \sum_{j=1}^{c} \sum_{i=1}^{N_j} (x_i^j - \mu_j)(x_i^j - \mu_j)^T \tag{5.14}$$

given that x_i^j is the i^{th} sample of class j, μ_j is the mean of class j, c is the number of classes and μ_j is the number of samples in class j

(ii) Between class scatter matrix

$$S_b = \sum_{j=1}^{c} (\mu_i - \mu)(\mu_j - \mu)^T \quad (5.15)$$

given that μ represents the mean of all classes.

5.9 Feature selection methods

These techniques can be categorised in various ways. Features can be selected using filters, wrappers or embedded systems.

Feature extraction – A new set of features is produced from the initial set of features using modification or composition.

Feature selection – A subset of the topmost significant attributes is selected.

Table 5.1 presents the different feature selection methods used and their descriptions.

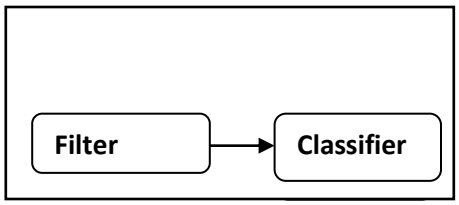
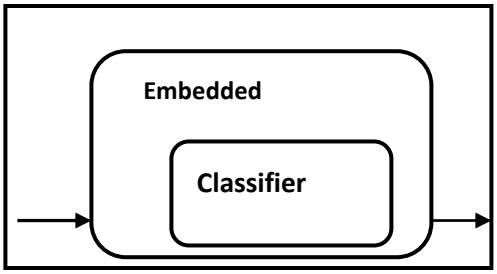
Table 5.1 Feature Selection Methods

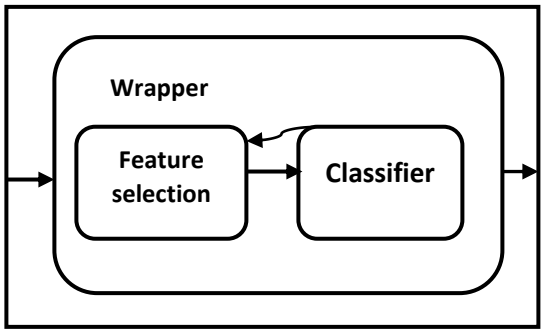
Type	Description	Method
	Statistics Based Methods	T-Test
		Correlation
		Regression
		Clustering
		Chi-Square
		Correlation-Based Feature Selection (CFS)
		Fisher Score

Filter Based Feature Ranking		Feature Weighting K-Means
		Localised Feature Selection Based Scatter Separability
	Information Theory Probability Based Methods	Information Gain
		Gain Ratio
		Mutual Information
		Symmetric Uncertainty
		Maximal Information Coefficient
		Minimum Redundancy Maximum Relevancy (MRMR)
		Fast Correlation based Filter (FCBF)
Wrapper Subset Selection	Naïve Bayes	
	Logistic Regression	
	IBk Nearest Neighbor	
Embedded	FS-Percepton	
	Support Vector Machine	
	C4.5	
	Random Forest	
Extraction Based Method	Principal Component Analysis	

Table 5.2 describes and compares the feature selection techniques.

Table 5.2 Feature Selection Techniques (Bolon-Canedo et al. 2013:483-519)

Method	Advantages	Disadvantages	Examples
	Independent from the classification algorithm Reduced computational expenses than wrappers High speed Superior generalisation capability	No interdependence with the classification algorithm	T-Test Chi-square ReliefF
Embedded 	Interrelation with the classification algorithm Reduced computational expenses than wrappers Measure feature dependencies	Feature selection is dependent on the classification algorithm	Sequential Forward Selection Sequential Backward Elimination

Wrapper 	*Interaction with the classifier* *Calculate feature dependencies*	*High computational cost* *Possibility of overfitting* *Feature selection is dependent on the classification algorithm*	*Wrapper Decision Tree* *Wrapper Support Vector Machine*

5.9.1 Filter

A filter is a pre-processing method that selects a subset of attributes. It is unrelated from the prediction algorithm and utilises the measurement methods including, distances between classes and statistical dependencies for feature selection (Wahono & Suryana 2013:153-166). This method is computationally cheap and popular (Wang & Liu 2016:119-128). However, the filter methods are inclined to select subsets that have many features, therefore a threshold is required to select a subset (Sanchez-Morono, Bolón-canedo & Alonso-Betanzos 2007: 483-519).

Filter methods may be univariate or multivariate. Univariate methods measure the weight of features considering their dependencies to classes. Multivariate methods measure the weight of features with their dependencies on classes and between each feature pair.

Filter methods include the Relief, Information Gain, Mutual Information, Symmetric Uncertainty and OneR.

5.9.1.1 Correlation-based feature selection

The Correlation-Based Feature Selection (CFS) is a filter method which grades attribute subsets based on a correlation-based heuristic evaluation function. This multivariate technique ranks the significance of attributes by evaluating the association of attributes between the class and with other attributes. Insignificant features are not selected, due to their low association with the class. Attributes that cause redundancy should be eliminated, as they are extremely associated with one or more of the features (Hall 1999:1-178; Karthikeyan & Thangaraju 2015: 1-6).

$$F_s = \left(\frac{xr_{cf}}{\sqrt{\mathrm{x} + \mathrm{x}(\mathrm{x}-1)r_{ff}}} \right) \qquad (5.16)$$

Given that $F_s$is the significance of the attribute subset,$r_{cf}$is the average linear correlation coefficient between these attributes and classes and $r_{ff}$is the average linear correlation between the different attributes.

5.9.1.2 Chi-square

The Chi-square test for independence examines if a statistically significant relationship exists between an independent attribute and a dependent attribute. The statistical method, X^2test is used, among other things, to evaluate the impartiality of two events. Events *A* and *B* are described as independent if *P(XY) = P(X)P(Y)* or, equally, *P(X|Y) = P(X)* and *P(Y|X) = P(B).*

In attribute selection, this technique is applied in testing if the existence of a variable and the existence of a certain class are independent. The null hypothesis (H_0) assumes that there is no dependence. The null hypothesis is that, if f of the instances have a certain value and g of the instances are in a particular class, $\frac{f.g}{n}$ instances have certain value and are in a specific class (*n* is the total number of instances in the dataset). The reason is that *f/n* instances have the value and *g/n* instances are in the class and if the probabilities are independent (i.e. the null hypothesis) their joint probability is their product (Ladha & Deepa 2011: 1787-1797).

The X_2 measures the divergence of the observed data values from the expected values.

$$X^2 = \sum_{m=1}^{k} \sum_{n=1}^{c} \frac{(O_{m,n} - E_{m,n})^2}{E_{m,n}} \quad (5.17)$$

Where *k* represents the amount of unique values of the attribute, *c* is the amount of classes. $O_{m,n}$is the quantity of instances with value m that are in class n, and $E_{m,n}$ is the expected amount of instances with value m and class *n*, based on *(f·g)/n*. High scores on the chi-square indicate that the variable and the class are dependent, (i.e, that the attribute is significant to the class).

5.9.1.3 T-Test

A paired T-test is a hypothesis test that compares the means between paired values in two samples. It incorporates the sample size and variability in the data and creates a number called a t-value. The numerator in the ratio is the signal, (i.e. the difference between the two means). The denominator is a measure of the variability or dispersion of the scores.

$$T = \frac{\bar{X}_1 - \bar{X}_2}{s} \quad (5.18)$$

Filter methods used in this study are the Linear Correlation, Information Gain and ReliefF. The Linear Correlation is a statistical method that measures the relationship between two features. LC may not be able to measure relationships that are non-linear. It is also not ideal for nominal data (Yu & Liu 2004).

The Information Gain is an information theoretic entropy based measure that selects relevant features based on the class attribute. It is measured by the uncertainty in identifying the class attribute when the value of the feature is known (Agarwal & Mittal 2013). It can identify both relevant and redundant features. However, it can't capture the interactions between features. The IG is also biased as it favours attributes with many values (Hall 1999).

The ReliefF is an extension of the Relief method. It not only deals with class problems but is more robust and capable of handling incomplete and noisy data.

The MIC is a method that calculates functional and non-functional relationships between two variables (Reshef et al. 2011). A wide range of associations can be such as linear, sinusoidal, exponential, or parabolic, can be detected by the MIC. The method also has the equitability property, i.e. similar scores are allocated to equally noisy relationships of different types (Fan, Li & Zhang, 2017).

5.9.2 Wrapper methods

The wrapper feature selection process consists of three parts:

- search approach
- evaluation operation
- performance operation.

The search approach hunts and selects features. The evaluation operation utilises a predetermined classifier to assess the set of attributes considered. The performance operation validates the chosen attributes.

The filter-based technique is computationally quicker than the wrapper method. Nevertheless, the wrapper method normally outclasses the filter technique in terms of the accuracy of the classification algorithm (Wang & Liu 2016:119-128).

Deterministic wrappers search through the feature space for features using the forward or backward method. In the forward selection process, the set is initially empty and the most relevant single attributes are selected and added to it. The attributes added are those that are not yet in the set and improve the classification accuracy.

Random wrappers, unlike the deterministic wrappers, search for the following subset of features is partially at random. Individual attributes or multiple attributes can be incorporated, eliminated or

substituted from the preceding attribute set. The randomised wrapper techniques emulate natural sensations comprising the biological evolutionary procedure such as genetic algorithm to select a features (Rathore & Gupta 2014: 1-10).

Due to the interaction of filter methods and the classifier, the classification accuracy is better than that achieved with filter methods.

5.9.2.1 Sequential forward selection

The Stepwise Method, is a search procedure that is also known as the Sequential Forward Selection (SFS). It starts with a null set and iteratively includes the greatest suitable feature *k+* to achieve the highest objective function *J(Fi+k+).*This feature is added to a set of existing features *Fi*.

SFS accomplishes best results if the optimal subset has few attributes. If the search has just begun and it is close to a null set, a big amount of states can be possibly assessed. If the set is nearly full, the area inspected by the method (when searching for candidate features) is smaller, as the majority of the attributes would have been chosen already. The search method resembles an ellipse to give emphasis to the point that there are less elements near the full or empty sets. The disadvantage of the SFS is that it cannot eliminate features that become unusable after other features have been incorporated.

5.9.2.2 Sequential backward elimination

It operates the other way around. The function is also known as the Sequential Backward Selection. The method commences with a full set and removes the least significant feature $x-$, thereby reducing the function's value $(J(M-x-))$ using a minimum reduction. The objective function may in some instances grow, due to the elimination of a feature. These types of functions are known as non-monotonic.

5.9.3 Embedded methods

Embedded approaches conduct feature selection during the training procedure and are generally inbuilt to the specified classification algorithms and hence may be more effective than the other approaches. (Ghanta & Rao 2015: 300-303).

5.9.3.1 Decision trees

In decision trees, a feature is represented by a node and potential values of a feature are specified at the branches emanating from the node. The tree uses feature values to perform the feature selection process. Tree ensembles, which consist of many trees are more accurate than single trees. However, tree ensembles have more incidents of feature redundancy. Most of the tree algorithms, select features split attributes based on the entropy or Information Gain. The IG favours features with numerous values. The C4.5 supresses this by using an alternative measure called Information Gain Ratio.

5.9.3.2 Naïve Bayes

The Naïve Bayes is a classification technique which is founded on the Bayesian networks theory and uses probability for predicting the class an instance is associated with, given the set of features defining the instance. Features are considered to contribute independently to the probability, regardless of correlations between them. The classifier learns from the training data which parameters are suitable for the classification task. The Bayes rule joins the prior probability of every variable and the likelihood to create a highest posterior probability that is used to predict a class.

The classification algorithm is denoted by:

$$fi(X) = \prod_{j=1}^{n} P(xj\backslash cj)P(ci) \qquad (5.19)$$

where $X = (x_1, x_2, \dots x_n)$ represents the vector of a feature, (i.e. x_1 is the value of feature X. and ,$j = 1, 2, \dots, N$ are the potential labels of the class. $P(x_j \backslash c_j)$ are conditional probabilities and $nP(c_i)$are prior probabilities).

$$fi(X) = \prod_{j=1}^{n} P(xj \backslash cj) P(ci) \qquad (5.20)$$

The MIC method that was used in this study is a filter-based attribute ranking method. Filters, unlike wrappers and embedded methods, are computationally inexpensive. They are also independent from the classification algorithms. The MIC can capture functional and non-functional associations.

5.10 Chapter summary

Selecting the most significant and non-redundant attributes improves the classification accuracy. Feature weighting allocates a score to an attribute based on its level of significance. The feature ranking technique then sorts the features. Attribute selection techniques select a number of features as specified by threshold. Feature weighting is transformed to feature ranking by sorting the weights, and ranked features can be selected to a feature subset. The following chapter presents the results from the experiments that were conducted in this study.

CHAPTER 6

PREDICTION MODEL EVALUATION

6.1 Introduction

This chapter presents the outcome of the software defect prediction experiments regarding the feature selection and classification algorithms performances. The previous chapter discussed the proposed hybrid algorithm for selecting the subset of metrics. Methods used in selecting features to be employed in defect prediction experiments were discussed.

The aim of the research test was to ascertain the effectiveness of the MICFastCR feature selection algorithm in software defect prediction. The software defects data was obtained from five OSS systems written in Java:

- Mylyn
- Equinox
- Eclipse PDE
- Apache Lucene
- Eclipse JDT

The data consists of process metrics that were used in this study to build a software prediction model. Previous research has proved that an intuitive selection of software metrics influences the model performance in the defect prediction process(Xu, Xuan, Liu & Cui 2016: 370-381; Liu, Chen, Liu, Chen, Gu & Chen 2014; Sharmin, Wadud & Nower 2015: 184-189).

The new model, MICFastCR is contrasted with other feature selection methods, ReliefF, Information Gain and Linear Correlation using performance measures. The validity of results in a research must

be investigated. This chapter addresses elements that may affect the validity of the research experiments and how to limit them. The evaluation of algorithms performance metrics can be ambiguous as a result of inherent variance. Therefore, the conclusions in this study are founded on the statistical tests for significance.

6.2 Statistical comparison of classification algorithms

Earlier feature selection-based defect prediction studies have indicated that a large amount of features may result in reduced classification accuracy (Liu, Chen, Liu, Chen, Gu & Chen 2014: 426-435; Sharmin, Wadud & Nower 2015: 184-189; Khan, Gias, Siddik, Rahman, Khaled & Shoyaib 2014: 1-4). In this research, the code for selecting attributes using the MIC, ReliefF, Information Gain and Linear Correlation was written and tested in the R programming tool. The output from the code displayed the feature weights that were ranked in order of importance. Attributes were selected from the original sets considering their scores ant the ones that had the least weights were eliminated from the feature subsets. The AUC and F-Measure are commonly applied in the assessment of the classification algorithms performance.

6.2.1 Data analysis

This research evaluates the efficiency of the feature selection methods using:

Win draw loss method

This method computes the number of times algorithm *i* performed better, equal or worse than algorithm *j* (compares pairs of methods). The method adds the quantity of data sets in which an algorithm is the overall winner.

Average

The mean value of a performance measure is calculated across all data sets.

Freidman and Nemenyi tests

A Friedman test is used to ascertain if the algorithms produce statistically different results (Friedman 1937: 675-701). If the test results show a statistical difference, the Nemenyi or Wilcoxon signed rank post hoc test identifies the algorithms that perform differently (Mende & Koschke 2009:1-10). The test compares the average ranks of the classification algorithms and inspects if amounts between a pair of classifiers differ and if the difference between their ranks is more than the critical difference (CD);

$$CD = q\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (6.1)$$

given that *k* is the quantity of algorithms, *N* the amount of datasets, $q\alpha$ is a critical value subject to the quantity of algorithms and the level of significance. The value of $q\alpha$depends on the Studentized range statistic divided by $\sqrt{2}$ and is tabulated in standard statistical textbooks.

6.2.2 Proportion of features selected

This part discusses the proportion of attributes selected by the feature selection algorithms. In this study, different filter methods that are used in selecting attributes are discussed. The most relevant features were selected. Table 6.1displays the percentages of features selected by the feature selection algorithms per dataset.

Table 6.1 Proportion of features selected by the algorithms

Dataset	MICFastCR	ReliefF	Info Gain	Linear Correlation
Equinox	37%	60%	75%	65%
Lucene	40%	60%	70%	71%
Mylyn	55%	57%	50%	55%
PDE	65%	65%	70%	65%
JDT	35%	40%	40%	57%
Average	46%	56%	61%	63%
Win/Draw/Loss		**4/1/0**	**4/0/1**	**3/2/0**

The Win/Draw/Loss outcomes show that classification using the MICFastCR subset produces the highest accuracy compared to other subsets. The Friedman Test was applied in validating the feature selection results. This non-parametric test was used to assess classification accuracy when using attribute reduced algorithms at significance level, $\alpha = 0.05$. Ranks are allocated to each classifier per data set. The test examines if the computed average ranks are significantly different from the average rank. The equation that is applied in determining if the algorithms performances are different was defined by (Mende & Koschke 2009:1-10);

$$X_F^2 = \frac{12N}{k(k+1)}\left(\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right) and\ F_F = \frac{(N-1)x\frac{2}{F}}{N(k-1)-x_F^2} \qquad (6.2)$$

where F_F is distributed according to the F-Distribution with *k*- and *(k-1)(N-1)* degrees of freedom.

The hypothesis of the test:

H_θ: *there is no difference in the performance of the feature selection algorithms*

H_1: *there is a difference in the performance of at least two feature selection algorithms*

Friedman's test result:

The proportion of attributes selected test revealed that feature selection algorithms did not perform significantly different. The Chi-squared value is 3.1957 and the *p*-value is 0.3624. The *p*-value obtained is 0.362 and greater than the *p*-value of 0.05 and therefore the H_θ is accepted. The conclusion is that the performance of the feature selection techniques in the proportion of attributes is not significantly different.

6.2.3 Running time of the feature selection algorithms

The analysis of the amount of time required to execute algorithms is essential. In the experiment, the average runtime in milliseconds of each feature selection algorithm was calculated. The

Win/Draw/Loss method indicates that the runtime of the ReliefF algorithm is less than that of other algorithms, (see Table 6.2).

Table 6.2 Runtime of the feature selection algorithms (milliseconds)

Dataset	Full	MICFastCR	ReliefF	Info Gain	Linear Correlation
Equinox	64071.59	61687.74	61713.3	63882.12	61700.7
Lucene	63504.11	62110.3	61674.54	64209.72	61660.5
Mylyn	62582.84	61684.74	61667.52	61916.52	61728.54
PDE	61459.09	61795.92	61744.14	62088.12	61795.92
JDT	61035.13	61668.9	61651.92	62802.92	63743.77
Average	**62,530.55**	**61,789.52**	**61,690.28**	**62,979.88**	**62,125.89**
Win/Draw/Loss	**3/0/2**		**1/0/4**	**5/0/0**	**3/1/1**

The Friedman test was applied in comparing the running time of the feature selection algorithms over five datasets. The results obtained, Chi-squared 6.586, *p*-value 0.16, indicate that there is no statistically significant difference in the performance of the feature selection algorithms.

6.3 Classification results

Classification in this case study was implemented using the Naïve Bayes, PART and J48 machine-learning algorithms in the WEKA application. The performance-evaluation measures included the Percentage Accuracy, Recall, Area Under the Curve and F-Measure and the Win/Draw/Loss methods. The Friedman test, succeeded by the Nemenyi test, as suggested by Demsarˇ (2006:1-30) were applied to statistically compare the feature selection algorithms and the classifiers.

6.3.1 Percentage accuracy

In the percentage accuracy experiment conducted using the Naïve Bayes classifier, the MICFastCR has the most wins. The percentage accuracy of the feature selection algorithms using the Naïve Bayes was compared, (see Table 6.3).

Table 6.3 Percentage accuracy using Naïve Bayes

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	68.826(2)	80.178 (5)	78.794 (4)	75.205 (3)	62.268 (1)
Lucene	77.365 (2)	84.103 (5)	79.017 (3)	82.044 (4)	75.564 (1)
Mylyn	76.553(2)	78.006 (4)	77.161 (3)	47.711 (1)	81.575 (5)
PDE	68.324(1)	82.686 (5)	81.884 (4)	75.07 (2)	79.476 (3)
JDT	70.301(1)	93.202 (4)	95.387 (5)	85.878 (3)	82.818 (2)
Average	***72.274***	***83.635***	***82.4489***	***73.1816***	***76.3402***
W/D/L	***5/0/0***		***4/0/1***	***5/0/0***	***4/0/1***
Average Rank	**1.6**	**4.6**	**3.8**	**2.6**	**2.4**

The Friedman test results show a statistically significant difference in the performance of the attribute selection algorithms. The Chi-squared value is 10.4 and the p-value, 0.034 and is significant at the 95% confidence level. Therefore, the null hypothesis is rejected.

The critical value for 5 algorithms at $p = 0.05$ is 2.728 (Demsarˇ 2006:1-30). The Critical Distance (CD) is calculated as described by Equation 6.1. In this test, the corresponding CD at *p*=0.05 is $2.728\sqrt{\frac{5.6}{6.5}}$ = 2.728. The CD is $2.459\sqrt{\frac{5.6}{6.5}}$ = 2.459 at *p*=0.10. The difference between the average ranks of two algorithms and the CD indicates that the MICFastCR performs significantly better than the Full set, (4.6 – 1.6 = 3 > 2.728). The ReliefF does not perform better than the Full set as their average ranks is smaller than the CD (3.8 – 1.6 = 2.2 < 2.728(CD value)).

The *p*-values were computed using the Nemenyi test to indicate the differences between pairs of algorithms. These are displayed in Table 6.4.

Table 6.4 Nemenyi Test - Perc Accuracy using Naive Bayes

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.034	-	-	-
ReliefF	0.18	0.931	-	-
LCorrel	0.855	0.266	0.751	-
InfoGain	0.931	0.18	0.628	1

According to the Nemenyi test, the MICFastCR subset differs highly to the Full set $p < 0.05\%)$ with a link value of 0.023.

In another experiment, the Percentage Accuracy was tested on the PART classifier. The MICFastCR outperforms other methods except the Information Gain. The Wins/Draw/Loss data is shown in Table 6.5.

Table 6.5 Percentage Accuracy using PART

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	81.812	82.503	83.022	83.334	84.519
Lucene	80.314	86.888	83.719	83.791	84.023
Mylyn	74.087	84.751	80.339	81.388	82.393
PDE	74.095	85.479	84.283	84.222	84.516
JDT	77.593	98.475	99.037	77.593	97.794
Average	**77.580**	**87.619**	**86.080**	**82.066**	**86.649**
W/D/L	**5/0/0**		**3/0/2**	**4/0/1**	**4/0/1**
Average Rank	**1.1**	**4.2**	**3.0**	**2.7**	**4.0**

Testing for statistical significance of differences between the feature selection algorithms was conducted using the Friedman test. The results prove that the difference in the performance of the methods is statistically significant, Chi-square was 12.96, *p*-value is 0.011. Consequently, the null hypothesis was rejected. The Nemenyi test was used to further identify pairs of algorithms that are significantly different. The CD was used for the pairwise comparisons. The MICFastCR and Full set have a statistically significant performance (4.2 – 1.1 = 3.1 > 2.728(CD)). The InfoGain and Full set performances differences are also significant (4.0 – 1.1 = 2.9 > 2.728).

The Nemenyi test calculated *p*-values for the same test as seen in Table 6.6.

Table 6.6 Nemenyi Test- PART CLassifier

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.011	-	-	-
ReliefF	0.317	0.751	-	-
LCorrel	0.497	0.562	0.998	-
InfoGain	0.031	1	0.855	0.691

The Full set and the MICFastCR subset had a statistically significant difference of 0.011, while the Full set and InfoGain subset had a statistically significant difference of 0.031.

Another test used the J48 classifier to compare the feature selection algorithms performance. The Win/Draw/Loss, Friedman and Nemenyi tests were applied. As demonstrated in Table6.7, the MICFastCR method has the majority wins compared to other algorithms.

Table 6.7 Perc Accuracy using J48

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	83.456	83.766	83.884	83.766	83.458
Lucene	83.95	84.385	84.255	84.139	84.138
Mylyn	84.581	84.635	84.624	84.689	84.683
PDE	84.643	84.904	84.663	84.663	84.903
JDT	97.794	98.495	98.959	97.653	97.794
Average	**86.885**	**87.237**	**87.277**	**86.982**	**86.995**
W/D/L	**5/0/0**		**3/0/2**	**3/1/1**	**4/0/1**
Average Rank	**1.2**	**4.1**	**3.6**	**2.9**	**2.8**

The Friedman was used to determine if the feature selection algorithms are statistically different. The Chi-squared value was 11.543 and the *p*-value was 0.021, which implies that there is a statistical difference between the attribute selection methods. The Nemenyi test used the CD to evaluate the differences. The Full and MICFastCR sets' performance difference is significant (4.1 – 1.2 = 2.9 > 2.728).

The results in Table 6.8 indicate pairwise comparisons using *p*-values. The statistical difference is between the Full and the MICFastCR sets (p-value = 0.031< 0.05).

Table 6.8 Nemenyi Test – Perc Accuracy using J48

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.031	-	-	-

ReliefF	0.317	0.855	-	-
LCorrel	0.266	0.897	1	-
InfoGain	0.751	0.434	0.957	0.931

6.3.2 Area under ROC curve

The Receiver Operating Characteristic (ROC) curve plots the True Positive (TP) against the False Positive (FP). The ROC curve can be used to locate a threshold for a classification algorithm which increases the true positives, while reducing the false positives. The AUC can be used to evaluate a classification algorithm's performance or equate it with other algorithms. The AUC in the present study was used to assess performances of the feature selection algorithms. Table 6.9 reports the AUC results obtained by the Naïve Bayes algorithm. The results were produced by the WEKA prediction process discussed in section 3.3.6.3.

Table 6.9 Area under ROC Using Naive Bayes

Project	Full	MICFastCR	ReliefF	Linear Correlation	Infor Gain
Equinox	0.641	0.656	0.643	0.644	0.65
Lucene	0.618	0.635	0.638	0.622	0.626
Mylyn	0.616	0.644	0.633	0.627	0.632
PDE	0.566	0.601	0.596	0.588	0.594
JDT	0.817	0.859	0.844	0.831	0.835
Average	**0.652**	**0.679**	**0.671**	**0.662**	**0.667**
W/D/L	**5/0/0**		**4/0/1**	**5/0/0**	**5/0/0**
Average Rank	**1**	**4.8**	**3.8**	**2.2**	**3.2**

The Friedman test was used to ascertain if the performances of the algorithms was statistically different. The Friedman test produces a *p*-value 0.0018 which is less than the critical threshold of 0.05. This implies that the performances are not random and therefore the performances were evaluated using the CD. The Full and MICFastCR sets' performance differences are significant (4.8 – 3.8 = 3 > 2.728).

The Nemenyi test *p* values for the same test are displayed in Table 6.10.

Table 6.10 Nemenyi Test - AUC using Naive Bayes

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.0014	-	-	-
ReliefF	0.0409	0.8555	-	-
LCorrel	0.7514	0.0703	0.4973	-
InfoGain	0.1796	0.4973	0.9751	0.8555

The AUC was also tested using the PART classifier. The Win/Draw/Loss indicates that the Information Gain and the MICFastCR algorithms perform better than the other classifiers, see Table 6.11.

Table 6.11 Area under ROC Curve Using PART

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	0.542	0.712	0.617	0.651	0.683
Lucene	0.618	0.552	0.631	0.617	0.537
Mylyn	0.544	0.605	0.592	0.603	0.601
PDE	0.682	0.782	0.652	0.632	0.602
JDT	0.838	0.884	0.888	0.854	0.833
Average	**0.645**	**0.707**	**0.676**	**0.671**	**0.651**
W/D/L	**4/0/1**		**3/0/2**	**4/0/1**	**5/0/0**
Average Rank	**1**	**4.4**	**3.8**	**2.2**	**3.2**

The Friedman test was used to establish if the differences were significant. The result showed that the Chi-squared value was 12.32 and the p-value was less than 0.015, which indicated that the differences were statistically different. The comparisons of performances using the CD were:

Full vs MICFastCR sets (4.4 – 1 = 3.4 > 2.728)

Full vs ReliefF (4.4 – 1 = 3.4 > 2.728)

The comparison of the differences between the average rankings and the critical difference prove that the algorithms' performances are statistically different. The Nemenyi test calculated the *p*-values of the algorithms, see Table 6.12.

Table 6.12 Nemenyi Test - AUC using PART

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.012	-	-	-
ReliefF	0.628	0.373	-	-
LCorrel	0.115	0.931	0.855	-
InfoGain	0.855	0.18	0.995	0.628

In a separate test, the feature selection algorithms were evaluated using the J48 classifier. As shown in Table 6.13, the proposed MICFastCR method has the most wins.

Table 6.13 Area under ROC Curve using J48

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	0.662	0.664	0.661	0.551	0.537
Lucene	0.599	0.612	0.607	0.58	0.705
Mylyn	0.598	0.799	0.698	0.599	0.634
PDE	0.601	0.714	0.658	0.599	0.675
JDT	0.839	0.884	0.4	0.856	0.875
Average	**0.660**	**0.735**	**0.605**	**0.637**	**0.685**
W/D/L	**5/0/0**		**5/0/0**	**5/0/0**	**4/0/1**
Average Rank	**1**	**4.6**	**3.4**	**2.4**	**3.2**

The results from the Friedman test show that the attribute selection methods' performances are statistically different. The Chi-squared value is 11.04, while the *p*-value is 0.026. The CD value was compared with the differences between the average ranks of the MICFastCR and Full sets (4.6 – 1 =3.6 > 2.728). The Bonferroni-Dunn test was applied to verify if one of the other algorithm's performances can be improved by tuning their parameters. The CD was computed using the Bonferroni-Dunn test at *p*=0.10 as $2.241\sqrt{\frac{5.6}{6.6}}$ to compare the ReliefF and the Full sets (3.4 - 1 = 2.4 >

2.2). The outcome denotes statistically significant differences between the ReliefF and the Full sets. The Nemenyi *p*-values test reveals that the sets that caused the differences are the MICFastCR and the Linear Correlation, MICFastCR and the Full set, see Table 6.14.

Table 6.14 Nemenyi Test- AUC using J48

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.07	-	-	-
ReliefF	0.975	0.266	-	-
LCorrel	0.995	0.023	0.855	-
InfoGain	0.751	0.628	0.975	0.497

The boxplot in Figure 6.1 displays the AUC results of the feature selection algorithms.

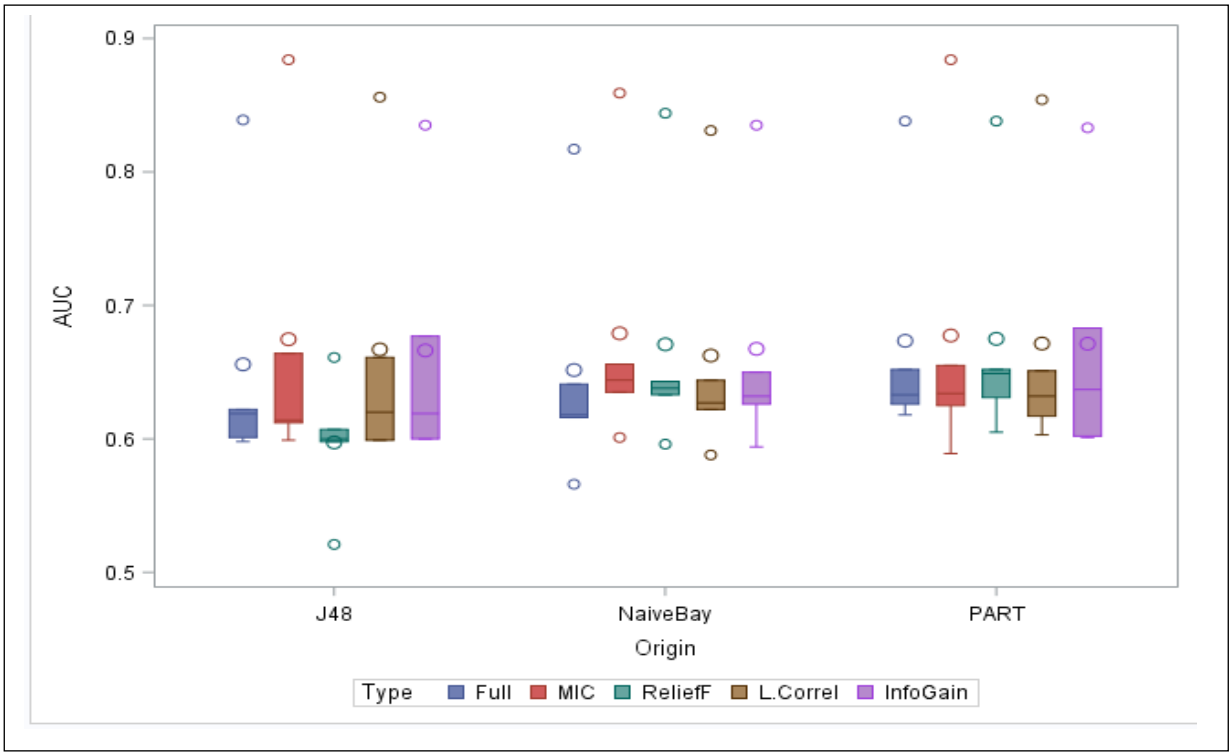


Figure 6.1 Boxplot: Area under ROC Curve

6.3.3 F-Measure

The F-Measure computes the harmonic mean of Precision and Recall. Table 6.15 displays the F-Measure values for the Naïve Bayes classification algorithm. The MICFastCR subset has the best Win/Draw/Loss values compared to the ReliefF, Linear Correlation and Information Gain subsets.

Table 6.15 F-Measure using Naive Bayes

Project	Full	MICFastCR	ReliefF	Linear Correlation	Infor Gain
Equinox	0.799	0.88	0.845	0.889	0.89
Lucene	0.836	0.888	0.894	0.884	0.871
Mylyn	0.251	0.856	0.263	0.257	0.296
PDE	0.888	0.9	0.905	0.894	0.88
JDT	0.963	0.976	0.97	0.966	0.967
Average	**0.747**	**0.900**	**0.775**	**0.778**	**0.781**
W/D/L	**5/0/0**		**3/0/2**	**4/0/1**	**4/0/1**
Average Rank	**1.2**	**4.2**	**3.8**	**2.8**	**3.0**

The Friedman test p-value result, 0.029 indicates that the performances are statistically different. The CD comparison of the Full and MICFastCR subsets reveal that the differences are significant (4.2 – 1.2 = 3 > 2.728). As observed in Table 6.16, the Full set and MICFastCR subsets have a significant difference of 0.02.

Table 6.16 Nemenyi Test – F-Measure using Naive Bayes

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.02	NA	NA	NA
ReliefF	0.18	0.93	NA	NA
LCorrel	0.86	0.27	0.75	NA
InfoGain	0.93	0.18	0.63	1.00

In another experiment run on the PART classifier, the outcome presented in Table 6.17, demonstrates that the ReliefF performs better than the MICFastCR in tests that were run using the JDT dataset. However, the MICFastCR has the best overall performance.

Table 6.17 F-Measure using PART

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	0.907	0.915	0.911	0.848	0.914
Lucene	0.91	0.913	0.912	0.874	0.914
Mylyn	0.914	0.917	0.915	0.861	0.916
PDE	0.913	0.917	0.915	0.856	0.916
JDT	0.981	0.992	0.995	0.988	0.989
Average	**0.925**	**0.931**	**0.930**	**0.885**	**0.930**
W/D/L	**5/0/0**		**4/0/1**	**5/0/0**	**4/0/1**
Average Rank	**1.8**	**4.6**	**3.4**	**1.1**	**4.0**

In this experiment, the Full and MICFastCR sets' differences are significant at p=0.05 (4.6 – 1.8 = 2.8 > 2.728) according to the CD and average rankings calculation.

In the experiment conducted using the J48 classifier, the ReliefF algorithm has the best Win/Draw/Loss records, see Table 6.18.

Table 6.18 F-Measure using J48

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Equinox	0.902	0.916	0.934	0.892	0.9
Lucene	0.913	0.916	0.915	0.914	0.914
Mylyn	0.877	0.917	0.925	0.893	0.903
PDE	0.917	0.918	0.917	0.917	0.918
JDT	0.982	0.995	0.992	0.988	0.989
Average	**0.918**	**0.932**	**0.937**	**0.921**	**0.925**
W/D/L	**5/0/0**		**3/0/2**	**5/0/0**	**4/1/0**
Average Rank	**1.6**	**4.5**	**4.0**	**1.9**	**3.0**

The *p*-value is 0.011 and smaller than the significance level of 0.05, therefore the Nemenyi test was conducted. Comparison was conducted using the CD derived from the Nemenyi test critical values. The Full and MICFastCR sets are significant (4.5 -1.6 =2.9 > 2.728). The Full and the ReliefF pair are insignificant at *p* = 0.05(4.1 – 1.9 = 2.1 < 2.728). Figure 6.2 visualises the F-Measure boxplot.

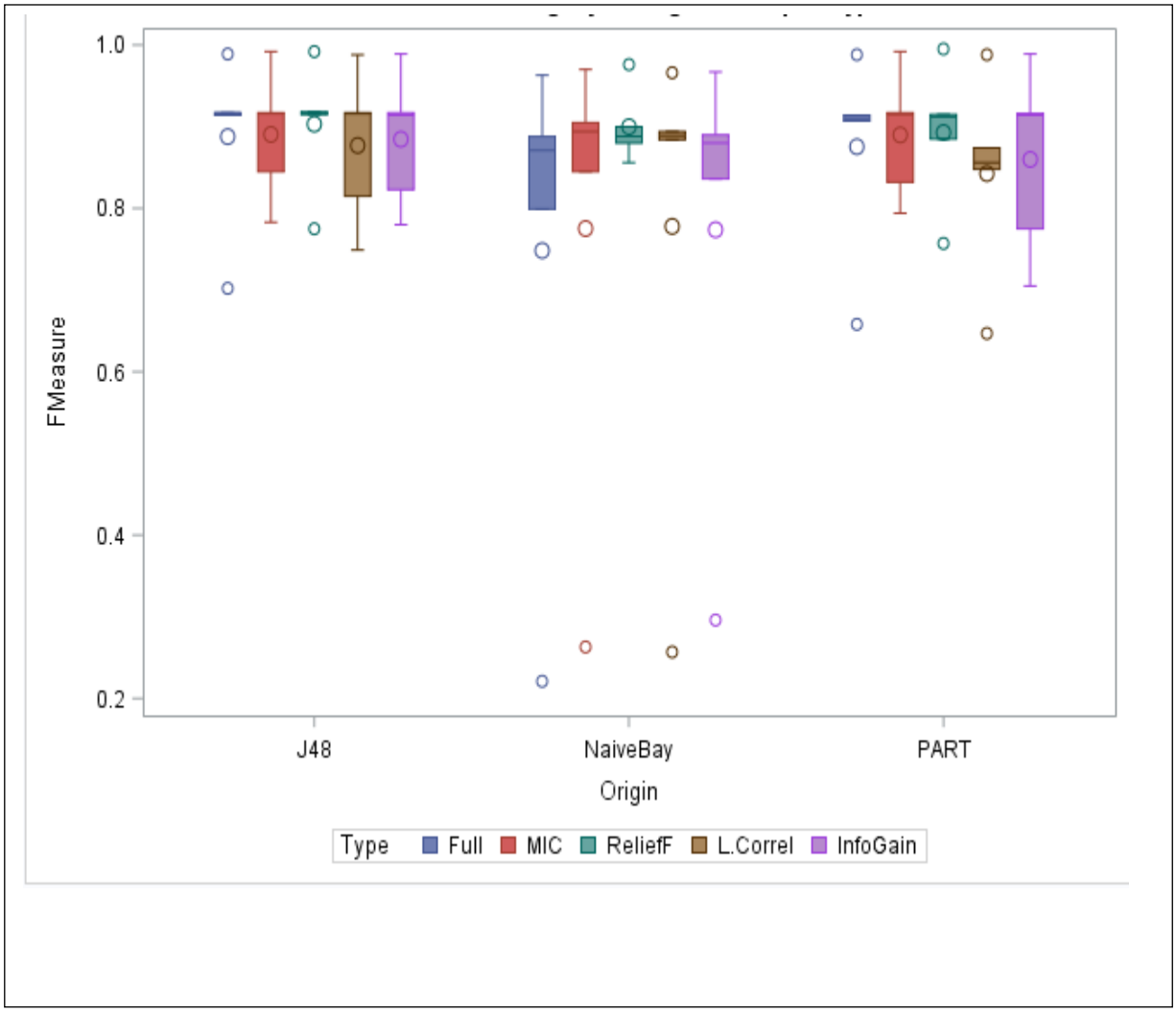


Figure 6.2 Boxplot: F-Measure

6.3.4 Root mean squared error

The Root Mean Square Error (RMSE) uses sample and population values to measure the differences between the values that were predicted and the values that were observed. The ReliefF classifier obtained the best result (0.263) in the test run using the JDT subset. However, the Linear Correlation and Information Gain has the best Win/Draw/Loss records against all the other algorithms, see Table 6.19.The Friedman computes the p-value for all algorithms as 0.034.

Table 6.19 RMSE using Naive Bayes

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.391	0.361	0.346	0.32	0.333
Lucene	0.36	0.349	0.352	0.354	0.351
Mylyn	0.682	0.632	0.342	0.595	0.586
PDE	0.413	0.391	0.398	0.408	0.405
JDT	0.297	0.266	0.263	0.248	0.257
Average	**0.429**	**0.400**	**0.340**	**0.385**	**0.386**
W/D/L	**5/0/0**		**2/0/3**	**2/0/3**	**2/0/3**
Average Rank	**1**	**3.2**	**3.6**	**3.4**	**3.8**

The CD and the rankings prove that the differences between the Full and MICFastCR sets are insignificant at p =0.05 (3.2 – 1 = 2.2 < 2.728) and at p =0.10 (3.2 – 1.0 = 2.2 < 2.459). However, the Full and InfoGain sets differences are significant at p=0.05 (3.8 – 1 = 2.8 > 2.728).The Nemenyi test produces p-values for the pairs of feature selection algorithms as presented in Table 6.20.

Table 6.20 Nemenyi Test - RMSE using Naïve Bayes

	Full	MICFastCR	ReliefF	LCorrel
	0.180	NA	NA	NA
ReliefF	0.023	0.931	NA	NA
LCorrel	0.855	0.266	0.751	NA
InfoGain	0.931	0.180	0.628	1.000

In the experiment conducted using the PART algorithm, the proposed MICFastCR followed by the Information Gain had the best RSME values (least error values).

Table 6.21 RMSE using PART Classifier

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.317	0.306	0.309	0.306	0.307
Lucene	0.309	0.302	0.304	0.304	0.3
Mylyn	0.302	0.297	0.299	0.301	0.298
PDE	0.369	0.362	0.365	0.368	0.361
JDT	0.255	0.253	0.254	0.301	0.257
Average	**0.310**	**0.304**	**0.306**	**0.316**	**0.305**
W/D/L	**5/0/0**		**5/0/0**	**4/1/0**	**3/0/2**
Average Rank	**1.4**	**4.3**	**2.9**	**2.4**	**3.8**

The *p*-value for all algorithms was 0.018, which is less than the significance level of 0.05. This required that further tests be run using the Nemenyi. Using the CD value 2.728 at *p*=0.05, the MICFastCR and Full sets' differences are significant (4.3 – 1.4 = 2.9 > 2.728). The pairwise comparison results are displayed in Table 6.22.

Table 6.22 Nemenyi Test –RMSE Using PART Classifier

$p.value				
	Full	MICFastCR	ReliefF	LCorrel
MIC	0.02	NA	NA	NA
ReliefF	0.18	0.93	NA	NA
LCorrel	0.86	0.27	0.75	NA
InfoGain	0.93	0.18	0.63	1.00

An experiment was run to calculate the RMSE using the J48 classifier. The results are displayed in Table 6.23.

Table 6.23 RMSE using J48

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.305	0.302	0.3	0.302	0.305
Lucene	0.302	0.297	0.299	0.3	0.3
Mylyn	0.296	0.295	0.295	0.295	0.295
PDE	0.375	0.358	0.361	0.361	0.361
JDT	0.257	0.256	0.256	0.256	0.256
Average	**0.307**	**0.302**	**0.302**	**0.303**	**0.303**
W/D/L	**5/0/0**		**2/2/1**	**2/3/0**	**3/2/0**
Average Rank	**1.1**	**4.1**	**3.7**	**2.6**	**2.7**

The p-value was 0.004. This indicates that the statistical differences were significant. The critical values, CD and rankings calculations indicated that the Full and MICFastCR differences are significant (4.1 – 1.1 = 3 > 2.728).

6.3.5 True positive rate

The True Positive rate calculates the fraction of values that are actually positive and were predicted to be positive. The MICFastCR algorithm had the best Wins/Draw/Loss records, in the Naïve Bayes algorithm experiment, as shown in Table 6.24.

Table 6.24 True Positives using Naive Bayes

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.805	0.877	0.929	0.942	0.938
Lucene	0.5	0.944	0.936	0.936	0.939
Mylyn	0.454	0.586	0.888	0.484	0.471
PDE	0.936	0.971	0.958	0.946	0.952
JDT	0.847	0.872	0.848	0.848	0.846
Average	**0.708**	**0.850**	**0.912**	**0.831**	**0.829**
W/D/L	**5/0/0**		**3/0/2**	**4/0/1**	**4/0/1**
Average Rank	**1.2**	**4.2**	**3.6**	**3.2**	**2.8**

The *p*-value was 0.033, which necessitated the running of the Nemenyi test. In the comparison using the CD, the Full and MICFastCR sets have significant differences (4.2-1.2 = 3>2.728).

The pairwise comparison values are shown in Table 6.25.

Table 6.25 True Positives *p*-values in Naïve Bayes

	Full	MICFastCR	ReliefF	LCorrel
MIC	0.023	-	-	-
ReliefF	0.115	0.975	-	-
LCorrel	0.266	0.855	0.995	-
InfoGain	0.497	0.628	0.931	0.995

The MICFastCR algorithm has the best performance in the experiment conducted using the PART classifier experiment. It has the highest average True Positives value contrasted with other attribute selection algorithms as depicted in Table 6.26.

Table 6.26 True Positives using PART

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.982	0.998	0.989	0.998	0.993
Lucene	0.985	0.99	0.988	0.99	0.993
Mylyn	0.989	0.995	0.992	0.993	0.994
PDE	0.989	0.997	0.991	0.991	0.998
JDT	0.801	0.849	0.849	0.849	0.849
Average	**0.949**	**0.966**	**0.962**	**0.964**	**0.965**
W/D/L	**5/0/0**		**4/1/0**	**2/3/0**	**2/1/2**
Average Rank	**1**	**4.1**	**2.4**	**3.4**	**4.1**

The p-value was 0.009 and therefore the Nemenyi test was conducted. The Full and MICFastCR differences were significant (4.1- 1 = 3.1 > 2.728). Differences are also observed between the Full and InfoGain set (4.1-1.0 = 3.1 > 2.728).

In the J48 machine-learning algorithm test, the ReliefF has the most wins as indicated by the Win/Draw/Loss data. The records are shown in Table 6.27 below.

Table 6.27 True Positives Using J48

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Equinox	0.873	0.899	0.991	0.85	0.991
Lucene	0.991	0.996	0.995	0.993	0.993
Mylyn	0.991	0.998	0.998	0.999	0.999
PDE	0.889	1	0.996	0.997	1
JDT	0.827	0.849	0.85	0.849	0.849
Average	**0.914**	**0.948**	**0.966**	**0.938**	**0.966**
W/D/L	**5/0/0**		**2/1/2**	**3/1/1**	**1/2/2**
Average Rank	**1.2**	**3.6**	**3.6**	**2.8**	**3.8**

The test using the Nemenyi critical values produced differences that were insignificant (3.6 – 1.2=2.4< 2.728). To verify if the algorithms' performances can be improved by tuning their parameters, the CD was calculated using the Bonferroni-Dunn test at p=0.10. The CD is $2.241\sqrt{\frac{5.6}{6.5}}$ = 2.241, which produces significant differences (3.6-1.2=2.4>2.241). The pairwise p-values are displayed in Table 6.28.

Table 6.28 Nemenyi Test - True Positives using J48

	Full	MICFastCR	ReliefF	LCorrel
MIC	0,022659	NA	NA	NA
ReliefF	0,179597	0,930677	NA	NA
LCorrel	0,855475	0,265889	0,751424	NA
InfoGain	0,930677	0,179597	0,627659	0,999644

The Win/Draw/Loss records from the experiments indicate that the MICFastCR had the best overall performance. The statistic difference results confirm the performance differences among the feature selection algorithms. Tests were conducted using the same feature selection algorithms, but on different sets of process metrics to test the validity of the results.

6.4 Threats to validity

In the recent years, machine-learning research has realised the need for the validation of the experiment results. This can be due to maturity in the research area and the increase in the design of real world applications (Demsarˇ 2006:1-30). This section determines if this study was valid by investigating the variables that influence this research and the generalisability of the results. Aspects that may affect the validity of a research must be controlled, as they may affect the validity of a research. The datasets used to verify the validity of this research are available on Github. The datasets are described in Table 6.29.

Table 6.29 Validation Test Dataset

Dataset	Description
Android	Linux kernel based Mobile Operating System developed by Google
Antlr4	Tool and supports building of lexers and parsers
Broadleaf	Open source Java eCommerce platform
Ceylon	This Java application is object oriented and has high readability
ElasticSearch	Distributed search and analytics engine
Hazelcast	Open source in memory data grid based on Java
Junit	Unit testing framework for Java

6.4.1 Threats to internal validity

This pertains to the research's ability to establish if cause and consequence relationships exist between independent variables and one or more dependent variables. To avoid bias in a classifier selection in this study, three classification algorithms derived from different categories were applied in software defect prediction. The Naïve Bayes is a probabilistic model, PART is a rule-based classification algorithm and J48 is a decision tree based classifier. The experimental and validation data were selected from separate repositories.

6.4.2 Threats to external validity

This investigates the possibility of generalising the research, (i.e. if the results of a sample in the study represent the entire population). The experiments were performed on groups of datasets, to determine the generality of the methods.

6.4.3 Construct validity

Construct validity describes how well a test measures what it claims to be measuring. The Percentage Accuracy, AUC and F-Measure performances were used to check the validity of this research.

Percentage accuracy

The algorithms that have good Win/Draw/Loss records in the experiment conducted using the Naïve Bayes classifier are the MICFastCR and ReliefF, see Table 6.30.

Table 6.30 Perc Accuracy using Naïve Bayes (Validation)

Project	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Android	67.529	70.68	98.116	77.412	68.81
Antlr4	94.523	96.629	94.607	97.573	94.966
Broadleaf	84.834	97.148	94.157	95.04	92.341
Ceylon	91.644	92.272	95.049	93.248	91.5
Elastic	86.598	98.021	94.337	86.144	92.619
Hazelcast	93.744	96.292	88.907	95.947	95.35
Junit	86.787	97.318	98.517	90.669	99.641
Average	**86.523**	**92.623**	**94.813**	**90.862**	**90.747**
W/D/L	**7/0/0**		**4/0/3**	**4/0/3**	**6/0/1**
Average Rank	**1.43**	**4.0**	**3.43**	**3.43**	**2.71**

The Chi-square value in the Friedman rank sum test is 10.971, the *p*-value is 0.0269. The CD at *p*=0.05 is $2.728\sqrt{\frac{5.6}{6.7}}$ = 2.306. In this experiment, the Full and MICFastCR sets differences are significant (4.0-1.43 = 2.57>2.306). The differences between the Full sets and other algorithms are insignificant. The Nemenyi test results indicate that the statistical differences are caused by the MICFastCR and Full sets that have a *p*-value difference of 0.023.

6.31 Nemenyi Test-Perc Accuracy using Naïve Bayes (Validation)

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.023	NA	NA	NA
ReliefF	0.180	0.931	NA	NA
LCorrel	0.855	0.266	0.751	NA
InfoGain	0.931	0.180	0.628	1.000

In another experiment conducted using the PART classification algorithm, the MICFastCR method has the best Win/Draw/Loss record. The outcome is presented in Table 6.32.

Table 6.32 Perc Accuracy using PART (Validation)

	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Android	82.941	87.124	99.338	86.039	85.268
Antlr4	96.449	97.213	96.712	89.079	96.854
Broadleaf	98.368	99.3	96.09	99.561	98.628
Ceylon	97.569	98.341	99.067	97.139	97.712
Elastic	97.113	99.175	97.883	98.144	96.371
Hazelcast	95.959	99.775	98.872	99.699	98.327
Junit	96.933	98.421	99.943	99.589	97.091
Average	**95.047**	**97.050**	**98.272**	**95.607**	**95.750**
W/D/L	**7/0/0**		**4/0/3**	**5/0/2**	**7/0/0**
Average Rank	**1.57**	**4.29**	**3.57**	**2.71**	**2.42**

There are statistical differences in the performances of the Full and MICFastCR sets (4.29-1.57=2.57>2.306). Validation tests were also conducted using the J48 classification algorithm, see Table 6.33.

Table 6.33 Perc Accuracy using J48 (Validation)

	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Android	82.928	99.338	87.83	86.039	85.268
Antlr4	96.404	97.712	97.303	97.079	97.079
Broadleaf	97.489	96.18	99.3	99.39	98.556
Ceylon	97.11	99.067	98.57	97.997	97.712
Elastic	96.907	97.797	99.175	97.938	96.082
Hazelcast	97.185	99.689	99.713	95.052	98.423
Junit	96.783	99.924	98.421	99.334	97.142
Average	***94.972***	***98.530***	***97.187***	***96.118***	***95.752***
W/D/L	***6/0/1***		***4/0/3***	***4/0/3***	***6/0/1***
Average Rank	***1.429***	***3.857***	***3.857***	***3.429***	***2.429***

The Friedman's test output has a Chi-squared value of 11.543 and a *p*-value of 0.0211. The *p*-value demonstrates that there is a statistical difference in the performances of the feature selection algorithms. The MICFastCR and Full sets' differences using the CD are significant (3.857-1.429=2.429>2.306).

Area Under the ROC Curve

Performance measures in terms of the AUC were calculated for use in validation tests. In the Naïve Bayes experiment, the MICFastCR has the most wins, see Table 6.34.

Table 6.34 AUC using Naive Bayes (Validation)

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Android	0.851	0.827	0.789	0.998	0.806
Antlr4	0.82	0.98	0.968	0.721	0.959
Broadleaf	0.841	0.978	0.885	0.944	0.807
Ceylon	0.816	0.949	0.913	0.94	0.867
Elastic	0.803	0.882	0.794	0.97	0.769
Hazelcast	0.835	0.975	0.965	0.872	0.957
Junit	0.844	0.94	0.987	0.832	0.904

Average	***0.830***	***0.933***	***0.900***	***0.897***	***0.867***
W/D/L	***6/0/1***		***6/0/1***	***5/0/2***	***7/0/0***
Average Rankings	***2.1***	***4.4***	***3.1***	***3.1***	***2.1***

The Friedman results shows a chi-squared value of 9.943 and a *p*-value of 0.041. The Full and MIC sets difference is approximately equal to the critical value (4.4-2.1=2.3 ≈ 2.306).

Table 6.35 shows that the difference between the pairs of sets.

Table 6.35 Nemenyi Test - AUC using Naïve Bayes (Validation)

	Full	MICFastCR	ReliefF	LCorrel
MICFastCR	0.022659	NA	NA	NA
ReliefF	0.179597	0.930677	NA	NA
LCorrel	0.855475	0.265889	0.751424	NA
InfoGain	0.930677	0.179597	0.627659	0.999644

An experiment was also conducted using the PART classifier. The MICFastCR has the greatest wins and the least losses, see Table 6.36.

Table 6.36 AUC Using PART (Validation)

	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Android	0.718	0.829	0.799	0.852	0.752
Antlr4	0.572	0.994	0.852	0.6	0.742
Broadleaf	0.878	0.759	0.997	0.654	0.984
Ceylon	0.597	0.92	0.912	0.931	0.818
Elastic	0.619	0.743	0.528	0.614	0.613
Hazelcast	0.651	0.977	0.912	0.899	0.866
Junit	0.733	0.986	0.829	0.814	0.916
Average	***0.681***	***0.887***	***0.833***	***0.766***	***0.813***
W/D/L	***6/0/1***		***6/0/1***	***5/0/2***	***6/0/1***
Average Rank	***1.7***	***4.3***	***3.3***	***3.0***	***2.7***

The Friedman result has a Chi-squared value of 3.31 and *p*-value of 0.50. The differences between the MICFastCR and Full sets are statistically significant (4.3-1.7=2.4>2.306).

In the validation test that was conducted using the J48 classification algorithm, the InfoGain set has more wins than the ReliefF and Linear Correlation.

Table 6.37 AUC using J48 (Validation)

	Full	MICFastCR	ReliefF	Linear Correlation	Info Gain
Android	0.61	0.829	0.747	0.712	0.677
Antlr4	0.497	0.794	0.5	0.65	0.751
Broadleaf	0.747	0.833	0.897	0.5	0.879
Ceylon	0.831	0.907	0.873	0.736	0.818
Elastic	0.5	0.558	0.612	0.7	0.782
Hazelcast	0.505	0.871	0.853	0.844	0.871
Junit	0.774	0.724	0.829	0.81	0.87
Average	***0.638***	***0.788***	***0.759***	***0.707***	***0.807***
W/D/L	***6/0/1***		***4/0/3***	***5/0/2***	***3/1/3***
Average Rank	***1.6***	***3.6***	***3.6***	***2.4***	***3.8***

The Friedman has a *p*-value of 0.033 and smaller than the level of significance. The InfoGain had the highest average ranking and the same number of wins and losses as the MICFastCR method. The differences between the Full and InfoGain sets are insignificant at *p*=0.05 (3.8-1.6=2.2<CD value 2.306).

F-Measure

Experiments were also conducted using the F-Measure to test the generalisation of results. Table 6.38 reveals that the proposed ReliefF subset produced the best outcome. It has the most wins when compared to other algorithms.

Table 6.38 F-Measure Using Naïve Bayes (Validation)

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Android	0.743	0.992	0.801	0.857	0.802
Antlr4	0.975	0.977	0.984	0.988	0.973
Broadleaf	0.961	0.970	0.986	0.975	0.654
Ceylon	0.958	0.975	0.962	0.965	0.957
Elastic	0.906	0.971	0.991	0.911	0.960
Hazelcast	0.967	0.932	0.982	0.980	0.977
Junit	0.927	0.998	0.996	0.953	0.998
Average	**0.920**	**0.974**	**0.957**	**0.947**	**0.903**
W/D/L	**6/0/1**		**3/0/4**	**4/0/3**	**5/1/1**
Average Rank	**1.6**	**3.6**	**3.9**	**3.6**	**2.4**

The Nemenyi test was run since the p-value was 0.026. The test results are in Table 6.45. The CD, Full and ReliefF sets differences are insignificant at p=0.05 (3.6-1.6=2.0 < 2.306). However the performances were significant at p=0.10. The critical value for 5 classifiers is 2.241 at p=0.10. The CD at p=0.10 is $2.241\sqrt{\frac{5.6}{6.7}}$ = 1.89. The differences between the Full and MICFastCR sets are 3.6-1.6=2>1.89(CD value).

The F-Measure performance was also assessed using the PART machine-learning algorithm, see Table 6.39.

Table 6.39 - F-Measure using PART (Validation)

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Android	0.901	0.997	0.942	0.926	0.934
Antlr4	0.982	0.994	0.986	0.985	0.984
Broadleaf	0.992	0.98	0.997	0.998	0.971
Ceylon	0.989	0.996	0.993	0.986	0.99
Elastic	0.986	0.99	0.996	0.991	0.981
Hazelcast	0.979	0.994	0.999	0.998	0.992
Junit	0.984	1	0.993	0.985	0.998
Average	**0.973**	**0.993**	**0.987**	**0.981**	**0.979**
W/D/L	**6/0/1**		**4/0/3**	**4/0/3**	**7/0/0**
Average Rank	**1.6**	**4.0**	**4.0**	**2.9**	**2.3**

The overall *p*-value is 0.038 and this called for the Nemenyi test to be run. The MICFastCR and ReliefF had equal average ranking of 4.0. Their static difference with the Full set is significant (4.0-1.6=2.4>2.306 (CD value)).

The F-Measure was also calculated in an experiment that was run on the J48 classification algorithm. As shown in Table 6.40, the ReliefF has a better score on the Broadleaf, Elastic and Hazelcast datasets. However, the MICFastCR has the best overall performance.

Table 6.40 F-Measure using J48 (Validation)

	Full	MICFastCR	ReliefF	LCorrel	InfoGain
Android	0.904	0.997	0.944	0.926	0.934
Antlr4	0.982	0.994	0.986	0.985	0.985
Broadleaf	0.987	0.981	0.997	0.997	0.969
Ceylon	0.991	0.996	0.994	0.985	0.99
Elastic	0.984	0.996	0.989	0.99	0.98
Hazelcast	0.986	0.999	0.975	0.998	0.992
Junit	0.997	1	0.993	0.985	0.985
Average	**0.976**	**0.995**	**0.983**	**0.981**	**0.976**
W/D/L	**6/0/1**		**5/0/2**	**5/0/2**	**7/0/0**
Average Rank	**2.1**	**4.6**	**3.4**	**2.8**	**2.1**

The *p*-value is 0.018 (less that the threshold of 0.05%), therefore the Nemenyi test was run. The difference of the rankings between the MICFastCR and Full sets is (4.6–2.1=2.5>2.306), therefore the performances are statistically significant.

The results from the validation experiments confirm that, in general there exists a significant difference between the performances of the MICFastCR, ReliefF, Information Gain and Linear Correlation algorithms. The validity results confirm the generality of the results. Statistical tests give reassurance concerning the validity and non-randomness of the outcome from experiments.

6.5 Chapter summary

This study proposed and experimentally evaluated the performances of the proposed Maximal Information Coefficient, Information Gain, Linear Correlation and ReliefF feature selection methods. The Friedman tests were used to analyse the results from the experiments. The statistical significance of the differences was assessed using the Nemenyi tests.

The experimental results reveal that the proposed MICFastCR, based on the Maximal Information Coefficient and FCBF methods produces the most optimal subset followed by the ReliefF, then the Information Gain. In the fraction of features selected and runtime experiments, the ReliefF had the best results compared to the proposed MICFastCR and other algorithms. However, in the Percentage Accuracy, Area Under the ROC Curve and F-Measure experiments, the new MICFastCR method outperforms most algorithms with statistical significance.

This implies that the predictions using attributes selected by the MICFastCR methods are more accurate. The next chapter discusses and presents the conclusion and future work.

CHAPTER 7

DISCUSSION AND CONCLUSION

7.1 Introduction

In machine learning, feature selection methods are employed in the identification of significant and non-redundant data. Data may be inconsistent and irrelevant and so must be cleansed. The selection of attributes is a pre-processing phase that helps reduce the dimensionality of data, thereby improving the prediction accuracy.

In this study, experiments were conducted using defect data obtained from an online repository. An organised review of relevant literature on software metrics and software defect prediction identified the approaches used in predicting defects in single version systems and software product lines. The purpose of this research was to present and evaluate the predictive capability of a hybrid algorithm invented using the Maximal Information Coefficient and FCBF. In order to establish this, feature selection algorithms, including the MICFastCR were applied in the selection of relevant and non-redundant defect data. The classification models, the Naive Bayes, PART and J48 served as the predictors. The performance of these machine-learning algorithms was evaluated and compared. The study took the statistical significances of results into consideration.

7.2 Discussion

The efficiency of the proposed hybrid model MICFastCR, using Maximal Information Coefficient and FCBF algorithms was evaluated. The most significant attributes were selected using the MICFastCR, ReliefF, Information Gain and Linear Correlation algorithms. These most important attributes were analysed by the machine learning algorithms in the software defect prediction process. The Friedman and Nemenyi tests were used to test the statistical significance of the results. The research also measured the proportion of attributes selected by the algorithms and the running time of the algorithms.

The outcome proved that the differences were statistically insignificant. The Percentage Accuracy, Area Under ROC curve, F-Measure results were statistically significant. The outcome of the study demonstrates the effectiveness of MICFastCR feature selection algorithm, when compared with the ReliefF, Linear Correlation and Information Gain algorithms.

The Maximal Information Coefficient is currently the best information theoretic technique (Kinney & Atwal 2014: 3354-3359). The MIC captures functional and non-functional associations. This method also has the advantage of resisting noise (Reshef et al., 2011:1518-1524). In previous studies conducted using the MIC, results indicate that the technique is a great measure of relevance (Zhao, Deng & Shi 2013: 70-79; Xu et al., 2016: 370-381).

There is a need to study and understand the complexity of the ever-increasing technology driven data sets. The MIC has the ability to inspect the relationships in data sets. As the MIC improved the software defect accuracy in this research, it should be used in other algorithms to determine if they can be more effective in prediction analysis.

7.3 Contribution to knowledge

This study considered questions stated in (section 1.9). These questions are briefly stated below.

RQ1. Which metrics are suitable for predicting defects in the versions of a software product line?

Literature review suggests that process metrics, though difficult to gather, are more accurate in predicting post-release defects in software, compared to other types of metrics. The metrics capture the modifications made to software. In this study, process metrics also known as historical metrics were selected to predict in software product lines.

RQ2*: Which information theoretic methods have been used in previous research*?

The techniques that are based on the entropy concept, include the Information Gain, Mutual Information, Symmetric Uncertainty and the Maximal Information Coefficient. The information concept $H(X) = -\sum_i P(x_i) log_2\big(P(x_i)\big)$ measures the uncertainty of an attribute.

RQ3: How is the performance of the information theory based methods compared to other algorithms?

Previous studies have used feature selection techniques for defect prediction. These include statistical, information theoretic, instance-based and probabilistic methods. The results from the tests indicate that the information-based theories are more accurate. The Maximal Information Coefficient selected significant features that resulted in high performance of the classifiers.

RQ4: Are the data-mining techniques consistently effective in predicting defects?

In this study, the Naïve Bayes, PART and J48 classifiers were applied in the prediction process. The performance of all the three classifiers was relatively consistent. The PART and J48 had the good prediction accuracy, notably in the AUC, TP, RMSE and F-Measure performance measures.

RQ5: How can a data redundancy removal technique be derived from the concept of predominant correlation?

Non-redundant attributes are selected from the list of relevant ones using the predominant correlation.

A feature f_i is said to be redundant *iff* f_j is an existing predominant feature,$SU(f_j, c) \geq SU(c, f_i)$ and $SU(f_i, f_j) \geq SU(f_i, c)$.

RQ6: How can a model that will predict defects in the next versions of the software applications be derived?

The proposed hybrid model selects significant attributes using the Maximal Information Coefficient. The approach shows that relevant feature selection and reduced dimensionality improved the classifiers' prediction accuracy. Redundant features are eliminated using the technique derived from

the FCBF. Experiments that were conducted proved that the method can effectively predict defects, achieving F-Measure values between 74.7% to 93.7% across all datasets.

7.4 Limitations of the study

The study had certain limitations. The datasets used in this study were obtained from a single OSS website (D'Ambros et al. 2012: 531-577). The validation data was also retrieved from open source websites. Data from company repositories was not included in this study. The results may be biased towards the defect reporting patterns of open source systems. Differences in program design may affect the applicability of results in the industry (Ullah & Khan 2011: 98-108).

The other limitation was that the project consisted of data from popular applications, Apache and Eclipse systems. Most of the validation data was also from popular systems. The less popular projects, which may apply dissimilar defect classification and resolution practices were excluded. Further, research will predict defects in both common and less known applications.

In this research, attributes were only selected using filters. Other feature selection methods such as embedded methods and wrappers were not included.

7.5 Conclusion

In this study, a new method for selecting attributes to be used in defect prediction was presented. The proposed MICFastCR algorithm is a hybrid method that selects significant features, for software defect prediction using the Maximal Information Coefficient. It eliminates redundant features based on the FCBF algorithm. The proposed algorithm and other widely known feature selection algorithms, Linear Correlation, Information Gain, Maximal Information Coefficient and ReliefF were applied on the same open source datasets.

In the experiments that were conducted in this study, the proposed algorithm outperformed other algorithms in most of the measures. The validity of the results was tested by conducting a study, where the same feature selection and classification algorithms were used on different open source

datasets. Similar results were observed. Generally, the MICFastCR algorithm is ideal in the improvement of classification accuracy in software projects. The ReliefF algorithm results were fairly good.

7.6 Future work

This study deliberated on information theoretic filters for selecting features. Other feature selection processes such as wrappers and embedded methods, discussed in sections 5.92 and 5.93, will be explored in the future work.

The interaction of information theoretic and search methods will be studied. Search methods that apply learning algorithms to improve the searching process have been used (Liu, Lin, Lin, Wu & Zhang 2017:11-22; Kannan & Ramaraj 2015:580-585) in previous studies. The future research will investigate if such methods can interrelate with information theoretic feature selection methods. The cost-effectiveness, including inspection costs of the algorithms will be investigated and compared to complement prediction accuracy. Improved accuracy does not imply better performance in terms of cost-effectiveness (Zhang & Cheung 2013: 643-646).

REFERENCES

Abaei, G. & Selamat, A., 2013. A survey on software fault detection based on different prediction approaches. *Vietnam Journal of Computer Science*, 1(2), pp.79–95. Available at: http://link.springer.com/10.1007/s40595-013-0008-z.

Abraham, R. & Simha, J.B., 2007. Medical datamining with a new algorithm for Feature Selection and Naïve Bayesian classifier. , pp.44–49.

Adikari, S., McDonald, C. & Campbell, J., 2009. Little Design Up-Front : A Design Science Approach to Integrating Usability into Agile Requirements Engineering. In J. . Jacko, ed. *Human-Computer Interaction*. Heidelberg: Springer-Verlag Berlin, pp. 549–558.

Agarwal, B. & Mittal, N., 2013. Optimal Feature Selection for Sentiment Analysis. *Springer-Verlag*, pp.13–24.

Ahmed, F., Mahmood, H. & Aslam, A., 2014. Green computing and Software Defects in open source software: An Empirical study. *ICOSST 2014 - 2014 International Conference on Open Source Systems and Technologies, Proceedings*, pp.65–69.

Aleem, S., Capretz, L.F. & Ahmed, F., 2015. BENCHMARKING MACHINE LEARNING TECHNIQUES FOR SOFTWARE DEFECT DETECTION. *International Journal of Software Engineering and Applications*, 6(3), pp.11–23.

Alemerien, K. & Magel, K., 2014. Examining the effectiveness of testing coverage tools: An empirical study. *International Journal of Software Engineering and its Applications*, 8(5), pp.139–162.

Aloysius, A. & Arockiam, L., 2012. Coupling Complexity Metric : A Cognitive Approach. *I.J. Information Technology and Computer Science,*, (August), pp.29–35.

Alshayeb, M., Eisa, Y. & Ahmed, M.A., 2014. Object-Oriented Class Stability Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine. *Arabian Journal for Science and Engineering*, 39(11), pp.7865–7876.

Ambros, M.D., Lanza, M. & Robbes, R., 2010. An Extensive Comparison of Bug Prediction Approaches. *IEEE*, pp.31–41.

Anand, P., 2015. Test_Cases. *IEEE*, pp.1111–1117.

Antony, D.A. & Singh, G., 2016. Dimensionality Reduction using Genetic Algorithm for Improving Accuracy in Medical Diagnosis. , (January), pp.67–73.

Babar, M., Vierimaa, M. & Oivo, M., 2010. Product-Focused Software Process. In *11th International Conference, PROFES 2010*. Limerick, pp. 1–407.

Bafna, P., Metkewar, P. & Shirwaikar, S., 2014. Novel Clustering approach for Feature selection. *American International Journal of Available online at http://www.iasir.net Research in Science, Technology, Engineering & Mathematics*, pp.62–67.

Barb, A.S. et al., 2014. A statistical study of the relevance of lines of code measures in software projects. *Innovations in Systems and Software Engineering*, 10(4), pp.243–260.

Bell, R.M., Ostrand, T.J. & Weyuker, E.J., 2013. The limited impact of individual developer data on software defect prediction. *Empirical Software Engineering*, 18(3), pp.478–505.

Bernstein, A., Ekanayake, J. & Pinzger, M., 2007. Improving Defect Prediction Using Temporal Features and Non Linear Models. *ACBernstein, A., Ekanayake, J. & Pinzger, M. 2007. Improving Defect Prediction Using Temporal Features and Non Linear Models. ACM.M*, pp.1–8.

Bettenburg, N. & Hassan, A.E., 2013. *Studying the impact of social interactions on software quality*,

Blake, C.. & Merz, C.., 1998. UCI repository of machine learning databases. Available at: http://www.ics.uci.edu/- mlearn/MLRepository. html.

Boehm, B. & Basili, V., 2005. Software Defect Reduction Top-10 List. In *Foundations of Empirical Software Engineering*. Springer, pp. 426–431.

Bolón-canedo, V., Sánchez-maroño, N. & Alonso-Betanzos, A., 2013. A review of feature selection methods on synthetic data. *Knowl Inf Syst*, pp.483–519.

Botterweck, G. & Pleuss, A., 2014. Evolution of Software Product Lines. In *Evolving Software*

Systems.

Bowes, D. et al., 2016. Mutation-Aware Fault Prediction. *ACM*, pp.330–341.

Caglayan, B. et al., 2015. Predicting defective modules in different test phases. *Software Quality Journal*, 23(2), pp.205–227.

Caglayan, B. et al., 2010. Usage of Multiple Prediction Models Based On Defect Categories. *ACM*, pp.1–9.

Capgemini Group, 2017. *WORLD QUALITY*, Capgemini.

Catal, C. & Diri, B., 2008. A fault prediction model with limited fault data to improve test process. *Springer-Verlag Berlin*, 5089 LNCS, pp.244–257.

Cavezza, D.G., Pietrantuono, R. & Russo, S., 2015. Performance of Defect Prediction in Rapidly Evolving Software. In *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. pp. 8–11. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7169444.

Cendrowska, J., 1987. PRISM : An algorithm for inducing modular rules. *Man-Machine Studies*, pp.349–370.

Chawla, S. & Nath, R., 2013. Evaluating Inheritance and Coupling Metrics. *International Journal of Engineering Trends and Technology (IJETT)*, 4(July), pp.2903–2908.

Chidamber, S. & Kemerer, C.F., 1991. Towards a Metrics Suite for Object Oriented Design. *ACM*, (1), pp.197–211.

Cochez, M. et al., 2013. How Do Computer Science Students Use Distributed Version Control Systems ? In *ICTERI 2013*. pp. 210–228.

Coelho, R.A., Guimaraes, F. dos R.N. & Esmin, A.A.A., 2014. Applying Swarm Ensemble Clustering Technique for Fault Prediction Using Software Metrics. In *2014 13th International Conference on Machine Learning and Applications*. pp. 356–361. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84924940251&partnerID=tZOtx3y1.

D'Ambros, M., Lanza, M. & Robbes, R., 2012. *Evaluating defect prediction approaches: A benchmark*

and an extensive comparison,

Daniel, B. & Boshernitsan, M., 2008. Predicting effectiveness of automatic testing tools. *ASE 2008 - 23rd IEEE/ACM International Conference on Automated Software Engineering, Proceedings*, 1, pp.363–366.

Davis, G., 2005. Advising and Supervising in Research in Information Systems. In D. Avison & J. Pries-Heje, eds. *A Handbook for Research Supervisors and Their Students*. Elsevier Butterworth Heinemann, pp.1–25.

Demsarˇ, J., 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7, pp.1–30.

Doshi, M. & Chaturvedi, S.K., 2014. Correlation Based Feature Selection (CFS) Technique to Predict Student Performance. *International Journal of Computer Networks & Communications (IJCNC)*, 6(3), pp.197–206.

Duarte, C.H.C., 2014. On the relationship between quality assurance and productivity in software companies. *Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry - CESI 2014*, pp.31–38. Available at: http://dl.acm.org/citation.cfm?doid=2593690.2593692.

Efron, B., 1979. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, pp.1–26.

Efron, B. & Tibshirani, R.., 1993. An Introduction to the Bootstrap. *Springer US*.

Ellerman, D., 2009. Counting distinctions : on the conceptual foundations of Shannon ' s information theory. *Springer*, pp.119–149.

Erfanian, A. & Darav, N.K., 2012. CBM-Of-TRaCE : An Ontology-Driven Framework for the Improvement of Business Service Traceability , Consistency Management and Reusability. *International Journal of Soft Computing And Software Engineering*, 2(7), pp.69–78.

Erturk, E. & Sezer, E.A., 2015. Software Fault Inference Based on Expert Opinion. *Journal of Software*, 10(6), pp.757–766.

Fan, Y., Li, X.I.N. & Zhang, P., 2017. Integrated Approach for Online Dynamic Security Assessment With Credibility and Visualization Based on Exploring Connotative Associations in Massive Data. *IEEEAccess*, 5, pp.16555–16567.

Fayyad, U. & Irani, K.B., 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. *Machine Learning*, pp.1022–1027.

Fenton, N.E. & Neil, M., 1999. Neil, M.: A critique of software defect prediction models. IEEE Trans. SW Eng. 25, 675-689. *IEEE Transactions on Software Engineering*, (June 2013), pp.675–689.

Ferzund, J., Ahsan, S.N. & Wotawa, F., 2008. Analysing bug prediction capabilities of static code metrics in open source software. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5338 LNCS, pp.331–343.

Fischer, G.W. & Dyer, J.S., 1998. Attribute weighting methods and decision quality in the presence of response error : A simulation study. *Journalin Behavioral Decision Making*, (June), pp.85–102.

Friedman, M., 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(200), pp.675–701.

Fukushima, T. et al., 2014. An empirical study of just-in-time defect prediction using cross-project models. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pp.172–181. Available at: http://dl.acm.org/citation.cfm?doid=2597073.2597075.

Gao, K. et al., 2011. Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms. *Software - Practice and Experience*, 41, pp.579–606.

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

Ghanta, S.K. & Rao, G.U.M., 2015. Handling High Dimensional Data using Novel Feature Subject Selection Algorithm for Clustering Problem. *International Journal of Advanced Technology and Innovative Research*, 7(2), pp.300–303.

Ghotra, B., Mcintosh, S. & Hassan, A.E., 2015. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. pp. 789–800.

Gill, N.S., 2005. Factors Affecting Effective Software Quality Management Revisited. *ACM SIGSOFT Software Engineering Notes*, 30(2), pp.1–4.

Gregor, S. & Hevner, A.R., 2013. Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, 37(2), pp.337–355.

Gupta, A. & Kumar, D., 2017. Fuzzy clustering-based feature extraction method for mental task classification. *Brain Informatics*, 4(2), pp.135–145.

Gupta, P., Jain, S. & Jain, A., 2014. A Review Of Fast Clustering-Based Feature Subset Selection Algorithm. *International Journal of Scientific & Technology Research*, 3(11), pp.86–91.

Guyon, I. & Elisseeff, A., 2003. An Introduction to Variable and Feature Selection 1 Introduction. *Journal of Machine Learning*, 3, pp.1157–1182.

Hall, M.A., 1999. *Correlation-based Feature Selection for Machine Learning*.

Hao, S. et al., 2016. Optimizing Correlation Measure Based Exploratory Analysis. In *2016 8th International Conference on Information Technology in Medicine and Education Optimizing*. IEEE Computer Society, pp. 635–639.

Harter, D.E., Kemerer, C.F. & Slaughter, S.A., 2012. Does software process improvement reduce the severity of defects? A longitudinal field study. *IEEE Transactions on Software Engineering*, 38(4), pp.810–827.

Hartley, R., 1928. Transmission of Information. *Bell System Technical Journal*, pp.535–563.

Hassan, A.E., 2009. Predicting Faults Using the Complexity of Code Changes. In *ICSE 09*. pp. 78–88.

He, Z. et al., 2013. Learning from open-source projects: An empirical study on defect prediction. *International Symposium on Empirical Software Engineering and Measurement*, pp.45–54.

Herzig, K., 2014. Using pre-release test failures to build early post-release defect prediction models. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp.300–311.

Hewett, R., 2011. Mining software defect data to support software testing management. *Applied Intelligence*, 34(2), pp.245–257.

Hneif, M. & Lee, S.P., 2011. Using guidelines to improve quality in software nonfunctional attributes. *IEEE Software*, 28(6), pp.72–77.

HP, 2011. *Short Release Cycles by Bringing Developers to Application Lifecycle Management*,

IEEE - SA Standards Board, 2010. IEEE Computer Society. , pp.1–15.

Islam, R. & Sakib, K., 2014. A Package Based Clustering for enhancing software defect prediction accuracy. In *2014 17th International Conference on Computer and Information Technology (ICCIT)*. pp. 81–86. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7073117.

Johannesson, P. & Perjons, E., 2014. Research Paradigms. In *An Introduction to Design Science*. pp. 1–179.

Johansson, U. & Niklasson, L., 2010. Improving GP Classification by Injection of Decision Trees. *IEEE*, pp.18–23.

John, G.H., Kohavi, R. & Pfleger, K., 1994. Irrelevant Features and the Subset Selection Problem. In *Machine Learning: Proceeds of the Eleventh International Conference*. pp. 121–129.

Jose, J. & Reeba, R., 2014. Fast for Feature Subset Selection Over Dataset. *International Journal of Science and Research*, 3(6), pp.380–383.

Kamei, Y. et al., 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *Transactions on Software Engineering*, 39(6), pp.757–773.

Kannan, S.S. & Ramaraj, N., 2015. A novel hybrid feature selection via Symmetrical Uncertainty ranking based local memetic search algorithm. *Knowledge-Based Systems*, 23(May), pp.580–

585.

Kapur, P.K. & Shrivastava, A.K., 2015. Release and Testing Stop Time of a Software : A New Insight. *IEEE*.

Karthikeyan, T. & Thangaraju, P., 2015. Genetic Algorithm based CFS and Naive Bayes Algorithm to Enhance the Predictive Accuracy. *Indian Journal of Science and Technology*, 8(27), pp.1–8.

Kastro, Y. & Bener, A.B., 2008. A defect prediction method for software versioning. *Software Quality Journal*, 16(4), pp.543–562.

Kaur, A., Kaur, K. & Kaur, H., 2015. An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. *IEEE*, (1–5).

Khan, J.I. et al., 2014. An attribute selection process for software defect prediction. In *3rd International Conference on Informatics, Electronics & Vision 2014*. pp. 1–4. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84904965432&partnerID=tZOtx3y1.

Khoshgoftaar, T.M. & Seliya, N., 2003. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering*, 8(3), pp.255–283.

Kienzle, J., 2003. Software Fault Tolerance : Fault Error. *Springer-Verlag Berlin*, pp.45–67.

Kinney, J.B. & Atwal, G.S., 2014. Equitability , mutual information , and the maximal information coefficient. *PNAS*, (PNAS), pp.3354–3359.

Kpodjedo, S. et al., 2010. Design evolution metrics for defect prediction in object oriented systems. *Empirical Software Engineering*, 16(1), pp.141–175. Available at: http://link.springer.com/10.1007/s10664-010-9151-7.

Kuhn, T.S., 1970. *The Structure of Scientific Revolutions*,

Ladha, L. & Deepa, T., 2011. Feature Selection Methods and Algorithms. *International Journal on Computer Science and Engineering*, 3(5), pp.1787–1797.

Lee, C., Gutierrez, F. & Dou, D., 2011. Calculating Feature Weights in Naive Bayes with Kullback-Leibler Measure. In *2011 11th IEEE International Conference on Data Mining*. pp. 1146–1151.

Lee, R.., 2007. Testing. In *Software Engineering: A Hands-On Approach*. pp. 191–216.

Lee, T. et al., 2016. Developer Micro Interaction Metrics for Software Defect Prediction. *IEEE*, 42(11), pp.1015–1035.

Lehr, T. et al., 2011. Rule based classifier for the analysis of gene-gene and gene-environment interactions in genetic association studies. *BioData Mining*, 4(1), pp.1–14. Available at: http://www.biodatamining.org/content/4/1/4.

Li, C., Zhao, H. & Xu, Z., 2017. Kernel C-Means Clustering Algorithms for Hesitant Fuzzy Information in Decision Making. *International Journal of Fuzzy Systems*, pp.1–10.

Li, M. et al., 2012. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, 19(2), pp.201–230.

Liparas, D., Angelis, L. & Feldt, R., 2012. Applying the Mahalanobis-Taguchi strategy for software defect diagnosis. *Automated Software Engineering*, 19(2), pp.141–165.

Liu, B., 2011. Supervised Learning. In *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data,*. Heidelberg: Springer-Verlag Berlin.

Liu, H., Wu, X. & Zhang, S., 2011. Feature Selection using Hierarchical Feature Clustering. In *CIKM'11*. Glasgow: ACM, pp. 979–984.

Liu, J. et al., 2017. Neurocomputing Feature selection based on quality of information. *Neurocomputing*, 225(June 2016), pp.11–22. Available at: http://dx.doi.org/10.1016/j.neucom.2016.11.001.

Liu, L. et al., 2015. Fuzzy Integral Based on Mutual Information for Software Defect Prediction. In *2015 International Conference on Cloud Computing and Big Data*. IEEE Computer Society, pp. 93–96. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7450536.

Liu, S. et al., 2014. FECAR : A Feature Selection Framework for Software Defect Prediction. In *2014 IEEE 38th Annual International Computers, Software and Applications Conference*. pp. 426–435.

Lu, H., Cukic, B. & Culp, M., 2014. A Semi-supervised Approach to Software Defect Prediction.

Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual, pp.416–425.

Lu, H., Kocaguneli, E. & Cukic, B., 2014. Defect prediction between software versions with active learning and dimensionality reduction. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp.312–322.

Madeyski, L. & Jureczko, M., 2015. Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal*, 23(3), pp.393–422.

Malhotra, R. & Khanna, M., 2013. Investigation of relationship between object-oriented metrics and change proneness. *International Journal of Machine Learning and Cybernetics*, 4(4), pp.273–286.

Mäntylä, M. V. et al., 2015. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5), pp.1384–1425. Available at: http://link.springer.com/10.1007/s10664-014-9338-4.

Mccabe, J., 1976. THOMAS J. McCABE. *IEEE Transactions on Software Engineering*, (4), pp.308–320.

McIntosh, S., Adams, B. & Hassan, A.E., 2012. The evolution of Java build systems. *Empirical Software Engineering*, 17(4–5), pp.578–608.

Mende, T. & Koschke, R., 2009. Revisiting the Evaluation of Defect Prediction Models. *ACM*, pp.1–10.

Menzies, T. et al., 2010. Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering*, 17(4), pp.375–407.

Misirli, A.T., Bener, A.B. & Turhan, B., 2011. An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, 19(3), pp.515–536.

Molina, L.C., Belanche, L. & Nebot, A., 2002. Evaluating Feature Selection Algorithms. In *Topics in Artificial Intelligence*. Springer Berlin Heidelberg, pp. 216–227.

Monden, A. et al., 2012. A Heuristic Rule Reduction Approach to Software Fault-proneness Prediction. , pp.838–847.

Moser, R., Pedrycz, W. & Succi, G., 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proceedings of the 13th international conference on Software engineering - ICSE '08*, pp.181–190. Available at: http://portal.acm.org/citation.cfm?doid=1368088.1368114.

Muthukumaran, K., Choudhary, A. & Murthy, N.L.B., 2015. Mining github for novel change metrics to predict buggy files in software systems. *Proceedings - 1st International Conference on Computational Intelligence and Networks, CINE 2015*, pp.15–20.

Natarajan, S. et al., 2015. Implementation of Clustering Based Feature Subset Selection Algorithm for High Dimensional Data. *International Journal of Computer Science and Information Technology Research*, 3(3), pp.366–372.

Nguyen, T.T., Nguyen, T.N. & Phuong, T.M., 2011. Topic-based defect prediction. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. pp. 932–935. Available at: http://portal.acm.org/citation.cfm?doid=1985793.1985950.

Novakovic, J., Strbac, P. & Bulatovic, D., 2011. Toward Optimal Feature Selection Using Ranking Methods and Classification Algorithms. *Yogoslav Journal of Operations Research*, 21(1), pp.119–135.

Ogasawara, H., Yamada, A. & Kojo, M., 1996. Experiences of the Software Quality Management Using Metrics through the Life-Cycle. In *Proceedings of IEEE 18th International Conference on Software Engineering*. pp. 179–188. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=493414.

Pagallo, G. & Haussler, D., 1990. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5, pp.71–99.

Pakkar, M.S., 2016. An integrated approach to grey relational analysis , analytic hierarchy process and data envelopment analysis. *Emerald*, 9(1), pp.71–86.

Pan, S.J., 2014. Transfer Learning. In *Data Classification: Algorithms and Applications*. pp. 537–570.

Pandeeswari, L. & Rajeswari, K., 2015. K-Means Clustering and Naive Bayes Classifier For Categorization Of Diabetes Patients. , 2(1), pp.179–185.

Paramshetti, P. & Phalke, D.A., 2014. Survey on Software Defect Prediction Using Machine Learning Techniques. *International Journal of Science and Research*, 3(12), pp.1394–1397.

Peffers, K. et al., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45–78.

Pelayo, L. & Dick, S., 2012. Evaluating stratification alternatives to improve software defect prediction. *IEEE Transactions on Reliability*, 61(2), pp.516–525.

Peng, H., Long, F. & Ding, C., 2005. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8), pp.1226–1238.

Pushphavathi, T.P., Suma, V. & Ramaswamy, V., 2014. A novel method for software defect prediction: Hybrid of FCM and random forest. *Electronics and Communication Systems (ICECS), 2014 International Conference on*, pp.1–5.

Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. In Morgan Kaufman.

Quinlan, J.R., 1986. Induction of Decision Trees. *Machine Learning*, pp.81–106.

Rahman, F. & Devanbu, P., 2013. How, and why, process metrics are better. In *Proceedings - International Conference on Software Engineering*. pp. 432–441.

Rana, Z.A., Awais, M.M. & Shamail, S., 2014. Impact of Using Information Gain in Software Defect Prediction Models. In *ICIC 2014*. Springer International Publishing, pp. 637–648.

Rathi, V.. G. & Palani, S., 2012. A Novel Approach for Feature Extraction and Selection on MRI Images for Brain Tumor Classification. In *CS & IT 05*. pp. 225–234.

Rathore, S.S. & Gupta, A., 2014. A Comparative Study of Feature-Ranking and Feature-Subset Selection Techniques for Improved Fault Prediction. *ACM*, pp.1–10.

Rathore, S.S. & Kumar, S., 2016. A decision tree logic based recommendation system to select software fault prediction techniques. *Computing*, pp.1–31.

Regha, R.S. & Rani, R.U., 2015. A Novel Clustering based Feature Selection for Classifying Student Performance. *Indian Journal of Science and Technology*, 8(April), pp.135–140.

Reshef, D.N. et al., 2011. Detecting Novel Associations in Large Data Sets. *Science*, 334, pp.1518–1524.

Ricky, M.Y., Purnomo, F. & Yulianto, B., 2016. Mobile Application Software Defect Prediction. *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp.307–313. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7473042.

Rivest, R., 1987. Learning decision lists. *Machine Learning*, pp.229–246.

Romito, N., 2013. New Genetic Algorithm with a Maximal Information Coefficient Based Mutation. *ACM*, pp.1–6.

Roszkowska, E., 2013. Rank ordering criteria weighting methods – a comparative overview. *OPTIMUM. STUDIA EKONOMICZNE*, 5(5), pp.14–30.

Salih, A., Salih, M. & Abraham, A., 2014. Novel Ensemble Decision Support and Health Care Monitoring System. *Journal of Network and Innovative Computing*, 2, pp.41–51.

Sanchez-Morono, N., Alonso-Betanzos, A. & Tombilla-Sanroman, M., 2007. Filter Methods for Feature Selection – A Comparative Study. In *IDEAL 2007*. Springer Berlin Heidelberg, pp. 178–187.

Sawadpong, P. & Allen, E.B., 2016. Software Defect Prediction using Exception Handling Call Graphs : A Case Study. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering Software*. pp. 55–62.

Seliya, N., Khoshgoftaar, T.M. & Van Hulse, J., 2010. Predicting faults in high assurance software. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, pp.26–34.

Setsirichok, D. et al., 2012. Classification of complete blood count and haemoglobin typing data by a C4.5 decision tree, a naïve Bayes classifier and a multilayer perceptron for thalassaemia screening. *Biomedical Signal Processing and Control*, 7, pp.202–212.

Shafiullah, G.M. et al., 2010. Rule-Based Classification Approach for Railway Wagon Health Monitoring. *IEEE*, p.1-.

Shannon, C.E., 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27, pp.379–423.

Sharmin, S., Wadud, M.A.-A. & Nower, N., 2015. SAL : An Effective Method for Software Defect Prediction. In *18th International Conference on Computer and Information Technology (ICCIT)*. pp. 184–189.

Shepperd, M., Bowes, D. & Hall, T., 2014. Researcher Bias : The Use of Machine Learning in Software Defect Prediction. *IEEE Transactions on Software Engineering*, 40(6), pp.603–616.

Shihab, E. et al., 2013. *Studying re-opened bugs in open source software*,

Singh, B., Kushwaha, N. & Vyas, O.P., 2014. A feature Subset Selection Technique for High Dimensional Data Using Symmetric Uncertainty. *Journal of Data Analysis and Information Processing*, (November), pp.95–105.

Singh, M., 2013. Software Defect Prediction Tool based on Neural Network. *International Journal of Computer Applications*, 70(22), pp.22–28.

Singh, P. & Verma, S., 2012. Empirical investigation of fault prediction capability of object oriented metrics of open source software. *IEEE*, pp.323–327.

Singh, P.K. & Sangwan, O.P., 2014. A process metrics based framework for Aspect Oriented Software to predict software bugs and maintenance. *IEEE*, pp.831–836.

Stillwell, W.G., Seaver, D.A. & Edwards, W., 1981. A Comparison of Weight Approximation Techniques in Multiattribute Utility Decision Making. *Organizational Behavior and Human Performance*, pp.62–77.

Sunindyo, W. et al., 2012. LNBIP 94 - Improving Open Source Software Process Quality Based on Defect Data Mining. *Springer-Verlag Berlin*, pp.84–102.

Syer, M.D. et al., 2015. Replicating and re-evaluating the theory of relative defect-proneness. *IEEE Transactions on Software Engineering*, 41(2), pp.176–197.

Taipale, O. et al., 2011. Trade-off between automated and manual software testing. *International Journal of Systems Assurance Engineering and Management*, 2(2), pp.114–125.

Tan, P.-N., Steinbach, M. & Kumar, V., 2006. Cluster Analysis : Basic Concepts and Algorithms. In *Introduction to Data Mining*. pp. 488–567.

Tan, X. et al., 2011. Assessing software quality by program clustering and defect prediction. *Proceedings - Working Conference on Reverse Engineering, WCRE*, pp.244–248.

Thakur, A. & Goel, A., 2016. A Hybrid Neuro Fuzzy Approach for Bug Prediction using Software Metrics. *International Journal of Engineering Trends and Technology (IJETT)*, 38(2), pp.85–92.

Thangaraj, M. & Vijayalakshmi, 2013. Performance Study on Rule-based Classification Techniques across Multiple Database Relations. *International Journal of Applied Information Systems (*, 5(4), pp.1–7.

Thawonmas, R. & Abe, S., 1997. A Novel Approach to Feature Selection Based on Analysis of Class Regions. *IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics*, 27(2), pp.196–207.

Thongtanunam, P. et al., 2016. Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review. *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, (1), pp.1039–1050.

Tiwari, R. & Singh, M., 2010. Correlation-based Attribute Selection using Genetic Algorithm. *International Journal of Computer Applications*, 4(8), pp.28–34.

Tse-Hsun Chen et al., 2012. Explaining software defects using topic models. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. pp. 189–198. Available at:

http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6224280.

Ullah, K. & Khan, S.A., 2011. A Review of Issue Analysis OF ISSUE ANALYSIS IN OPEN SOURCE SOFTWARE DEVELOPMENT. *Journal of Theoretical and Applied Information Technology*, pp.98–108.

Ullah, N., 2015. A method for predicting open source software residual defects. *Software Quality Journal*, 23(1), pp.55–76.

Untan, N. et al., 2014. 2014 2nd International Conference on Technology, Informatics, Management, Engineering & Environment. In *IEEE*. pp. 228–233.

Valles-Barajas, F., 2015. A comparative analysis between two techniques for the prediction of software defects: fuzzy and statistical linear regression. *Innovations in Systems and Software Engineering*, 11(4), pp.277–287.

Veeralakshmi, V., 2015. Ripple Down Rule learner (RIDOR) Classifier for IRIS Dataset. , 4(3), pp.79–85.

Wahono, R.S. & Suryana, N., 2013. Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction. *International Journal of Software Engineering and Its Applications*, 7(5), pp.153–166.

Wang, H., Khoshgoftaar, T.M. & Seliya, N., 2011. How many software metrics should be selected for defect prediction? In *Proceedings of the 24th International Florida Artificial Intelligence Research Society, FLAIRS - 24*. pp. 69–74. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-80052408958&partnerID=tZOtx3y1.

Wang, H. & Liu, S., 2016. An Effective Feature Selection Approach Using the Hybrid Filter Wrapper. *International Journal of Hybrid Information Technology*, 9(1), pp.119–128.

Wang, J., Shen, B. & Chen, Y., 2012. Compressed C4.5 Models for Software Defect Prediction. *2012 12th International Conference on Quality Software*, 2(1), pp.13–16. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84869118624&partnerID=tZOtx3y1.

Watanabe, L. & Rendell, L., 1991. Learning Structural Decision Trees from Examples. *Learning and Knowledge Acquisition*, pp.770–776.

WEKA, 2016. *Weka – Evaluation : Assessing the performance*,

Weyuker, E.J. & Ostrand, T.J., 2008. Do too many cooks spoil the broth ? Using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, (May), pp.1–11.

Wolf, L. & Shashua, A., 2003. Feature Selection for Unsupervised and Supervised Inference : the Emergence of Sparsity in a Weighted-based Approach. In *Ninth IEEE International Conference on Computer Vision (ICCV 03)*. IEEE Computer Society, pp. 0–6.

Wong, T., 2015. Performance evaluation of classi fi cation algorithms by k -fold and leave-one-out cross validation. *Pattern Recognition*, 48, pp.2839–2846.

Wu, W., Gao, Q. & Wang, M., 2006. Extended Fast Feature Selection for Classification Modeling. In *Proceedings of the 10th WSEAS International Conference on COMPUTERS*. pp. 13–18.

Xia, Y., Yan, G. & Zhang, H., 2014. Analyzing The Significance of Process Metrics for TT & C Software Defect Prediction. *IEEE*, pp.77–81.

Xu, Z. et al., 2016. MICHAC : Defect Prediction via Feature Selection based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution and Engineering*. pp. 370–381.

Yang, J. et al., 2016. Iterative ensemble feature selection for multiclass classification of imbalanced microarray data. *Journal of Biological Research - Thessaloniki -BioMedCentral*, 23(1), pp.1–9.

Yang, X. et al., 2015. Deep Learning for Just-in-Time Defect Prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*. pp. 17–26. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7272910.

Yang, Y. et al., 2015. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *IEEE Transactions on Software Engineering*,

41(4), pp.331–357.

Yu, L. & Liu, H., 2004. Efficient Feature Selection via Analysis of Relevance and Redundancy. *Journal of Machine Learning*, 5, pp.1205–1224.

Yu, L. & Liu, H., 2003. Feature Selection for High-Dimensional Data : A Fast Correlation-Based Filter Solution. In *Twentieth Internation Conference on Machine Learning (ICML-2003)*. pp. 1–11.

Yu, L., Mishra, A. & Mishra, D., 2014. An Empirical Study of the Dynamics of GitHub Repository and Its Impact on Distributed Software. *Springer-Verlag*, pp.457–466.

Yu, Q. & Jiang, S., 2016. Which is More Important for Cross-Project Defect Prediction : Instance or Feature ? In *2016 International Conference on Software Analysis, Testing and Evolution*. pp. 90–95.

Zhang, H. & Cheung, S.C., 2013. A Cost-effectiveness Criterion for Applying Software Defect Prediction Models. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. pp. 643–646. Available at: http://doi.acm.org/10.1145/2491411.2494581.

Zhang, L.F. & Shang, Z.W., 2011. Classifying feature description for software defect prediction. *International Conference on Wavelet Analysis and Pattern Recognition*, pp.138–143.

Zhang, Y. & Yang, Y., 2015. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187, pp.95–112.

Zhang, Z.-W., Jing, X.-Y. & Wang, T.-J., 2016. Label propagation based semi-supervised learning for software defect prediction. *Automated Software Engineering*, pp.1–15. Available at: http://link.springer.com/10.1007/s10515-016-0194-x.

Zhao, X., Deng, W. & Shi, Y., 2013. Feature Selection with Attributes Clustering by Maximal Information Coe ffi cient. *Procedia Computer Science*, 17, pp.70–79. Available at: http://dx.doi.org/10.1016/j.procs.2013.05.011.

Zhihua, L. & Wenqu, G., 2015. A redundancy-removing feature selection algorithm for nominal data. *PeerJ Computer Science*, pp.1–17.

Zou, P.X.W., Sunindijo, R. & Dainty, A.R.J., 2014. A mixed methods research design for bridging the gap between research and practice in construction safety. *Safety Science*, 70, pp.316–326. Available at: http://dx.doi.org/10.1016/j.ssci.2014.07.005.

Zubrow, D. & Clark, B., 2001. How Good Is the Software : A Review of Defect Prediction Techniques. In *Software Engineering Symposium*. pp. 1–7.

APPENDIX A: TOOLS & METHODS

This section describes the tools and method used in the experiments for this study.

1. Software Tools

In this study, the R application was used write code that assigned the attribute importance to the features. R is an open source programming language for statistical and graphical computing. It was created in 1993 and has improved over the years. The features that had high level of importance were regarded as the most relevant features and retained. Features with the least importance values were eliminated. The program code for redundancy elimination was written in Java.

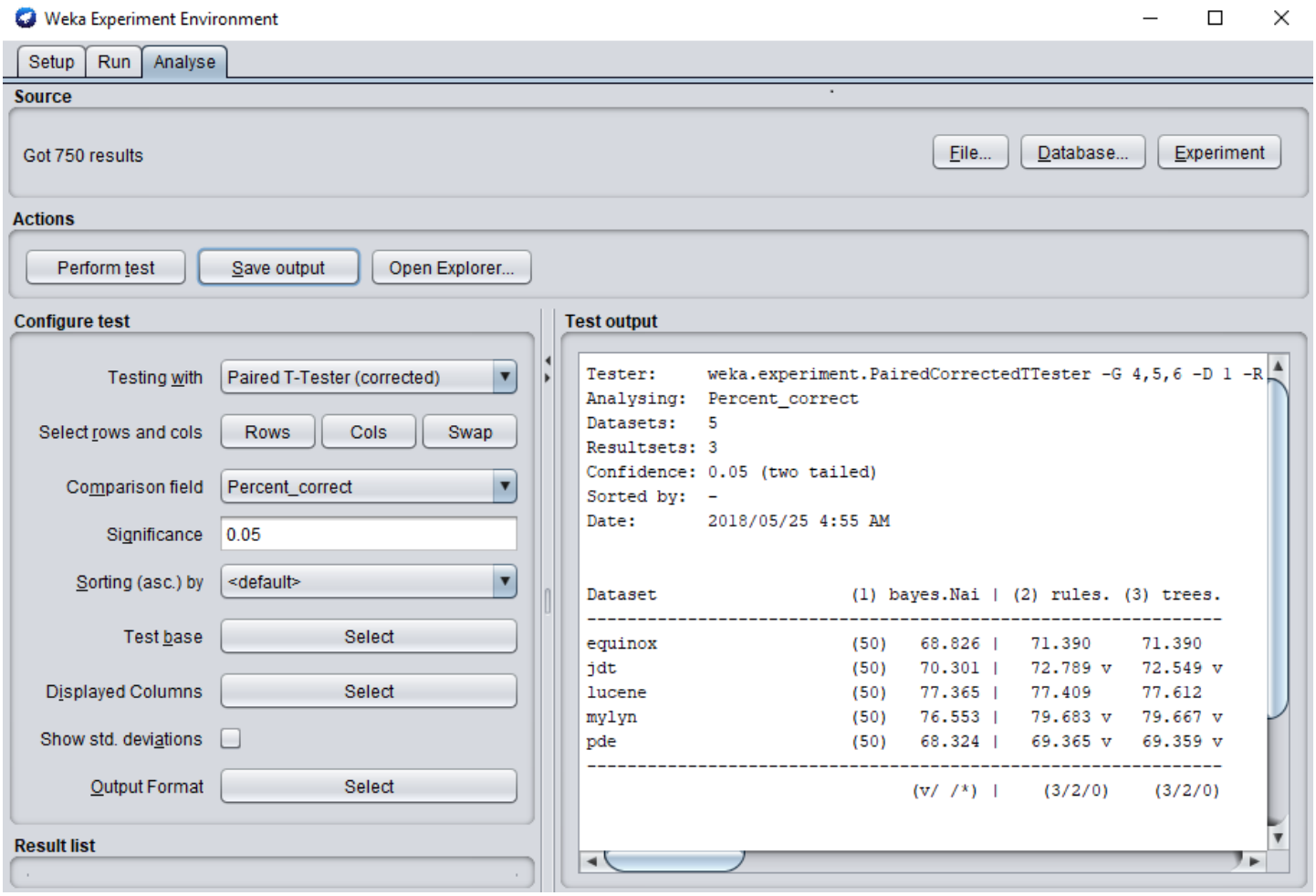
WEKA – Waikato environment for knowledge analysis

The Weka is a data mining and machine-learning tool that was designed and is maintained by the University of Waikato. It implements its machine-learning algorithms in Java.

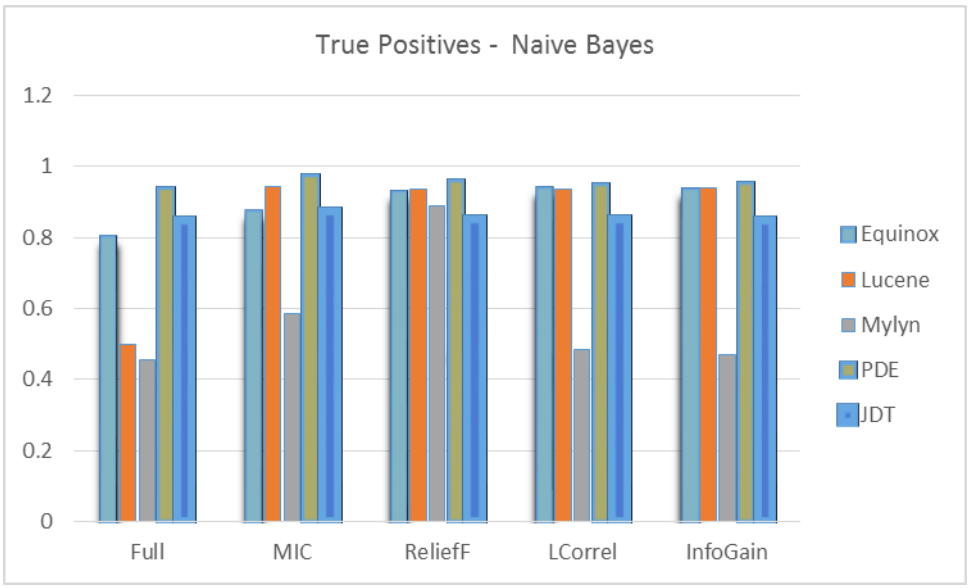
Performance measures

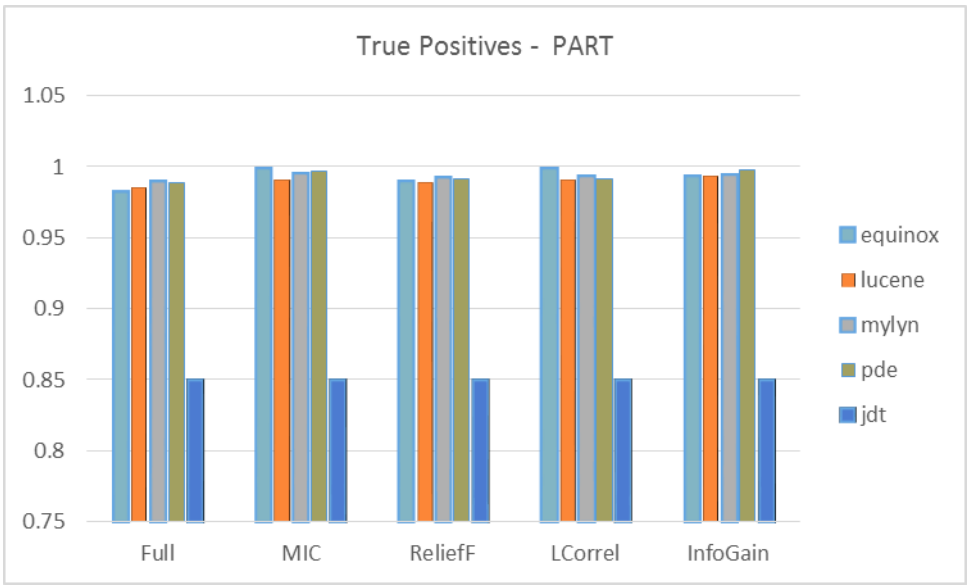
The performance measures created in this study included the Area Under the ROC Curve, Percentage Accuracy, Precision, Recall, F-Measure, True Positive and the Root Mean Squared Error. True Positives is the percentage of actual positive values that were predicted as positive. As shown in the figures below, the J48 classifier had the highest Percentage Accuracy and True Positive values.

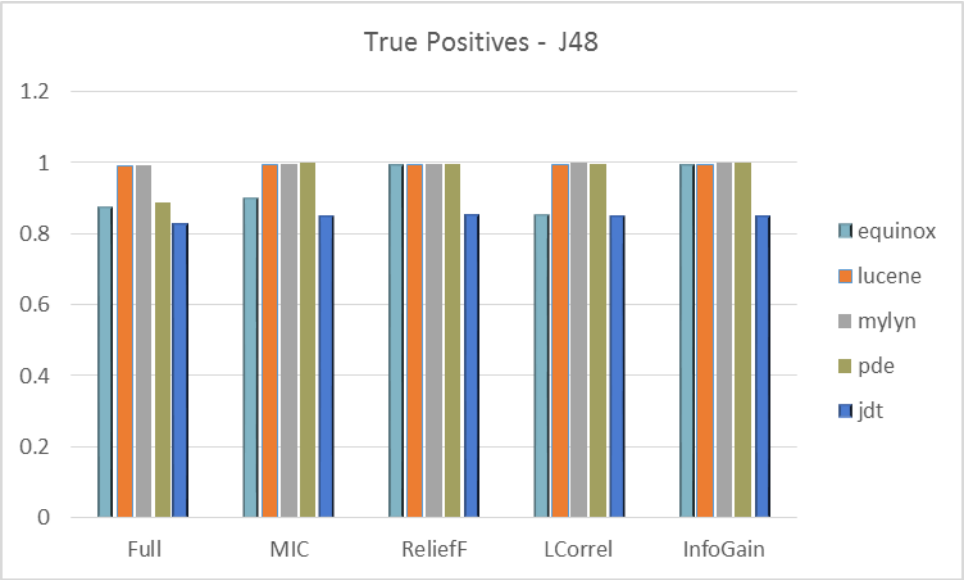
Performance Measures – Percentage Accuracy (WEKA)



Performance Measures – True Positives


True Positives - Naive Bayes
1.2
1
0.8
0.6
0.4
0.2
0
Full
MIC
ReliefF
LCorrel
InfoGain
Equinox
Lucene
Mylyn
PDE
JDT


True Positives - PART
1.05
1
0.95
0.9
0.85
0.8
0.75
Full
MIC
ReliefF
LCorrel
InfoGain
equinox
lucene
mylyn
pde
jdt



Statistical Significance Script (R Code)

```
library(PMCMR)
require(PMCMR)
library(mlr)

 naive <- c(0.391, 0.361, 0.346, 0.32, 0.333, 0.36, 0.349, 0.352, 0.354, 0.351, 0.682, 0.632, 0.342, 0.595, 0.586, 0.413, 0.391, 0.398, 0.408, 0.405, 0.297, 0.266, 0.263, 0.248, 0.257)
 rmse_naive <- matrix(naive, nrow = 5, byrow = TRUE,
                      dimnames = list(c("equinox", "lucene", "mylyn", "pde", "jdt"),
                                      c("Full", "MIC", "ReliefF",  "LCorrel", "InfoGain")))

rmse_naive
f1 <- friedman.test(rmse_naive)
print (f1)
# Post-hoc tests are conducted only if omnimus Kruskal-Wallis test p-value is 0.05 or less.
if ( f1$p.value < 0.05 )
{
n1 <- posthoc.friedman.nemenyi.test(rmse_naive)
}
n1;
# alternate representation of post-hoc test results
summary(n1);
```

APPENDIX B : CERTIFICATES & PUBLICATIONS

Editing Certificate

Samuel Mahlangu

P O Box 85

Madlayedwa

0460

23 January, 2017

To whom it may concern

This is to declare that **Samuel Mahlangu**, from Language Services at the above-mentioned address has edited an academic work of Ms Bongeka Mpofu titled as follows: **Software defect prediction using maximal information coefficient and correlation-based filter feature selection .**The author is kindly requested to make the changes suggested and to attend to the editor's queries.

Please direct any enquiries regarding the editing work of this academic work to me.

Kind regards
Samuel Mahlangu

ETHICS CLEARANCE

UNISA university of south africa

UNISA COLLEGE OF SCIENCE, ENGINEERING AND TECHNOLOGY'S (CSET) RESEARCH AND ETHICS COMMITTEE

26 September 2017

Ref #: 070/BM/2017/CSET_SOC
Name: Bongeka Mpofu
Staff #: 49131699

RECEIVED
2017 -09- 26
Office of the Deputy Executive Dean
College of Science, Engineering & Technology

Dear Ms Bongeka Mpofu

Decision: Ethics Approval for 5 years
(No humans involved)

Researchers: Bongeka Mpofu, 49131699@mylife.unisa.ac.za, +27 11 348 2975

Supervisor (s): Prof E. Mnkandla, mnkane@unisa.ac.za, +27 11 670 9059

Proposal: Software Defect Prediction based on Maximal Information Coefficient and Fast Correlation Based Filter Feature Selection

Qualification: PhD Computer Science

Thank you for the application for research ethics clearance by the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee for the above mentioned research. Ethics approval is granted for a period of five years from 26 September 2017 to 26 September 2022.

1. The researcher will ensure that the research project adheres to the values and principles expressed in the UNISA Policy on Research Ethics.
2. Any adverse circumstance arising in the undertaking of the research project that is relevant to the ethicality of the study, as well as changes in the methodology, should be communicated in writing to the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee. An amended application could

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za

be requested if there are substantial changes from the existing proposal, especially if those changes affect any of the study-related risks for the research participants.

3. The researcher will ensure that the research project adheres to any applicable national legislation, professional codes of conduct, institutional guidelines and scientific standards relevant to the specific field of study.
4. Only de-identified research data may be used for secondary research purposes in future on condition that the research objectives are similar to those of the original research. Secondary use of identifiable human research data require additional ethics clearance.
5. Permission to conduct research involving UNISA employees, students and data should be obtained from the Research Permissions Subcommittee (RPSC) prior to commencing field work.

Note:
The reference number 070/BM/2017/CSET_SOC should be clearly indicated on all forms of communication with the intended research participants, as well as with the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee

Yours sincerely

Adèle da Veiga

Dr. A Da Veiga

Chair: Ethics Sub-Committee School of Computing, CSET

Prof I. Osunmakinde

Director: School of Computing, CSET

Prof B. Mamba

Executive Dean: College of Science, Engineering and Technology (CSET)

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za

Defect Prediction based on Maximal Information Coefficient and Fast Correlation Based Filter Feature Selection

Bongeka Mpofu
School of Computing
University of South Africa
bongielondy@yahoo.com

Ernest Mnkandla
School of Computing
University of South Africa
mnkane@unisa.co.za

***Abstract*— Software fault prediction is an important quality assurance activity. A mechanism that correctly predicts and classifies modules saves resources, time and developers' effort. Datasets that are used in defect prediction may contain non-significant and redundant attributes that may affect the accuracy of machine learning algorithms. In this research, relevant attributes were chosen using the Maximal Information Coefficient and redundant attributes were eliminated using Fast Correlation Based Filter. Defect prediction was conducted on four open source bug datasets Equinox, Mylyn, PDE and Lucene. The classification algorithms, Naïve Bayes, IBk and J48 were applied to the selected datasets. The classifiers' performance was analysed using the 10 fold cross validation. The outcome proved that the Maximal Information Coefficient comparatively had the best performance in feature selection.**

Keywords—defect prediction; feature selection; software metrics; relevant metrics; redundancy; machine learning algorithms; filter; wrapper; embedded; information theory

I. INTRODUCTION

A. Software testing and quality

Software defect prediction is the use of metrics to detect modules that are more susceptible to errors. The identified modules are tested effectively and most software testing resources are reserved for them [1], [2], [3], [4]. Defective modules affect the quality and reliability of software and increase development time and software testing costs [5]. Improving the quality of software [6] minimises software rework thereby saving development resources. Software quality properties should be considered, inferred to and predicted. Evaluation and verification of quality is conducted during the development, deployment and execution stages [7]. Locating and fixing defects is costly. Software defect prediction locates the most defective modules without spending too much resources and time in the quality assurance activities.

Feature selection, also known as variable or attribute selection, is a technique that selects relevant attributes and eliminates redundant ones. Irrelevant and redundant attributes may decrease the effectiveness of the software defect model [8]. The chosen relevant features are used in the prediction process to improve the performance of the classification algorithms [9]. In this study, the importance of features or attributes will be evaluated using correlation, information theory and its concepts.

B. Information Theory

Information theory is applied in assessing and defining the amount of information in a message. In 1948 Shannon's entropy that is based on information theory was introduced as a measure of the uncertainty of random variables [10].

If x has N values given by $\{x_1,...x_N\}$, the amount of information associated with output x_i is defined as;

$$I(x_i) = -log_2 p(x_i) \quad (1)$$

where the output x_i occurs with the probability $p(x_i)$.

The measurement is in bits of information. Methods including Decision Trees, Information Gain and Symmetrical Uncertainty are based on entropy. Hassan [11] used the concepts of information theory to measure the complexity of code change in files over a specific period.

II. BACKGROUND STUDY

Several studies have been conducted in defect prediction [12], [3], [13], [14]. Feature selection algorithms based on information theory have been developed [15], [16], [17]. Many researchers have used software metrics to quantify defects and create tools for defect detection.

A. Software Metrics

Static code metrics and the Support Vector Machine algorithm were used in a defect prediction study [18]. The static code metrics were extracted from modules in National Aeronautics and Space Administration, (NASA) datasets. Pre-processing techniques such as the elimination of insignificant attributes, filling in missing values, normalisation and balancing the data were conducted. The results showed a defect prediction accuracy of 70% for all the datasets.

Moser, Pedrycz and Succi [19] investigated the prediction of defects using a large Eclipse data set. They concluded that the

change metrics were strong indicators of defects. The group of 18 change metrics produced correct output for three Eclipse project versions. The results were better than those of classifiers based on code metrics. They concluded that files that have been revised numerous times are generally defect prone. The researchers observed that files from source code repositories are likely to be defect free as a result of corrections and modifications applied to them. The results also proved that correcting bugs is may create errors and that refactored files improves software quality.

A dataset based on Chidamber and Kemerer's object oriented metrics was employed to locate defects in software. The data was tested using the Levenberg-Marquardt based neural network classifier [20]. The model had higher accuracy when compared with other function based neural network predictors.

B. *Classification*

A clustering ensemble using Particle Swarm Optimization (PSO) was used to predict defects [21]. Different clustering methods generate clusters which are integrated by the ensembles to produce a single cluster. PSO attempts to find the best solution through the imitation of some ideas derived from fish schooling, flocking patterns of birds, animal herds and other groups. In the study, the PSO using the Manhattan similarity measure located fault prone modules comparable to or better than the other algorithms.

The performance of the fuzzy linear regression and the statistical linear regression methods was compared in a defect prediction study conducted by [22]. The researcher used the KC1 NASA dataset for the analysis. Uncertainty is represented as randomness in statistical linear regression, while uncertainty is represented as fuzziness by the fuzzy linear regression model. The Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE) were used to compare the statistical linear and fuzzy models. The RMSE and the MAE values for the fuzzy regression model were bigger than those of the statistical regression model. Therefore, the statistical regression method produced better prediction results than the fuzzy regression model.

A hybrid algorithm of the Random Forest (RF) and Fuzzy C Means (FCM) clustering was designed [6]. The RF executed the pre-processing of attributes and ranked attributes in order of importance. Afterwards, the data was loaded into the FCM method, which created models for predicting defects. The output showed that the hybrid technique was more efficient and noncomplex, allowing better prediction of software defects.

A novel algorithm for selecting features using FEature Clustering And feature Ranking (FECAR) was employed the select highly important attributes to be used in locating defects. The method first clustered attributes using correlation and then selected relevant attributes from each cluster. Symmetric Uncertainty (SU) was used as the correlation measure and Information Gain, ReliefF and Chi-Square were used to select the relevant attributes [23].

MICHAC, a Maximal Information Coefficient and Hierarchical Clustering algorithm was developed by [24] to select relevant attributes using MIC and eliminate redundancy using clustering. Projects using the NASA datasets were used for conducting the research. The results of the new method were contrasted again those of the Chi-Square, Gain Ratio, ReliefF, TC and FECAR using the evaluation metrics F-measure and Area Under the Curve. The method produced better results than the other classifiers.

III. METHODOLOGY

A. *Software Metrics*

According to Shang [25], product metrics are the snapshot of data at a particular time. Process or change metrics reflect the historical changes over time. In this study, some of the change metrics used to predict changes in software are listed in Table 1.

TABLE I. CHANGE METRICS

Metrics	Metrics Definitions
linesAddedUntil	the sum of lines of source code added to a file
ageWithRespectTo	duration of file counting backwards from a specific creation or modification date
avgLinesRemovedUntil	average lines of code removed from a file
codeChurnUntil	cumulative lines of code added, removed and modified
numberOfAuthorsUntil	sum of developers who modified a file
majorBugs	cumulative major bugs located in the file
nonTrivialBugs	number of significant bugs
numberOfVersions	number of released versions of a file
numberOfFixes	count of fixes on specific file
numberOfRefactorin	count of program structure modifications

B. *Bug data sets*

Open source software, Apache Lucene, Mylyn, Equinox and PDE was used to predict the modules that are defect prone. The number of modules in Equinox were 324 modules. Lucene had 691 modules, Mylyn, 1862 and PDE had 1497 modules. Each of the datasets had 22 attributes.

C. *Relevant and Non Redundant Features*

Feature selection enables the selection of useful features. There most common classes of feature selection algorithms are filter, wrapper and embedded methods [26].

Filters

Filters measure the importance of attributes by assessing the basic characteristics of the data. The evaluation is independent of the classification algorithm [27]. Usually, a feature relevance score is calculated based on some measurement, and features with the least scores are removed. Selected features may be ranked using the assigned score [28] and subsequently, input in

a classifier. Examples of filters include the Linear Correlation (LC), T-test, Information Gain (IG) and Relief algorithms.

Wrappers

Wrappers apply feedback from classifiers to determine which features to select [29]. The filter based method is computationally faster than the wrapper method. However, wrappers are more accurate than filters [30]. Sequential Forward Selection, Sequential Backward Elimination and Beam Search are some of the wrapper techniques.

Embedded

Embedded methods perform feature selection during training and are generally inbuilt to the specific learning algorithms, and thus may be more accurate than the other methods [31]. Common embedded techniques include the Lasso methods and decision tree algorithms: C4.5 and Random Forest.

Feature selection enables the selection of useful features. In this study, attributes were selected via the MIC, LC and ReliefF. SU, also known as the Fast Correlation Based Filter, reduced redundancy in the data. The feature selection procedure is shown in Figure 1.

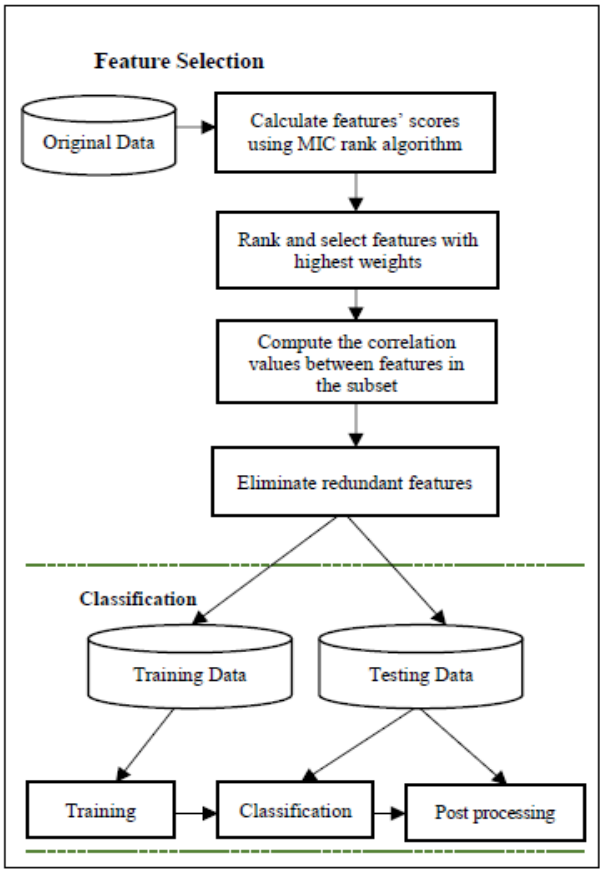


Fig. 1. Feature selection procedure

Linear Correlation

The linear correlation coefficient measures the relationship between two features [8]. In Equation 2, *cov()* is the covariance, *D()* is the variance and is defined as;

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}} \quad (2)$$

The value of ρ ranges between -1 and 1 inclusive. If the two features *X* and *Y* are fully correlated, ρ takes the value of -1 or 1. If the two features X and Y are independent the value will be 0. It is a symmetrical measure for the two variables.

Information Gain

Given two variables A and B, the conditional entropy H(B|A) measures the remaining uncertainty about B when A is known. In probability theory and information theory, Information gain (IG), also known as relative entropy, or Kullback-Leibler divergence is the degree of variation between two probability distributions [31]. The entropy (H) of a random attribute is a degree of its uncertainty. An entropy for a random attribute X with N outcomes is defined as;

$$H(X) = -\sum_{i=1}^{n} p(xi) log_b p(xi) \quad (3)$$

Information gain evaluates attribute *X* by calculating the magnitude of information it provides about the class distribution *Y*. It is interpreted as;

$$I(X) = H(P(Y) - H(P(Y/X)) \quad (4)$$

Maximal Information Coefficient

Similarity between features is measured using the relationship coefficient. Pearson coefficient is one of the relationship algorithms but it can only capture linear relationships and does not possess the superposition property [32]. It fails to cater for functional sin or cubic. The MIC was proposed by [33] and caters for functional and non-functional associations. It is based on the concepts of information theory. It symbolises the nonlinear relationship between two variables and is a real number whose values range from 0 to 1, where 0 represents uncorrelation and 1 represents complete correlated, noiseless functional relationship. Mutual information, the extent of how much the two variables *X* and *Y* share is represented by;

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (5)$$

The data points of variables X and Y on a grid yield the maximum resolution size. The mutual information for X and Y is calculated as;

$$I(X;Y) = \sum_{X,Y} p(X,Y)\, log_2 \frac{p(X,Y)}{p(X)p(Y)} \quad (6)$$

where the joint distribution of variables X and Y is $p(X,Y)$. The computation of the normalised mutual information value is based on pairs of integers (x,y) on the grid. The highest of the normalised values is known as the MIC and is defined as;

$$MIC(X;Y) = \max_{X,Y_{total}<B} \frac{I(X;Y)}{log_2(\min(X,Y))} \quad (7)$$

The advantages of MIC is that it can explore hidden relationships between variables and reduce noise [24].

Fast Correlation Based Filter

The Fast Correlation Based Filter (FCBF) is a multivariate feature selection algorithm that uses Symmetric Uncertainty to calculate dependencies of features and selects the best subset in high dimensional data using the backward selection technique with sequential search strategy [31]. The selection ends if there are no more features to eliminate. In information theory, Symmetrical Uncertainty (SU) is a normalised measure that evaluates the dependencies of features using entropy and conditional entropy. The entropy of X given that X is a random variable and the probability of x is $P(x)$ is defined as;

$$H(X) = -\sum_i P(x_i) log_2(P(x_i)) \quad (8)$$

The conditional entropy, also known as the conditional uncertainty of X after given the values of an attribute variable Y is;

$$H(X\backslash Y) = -\sum_j P(y_j) \sum_i P(x_i\backslash y_j) log_2(P(x_i\backslash y_j)) \quad (9)$$

The SU figure of 0 implies that features are totally independent while an SU amount of 1 signifies that a feature can totally predict the value of another feature.

D. Classification Algorithms

Naïve Bayes

The classification algorithm Naïve Bayes is founded on the Bayesian networks [34] theory and uses probability for predicting the class an instance is associated with, given the set of features defining the instance [35]. Features are considered to contribute independently to the probability regardless of correlations between them. The classifier learns from the training data which parameters are suitable for the classification task. The Bayes rule joins the prior probability of every variable and the likelihood to create a highest posterior probability that is used to predict a class. The classifier is defined by:

$$fi(X) = \prod_{j=1}^{n} P(xj\backslash cj)P(ci) \quad (10)$$

where $X = (x1, x2, ...xn)$ represents the vector of a feature, i.e. $x1$ is the value of feature X. and cj , $j = 1, 2, ..., N$, are the potential labels of the class, $P(xj\backslash cj)$ are conditional probabilities and $nP(ci)$ are prior probabilities.

K Nearest Neighbors

The K nearest neighbors (KNN) classifiers, also called instance based classifier, IBk, belong to the class of lazy learners because they do not induce categorisation models from training data [29]. Categorisation is obtained by comparing a new instance with existing ones in the training dataset. The k nearest neighbors is obtained based on the distance metric, generally the Euclidean distance. The nearest present instance is used to allocate the class for the test sample.

J48

It is a variation of C4.5, which is a standard decision tree classifier presented by Quinlan and is used to build a decision tree using a training dataset [36]. It uses the concept of information entropy. The decision tree approach is most useful in classification problems. With this technique, a tree is created to represent the classification procedure. Once the tree is built, it is applied to the elements in the database.

Algorithm

Input: $F(f_1, f_2....F_n, C)$ - the set of features and the threshold Ts

Output S : Selecting the feature subset

//Section 1: Eliminate irrelevant features

//calculate the mutual information score of two features

//normalise score from 0 to 1

$MIC(f_1, f_2) = norm(I(X,Y))$

for i = 1 to n do

if ($MIC_{i,c} \geq T$)

F = F U {fi}

//Section 2: Removing redundant features based on

symmetric uncertainty value between class and features

Input: F(f1, f2.....Fn, C) - the set of features

Output S : Selecting the feature subset

for i = 1 to m do

if (SUi,c ≥ Ts)

F = F U {fi}

Return S

Model Evaluation Techniques

The Weka contains some of the following model evaluation techniques:

Training Dataset

The machine learning algorithm is assessed on its capability to predict the class of the instances it was trained on.

Supplied Test Set

The classification algorithm is assessed on its prediction accuracy of the class of a set of instances.

Percentage Split

The classifier is evaluated on its accuracy in predicting a specific segment of the data. The amount of data tested depends on the value input in the % field.

Cross Validation

This validation method reduces the bias related to the random sampling of data samples. The dataset is split into equal sized *k* partitions or folds. The classifiers are trained on *k-1* partitions. The partition that was eliminated is used to test the efficiency of the learned model in the dataset of the partitions used in the test. The average performance of all k models is then calculated.

Performance Evaluation

The accuracy of classification algorithms is the measurement of correctness in prediction using the test sets.

Mean absolute error

The Mean Absolute Error (MAE) calculates the average amount of the errors. It measures accuracy for continuous variables and evaluates the degree of average deviation in prediction [37]. The MAE value nearer to zero is regarded as having the better prediction capability. It is interpreted as;

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|\hat{m_i} - m_i|) \quad (11)$$

where *n* is the sum of the tests, ^ *mi* is the predicted value and *mi* is the observed value.

Recall

Recall represents the TP rate, i.e. s all the defective modules that the classifier can locate, True positives / Actual positives. It is defined as;

$$Recall = \frac{tp}{tp+fn} \quad (12)$$

Precision

Precision is TP /positively predicted. It evaluates the number that were predicted to be defected prone and turned out to be defective. It is represented by;

$$Precision = \frac{tp}{tp+fp} \quad (13)$$

The Recall and Precision amounts span from 0 to 1. The top values signify more accurate prediction. If both Recall and Precision are equivalent to 1, it implies that the predictor can predict all defect-prone modules without False Negative or False Positive [37]. Realistically, the values of Recall and Precision are generally mutually exclusive, i.e., high Recall value is often with low Precision value. Achieving both high Recall and Precision at the same time is unlikely.

F-Measure

The F – Measure is a test that integrates precision and recall and is calculated as;

$$F = 2.\frac{precision.recall}{precision+recall} \quad (14)$$

AUC

The Area Under the ROC Curve (AUC) evaluates accuracy. It assesses the amount of discrimination achieved by the MAE. The value of AUC spans from 0 to 1 and random prediction has AUC of 0.5. The advantage of AUC is that it is not affected by decision threshold like Precision and Recall. The AUC with maximal values signify better prediction [17].

IV. RESULTS

The Percentage Accuracy, Recall (RC), Area Under the Curve (AUC) and F-Measure were calculated in WEKA. The performance of the classifiers, Naïve Bayes, IBk and J48 using data sets of features selected using Linear Correlation, MIC and ReliefF are shown in Tables II, III and IV. These subsets are compared with a full set of attributes.

TABLE II. CLASSIFICATION ACCURACY - NAÏVE BAYES

		Perc. Acc.	*RC*	*AUC*	*FM*
Equinox	Full	55.586	0.599	0.517	0.653
	LC	**66.543**	**0.752**	0.516	**0.752**

		Perc. Acc.	*RC*	*AUC*	*FM*
	MIC	64.136	0.721	**0.54**	0.734
	RelfF	66.08	0.745	0.538	0.75
Lucene	Full	66.544	0.745	**0.511**	0.756
	LC	70.181	0.801	0.498	0.801
	MIC	**75.956**	**0.878**	0.501	**0.845**
	RelfF	71.818	0.821	0.5	0.821
Mylyn	Full	41.729	0.383	0.518	0.432
	LC	44.334	0.421	0.502	0.421
	MIC	46.112	0.443	0.522	0.494
	RelfF	**64.033**	**0.702**	**0.537**	**0.702**
PDE	Full	71.635	0.811	0.489	0.807
	LC	**79.912**	**0.929**	0.498	**0.929**
	MIC	77.953	0.903	**0.5**	0.863
	RelfF	79.865	0.928	0.48	0.928

The ReliefF had the highest Recall percentage in the tests conducted using the Naïve Bayes classifier. The average Recall values for the MIC, LC, ReliefF and Full datasets were 0.736, 0.725 and 0.799. In the Naïve Bayes, The MIC and LC had one Win for the Percentage Accuracy compared to ReliefF that had 2 wins. The LC had 2 Wins for the F-Measure. The ReliefF and MIC each had one. All the subsets had 1 Win and 3 Losses for the AUC created by the Naïve Bayes.

TABLE III. CLASSIFICATION ACCURACY - IBK

		Perc. Acc.	*RC*	*AUC*	*FM*
Equinox	Full	72.963	0.869	0.47	0.845
	LC	73.765	0.878	0.479	0.85
	MIC	**75.093**	**0.891**	**0.514**	**0.859**
	RelfF	74.074	0.876	0.504	0.851
Lucene	Full	76.961	0.882	0.525	0.866
	LC	77.164	**0.896**	0.517	0.868
	MIC	**77.221**	0.888	**0.551**	0.869
	RelfF	76.947	0.892	0.524	**0.868**
Mylyn	Full	75.494	0.866	0.5	0.857
	LC	**75.623**	0.868	0.496	0.858
	MIC	76.547	**0.877**	**0.509**	**0.868**
	RelfF	76.015	0.876	**0.509**	0.861
PDE	Full	75.11	0.857	**0.51**	0.854
	LC	**75.825**	0.868	0.508	**0.859**
	MIC	75.104	0.863	0.499	0.855
	RelfF	75.685	**0.869**	0.509	0.858

In the IBk, the MIC had 2 wins, while the ReliefF had none and the LC had one. The MIC and the LC each had 2 wins for the F-Measure while the other subsets had losses. The MIC subset had 2 highest scores/Wins and a Tie for the AUC. The ReliefF had one Tie while the Full dataset only had none

TABLE IV. CLASSIFICATION ACCURACY - J48

		Perc. Acc.	*RC*	*AUC*	*FM*
Equinox	Full	82.438	0.986	**0.533**	0.906
	LC	82.654	**0.99**	0.532	**0.908**
	MIC	**82.747**	0.986	0.532	**0.908**
	RelfF	82.407	0.987	0.53	0.906
Lucene	Full	83.865	0.994	**0.507**	0.912
	LC	83.879	0.994	0.504	0.912
	MIC	84.183	**0.999**	0.501	**0.914**
	RelfF	83.792	0.993	0.503	0.912
Mylyn	Full	84.517	0.997	0.498	0.916
	LC	84.635	0.998	**0.499**	**0.917**
	MIC	84.608	0.998	0.498	**0.917**
	RelfF	**84.672**	**0.999**	0.498	**0.917**
PDE	Full	84.723	0.998	**0.5**	0.917
	LC	**84.903**	**1**	**0.5**	**0.918**
	MIC	84.83	0.999	0.499	**0.918**
	RelfF	**84.903**	**1**	**0.5**	**0.918**

The MIC subset and the ReliefF had Ties in Recall values of the J48 classifier. The MIC had the 2 highest scores for the prediction accuracy. The MIC and the ReliefF had the same number of Wins, Ties and Losses for the Recall. The MIC had Wins for the F-Measure and AUC while the ReliefF subset had none.

TABLE V. WINS/TIES/LOSSES

	P.Acc	*Recall*	*F.Measure*	*AUC*
ReliefF	3/1/8	3/1/8	1/2/9	1/2/9
MIC	6/0/6	4/1/7	4/3/5	4/1/7
LC	2/1/9	3/2/7	4/3/5	1/1/10
Full	0/0/16	0/0/16	0/0/16	3/1/8

In the Wins/Ties/Loss Table, MIC had the most wins, which represent the number of times the subset had the best classification accuracy from the algorithm. In the statistical package SPSS, paired samples t-test, the number of Wins needed for a statistical significance was 4.

TABLE VI. WINS/TIES/LOSSES BY CLASSIFIER

	Naïve Bayes	*IBk*	*J48*
Full	1/0/15	1/0/15	2/1/13
MIC	5/0/11	8/1/7	4/1/11
ReliefF	4/0/12	2/1/13	2/3/11
LC	6/0/10	4/0/12	1/6/9

V. CONCLUSION

Defect prediction is a vital process in software quality assurance activity. In this study, the effectiveness of feature selection algorithms in defect prediction was investigated. The proposed algorithm selected the best features using the MIC.

Redundant features were removed using the FCBF. The performance of the algorithm was compared with other techniques that are based on the Information Theory. The results showed that the MIC selected subset had the highest values compared to other subsets in results produced by the classification algorithms. Selected features were used by the classifiers for testing prediction accuracy.

Future work will include the testing of more datasets that have more features. Other feature selection methods, including wrappers and embedded methods will be applied in the selection of relevant features.

REFERENCES

[1] Bettenburg, N. & Hassan, A.E. 2013. Studying the impact of social interactions on software quality.

[2] Yang, X., Tang, K. & Yao, X. 2015. A learning-to-rank approach to software defect prediction. IEEE Transactions on Reliability, 64(1): 234–246.

[3] Arnmah, G.K., Luo, G. & Qin, K. 2013. Multi_level data pre_processing for software defect prediction. 2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering: 170–174. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6703111.

[4] Caglayan, B., Tosun Misirli, A., Bener, A.B. & Miranskyy, A. 2015. Predicting defective modules in different test phases. Software Quality Journal, 23(2): 205–227.

[5] Kumar, S.V. & Jacob, S.G. 2012. Predicting Fault –Prone Software Modules Using Feature Selection and Classification through Data Mining Algorithms 1. : 1–4.

[6] Pushphavathi, T.P., Suma, V. & Ramaswamy, V. 2014. A novel method for software defect prediction: Hybrid of FCM and random forest. Electronics and Communication Systems (ICECS), 2014 International Conference on: 1–5.

[7] Jiao, W. 2011. Using autonomous components to improve runtime qualities of software. IET Software, 5(1): 1. http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2010.0001.

[8] Xia, Y., Yan, G., Jiang, X. & Yang, Y. 2014. A new metrics selection method for software defect prediction. 2014 IEEE International Conference on Progress in Informatics and Computing: 433–436. http://www.scopus.com/inward/record.url?eid=2-s2.0-84919359671&partnerID=tZOtx3y1.

[9] Rahman, M.H., Sharmin, S., Sarwar, S.M. & Shoyaib, M. 2016. Software Defect Prediction Using Feature Space Transformation. Proceedings of the International Conference on Internet of things and Cloud Computing - ICC '16: 1–6. http://dl.acm.org/citation.cfm?doid=2896387.2900324.

[10] Liu, J., Lin, Y., Lin, M., Wu, S. & Zhang, J. 2017. Neurocomputing Feature selection based on quality of information. Neurocomputing, 225(June 2016): 11–22. http://dx.doi.org/10.1016/j.neucom.2016.11.001.

[11] Hassan, A.E. 2009. Predicting Faults Using the Complexity of Code Changes. In ICSE 09. 78–88.

[12] Dhiman, P. & Chawla, R. 2012. A Clustered Approach to Analyze the Software Quality Using Software Defects. 2012 Second International Conference on Advanced Computing & Communication Technologies: 36–40. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6168329.

[13] Bell, R.M., Ostrand, T.J. & Weyuker, E.J. 2013. The limited impact of individual developer data on software defect prediction. Empirical Software Engineering, 18(3): 478–505.

[14] Liu, L., Li, K., Shao, M. & Liu, W. 2015. Fuzzy Integral Based on Mutual Information for Software Defect Prediction. In 2015 International Conference on Cloud Computing and Big Data. IEEE Computer Society: 93–96.

[15] Gao, K., Khoshgoftaar, T. & Wald, R. 2010. Combining Feature Selection and Ensemble Learning for Software Quality Estimation. In Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference. 47–52.

[16] Singh, V.B., Chaturvedi, K.K., Khatri, S.K. & Kumar, V. 2015. Bug prediction modeling using complexity of code changes. International Journal of Systems Assurance Engineering and Management, 6(1): 44–60.

[17] Seliya, N., Khoshgoftaar, T.M. & Van Hulse, J. 2010. Predicting faults in high assurance software. Proceedings of IEEE International Symposium on High Assurance Systems Engineering: 26–34.

[18] Gray, D., Bowes, D., Davey, N., Sun, Y. & Christianson, B. 2009. Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. Engineering Applications of Neural Networks. Springer Berlin Heidelberg: 223–234.

[19] Moser, R., Pedrycz, W. & Succi, G. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. Proceedings of the 13th international conference on Software engineering - ICSE '08: 181. http://portal.acm.org/citation.cfm?doid=1368088.1368114.

[20] Singh, M. 2013. Software Defect Prediction Tool based on Neural Network. , 70(22): 22–28.

[21] Coelho, R.A., Guimaraes, F. dos R.N. & Esmin, A.A.A. 2014. Applying Swarm Ensemble Clustering Technique for Fault Prediction Using Software Metrics. In 2014 13th International Conference on Machine Learning and Applications. 356–361. http://www.scopus.com/inward/record.url?eid=2-s2.0-84924940251&partnerID=tZOtx3y1.

[22] Valles-Barajas, F. 2015. A comparative analysis between two techniques for the prediction of software defects: fuzzy and statistical linear regression. Innovations in Systems and Software Engineering, 11(4): 277–287.

[23] Liu, S., Chen, X., Liu, W., Chen, J., Gu, Q. & Chen, D. 2014. FECAR : A Feature Selection Framework for Software Defect Prediction.

[24] Xu, Z., Xuan, J., Liu, J. & Cui, X. 2016. MICHAC : Defect Prediction via Feature Selection based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Engineering. 370–381.

[25] Shang, W., Nagappan, M. & Hassan, A.E. 2013. Studying the relationship between logging characteristics and the code quality of platform software. Empirical Software Engineering, 20(1): 1–27.

[26] Saeys, Y., Inza, I. & Larranaga, P. 2007. Gene expression A review of feature selection techniques in bioinformatics. Bioinformatics, 23(19): 2507–2517.

[27] Gigli, A., Lucchese, C., Nardini, F.M. & Perego, R. 2016. Fast Feature Selection for Learning to Rank. ACM: 167–170.

[28] Ji, Z., Meng, G., Huang, D., Yue, X. & Wang, B. 2015. NMFBFS : A NMF-Based Feature Selection Method in Identifying Pivotal Clinical Symptoms of Hepatocellular Carcinoma. Hindawi, 2015: 1–11.

[29] Gao, K., Khoshgoftaar, T., Wang, H. & Seliya, N. 2011. Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms. Software - Practice and Experience, 41: 579–606.

[30] Wang, H. & Liu, S. 2016. An Effective Feature Selection Approach Using the Hybrid Filter Wrapper. *International Journal of Hybrid Information Technology*, 9(1): 119–128

[31] Ladha, L. & Deepa, T. 2011. Feature Selection Methods and Algorithms. International Journal on Computer Science and Engineering, 3(5): 1787–1797.

[32] Zhao, X., Deng, W. & Shi, Y. 2013. Feature Selection with Attributes Clustering by Maximal Information Coe ffi cient.

Procedia Computer Science, 17: 70–79. http://dx.doi.org/10.1016/j.procs.2013.05.011

[33] Reshef, D.N., Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., Mcvean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M. & Sabeti, P.C. 2011. in Large Data Sets. , 1518.

[34] Abraham, R. & Simha, J.B. 2007. Medical datamining with a new algorithm for Feature Selection and Naïve Bayesian classifier. : 44–49.

[35] Singh, P. & Verma, S. 2012. Empirical investigation of fault prediction capability of object oriented metrics of open source software. *JCSSE 2012 - 9th International Joint Conference on Computer Science and Software Engineering*: 323–327.

[36] Hewett, R. 2011. Mining software defect data to support software testing management. *Applied Intelligence*, 34(2): 245–257.

[37] Patil, T.R. & Sherekar, S.S. 2013. Performance Analysis of Naive Bayes and J48 Classification Algorithm for Data Classification. *International Journal Of Computer Science And Applications*, 6(2): 256–261.

Software Defect Prediction Using Process Metrics ElasticSearch Engine Case Study

Bongeka Mpofu
School of Computing
University of South Africa
Johannesburg, South Africa
bongielondy@yahoo.com

Enerst Mnkandla
School of Computing
University of South Africa
Johannesburg, South Africa
mnkane@unisa.ac.za

Abstract—New technology advances provide more sophisticated features which increasingly require complex software. The growing complexity of software presents a significant challenge to software quality and reliability. The prediction of defects in software is aimed at maintaining software quality without allocating a lot of resources in the quality assurance activities. The distribution of defects in modules is not uniform. Defect prediction models can locate the most defective modules. In this study, software defect prediction was conducted using the source code of the ElasticSearch engine. The Logistic Regression, Naïve Bayes and J48 techniques were used to learn from previous versions and predict defects in upcoming releases. The J48 outperformed the Logistic Regression and Naïve Bayes. The quality and accuracy of the prediction models was evaluated using basic performance measures and the 10-fold cross validation.

Keywords—software quality; bugs; defect prediction; process metrics; multiple linear regression, clustering, accuracy, precision, recall

I. INTRODUCTION

A. Software Quality Management

Software engineering was primarily aimed at accomplishing system functional requirements. This gradually included quality attributes of the systems as a result of the growth in business and industrial considerations [1]. Quality software not only fulfils its functionalities but does so effectively, safely, and productively while also giving users a certain level of satisfaction [2]; [3]. To ensure quality, it is imperative to conduct intensive software testing. Most of the tools and resources of software development and maintenance are linked to software testing [4]; [5].

B. Software Defects

Defects are caused by errors in logic or coding which result in failure or unpredicted results. Software defects have a negative impact on the quality of software [6]. They may cause the delay of a product release, loss of reputation and increased development costs [7]. Research has also focused on software defect prediction, (SDP) [8]; [9]. It is a proactive approach that predicts the fault-proneness of modules and allows software developers to allocate the limited resources to the defect prone modules so that high reliability software can be produced on time and within budget [10]; [11]. Software defect prediction techniques can be based on the source code and defect data of current and previous applications [12]. Verification, validation and testing can be executed throughout the software development process. Defects found in the earlier stages of software development can be corrected with minimal expense [13]. According to [14], higher levels of software process improvement significantly reduce the likelihood of high severity defects. This is beneficial when the requirements are clear, complete and unambiguous.

978-1-5090-2576-3/16/$31.00 ©2016 IEEE.

C. Software Versioning

Software versioning is the development of software with the same name with some features or functions introduced [15]. Unique codes are allocated to successive and unique states of computer software. These are usually decided on the basis of the perceived significance of changes between versions without any clear criterion [16].

A new software version helps to define a threshold which makes a transition from a current state to a new state. Version control plays an important role in software industries to manage changes and co de development where a team of developers constantly change source files to implement technical specifications [17]. New versions must be better than the preceding ones and not introduce new bugs.

Successive versions of software defect test tools that match the advancement of mobile devices and applications on the market have been developed [18]. The ElasticSearch is one of the open source applications. Predicting the defects in applications will enable developers to focus on the fault-prone code and allocate scarce resources to the problem areas thereby improving test efficiency and reducing development cost and time [3]; [5].

II. RELATED WORK

Surveys on defect prediction models have been conducted by [7]; [19]. [20]; [21]; [7] research showed that there have been great advances in software defect prediction which pays off in a significant way for software quality.

A. Static Code Metrics

[19] conducted the prediction of defects using a large data set for the Eclipse project. Their set of 18 change metrics generated very accurate results for three releases of the Eclipse project. The results outperformed predictors based on

code metrics. They concluded that files with high revision numbers are by nature defect prone. They observed that files that are part of large CVS commits are likely to be defect free, that bug fixing activities are likely to introduce errors and that refactoring improves software quality.

B. Process Metrics

A combination of code metrics and process metrics was used to detect software defects in aerospace tracking telemetry and control. The results showed that the combination set improved the performance of defect prediction. History change was proved to have a huge impact on the prediction performance [22]. [23] compared the performance, stability, portability and stasis of different sets of metrics. The results revealed that code metrics despite their widespread use are generally less suitable than process metrics. They found out that code metrics have high stasis, i.e., they don't change much from release to release. That results in stagnation in the prediction models, causing the same files to be reported as defective, however these continually defective files may turn out to contain less defects.

In a study conducted by [24] on the defect prediction on three chosen open source software, written in Java language. The performance of the predicted models was evaluated using receiver operating characteristic analysis. The study shows that machine learning methods are comparable to regression techniques. It was suggested that testing based on change proneness of a software leads to better quality by targeting the most change prone classes.

[25] conducted an investigation to evaluate the process metrics' ability to improve prediction models. Data from a wide range of software projects was used. The prediction models trained on a data set containing product metrics and as well as Number of Distinct Committers (NDC) were significantly better than those without the NDC. The models trained on a data set containing product models and as well as Number of Modified Lines (NML) were significantly better than those without the NML. Experiments conducted by [26] suggested that history change process metrics have obvious effect in improving the prediction performance of software.

C. Defect Prediction Models

Various techniques have been used for developing software fault prediction models. A study conducted by [9] predicted defects in different test phases. Change and static metrics were obtained from historical software of a large scale enterprise. They built a learning-based model for testing each phase. The outcome proved that a model for predicting faults in each testing phase improves the fault prediction performance and reduces the defect detection time. Their analysis indicated that defects can be located 7 months earlier if prediction is conducted in each phase.

To address the non-availability of historical fault data for software defect prediction, sample based methods were proposed by [12]. A small percentage of the modules was used to build a fault prediction model for detecting defects in the rest of the modules. Random sampling with conventional machine learners, random sampling with a semi-supervised learner and active sampling with active semi-supervised learner were used for selecting the samples. The results showed that the methods were effective for defect prediction.

[27] built logistic regression models for the Eclipse bug data set to predict whether files have post-release defects. Logistic regression models predict likelihoods between 0 and 1. If the predicted likelihood was greater than 0.5, they regarded the file as defect-prone, else as defect-free. The experiments indicated the presence of defects in packages and that the combination of complexity metrics can predict defects. It was advised that the more complex the code is, the more defects it has.

In a study on software with few defects, the Random Forest defect prediction models had better predictive ability than other learning techniques such as Naive Bayes, Logistic Regression, and Decision Trees [28]. [29] used the data classifiers J48 and Naïve Bayes to compare the accuracy and cost analysis. The experiments results showed that the J48 had more classification accuracy and it was also more cost efficient than the Naïve Bayes classifier.

[30] used the Logistic Regression, Naïve Bayes Tree, CART Decision Tree and Gaussian Naive Bayes to build bug prediction models using change metrics. The experiments were conducted on five different versions of Eclipse JDT project. The change metrics had a recall of 90.4% using Logistic Regression. [19] used the Naïve Bayes classification models to compare the efficiency of product and process metrics in the Eclipse data. The change model outperformed the code model with respect to all accuracy indicators, for the Eclipse 3.0.

The Gaussian Naïve Bayes, Decision Trees, Logistic Regression and the Naïve Bayes Tree were used by [30] for predicting defects using change metrics on different versions of Eclipse data. The Naïve Bayes Tree is a hybrid of the Naïve Bayes and the Decision Tree algorithms and supposedly has better predictive capabilities than the component algorithms alone. The NB Tree performed better than other models.

III METHODOLOGY

The source code of the ElasticSearch, an open source search engine was used in this study to predict software defect prone modules. The ElasticSearch code is written in Java and its data which was obtained from the GitHub online repository. The data was pre-processed and formatted using Waikato Environment for Knowledge Analysis (WEKA) data mining tool where the selected data is converted into Attribute Relation File Format (ARFF) which is readable from WEKA. The transformed data thus obtained is subjected to data mining process. The ElasticSearch Github repository has over 23,000 commits spanning from 2010 to 2016.

D. Metrics

In defect prediction studies, static and process metrics are used. Static code metrics are the metrics that can be extracted directly from the source code, such as, the lines of code (LOC) and complexity [31]. Process metrics are metrics that are based on historic source code over time [32]. These metrics are extracted from the source code module and include, for example, the number of additions and deletions from the

source code, the number of distinct committers and the number of modified lines.

TABLE V. TABLE 1. METRICS USED

Metric	*Description*
Commits	Number of commits that have modified a file. Justification: The more a file is modified, the more probability of a defect
Additions	Number of lines added per file revision. Justification: The more a file is modified, the more probability of a defect
Deletions	Number of lines deleted per file revision. Justification: The more a file is modified, the more probability of a defect
Bugs	Bugs fixed per file revision. Justification: Files that have been repeatedly fixed for bugs are more defect prone
Commit60	The number of times a file was changed in the last 60 days of its release. Justification: Files that are changed more before their release have more post release defects
AGE	Age of the file. The time interval between the current file version and the previous file version. Justification: A longer time interval increases the probability of defects
Authors	Number of developers that modified the file. Justification: The more number of developers that modify a file increases the probability of defects
EXP	Experience of the developer, measured by the number of commits by the developer. Justification: The more experienced a developer is, the less chances of defects
Number of Modified Files	Number of files modified by the commit. Justification: A large number of modified files increase the probability of defects
Entropy	Scattering of changes throughout the files. Justification: A commit that modifies many lines in a single file has lower entropy than one modifying a few lines in many files

Entropy is an information theory based measure that is defined as a measure of uncertainty in the code changes [33]. Entropy has been used to quantify code changes and predict defects. The Shannon Entropy is defined as

$$(\mathrm{H})\mathrm{n}(\mathrm{P}) = -\sum_{k=1}^{n}(Pk\ log2Pk) \qquad (1)$$

where $\mathrm{P_k} \geq 0$

$$\sum_{k=1}^{n} \mathrm{Pk} = 1$$

where n is the number of files and the value of k varies from 1 to n. Pk is the probability that the file k changes during the considered time interval.

E. Fault Prediction Techniques

Logistic Regression

Logistic regression (LR) is a statistical method for dealing with the formulation of the mathematical model depicting relationship among variables, which can be used for the values of each independent variable. Multivariate LR is done to construct a prediction model for the change proneness of classes. The multivariate LR formula can be defined as (2).

$$prob\ (X1, X2 \ldots . Xn)^n = \frac{e(A0 + A1X1 + \cdots + AnXn)}{1 + e(A0 + A1X1 + \cdots + AnXn)} \qquad (2)$$

where Xi, i = 1,2…. n are the independent variables. Prob is the probability of detecting whether the class has changed.

J48

It is a variation of C4.5, which is a benchmark decision tree learning algorithm proposed by Quinlan is used to build a decision tree based on a set of training data. It uses the concept of information entropy [34]. The decision tree approach is most useful in classification problems. With this technique, a tree is constructed to model the classification process. Once the tree is built, it is applied to each tuple in the database. C4.5 evolved from ID3, which uses the splitting criterion. The ID3 improves its performance by using information theory to select the most informative attribute and handles data with missing attributes.

Naïve Bayes

The Naive Bayes (NB) learner is based on the Bayesian rule of conditional probability, and assumes that predictor attributes are independent of each other given the class. Naïve Bayes classification gives an assumption called "class conditional independent", which assumes that the influence of a given class's attributes is independent. In large-scale data processing, Naïve Bayes method shows high accuracy and high speed [35]. Naïve Bayes is another very commonly used learner in the software engineering domain, and has been shown to perform well for defect prediction [34]. The Naïve Bayes is based on the Bayes rule below [36]:

$$P(cj/x) = P(cj)P(x/cj)/P(x) \qquad (3)$$

where cj is a member of the set of values, cj represents "fault-prone" or "fault-free" in this study, x represents a test instance. Naïve Bayes computes the conditional probability of the test instance being labelled cj. The cj with the highest probability is chosen as the label for x.

F. Performance Evaluation

The accuracy of classifiers is the percentage of correctness of prediction among the test sets. For each subset, a classification model is constructed using the nine of the 10 folds and tested on the tenth one to obtain a cross-validation

estimate of its prediction accuracy. The 10 cross-validation estimates are then averaged to provide an estimate for the accuracy constructed from all the data.

TP = true positives: number of examples predicted positive that are actually positive

FP = false positives: number of examples predicted positive that are actually negative

TN = true negatives: number of examples predicted negative that are actually negative

FN = false negatives: number of examples predicted negative that are actually positive

Recall is the TP rate, what fraction of those that are actually positive were predicted positive? TP / actual positives

Precision is TP /predicted positive, what fraction of those predicted positive are actually positive?

$$Recall = \frac{tp}{tp+fn} \quad (4)$$

$$Precision = \frac{tp}{tp+fp} \quad (5)$$

The values of Recall and Precision range from 0 to 1 and higher values indicate better prediction results. In the best case, both Recall and Precision are equal to 1, which means the predictor detects all defect-prone classes without FN or FP. In practice, the values of Recall and Precision are usually mutually exclusive, i.e., high Recall value is often with low Precision value, and it is hard to achieve both high Recall and Precision at the same time [37].

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn} \quad (6)$$

The F – Measure is a measure that combines precision and recall.

$$F = 2.\frac{precision.recall}{precision+recall} \quad (7)$$

Mean absolute error (MAE)

The MAE measures the average magnitude of the errors in a set of forecasts, without considering their direction. It measures accuracy for continuous variables and is the measure of average deviation percentage in prediction. The MAE value closer to zero is considered as the better prediction capability by the model. MAE is defined as

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|\hat{m_i} - m_i|) \quad (8)$$

where n is the number of experiments, ^ mi is the predicted value and mi is the observed value. Area Under the ROC Curve (AUC) is used to evaluate the degree of discrimination achieved by the model.

Root mean squared error (RMSE)

The RMSE is a quadratic scoring rule which measures the closeness with which a model predicts the observation. A lower value of RMSE provides better goodness of fit.

$$RSME = \sqrt{Bias * Bias + Variation * Variation} \quad (9)$$

Bias is the average prediction of error. Variation is the standard deviation of prediction error. The Area Under the ROC Curve (AUC) is used to evaluate the degree of discrimination achieved by the model. The value of AUC is ranged from 0 to 1 and random prediction has AUC of 0.5. The advantage of AUC is that it is insensitive to decision threshold like precision and recall. The higher AUC indicates a better prediction.

G. 10-Fold Cross Validation

This validation method reduces the bias associated with the random sampling of data samples used in comparing the predictive accuracy of two or more methods. When using the cross-validation, some of the data is removed before the training begins. After the training is complete, the data that was removed can be used to test the performance of the learned model on "new" data. The cross-validation procedure randomly divides the dataset into 10 disjoint subsets, with each fold containing approximately the same number of records.

IV RESULTS

Accuracy, sensitivity and specificity were used to evaluate the performance of prediction models. The Mean Absolute Error (MAE) was measured for all the techniques. This was used to check the model's effectiveness.

TABLE VI. TABLE 2. MEAN ABSOLUTE ERROR AND ROOT MEAN SQUARED ERROR

	Mean Absolute Error			Root Mean Squared Error		
	L.R	J48	Naïve Bayes	L.R	J48	Naïve Bayes
v2.1.1	0.114	0.073	0.143	0.337	0.231	0.332
v2.1.2	0.119	0.029	0.083	0.345	0.148	0.258
v2.3.4	0.203	0.072	0.080	0.450	0.191	0.249

The J48 modelling technique had the best results with a value of 0.073 for v2.1.1 and 0.029 for v2.1.2 and 0.072 for v2.3.4. It had the lowest MAE and RMSE values.

TABLE VII. TABLE 3. RELATIVE ABSOLUTE ERROR AND ROOT RELATIVE SQUARED ERROR

	Relative Absolute Error			Root Relative Squared Error		
	L.R	J48	Naïve Bayes	LR	J48	Naïve Bayes
v2.1.1	38.7 %	24.82 %	48.51 %	87.87 %	60.38 %	86.68%
v2.1.2	40.9%	9.92 %	28.66 %	90.72 %	39.03 %	67.97 %
v2.3.4	68.2%	24.16 %	26.91 %	82.24%	50.63 %	65.91 %

In Table 3, the relative errors using the J48 had the least values for all the ElasticSearch versions.

TABLE VIII. TABLE 4. TRUE POSITIVE RATE AND FALSE POSITIVE RATE

	TP Rate			FP Rate		
	L.R	J48	Naïve Bayes	L.R	J48	Naïve Bayes
v2.1.1	0.714	0.857	0.679	0.075	0.048	0.124
v2.1.2	0.703	0.946	0.811	0.071	0.005	0.064
v2.3.4	0.696	0.957	0.913	0.165	0.012	0.024

According to the results in Table 4, the Naïve Bayes model had the least number of positives that were actually predicted positive. The J48 model had the highest TP Rate and least FP Rate.

The results in Table 5 show that the mean precision result for the J48 was greater than 0.9 and the mean recall for the model was 0.92.

TABLE IX. TABLE 5. PRECISION AND RECALL

	Precision			Recall		
	L.R	J48	Naïve Bayes	L.R	J48	Naïve Bayes
v2.1.1	0.787	0.854	0.632	0.714	0.857	0.679
v2.1.2	0.741	0.941	0.791	0.703	0.946	0.811
v2.3.4	0.747	0.920	0.894	0.696	0.957	0.913

F- Measure and ROC

The J48 model had the highest F-Measure mean of 0.938 for version 2.1.2. The Logistic Regression and the Naïve Bayes model had the F-Measure mean of 0.72 and 0.79 respectively for the same version.

TABLE X. TABLE 6. F- MEASURE

	Logistic Regression	J48	Naïve Bayes
v2.1.1	0.731	0.839	0.649
v2.1.2	0.720	0.938	0.793
v2.3.4	0.720	0.937	0.898

The evaluation results also demonstrated the effectiveness of the J48 model. The J48, Logistic Regression and Naïve Bayes had an average ROC Area mean of 0.952, 0.879 and 0.86 respectively for all the three versions.

V. DISCUSSION AND CONCLUSIONS

Software quality can be improved by defect prediction. The paper analysed the characteristics of the ElasticSearch process metrics that was extracted from the Github repository. The complexity of code change and how it can be quantified by entropy was discussed. Defect prediction models, i.e. the Logistic Regression, Naïve Bayes and J48 classifiers were designed. Experiments were conducted using full test and cross validation (10 folds). The Mean Absolute Error and Root Mean Squared Error to measure the effectiveness of the performance of the models. The evaluation results show that the model using the J48 had a strong prediction performance.The future work will propose a novel prediction model based on the J48 classifier and process metrics.

REFERENCES

[1] Hneif, M., & Lee, S. P. (2011). Using guidelines to improve quality in software nonfunctional attributes. IEEE Software, 28(6), 72–77. http://doi.org/10.1109/MS.2010.157

[2] Duarte, C. H. C. (2014). On the relationship between quality assurance and productivity in software companies. Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry - CESI 2014, 31–38. http://doi.org/10.1145/2593690.2593692

[3] Kapur, P. K., & Shrivastava, A. K. (2015). Release and Testing Stop Time of a Software : A New Insight. IEEE.

[4] Taipale, O., Kasurinen, J., Karhu, K., & Smolander, K. (2011). Trade-off between automated and manual software testing. International Journal of Systems Assurance Engineering and Management, 2(2), 114–125. http://doi.org/10.1007/s13198-011-0065-6

[5] Misirli, A. T., Bener, A. B., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. Software Quality Journal, 19(3), 515–536. http://doi.org/10.1007/s11219-010-9128-1

[6] Azeem, N. (2011). Defect Prediction Leads to High Quality Product. Journal of Software Engineering and Applications, 04(11), 639–645. http://doi.org/10.4236/jsea.2011.411075

[7] Strate, J. D., & Laplante, P. A. (2013). A literature review of research in software defect reporting. IEEE Transactions on Reliability, 62(2), 444–454. http://doi.org/10.1109/TR.2013.2259204

[8] Bell, R. M., Ostrand, T. J., & Weyuker, E. J. (2013). The limited impact of individual developer data on software defect prediction. Empirical Software Engineering, 18(3), 478–505. http://doi.org/10.1007/s10664-011-9178-4

[9] Caglayan, B., Tosun, Misirli A., Bener, A. B., & Miranskyy, A. (2015). Predicting defective modules in different test phases. Software Quality Journal, 23(2), 205–227. http://doi.org/10.1007/s11219-014-9230-x

[10] Zhang, L. F., & Shang, Z. W. (2011). Classifying feature description for software defect prediction. International Conference on Wavelet Analysis and Pattern Recognition, 138–143. http://doi.org/10.1109/ICWAPR.2011.6014475

[11] Wang, J., Shen, B., & Chen, Y. (2012). Compressed C4.5 Models for Software Defect Prediction. 2012 12th International Conference on Quality Software, 2(1), 13–16. http://doi.org/10.1109/QSIC.2012.19

[12] Li, M., Zhang, H., Wu, R., & Zhou, Z. H. (2012). Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19(2), 201–230. http://doi.org/10.1007/s10515-011-0092-1

[13] Dhiman, P., & Chawla, R. (2012). A Clustered Approach to Analyze the Software Quality Using Software Defects. 2012 Second International Conference on Advanced Computing & Communication Technologies, 36–40. http://doi.org/10.1109/ACCT.2012.1

[14] Harter, D. E., Kemerer, C. F., & Slaughter, S. A. (2012). Does software process improvement reduce the severity of defects? A longitudinal field study. IEEE Transactions on Software Engineering, 38(4), 810–827. http://doi.org/10.1109/TSE.2011.63

[15] Kastro, Y., & Bener, A. B. (2008). A defect prediction method for software versioning. Software Quality Journal, 16(4), 543–562. http://doi.org/10.1007/s11219-008-9053-8

[16] Wang, X., Guarino, N., Guizzardi, G., & Mylopoulos, J. (2014). Software as a Social Artifact: A Management and Evolution Perspective. Conceptual Modeling -- ER 2014, 8824, 321–334. http://doi.org/10.1007/978-3-319-12206-9

[17] Badr, Y., & Caplat, G. (2010). Software versioning and evolution in digital ecosystems. 4th IEEE International Conference on Digital Ecosystems and Technologies - Conference Proceedings of IEEE-DEST 2010, 381–386. http://doi.org/10.1109/DEST.2010.5610616

[18] Nguyen, B. N., Robbins, B., Banerjee, I., & Memon, A. (2014). GUITAR: An innovative tool for automated testing of GUI-driven software. Automated Software Engineering, 21(1), 65–105. http://doi.org/10.1007/s10515-013-0128-9

[19] Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. Proceedings of the 13th International Conference on Software Engineering - ICSE '08, 181. http://doi.org/10.1145/1368088.1368114

[20] Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher Bias : The Use of Machine Learning in Software Defect Prediction. IEEE Transactions on Software Engineering, 40(6), 603–616. http://doi.org/10.1109/TSE.2014.2322358

[21] Singh, P. K., Agarwal, D., & Gupta, A. (2015). A Systematic Review on Software Defect. IEEE, 1793–1797.

[22] Xia, Y., Yan, G., Jiang, X., & Yang, Y. (2014). A new metrics selection method for software defect prediction. 2014 IEEE International Conference on Progress in Informatics and Computing, 433–436. http://doi.org/10.1109/PIC.2014.6972372

[23] Rahman, F., & Devanbu, P. (2013). How, and why, process metrics are better. Proceedings - International Conference on Software Engineering, 432–441. http://doi.org/10.1109/ICSE.2013.6606589

[24] Malhotra, R., & Khanna, M. (2013). Investigation of relationship between object-oriented metrics and change proneness. International Journal of Machine Learning and Cybernetics, 4(4), 273–286. http://doi.org/10.1007/s13042-012-0095-7

[25] Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal, 23(3), 393–422. http://doi.org/10.1007/s11219-014-9241-7

[26] Xia, Y., Yan, G., & Zhang, H. (2014). Analyzing The Significance of Process Metrics for TT & C Software Defect Prediction. IEEE, 77–81.

[27] Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting Defects for Eclipse. In Third International Workshop on Predictor Models in Software Engineering (PROMISE'07).

[28] Koroglu, Y., Sen, A., Kutluay, D., Bayraktar, A., Tosun, Y., Cinar, M., & Hasan, K. (2016). Defect Prediction on a Legacy Industrial Software : A Case Study on Software with Few Defects. In 2016 4th International Workshop on Conducting Empirical Studies in Industry (pp. 14–20).

[29] Patil, T. R., & Sherekar, S. S. (2013). Performance Analysis of Naive Bayes and J48 Classification Algorithm for Data Classification. International Journal Of Computer Science And Applications, 6(2), 256–261.

[30] Muthukumaran, K., Choudhary, A., & Murthy, N. L. B. (2015). Mining github for novel change metrics to predict buggy files in software systems. Proceedings - 1st International Conference on Computational Intelligence and Networks, CINE 2015, 15–20. http://doi.org/10.1109/CINE.2015.13

[31] Chidamber, S., & Kemerer, C. F. (1991). Towards a Metrics Suite for Object Oriented Design. ACM, (1), 197–211.

[32] Herzig, K. (2014). Using pre-release test failures to build early post-release defect prediction models. Proceedings - International Symposium on Software Reliability Engineering, ISSRE, 300–311. http://doi.org/10.1109/ISSRE.2014.21

[33] Singh, V. B., Chaturvedi, K. K., Khatri, S. K., & Kumar, V. (2015). Bug prediction modeling using complexity of code changes. International Journal of Systems Assurance Engineering and Management, 6(1), 44–60. http://doi.org/10.1007/s13198-014-0242-5

[34] Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2010). Predicting faults in high assurance software. Proceedings of IEEE International Symposium on High Assurance Systems Engineering, 26–34. http://doi.org/10.1109/HASE.2010.29

[35] Xia, Y., Yan, G., & Si, Q. (2013). A study on the significance of software metrics in defect prediction. Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013, 2, 343–346. http://doi.org/10.1109/ISCID.2013.199

[36] He, Z., Peters, F., Menzies, T., & Yang, Y. (2013). Learning from open-source projects: An empirical study on defect prediction. International Symposium on Empirical Software Engineering and Measurement, 45–54. http://doi.org/10.1109/ESEM.2013.20

[37] He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering (Vol. 19). http://doi.org/10.1007/s10515-011-0090-3