



UNIVERSITY OF NAIROBI
FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTING AND INFORMATICS

**BLACK BOX TEST CASE PRIORITIZATION FOR REGRESSION
TESTING USING BAYESIAN NETWORKS**

W SELPHA ATEMBA

P53/34646/2019

SUPERVISOR:

DR ANDREW M. KAHONGE

**A research project submitted in partial fulfillment of the requirement for the award of
Master of Science Degree in Distributed Computing Technologies of the
University of Nairobi**

DECEMBER 2022

Declaration

This research project is my original work and to the best of my knowledge this work has not been submitted for any other award in any other University.


SAtemba

W Selpha Atemba

Date 9th March 2023

This project report was submitted in partial fulfillment of the requirement of the Master of Science in Distributed Computing Technologies of the University of Nairobi with my approval as the University mentor.

Dr. Andrew M. Kahonge
Department of Computing and Informatic

Signature  _____

Date: 9th March 2023

Acknowledgement

I give all glory to Almighty God for his graces and Mercies that kept me going even when I almost gave up. My heartfelt gratitude to my supervisor , Andrew M. Kahonge for his formidable support and guidance throughout the project, I wouldn't have come this far was it not for his encouragement. A special thanks to Dr. Christopher A. Moturi for the support and mentorship throughout the project work. A big thank you to my panel chair Dr. Lawrence Muchemi and all the panelists for the opportunity to do the research project, the comments and the feedback that improved my work every day. And to my amazing family; my lovely husband, Morris Kimathi, our two adorable sons; Bright Mwenda and George Mbatia, my parents and siblings for the love and support. Last but not least , I am grateful to family ,friend and colleagues for the encouragement and support during the entire period of project work.

Abstract

In testing, regression testing can be defined as the re-execution of all test cases previously executed to rule out the fact that some functionalities that were working previously have been broken by the newly introduced fixes or system changes. The impact is adverse if the software system involves money or life critical systems. Constraints like time and resources cannot allow re-execution of the whole collection of test cases that were executed previously and due to security reasons system codebase is never availed to the testing team and if it is availed, the testing team might not be technically competent to extract value from the code. The study investigated existing implementations of test case prioritization and had incorporated machine learning algorithm as part of the implementation. The study implemented a black box test case prioritization model using Bayesian Networks, after which a prototype was developed to prioritize test cases using the model. The study also validated that the developed prototype is effective in prioritizing test cases. Based on the methodology selected for the study, data was collected from public dataset, Kaggle all the variables under study were available in the data. Analysis was done on the data to visualize distribution of data. Feature engineering was done to the features to improve the performance of the model. The model was implemented using the established correlations between dependent variable; likelihood of detecting bugs against the independent variables; complexity value of the developed system, the level of experience for the developer who participated in the development of the system under test, the change history of the system, the bug history of the system and the tester assessment for likelihood of detecting bugs based on experience .A prototype was developed and tests done to validate the effectiveness, simulations were also carried out using the test data. The model was evaluated to establish its effectiveness in prioritizing test cases. The Bayesian Networks model performed slightly better in classification accuracy and confusion matrix when compared to Gaussian Naïve Bayes and Support Vector Machine respectively.

The study achieved the set-out objectives by carrying out systematic literature review on previous work and identifying the gap in regression testing for black box test case prioritization(Catal & Mishra, 2012), a model was implemented, and a prototype developed to deal with the issue of black box test case prioritization for regression testing. The effectiveness of the developed model was evaluated against other models and BN model was slightly better

than the other models. The study achieved its objectives, with the proposed solution software development teams will be able to prioritize test cases without the need to access source code using minimum training data. This will ensure high quality software is released and reduce the risk of defect leakage which can cause harm or threaten lives.

Keywords: Regression testing, Black box testing, Test case Prioritization, Bayesian Networks

The Table of Contents

Declaration	ii
Acknowledgement	iii
Abstract	iv
Abstract	v
Table of Contents	vi
Table of Contents	vii
Table of Contents	viii
List of Tables	ix
List of figures	x
Abbreviations	xi
Definition of Key Terms	xii
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement and Justification	2
1.3 Research Objectives	3
CHAPTER 2: LITERATURE REVIEW	4
2.1 Introduction	4
2.2 White box test case prioritization techniques	4
2.2.1 Genetic Algorithm	4
2.2.2 Bayesian Networks	5
2.2.3 Support Vector Machine map	6
2.3 Black box test case prioritization techniques	6
2.3.1 Unsupervised Neural Network	6

2.3.2 Neural Networks	7
2.3.3 Support Vector Machine Rank	7
2.3.4 Combined Machine Learning.	8
2.4 The Gap in test case prioritization	8
2.5 The proposed solution	8
2.6 Conceptual framework	9
CHAPTER 3: METHODOLOGY	10
3.1 Introduction	10
3.2 Study population	10
3.3 Data Collection	11
3.4 Variables	11
3.4.1 Dependent variables	11
3.4.1.1 Likelihood of detecting bugs	11
3.4.2 Independent variables	11
3.4.2.1 Change history	11
3.4.2.2 Bug history	12
3.4.2.3 Complexity value of the module	12
3.4.2.4 Tester assessment	12
3.4.2.5 Developer experience	12
3.5 Data Analysis	13
3.5.1 Dependent Variable Distribution Analysis	13
3.5.2 Independent variable analysis, to spot outliers	14
3.5.3 Establishment of Variable Correlations	15
3.5.4 Data Cleaning	16
3.6 Creation of BN model	17

3.7 Implementation and Prototyping	17
3.8 Model and Prototype Evaluation	18
CHAPTER 4: RESULTS AND DISCUSSIONS	19
4.1 Introduction	19
4.2 Model results evaluation	19
4.2.1 Classification accuracy	19
4.2.2 Confusion Matrix	20
4.2.3 Classification report	22
4.3 Prototype results evaluation	23
4.3 Discussion	24
4.4 Model Verdict	25
CHAPTER 5: SUMMARY,CONCLUSION AND RECOMMENDATIONS.	26
5.1 Summary of Findings	26
5.2 Conclusion	28
5.3 Recommendation	28
5.4 Future research	29
References	31

List of Tables

Table 1: Classification accuracy.....	14
Table 2: Classification report 1.....	24
Table 3: Prototype evaluation expected results.....	24
Table 4: Prototype evaluation actual results.....	25
Table 5: Classification report 2.....	26

List of figures

Figure 1: Conceptual framework.....	9
Figure 2: Research process.....	10
Figure 3:Dependent variable distribution analysis graph.....	14
Figure 4:Tester assessment distribution before cleaning.....	15
Figure 5: Feature correlation with each other heatmap.....	16
Figure 6: Script to replace outliers.....	17
Figure 7:Tester assessment distribution after cleaning.....	17
Figure 8: Belief network for dependent and independent variable	18
Figure 9: Belief network test case prioritization prototype	19
Figure 10: Confusion matrix for Bayesian Network.....	21
Figure 11: Confusion matrix for Gaussian Naïve Bayes.....	22
Figure 12: Confusion matrix for support vector Machine.....	23

Abbreviations

iOS - i-device Operating System

ST – Software Testing

SDLC – Software Development Life Cycle

SUT – System Under Test

API – Application programming Interface

BN – Bayesian Networks

BBN – Bayesian Belief Networks

AFPD – Average percentage of faults detected

NN – Neural Networks

Definition of Key terms

Software development life cycle(SDLC) – The model used in software development where a process that is systematic is followed. This approach ensures that required quality standards and the correctness expected to be in the system is achieved.

Software testing - The process of evaluating a system or the software with the intent of establishing if it meets the requirements specifications or not.

Regression testing - A type of testing in the SDLC that is executed after every change of the system i.e., fixing of a defect in the system, updating the existing functionalities and adding new functionalities to the system.

Test case prioritization – The art of arranging test cases in a specific order with the intention of having the most important test cases come before the less important. The arrangement could be based on different factors like code coverage and functionality criticality.

Bug - A malfunction in the system or an error that may cause system components or the entire system fail to perform its required functions.

Bayesian Networks – A model type which combines probability and graphs. It computes probability using Bayesian Inferences. It's main aim is to model dependence that depends on conditions bringing about cause, it does this by representing edges in a directed graph to show dependence that is conditional.

Machine learning algorithm – The method by which systems that are intelligent artificially(AI) predicts some outcomes using given data as input.

CHAPTER 1: INTRODUCTION

1.1 Background

The history of software development is dated back to 1948 on June 21 at 11:00 a.m. when Tom Kilburn ran his first piece of software at the University of Manchester in England(Booch. G, 2018). Programming languages like Fortran, Cobol, BASIC, and C arrived in the two decades that followed. Today software development has become ubiquitous and recent programming languages like Java and Python have been used in different software development projects. Others like Go and Swift programming language for iOS by Apple are quite new in the industry(M. S. Mahoney, 2004).

Software development methodologies, on the other hand, have evolved immensely. It started back in the 1950s with structured programming, waterfall model, iterative and incremental models, prototyping, spiral model, V-model, rapid application development, and finally agile in the 1990-2000s(Misra et al., 2012). The agile methodology has also evolved over the years from a scrum, which basically entails practices like lean software development methodology, Kanban, extreme programming(XP)methodology, continuous integration(CI) practices, continuous delivery(CD) to scrum-of-scrums which is a scrum at scale i.e. more than one team. (Tom DeMarco, 2000).

Software testing (ST); the process of running a software program with a specific intention of finding defects, which is an integral part of the Software Development has also evolved over the years. It started in the 1940s with debugging, then followed the demonstration phase in the 1950s, then destruction, then evaluation, and finally the prevention phase from the 1980s to the present. There are different types of testing i.e. unit testing, component testing, smoke testing regression testing, integration testing, API testing white-box, and black-box testing(H. Freeman, 2002) only to mention a few.

Regression testing is the re-execution of test cases when requirements are changed, functionality is enhanced, or a defect is fixed. This makes sure that any changes do not have any unexpected consequences, and the system under test (SUT) still works as per the requirements. Regression

testing can be done either manually or be automated. Techniques for regression testing are re-test all where all the test cases are re-executed (Malishevsky et al.,2002), test case selection where test cases associated with the changed modules are selected and executed against the SUT (Kazmi et al.,2017), and “test case prioritization where the test cases of the regression test suite are reordered based on some particular criteria so that test cases that have high likelihood of detecting defects are executed first” (Khanna, 2016) making it possible to catch maximum errors with the available resources.

1.2 Problem Statement and Justification

With the current software development methodologies like the agile methodology, organizations are making frequent releases, and this means vibrant and more proactive testing is needed to cover all possible defects that could be introduced with the changes to the software that could affect the functionality or performance of the system. Regression testing is needed whenever a software system is reorganized or modified.70% of the testing cost is consumed by regression testing alone(Labuschagne et al., 2017). This is not easily achievable with the growing amount of software and test cases, having limited resources like time, money, and human resources involved in testing. Current test case prioritization techniques require interaction with the actual code (White box) to track code changes and map them to test cases and use that to determine the scenarios that should be executed first for earlier bug detection. System testing for complex systems does not allow interaction with or there could be a lack of trained resources to analyze source code hence techniques that do not require access to source code (black box) test case prioritization techniques become necessary. Most machine learning test case prioritization techniques implemented previously require a lot of training data to effectively prioritize test cases. Training data on the other hand might be insufficient during regression testing phase. This research paper proposes a black box test case prioritization technique that is built using machine learning algorithm: Bayesian Networks. The solution will use change history of the system being tested, bug history of the system being tested developer experience, tester assessment and complexity value of the module under development for training the model which will later be used to predict and thereafter

prioritize test cases based on the ability to identify bugs as early in the early stages during the system testing, as a solution to challenges associated with regression testing. This approach will automatically prioritize test cases that are important and have the highest probability of detecting errors in the updated software. This will improve the effectiveness of testing by detecting defects early, hence reducing cost and improving the quality of software released.

1.3 Research Objectives

The study intends to achieve the following set objectives

- I. To investigate existing techniques of prioritizing test cases with a focus on the algorithms that use machine learning.
- II. To implement a Blackbox test case prioritization prototype for regression testing using Bayesian Networks (BN).
- III. To validate the effectiveness of the developed prototype in identifying bugs

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

Existing techniques for prioritizing test cases are explored to establish their how they influence this study. Methodologies of prioritizing test cases using machine learning algorithms will be analyzed to determine their effectiveness in test case prioritization. Existing algorithms for prioritizing test cases using Bayesian Networks will be explored to prove their relevance to the topic of study. Limitations and gaps of previous implementations will also be explored.

The gap identified from literature review will inform the approach and design for implementation of the proposed solution

2.2 White box test case prioritization techniques

2.2.1 Genetic Algorithm

Genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. It reflects the process of natural selection where the fittest individuals are selected for reproduction(Sieja et al., 2019) so that they can reproduce offspring for the next generation.

Gohner and Abeleand used genetic algorithms for ordering test cases based on priorities. Their methodology uses software agents which are described as components of a software application used to predict the fault revealing probability of each test case and the fault proneness of each module. Their methodology also incorporates fuzzy logic rules. The fuzzy logic rules state that; complex modules are likely to have more faults, cases that have identified defects previously are most likely to identify defects in subsequent executions and that components that have been faulty

in the past are also likely to be faulty in future. The methodology uses genetic algorithm to update the weight of rules of fuzzy logic in that it uses the difference between the faults predicted and faults found as input. It uses cases that are likely to identify most defects.(Mahoney, M.S.,2004)

Ramingwong and Konsaard proposed a genetic algorithm methodology that has modifications for test cases prioritization. Prioritization of test cases Itest cases are prioritized based on code coverage. (Konsaard & Ramingwong, 2015)

A. Ahmed and M. Shaheen proposed a more advanced genetic algorithm-based test case prioritization technique(97). It uses a control flow graph for each test case, it measures three metrics: the coverage degree of each statement, conditions and multiple conditions. All the metrics are then integrated through genetic algorithm and after each iteration the fitness function of the algorithm outputs a value(81). The order of execution is determined by calculating the values for each test case.(A. A. Ahmed, 2012)

2.2.2 Bayesian Networks

Bayesian networks (BN) are models that combine graphs and probability using inferences of Bayesian computations of probability. Bayesian networks operate by modelling dependence that is conditional which leads to causation, it usually is represented by a dependence that is determined by conditions on edges that appear on a graph that is directed.

Tahvildari and Mirab came up with technique to order test cases based on priorities using probabilities by using BN to integrate software program's Whitebox details into a model that is consolidated . This methodology uses the following data as input; fault proneness of the system, degree of coverage and the changes made in the software system. Using BN a relationship for the data is established based on initial outcomes a probability is generated for each test scenario with algorithms that have probability inferences. Scenarios of testing are then ordered using the created probability values. Performance of the algorithm's performance is measured using the average percentage of faults detected (APFD). (Mirarab & Tahvildari, 2007)

Tahvildari and Mirab went ahead and improved their mechanism by prioritizing test cases based on metrics that determine quality of code, changes that have taken place and the coverage of the test scenarios. They gather information by comparing different versions of a file and comparing bytecodes. If a test case covers similar code modules as the previous test case, then the test case will receive a low priority score (S.MirarabandL, 2016)

Huang et al. proposes a more advanced methodology that implements code coverage based grouping using BN. The grouping of test case is determined by the similarity in code coverage and the likelihood of failure using Bayesian Networks(BN).The methodology rules out the possibility of similar test cases being executed with the same probability.(Huang et al., 2021)

2.2.3 Support Vector Machine map

Support Vector Machine (SVM) is a machine learning technique that can be applied to perform both classification and regression tasks.

Xie and Busjaeger propose SVM map for test case prioritization .The methodology uses different heuristic techniques like code coverage, test age, test-failure, similarity in text between tests and changes, fault history and test-failure history. These information is utilized in training the ranking model which is used in test case prioritization . (Busjaeger & Xie, 2016)

2.3 Black box test case prioritization techniques

2.3.1 Unsupervised Neural Network

Neural networks(NN) belongs to a category of machine learning algorithms. They take inspiration from human brain and replicate the way biological neurons communicate with each other. Unsupervised NN are the ones that do not use training data. They learn from experience.

Gokce et al suggested a technique utilizing neural networks for prioritizing test cases. The test cases are arranged according to their degree of preference. The degree of preference is calculated based on various factors, including the average frequency of usage, the degree of balance, the distance, the number of belongings, the number of sub-nodes and the number of layers of occurrence. Furthermore, the work was expanded by incorporating unsupervised Neural Network and fuzzy c-means for clustering events. (Gökçe et al., 2006)

2.3.2 Neural Networks

Neural networks belong to a category of machine learning techniques and are designed to imitate the way biological neurons communicate in the human brain.

Spieker et al. suggest a novel approach to prioritize test cases, it utilizes reinforcement learning and Neural Networks(NN). Reinforcement learning (RL) learns from experience while NN uses variables like duration of execution, last execution and failure history uses agents, the agent interacts with the environment to gather data, which is represented as a state, and then chooses an appropriate action based on the state. After the state execution, the agent learns from the feedback received and adjusts behavior accordingly. Positive feedback is reinforced while negative feedback is discouraged. In this approach, priority is given to test cases that have previously identified faults, with a focus on executing those with a higher capability of detecting defects first. (Spieker et al., 2017)

2.3.3 Support Vector Machine Rank

The SVM Rank is a version of the Support Vector Machine algorithm that is specifically designed for solving ranking problems by training a model to learn how to rank items.

Lachman et al. proposes black box test case prioritization method using SVM Rank. It uses requirements coverage, cost of test execution, age and count of failures and requirements priority. The method also entails parsing through requirements text. A computation of all words is done, and each word will represent a feature. Using SVM Rank algorithm, a ranked classification model

is trained on the data, which is then evaluated for its effectiveness using the APFD metric as chosen by the authors. (Lachmann et al., 2016)

2.3.4 Combined Machine Learning.

An approach for test case prioritization was developed using several machine learning algorithms, including Neural Networks(NN), k-nearest neighbor(KNN) and logistic regression. These algorithms' results were merged to create an ensemble learner.

Combinatorial ensemble learning was used to create ensemble learner, which was trained using historical data. The classification results of the latest versions are combined, and this helps in improving the quality of prioritization. Classifiers are trained based on insights gathered from past executions. The combinatorial ensemble uses classifiers for the same version of test cases. The priority value of a test case is determined by each individual classifier and these priorities are aggregated to establish the overall priority. (Lachmann, 2018).

2.4 The Gap in test case prioritization

From the literature review done, it was observed that most prioritization techniques require access to code (white box) which at most times is not possible. This called for a technique that can prioritize test cases without the need to access the source code. During the literature review, methods were identified that can prioritize test cases effectively without requiring access to source code, but these techniques rely heavily on having a significant amount of training data.

2.5 The proposed solution

To solve the above found challenge, this study proposes a solution that can prioritize test cases without need to access the source code using minimal training data; black box test case prioritization using Bayesian Networks.

2.6 Conceptual framework

The model construction is a combination of variables used in previous studies; change history, bug history, complexity value of the module and developer experience and additional variables; tester assessment which will be the additional variable that will be explored in this study. The Bayesian Network model construction will use dependent and independent variable. The value of dependent variable; likelihood of detecting bugs will be determined by independent variables; change history, bug history, complexity value of the module, tester assessment and the developer experience.

Figure 1 displays the conceptual framework of the Bayesian Network(BN) model for test case prioritization.

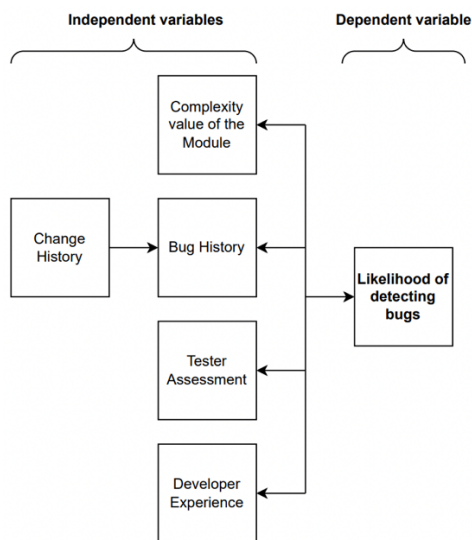


Figure 1: Conceptual framework

CHAPTER 3: METHODOLOGY

3.1 Introduction

In this section we give a detailed description of how we achieved the set-out objectives for this study. We conducted a theoretical analysis of the literature to investigate the implementation and limitations of test case prioritization techniques, with a specific emphasis on approaches employing machine learning. We examined both black box and white box prioritization methods. We collected test data for fintech systems. The data collected was cleaned in preparation for training. A BN algorithm model was designed, and subsequently conducted simulations and prototyping on the model. The BN model was then trained and evaluated.

The methodology utilized in this study is depicted in the figure below.

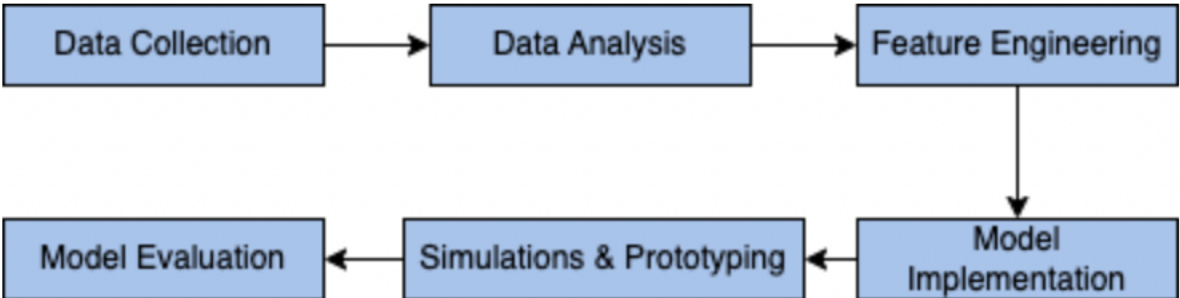


Figure 2: The process of Research

3.2 Study population

This research focused on fintech software system. Finance happens to be a fast-growing industry which has quickly picked technology advancements and innovation making it one of the fastest growing industries that has embraced technology. Security being of the challenges brought about by technology, fintech is not left out to that regard. Fintech systems require comprehensive

testing to remove any chances of security breaches or defects that can compromise customers money, hence the need for the study for an advanced mechanism to easy identify bugs and reduce the risk of defect leakage.

3.3 Data Collection

The training and testing data for the research was extracted from fintech systems test cases data in the Kaggle dataset (Roshan, 2019). The datasets contain fintech systems features test cases and columns that were used in this study. The input variables under study includes complexity value of module of the SUT, the change history of the SUT, the bug history of the SUT and the experience of the developers who developed the SUT. The prediction that a test case will detect a bug based on tester assessment and experience data was appended.

3.4 Variables

3.4.1 Dependent variables

3.4.1.1 Likelihood of detecting bugs

The dependent variable likelihood of detecting bugs is Boolean, it has two values: true or false. The independent variables will determine whether a test case has the capability of detecting defects or not by having a value of true or false.

3.4.2 Independent variables

3.4.2.1 Change history

Change history defines the amount of change a system or a module has undergone since its development. It has a value of 1 to 5, 1 being little or no change and 5 being maximum changes. A system module with less changes has low chances of having bugs while a module that has undergone many changes has high chances of having bugs (Nagappan et al., 2010).

3.4.2.2 Bug history

Bug history defines the historical defects track of a module or the SUT. It describes whether a system module has had many bugs during the previous executions or not. It has a value of 1 to 5, 1 being minimal or no bugs and 5 being the greatest number of bugs. A system module with high bug history is likely to have more bugs while a system module with a low bug history is likely to have less or no bugs (Zimmermann et al., 2008).

3.4.2.3 Complexity value of the module

The complexity value of a module defines the ease of understanding and developing a module. It states if it will be easy or hard to develop a module. It has a value of 1 to 5, 1 being very simple and easy to develop while 5 is very complex and difficult to develop. Complex system modules have high probability of having bugs than less complex system module. (Yu et al., 2010).

3.4.2.4 Tester assessment

Tester assessment is the prediction of a tester on the likelihood of a system module in having bugs based on their experience. It has a value of 1 to 5, 1 being less likely or not likely to detect any bug and 5 being very likely to detect bugs. The tester assessment is the additional variable that was added to existing and previously studied variables of bug detection.

3.4.2.5 Developer experience

The developer experience defines the expertise of the person developing the module of the system under test. A developer can be experience or inexperienced. It has a value of 1 to 5, 1 being less experienced and 5 being very experienced. Developer experience is inversely proportional to test case likelihood of detecting bugs, the more experienced the developer the less defects a developed module is likely to have. (Kini et al., 2018).

3.5 Data Analysis

To get the best results out of the Bayesian Network yet to be constructed, Exploratory data analysis was conducted. This consisted of the following activities:

- i. Dependent variable distribution analysis, to make sure the dataset is not skewed towards one variable
- ii. Independent variable distribution analysis, to easily spot outliers
- iii. Establishment of variable correlations, to how the variables relate to each other. If two variables have a strong correlation, then their nodes can be connected by an edge. Given the small number of independent variables, all the variables were used to build the belief network.

3.5.1 Dependent Variable Distribution Analysis

To avoid building a network based on a skewed dataset, the dependent variable's distribution was first investigated. This was done by plotting a graph of the count of both outcomes (likelihood of detecting bugs and likelihood of not detecting bugs). Figure 3 shows the analysis graph of dependent variable distribution.

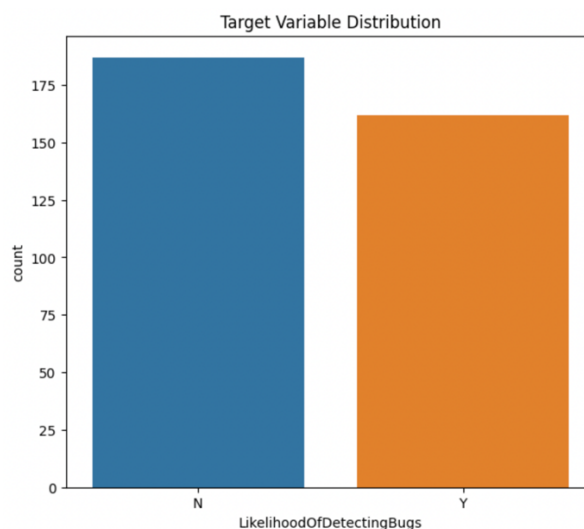


Figure 3: Dependent variable distribution analysis graph

The analysis results showed that the distribution between positive and negative dependent variables is almost even. Based on these results, there was no need to employ either oversampling or under sampling techniques to balance the dataset.

3.5.2 Independent variable analysis, to spot outliers

Given that all the independent variables were labels in nature (either 0,1,2,3,4 or 5), bar graphs were preferred to boxplots in order to establish the existence of outliers. Figure 4 shows tester assessment distribution results

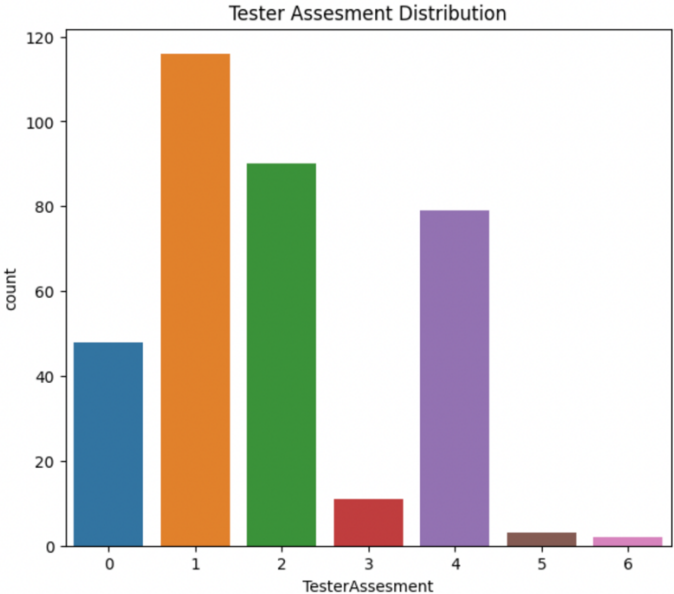


Figure 4: Tester assessment distribution before cleaning

The results show the existence of outliers (records existing with the label '6'). A similar graph was plotted for the other variables, and the outliers were replaced with the most immediate label (if outlier is label 6, then replace with label 5)

3.5.3 Establishment of Variable Correlations

Correlation between the registered features (Change history, Bug history, developer experience, complexity value of module and tester assessment) was plotted using a heat map, and the resultant plot informed the design of our Bayesian network. Figure 5 shows feature correlation with each other heatmap.

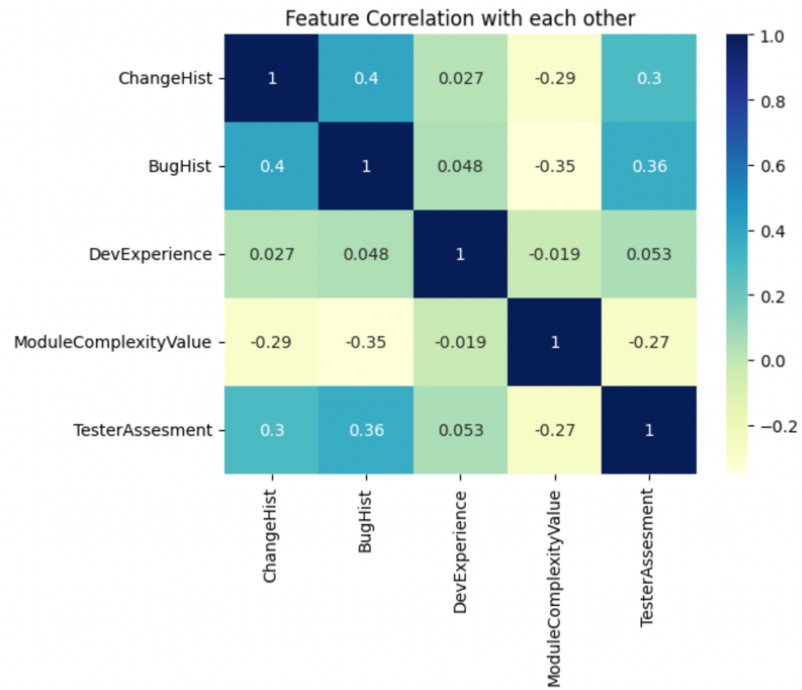


Figure 5: Feature correlation with each other heatmap

The above correlation heatmap offers invaluable insight that further informed this research. Correlation values greater than 0 imply a positive correlation; an increase in one variable tends to be accompanied by an increase in the other variable as well. If the correlation value is less than 0, it indicates a negative correlation, which implies that as one variable increases, the other variable tends to decrease.

From the heatmap, we deduce that there is a strong correlation between bug history and change history (0.4). As such, in our network, one of these variables will be modeled as the parent node, and the other as a child node.

The other independent variables have little correlation with each other and will therefore be modelled as independent nodes.

3.5.4 Data Cleaning

In data cleaning, the dataset was checked for duplicates, and these were removed. Since Bayesian belief networks are probabilistic, a row of data (both the independent and dependent variables) had to be similar to another row in order to be considered a duplicate. Such duplicates were dropped from the dataset.

```
df_dep_variables['TesterAssesment'] = df_dep_variables['TesterAssesment'].replace(to_replace=6, value = 5)
```

Figure 6: Script to replace outliers

From the previous exploratory phase, it was noticed that the outliers only appeared past the maximum label (5), and not the minimum. As such, the outlier values were replaced with the dataset’s maximum label. Figure 7 shows tester assessment distribution after cleaning.

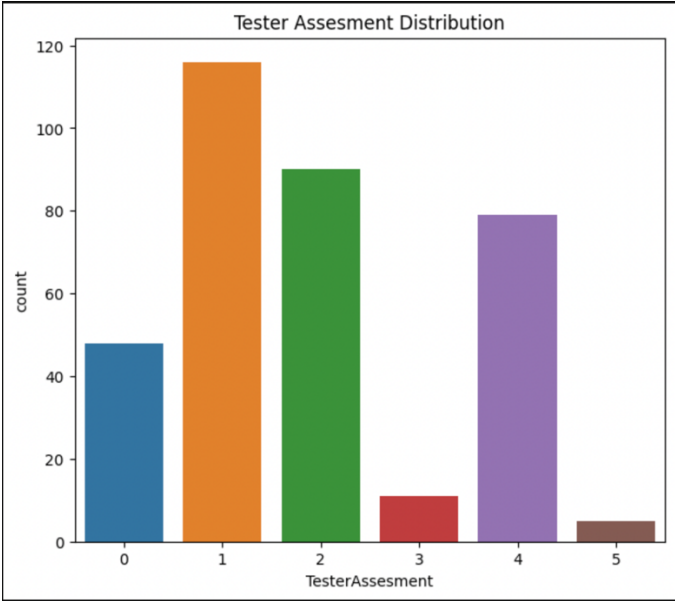


Figure 7: Tester assessment distribution after cleaning

3.6 Creation of BN model

The creation of a Bayesian network starts with the identification of nodes, and how these connect to edges. Since LikelihoodOfDetectingBugs is our target variable, it will be the end node, and the other nodes will connect to it either directly or indirectly, through other nodes.

From our earlier findings from the data analysis phase, we know that there are variables that have a strong correlation(change history and bug history). These will therefore be connected as parent and child nodes, and then branch will be connected to the LikelihoodOfDetectingBugs node. Figure 8 is the designed Bayesian Network; it shows the interaction of the various independent and target variables.

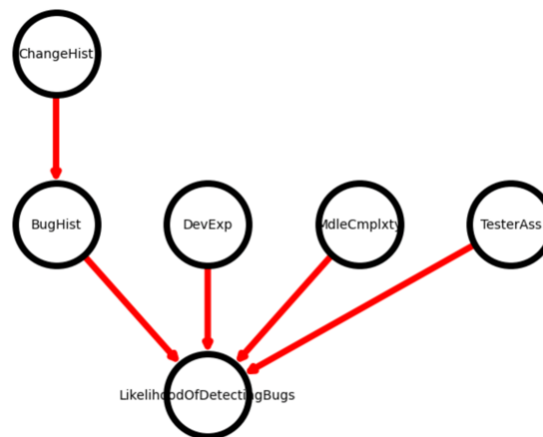


Figure 8: Bayesian Network for dependent and independent variables

3.7 Implementation and Prototyping

Python programming language was utilized in the research to create model architectures. The environment was set up on a personal computer that was running windows 10 Operating system. The programming language used was Python 3.6 with the libraries used for model creation were Pandas and NumPy for data manipulation, networkx, matplotlib and seaborn for data

visualization, pybbn for creating Bayesian Belief Networks. Flask framework was used for prototyping.

The BN model was trained using 80% of the data available; 800 records. The prototype requires that the user captures system under test information in relation to the test case being run, it will then compute the likelihood of a test case detecting bugs and display the results. Figure 9 shows Bayesian Network test case prioritization prototype user interface.

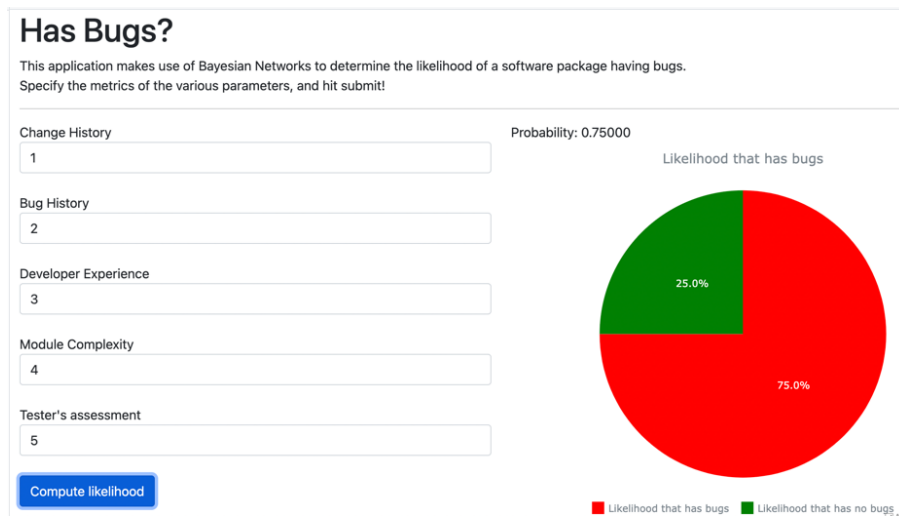


Figure 9: Bayesian Network test case prioritization Prototype

3.8 Model and Prototype Evaluation

The BN model was evaluated against Gaussian Naïve Bayes and SVM algorithms. The module's effectiveness was evaluated using metrics such as classification accuracy and a confusion matrix. The prototype developed from the module was tested using 20% of the testing data acquired from Kaggle dataset.

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1 Introduction

Bayesian Network's performance is measured during testing. In this chapter the focus is on determining whether the model's performance is satisfactory and whether it can enhance early identification of defects in software systems. The study evaluates the performance of Bayesian Networks(BN) for test case prioritization by comparing it to other machine learning algorithms, namely Gaussian Naïve Bayes and Support Vector Machine(SVM). Lastly, the paper discusses the efficiency of the proposed approach for constructing models to prioritize test cases in the software industry.

4.2 Model results evaluation

BN's performance is evaluated against other algorithms to determine the accuracy. The performance of other algorithms is measured as well. Various evaluation measures were employed to measure the model's performance, including the following classification metrics: classification accuracy, confusion matrix and classification report.

4.2.1 Classification accuracy

The classification accuracy metric is utilized to determine the proportion of the correctly predicted labels out of all the predictions made by the model. BN classifier performed slightly better than its counterparts, Support Vector Machine and Gaussian Naïve Bayes algorithm. Bayesian Network was the best followed by Support Vector Machine, Gaussian Naïve Bayes was the lowest of the three in classification accuracy. Table 1 shows classification accuracy scores of models under test.

Model	Accuracy Score
Bayesian Network	0.83
Gaussian Naïve Bayes	0.80
Support Vector Machine	0.81

Table 1: Classification Accuracy

4.2.2 Confusion Matrix

To present the model’s performance, a confusion matrix was generated, which depicts the number of correct predictions made by the classifier and where the classifier became confused. The confusion matrices for various algorithm classifiers are illustrated in the diagrams below.

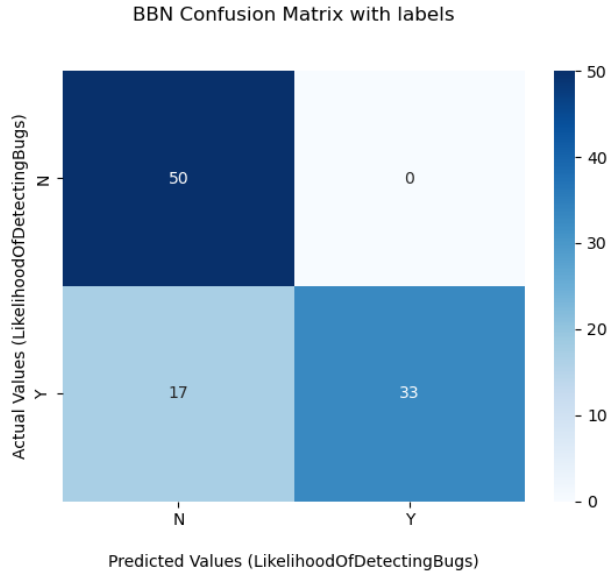


Figure 10: Confusion matrix for Bayesian Network

Figure 10’s confusion matrix illustrates that BN correctly classified 50% of test cases without bugs and 33% as with bugs. The BN classifier however got confused and classified 17% of test cases with bugs as without bugs and did not classify any test case without bugs incorrectly scoring 0%.

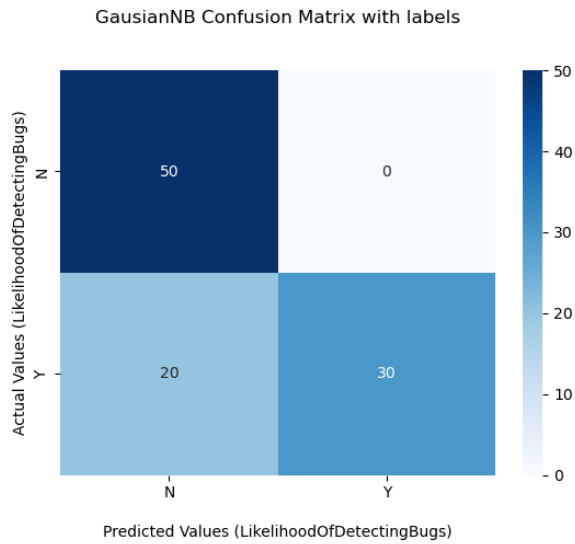


Figure 11: Confusion matrix for Gaussian Naïve Bayes

The confusion matrix in figure 11 shows that Gaussian Naïve Bayes gets confused and classifies 20% of test cases with bugs as without bugs and does not classify any without bugs test cases incorrectly scoring a value of 0%. The classifier correctly classifies 50% of without bugs test cases and 30% of with bugs correctly.

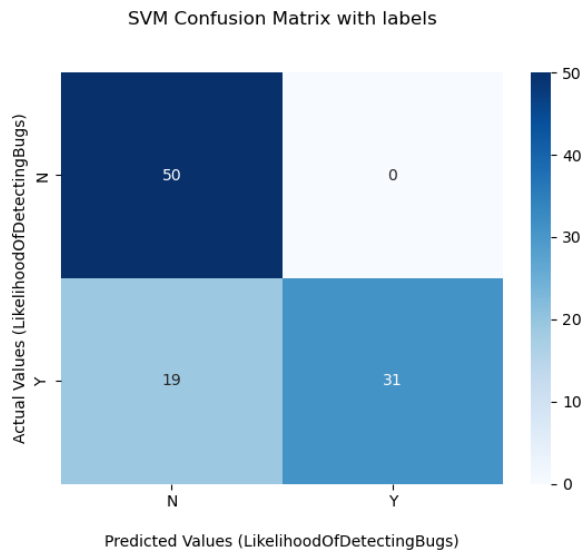


Figure 12: Confusion matrix for Support Vector Machine

In figure 12 the support vector machine classifier classifies correctly 50% of test cases without bugs and 31% of test cases with bugs. The classifiers, however, gets confused and classifies 19% of test cases with bugs as without bugs and can't predict any test cases without bugs as with bugs leading to a score of 0%.

4.2.3 Classification report

Additional metrics, such as precision, recall and F1 score, were examined to provide further evidence of the BN model's performance. Precision is calculated by dividing the number of correct predictions made by the model by the total number of predictions. Recall is determined by dividing the number of true positives by the total number of true positives plus false negatives. F1 score combines precision and recall metrics into a single metric. Thus, if both precision and recall metrics are high, F1 score will also be high, while if both metrics are low, F1 score will also be low.

In precision, BN performed slightly better than SVM which was better than Gaussian Naïve Bayes. Similarly in recall, F1-Score and accuracy BN performed slightly better than the other algorithms.

Table 2 below shows the classification report for BN, Gaussian Naïve Bayes, and Support Vector Machine algorithms.

Model Name	Precision	Recall	F1-Score	Accuracy
Bayesian Network	0.8731	0.83	0.8249	0.83
Gaussian Naïve Bayes	0.8571	0.80	0.79167	0.80
Support Vector Machine	0.8623	0.81	0.8029	0.81

Table 2: Classification report 1

4.3 Prototype results evaluation

The prototype developed using BN model was tested using 300 records of the testing data. Two tables were generated, one for the expected results and the other for the actual results produced after the test. Table 3 shows the expected results.

Change History	Bug History	Developer Experience	Module Complexity	Tester Assessment	Likelihood of detecting bugs
1	2	3	4	5	Y
1	1	3	4	5	Y
1	1	1	1	1	N
1	2	2	2	2	Y
5	5	5	4	4	Y
2	1	1	2	5	N
2	2	1	1	5	N
1	2	1	2	4	N
2	1	2	1	3	N
1	1	1	1	2	N
1	1	1	2	2	N

Table 3: Prototype evaluation expected results

The prototype was subjected to tests using the test data. Table 4 below shows the actual results from running the tests. From the test data only one record is classified incorrectly. The prototype simulations confirm that BN is effective in prioritizing test cases.

Change History	Bug History	Developer Experience	Module Complexity	Tester Assessment	Likelihood of detecting bugs
1	2	3	4	5	Y
1	1	3	4	5	Y
1	1	1	1	1	N
1	2	2	2	2	Y
5	5	5	4	4	Y
2	1	1	2	5	N
2	2	1	1	5	N
1	2	1	2	4	N
2	1	2	1	3	N
1	1	1	1	2	N
1	1	1	2	2	Y

Table 4: Prototype evaluation actual results

4.3 Discussion

Bayesian Network for Test case prioritization is the proposed solution to regression testing challenges when there is no access to source code of the system under test. Bayesian networks algorithm allows for test case prioritization with very minimal variables of the system under test as input without the need to access source code. It classifies accurately using minimum amount of data. The efficiency of BN in classification against other similar algorithms are discussed. We tested the effectiveness of the model on open-source data on Kaggle datasets (Roshan,2019).

From the confusion matrices BN classifier has a lower confusion 17% compared to Gaussian Naïve Bayes and support vector machine classifiers which have 20% and19% accordingly. This shows how effective the model is when it comes to classification given the fact that less data was used for training the model. (Luis et al, 2000)

The classification report and the classification accuracy show similar outputs shows similar reports as Bayesian Network algorithm performs better than support vector machine and Gaussian Naïve Bayes algorithms.

The experiments prove beyond doubt that BN algorithm will be the best solution for test case prioritization when we have limited training data and no access to the source code. It performs better than alternative models and gives accurate results for efficiency and effectiveness.

4.4 Model Verdict

Tests were conducted to establish a benchmark on the Bayesian Network algorithm and its performance in classifying test case's ability to detect bugs.

As the results indicate, the machine learning models exhibit diverse performance levels when tested with the dataset. F1-Score was selected as the main measure of performance since it shows the average of all other measurement metrics. F1 – Score shows BN outperforms other algorithms. The results further show that BN algorithm performs better even with minimum amount of data. The table below shows the comparison of performance between BN and other two algorithms using F1- Score; a higher F1 – Score means the model has a better performance and vice-versa for lower scores.

Model Name	F1-Score	Rank
Bayesian Network	0.8249	1
Gaussian Naïve Bayes	0.79167	3
Support Vector Machine	0.8029	2

Table 5: Classification report 2

CHAPTER 5: SUMMARY, CONCLUSION AND RECOMMENDATIONS.

5.1 Summary of Findings

Objective 1: To investigate test case prioritization techniques with a focus on those that incorporate machine learning algorithms.

For machine learning-based techniques for prioritizing test cases, access to source code and a considerable amount of training data is necessary to predict likelihood of a test case in detecting bugs, this poses a challenge since the requirements for prioritization are not always guaranteed. The literature review conducted on test case prioritization in a systematic manner for techniques that incorporate machine learning algorithms was carried out. The review sorted the literature into two categories; techniques that necessitate access to the system under test(SUT) source code(white box), and techniques that do not require access to the SUT's source code(black box) Under white box test case prioritization Genetic algorithm, Bayesian Networks(BN) and Support Vector Machine(SVM) algorithms are explored. Genetic algorithm uses software agents, code coverage of the SUT and conditional coverage variables to predict test case probability of a test case in identifying bugs. Bayesian networks uses system's fault proneness, code coverage, system changes, metrics related to quality, code changes, and coverage levels variables in prioritizing test cases. Support vector machine uses code coverage, test age, test failure, Test-case prioritization involves utilizing variables such as similarity between teste and changes, fault history, and test-failure history to prioritize test cases.

Regarding black-box test case prioritization, unsupervised neural networks utilize variables such as frequency of usage, balancing degree, distance, number of belongings, number of sub-nodes, and number of occurrence layers to prioritize test cases. Neural networks use variables such as execution duration, last execution, and failure history. Through interaction with the environment, agents learn from positive feedback and avoid negative feedback. Support Vector Machine Rank uses requirements, cost of test execution, requirements priority age and count of failures as variables to predict bug detection and prioritize test cases.

Objective 2: To implement a Blackbox test case prioritization prototype for regression testing using Bayesian Networks (BN).

A prototype was developed for black box regression test case prioritization using Bayesian Networks model. Python programming language was used with windows 10 operating system as the main implementation platform. For data manipulation, the Pandas and NumPy libraries were utilized, whereas the network, matplotlib, and seaborn libraries were used for data visualization, pybbn library was used for creating Bayesian Belief Networks. Flask framework was used for prototyping. The model presented a user interface where the user is supposed to key in system details and a prediction of whether a bug will be detected or not is given. Using this information, a tester can prioritize test cases.

Objective 3: To validate the effectiveness of the developed prototype in identifying bugs

BN test case prioritization technique prioritizes test cases effectively, BN model is also accurate in classification. Data from Kaggle was split, 20% of the data was used for testing the prototype's effectiveness in test case prioritization. The test confirmed that the developed prototype is effective in prioritizing test cases without the need to access source code using minimal training data. BN's classification accuracy is measured against other two algorithms and BN performs slightly better in classification and confusion matrix when compared to the other two algorithms.

5.2 Conclusion

This study proposed Bayesian Networks algorithm as a solution to blackbox regression testing. The solution can accurately prioritize test cases without the need to access SUT's source code and using minimum training data. This solves the problem of the need to access SUT's source code to be able to prioritize test cases. (S. Mirarab, 2020).BN also identifies conditional dependencies between nodes of the variables of study and uses the information to map relationships between nodes which in turn build a relationship model that can predict probabilities for variables under study. Consequently, BN succeeded in prioritizing test cases with greater precision. In comparison to other algorithms put forth by researchers, BN achieved a lower false-positive rate as well as a high F-1 score(T.Xie, 2016).

The BN model proposed in this study was tested and determined to be effective for prioritizing test cases in black-box regression testing.. The model allows all levels of testers performing regression testing be able to prioritize test cases effectively since there is no need to access the SUT's source code nor have technical knowledge of the software source code. The software testers need to have some basic knowledge of the required variables and they can prioritize test cases.

This solution is efficient and effective for the current software development methodologies where releases are more frequent especially when the system under test handles sensitive information or performs critical operations that can save life or involves money. With constrained resources any testing team can achieve maximum quality for the product with minimum effort.

5.3 Recommendation

The research found out that most complex and critical systems do not allow for source code to be shared to the testing team during regression testing. Thus, test case prioritization techniques that do not necessitate access to source code are recommended for easier identification of defects and delivery of quality products to users.

5.4 Future research

Technology is evolving quickly, and all industries are now integrating with technology, this study was limited by training data. For future studies, a variety of different industry datasets can be used to train and validate the effectiveness of the model. This also allows for a wider coverage in scope for all the available industries. Training and validation can further be broken down into dataset from the same industry with different programming languages for the systems and different localities, or different language implementations of application under study. That way the model can be applicable globally for different application implementations.

An experiment to test the validity of tester assessment in predicting the likelihood of a software module having bugs is recommended for future research since the variable was introduced during this study and further tests will validate its effectiveness in the required task.

References

1. DeMarco, T., Lister, T., Rosenberg, L., Gallo, A., Hammer, T., Parolek, F., ... & Barbour, R. Software Technology Engineering. *crosstalk*, 801, 775-5555.
2. Mahoney, M. S. (2004). Finding a history for software engineering. *IEEE Annals of the History of Computing*, 26(1), 8-19.
3. Mece, E. K., Paci, H., & Binjaku, K. (2020). The application of machine learning in test case prioritization-a review. *European Journal of Electrical Engineering and Computer Science*, 4(1).
4. Konsaard, P., & Ramingwong, L. (2015, June). Total coverage-based regression test case prioritization using genetic algorithm. In *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON)* (pp. 1-6). IEEE.
5. Noghabi, H. B., Ismail, A. S., Ahmed, A. A., & Khodaei, M. (2012). Optimized query forwarding for resource discovery in unstructured peer-to-peer grids. *Cybernetics and Systems*, 43(8), 687-703.
6. Mirarab, S., & Tahvildari, L. (2007, March). A prioritization approach for software test cases based on bayesian networks. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 276-290). Springer, Berlin, Heidelberg.
7. Felding, E. (2022). Mathematical Optimization for the Test Case Prioritization Problem.
8. Busjaeger, B., & Xie, T. (2016, November). Learning for test prioritization: an industrial case study. In *Proceedings of the 2016 24th ACM SIGSOFT International symposium on foundations of software engineering* (pp. 975-980).
9. Li, Z., Huang, M., Liu, G., & Jiang, C. (2021). A hybrid method with dynamic weighted entropy for handling the problem of class imbalance with overlap in credit card fraud detection. *Expert Systems with Applications*, 175, 114750.

10. Gökçe, N., Eminov, M., & Belli, F. (2006, November). Coverage-based, prioritized testing using neural network clustering. In *International Symposium on Computer and Information Sciences* (pp. 1060-1071). Springer, Berlin, Heidelberg.
11. Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017, July). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 12-22).
12. Lachmann, R., Schulze, S., Nieke, M., Seidl, C., & Schaefer, I. (2016, December). System-level test case prioritization using machine learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 361-368). IEEE.
13. Lachmann, R. (2018, June). Machine learning-driven test case prioritization approaches for black-box software testing. In *The European test and telemetry conference, Nuremberg, Germany*.
14. Shirzadi, A., Solaimani, K., Roshan, M. H., Kavian, A., Chapi, K., Shahabi, H., ... & Bui, D. T. (2019). Uncertainties of prediction accuracy in shallow landslide modeling: Sample size and raster resolution. *Catena*, 178, 172-188.
15. Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., & Murphy, B. (2010, November). Change bursts as defect predictors. In *2010 IEEE 21st international symposium on software reliability engineering* (pp. 309-318). IEEE.
16. Zimmermann, T., Nagappan, N., & Zeller, A. (2008). Predicting bugs from history. In *Software evolution* (pp. 69-88). Springer, Berlin, Heidelberg.
17. Yu, S., & Zhou, S. (2010, April). A survey on metric of software complexity. In *2010 2nd IEEE International conference on information management and engineering* (pp. 352-356). IEEE.
18. Kini, S. O., & Tosun, A. (2018, September). Periodic developer metrics in software defect prediction. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 72-81). IEEE.

19. Malishevsky, A. G., Rothermel, G., & Elbaum, S. (2002, October). Modeling the cost-benefits tradeoffs for regression testing techniques. In *International Conference on Software Maintenance, 2002. Proceedings.* (pp. 204-213). IEEE.

20. Kazmi, R., Jawawi, D. N., Mohamad, R., & Ghani, I. (2017). Effective regression test case selection: A systematic literature review. *ACM Computing Surveys (CSUR)*, 50(2), 1-32.