# *“Software Defect Prediction with Newer and Efficient Techniques”*

A PROJECT REPORT

SUBMITTED IN THE PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

Submitted By

**Aman Sharma**

**(2K19/SWE/22)**

Under the supervision of

**Dr. Ruchika Malhotra**
Head of Department
Department of Software Engineering
Delhi Technological University, Delhi

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

JULY, 2021

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

## CANDIDATE'S DECLARATION

I, Aman Sharma, 2K19/SWE/22 student of M.Tech (SWE), hereby declare that the project entitled "Software Defect Prediction with Newer and Efficient Techniques" which is submitted by me to the Department of Software Engineering, Delhi Technological University, Shahbad Daulatpur, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology in Software Engineering, has not been previously formed the basis for any fulfilment of the requirement in any degree or other similar title or recognition.

This report is an authentic record of my work carried out during my degree under the guidance of Dr. Ruchika Malhotra.

Aman

Place: Delhi **Aman Sharma**

Date: 29th July 2021 **(2K19/SWE/22)**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

## **CERTIFICATE**

I hereby certify that the project entitled "Software Defect Prediction with Newer and Efficient Techniques" which is submitted by Aman Sharma, 2K19/SWE/22 to the Department of Computer Science & Engineering, Delhi Technological University, Shahbad Daulatpur, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology in Software Engineering, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Place: Delhi

Date:

Dr. Ruchika Malhotra
(SUPERVISOR)
(Head Of Department)
(Associate Dean IRD, DTU)

## ACKNOWLEDGEMENT

I am very thankful to **Dr. Ruchika Malhotra** and all the faculty members of the Department of Computer Science and Department of Software Engineering at DTU. All of them provided us with immense support and guidance for the project.

I would also like to express my gratitude to the University for providing us with the laboratories, infrastructure, testing facilities, and environment which allowed us to work without any obstructions.

I would also like to appreciate the support provided to us by our lab assistants, seniors, and our peer group who aided us with all the knowledge they had regarding various topics.

**Aman Sharma**

**(2K19/SWE/22)**

# Abstract

For a long time, software defects have been a big problem in the software optimization and development processes. Various processes have been hindered in the past due to a small defect, which if predicted or corrected at that moment could have resulted in exponential efficiency and enormous commercial benefits. With that in mind it is a viable option of the current time to come up with a process, service or algorithm that can help in software defect prediction.

In a SDLC, it is a crucial part in the testing phase to identify the areas that are prone to problem and may ultimately lead to defects which may cause a very massive problem in the later stages of development. These problems may lead to a very important component of the algorithm to malfunction and at that stage can also make it costly to repair in terms of both finance and efforts.

Time and cost have both been important development considerations when taking up and developing a project and if it is prolonged after the expected deadlines can lead to massive drop off in efficiency and vulnerability on the execution side of things. It is always better to predict the areas of the problems that may creep in and use algorithms that may help do such process in an easier and effective way. Moreover, we can also use these algorithms with greater and much accurate results when the training has been done for long periods of time as it can make the procedure for training much more robust and precise.

This study helps in the analysis of the previously applied activities in the field of defect prediction on a widely known dataset and then focusses to optimize on these techniques using complementary methods that have been widely used in other disciplines. Newer techniques show promising results.

# Contents

# List of Figures

## **List of Tables**

# <u>List of symbols and abbreviations</u>

| **Abbreviations** | **Full Form** |
|---|---|
| SDLC | Software Development Life Cycle |
| LOC | Lines of Code |
| NB | Naïve Bayes |
| BNB | Bernoulli Naïve Bayes |
| GNB | Gaussian Naïve Bayes |
| MNB | Multinomial Naïve Bayes |
| LGBM | Light Gradient Boost Model |
| MLP | Multiplayer Perceptron |
| FNN | Feedforward Neural Network |
| RNN | Recurrent Neural Network |
| ANN | Artificial Neural Networks |
| DNN | Deep Neural Networks |
| HT | Hyperparameter Tuning |

# CHAPTER 1

# INTRODUCTION

## 1.1 General

In the industries today, it is important to have a method to gather the information about the product or software developed so that all its vulnerabilities and improvements, to some extent, are known beforehand. To achieve this goal various software development companies and government organisations in such fields, directly or indirectly, research on the field of software defect prediction to improve efficiency and reduce the cost of development in the long run.

This process helps them to acquire an understanding of the long-term cost of development of a software and provide a keen view of the entire project along with its vulnerabilities beforehand. Thus, industries that have a development component attached to them also have these studies.

Majority of the end of previous century and the start of the current century, if taken into consideration have increased the size of code manifold. The LOC have increased drastically over the years to the point where now the code can be considered to the point of unreadable or in case of an error, undetectable. To circumvent these problems there needs to be and algorithm, preferably something related to machine learning that can traverse through all these LOC and predict with maximum possible accuracy the defects that may be present in the provided dataset.

As shown in the given depiction(fig 1), there has been sharp rise in the lines of code even in the past few years to an enormous extent. The average codebase of the given project which was named PerfectTablePlan, shows that the LOC has been showing and upward trend over the years and if such process continues, it may be the case that the readability and correctness of the code may become an issue which, as we know, are the basic pillars of an average codebase development and must be at the core of every finished or launched product.

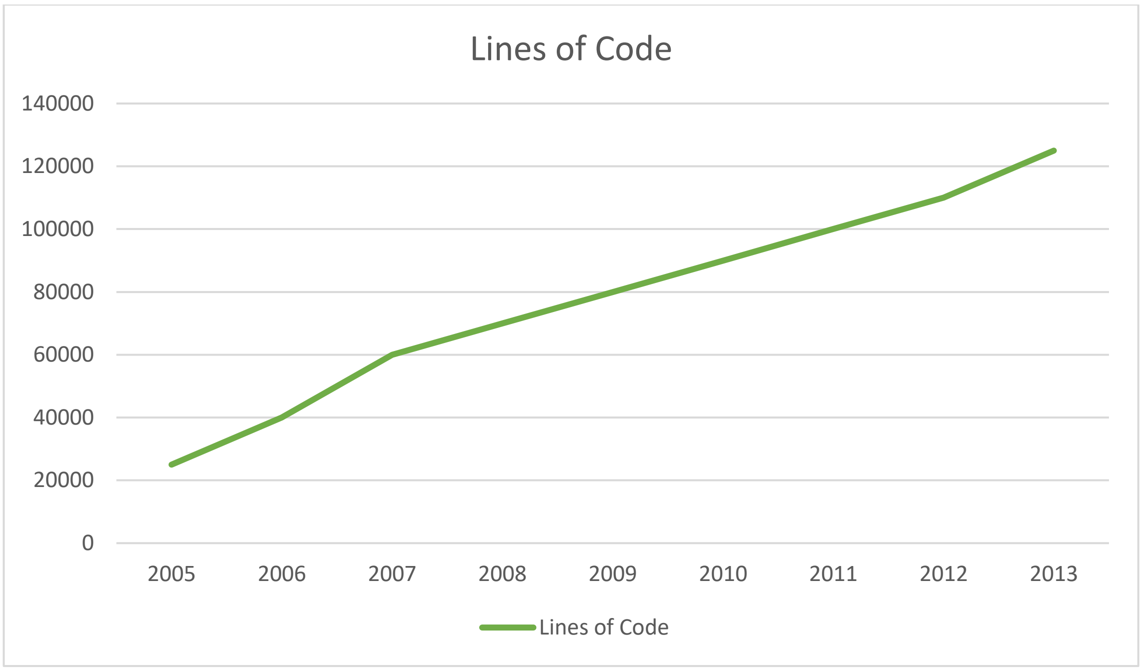

*Fig. 1: LOC vs date Comparison for PerfectTablePlan*

### 1.2 Problem Formulation

As much research have been made in the field of machine learning, deep learning, and software defect prediction along with prediction algorithms in other domains, there is a need to combine all these changes and view them through a singular perspective and come up with an approach that makes all these advancements valuable w.r.t the field of software defect prediction.

The problem that formed the core of this study are as follows:

1. What are the current algorithms used in software defect prediction?
2. What are the basic empirical metrics used in the previous algorithms?
3. Are there any processes that can make the existing algorithms better based on the same metrics?
4. Is over training and algorithm leading to better results?
5. Are there any more empirical studies that can be made on the current data to improve its understandability?
6. Is there a way to provide adjacent study results to the current algorithm so that it makes them more efficient?

## 1.3 Objectives of the Project

In this project, the following objectives need to be achieved:

1. Understanding adjacent studies to find the more efficient way to get the best possible results.
2. Using the pre-defined models in ready to run software.
3. Developing an algorithm based on the approaches taken into consideration.
4. Using same metrics as in previous studies to evaluate the developed algorithm so that the comparison can be easier.
5. Addressing the issues and providing solutions that may be present or creep into the datasets.
6. Trying to train the developed algorithm in different values and observe the results.

# CHAPTER 2

# LITERATURE REVIEW

In this section, we will look at the various understandings of algorithms that were considered during the development of the current model and the various formulae that have been used in those algorithms as well. Later, we will try to conclude the basic points upon which the new model can be developed and will also help us to develop the methodology for this algorithm.

We have the popularly used machine learning techniques for software defects prediction:

## 2.1. Xgboost

1. Xgboost a gradient based decision tree boosting approach developed in 1995 by Tianqi Chen. As stated in [20], xgboost can be used to enhance the abilities of the machine so that more resources can be available for computing, tuning, and enhancing.
2. The three techniques incurred in boosting using xgboost are: stochastic, regularized, and gradient boosting. This method can be applicable on trained model's added data and is Sparse Aware i.e., takes care of any missing value. This means if there is any value in the dataset that is missing then this algorithm can be applied and an aggregate value from the dataset can be supplemented in the datasets and process can take place without any hindrance.

## 2.2. Catboost

1. Catboost is again a decision tree-based data gradient boosting model. It seamlessly integrates with varying datapoints and datasets and gives the result that can be used further in the increment of performance and outcome of any process. Due to these properties, Catboost is a widely used algorithm for machine learning as well as deep learning process.
2. This process/algorithm was developed by Yandex in 2017 and is opensource so can be applied to any dataset across the board to get the desired results. It is

fundamentally an algorithm to reduce the complexity of LGBM that would provide similar answers in lesser number of iterations.

## 2.3. NB

This is a probability-based approach based on the bayes theorem. It contains aggregation of simple probability classifiers that can be used in the highly scalable learning problems. These problems can include feature and predictor-based requirements which proves feasible for our study. Prominent approaches to develop Naïve Bayes algorithm classifiers are:

### 2.3.1 BNB

This is the algorithm present in the scikit machine learning modules and can therefore be applied to any available dataset. It is based on the naïve assumption of conditional independence between the given value and the variable quantity. It is responsible to apply algorithm to multivariate data according to Bernoulli distributions. Decision rule for BNB:

$$P\,(a_i \mid b) = P\,(i \mid b)\, a_i + (1 - P\,(i \mid b))\,(1 - a_i)$$

### 2.3.2 GNB

Much like other prominent algorithms, GNB is also present in the scikit machine learning modules and can be directly imported and applied to any dataset to get the required metrics. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi s_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2 s_y^2}\right)$$

Here, $s_y$ and $\mu_y$ are estimated using maximum likelihood.

### 2.3.3 MNB

This algorithm is responsible for applying on multinomially distributed data. It can also be applied from scikit machine learning modules much like its counterparts.

The parameters $\theta_y$ is projected by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T,

And $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

## 2.4. LGBM

1. LGBM is another boosting based algorithm based on decision tree gradient boosting. This algorithm assumes that when K-Means algorithm uses the one-pass over input data, it optimizes K-means to increase the production of centroids.
2. It is an ensemble technique that can be used to combine smaller and weaker models into a stronger and more robust model. Due to these features this feature will also be useful for this study.

## 2.5. MLP

This is special type of feedforward neural network that can give flexible results depending upon the requirements of the dataset and process methodology. Multilayer Perceptron can be easily applied on the current dataset using WEKA and applying a few extensions within it. MLP can then be used to modify itself to develop and acquire the structure of the neural network and the metrics can be recorded accordingly. Multiple layers can be interconnected like in a directed graph and each layer after the input node layer can have its own independent functions.

## 2.6. Neural Networks

This are the networks that are used in the digital algorithms to mimic the effects of biological neural networks present in organic lifeforms.

### 2.6.1 FNN

This network is responsible to send the signal in a single direction. The signal can never be sent back wiz. Feedback mechanism is not possible in this algorithm.

### 2.6.2 RNN

These networks are an extension to FNN. They can use their internal memory to process variable inputs.

### 2.6.3 ANN

These networks are more in line to the fundamental neural networks and generally inspired by real life neural networks.

### 2.6.4 DNN

DNN are very distinct neural networks that have multiple intermittent layers between the input and output layers. Although having same components as any neural network it resembles more of a multilayer perceptron in its implementation.

The following concluding points can be gathered from the information gathered above:

1. Initial studies did not use the implementation of some of the newer algorithms that have been developed over the last half decade.
2. A combination of these algorithms may lead to a better result.
3. Another approach to improve the accuracy of the algorithms mentioned above can be to apply a data filtration mechanism based on a tree approach.
4. This approach can make a tree of the results that are made at different levels of depth and the result obtained from such depths can be recorded.

5. This result can then be recorded and the result which was the best in terms of time and closest to the best possible value can be considered as the desired outcome.
6. For developing such a mechanism, a concept of hyper parameter tuning can be given the spotlight as it can help us to change the depth of the tree at a dynamic rate and produce the results that can be recorded.
7. Some of the algorithms discussed above may be too taxing in term of cost and processing and if better results can be obtained with other algorithms, then such costly algorithms may be discarded from results.
8. As more training is given to the model, it is to be observed whether the change in the volume of training set by a few points, provide us with a significantly better result or not.
9. If we want to balance out the data of the datasets (as different datasets have different volumes), then newer approach such as weighted average can also be taken instead of just macro average that was taken in the studies that have taken place before.
10. The datasets that are taken into consideration in other studies can then also be taken out from the current project and a comparative analysis can be assembled.

# CHAPTER 3

# METHODOLOGY

This section process that was used in the implementation of this project along with the technique outline that were finally considered to be implemented. With the development of various software development life cycles (SDLC) we can surely follow many approaches as far as development of the project is concerned. A broad understanding of the methodology followed in this study is given in Fig. 2.

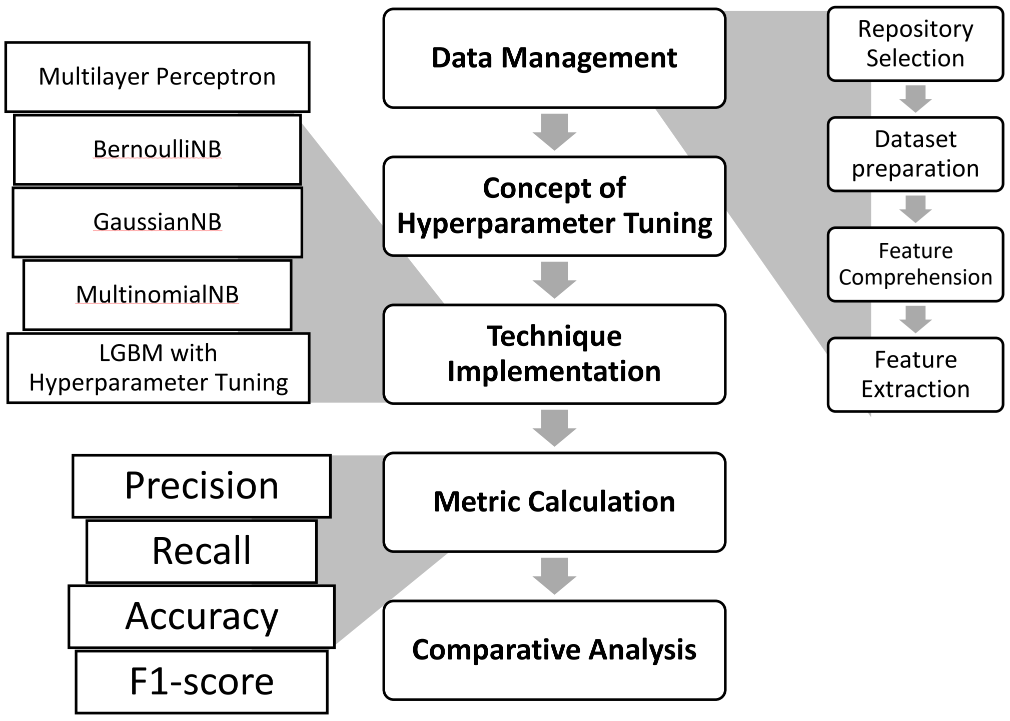

*Fig. 2: Methodology followed in this study*

## 3.1 Data Management

This subsection deals with the data management steps:

### 3.1.1 Repository Selection

There are two widely used repositories present in the open-source category in relation to software defect prediction namely: NASA repository and PROMISE repository. Due to more implementations and studies conducted in the NASA software engineering repository, it can be considered as the standard and can therefore be listed for this study. It is by no means to discredit the other repositories and are also encouraged to use in the further research process for a comparative analysis.

### 3.1.2 Dataset Preparation

Unwanted data such as noise and ambiguous data can be removed from the data. The given dataset is very well organized and is therefore does not have any empty or missing entries. The dataset does not have any noisy data in and is organized to a recognizable quality. 12 datasets out of the 13 datasets are selected from the repository. We also need to ensure that the variable names in the provided dataset does not have any keywords that can hinder in the execution of the algorithms.

| Dataset Name | Number of Features | Empty Features |
|---|---|---|
| **CM1** | 22 | none |
| **JM1** | 22 | none |
| **KC1** | 22 | none |
| **KC2** | 22 | none |
| **KC3** | 40 | none |
| **MC1** | 39 | none |
| **MC2** | 40 | none |
| **MW1** | 38 | none |
| **PC1** | 22 | none |
| **PC2** | 37 | none |
| **PC3** | 38 | none |
| **PC4** | 38 | none |

*Table 1: Analysis of datasets present in the NASA Dataset repository*

### 3.1.3 Feature comprehension

To create a uniform environment to unify all the dataset to process through a multitude of algorithms together. The minimum number of features in the given dataset is 22 as shown in Table 1.

The datasets have numerous features and from them we get 22 which can be used to process every dataset. Alternate names for same features are also observed across datasets.

To make the features viable, we need to see that all the features that are present in the minimum dataset are present in one form (variable name) or another in all the datasets. To assure this, we needed to find some common attributes between all the datasets with different name irrespective of the order and then we can access the different kind of feature forms that are present in the dataset.

As more and more features are added across the dataset, we must comprehend only the core 22 as shown in table 2.

### 3.1.4 Feature Extraction

Out of the provided 22 features which are common for all the datasets, only a few features from those are required for the actual implementation that will be expanded upon in the following sections are only required for the calculations. Next, we convert each of the feature to float explicitly to allow an easier calculation to get the relevant empirical results.

| Feature Name | Alternate Name | Description (numerical %) |
|---|---|---|
| **loc** | LOC_TOTAL | McCabe's line count of code |
| **v(g)** | CYCLOMATIC_COMPLEXITY | McCabe "cyclomatic complexity" |
| **ev(g)** | ESSENTIAL_COMPLEXITY | McCabe "essential complexity" |
| **iv(g)** | DESIGN_COMPLEXITY | McCabe "design complexity" |
| **n** | HALSTEAD_LENGTH | Halstead total operators + operands |
| **v** | HALSTEAD_VOLUME | Halstead "volume" |
| **l** | HALSTEAD_LEVEL | Halstead "program length" |

| | | |
|---|---|---|
| **d** | HALSTEAD_DIFFICULTY | Halstead "difficulty" |
| **i** | HALSTEAD_CONTENT | Halstead "intelligence" |
| **e** | HALSTEAD_EFFORT | Halstead "effort" |
| **b** | HALSTEAD_ERROR_EST | Halstead "Number of bugs" |
| **t** | HALSTEAD_PROG_TIME | Halstead "time estimator" |
| **lOCode** | LOC_EXECUTABLE | Halstead's line count |
| **lOComment** | LOC_COMMENTS | Halstead's count of lines of comments |
| **lOBlank** | LOC_BLANK | Halstead's count of blank lines |
| **lOCodeAndComment** | LOC_CODE_AND_COMMENT | Line count for code and comment |
| **uniq_Op** | NUM_UNIQUE_OPERATORS | unique operators |
| **uniq_Opnd** | NUM_UNIQUE_OPERANDS | unique operands |
| **total_Op** | NUM_OPERANDS | total operators |
| **total_Opnd** | NUM_OPERANDS | total operands |
| **branchCount** | BRANCH_COUNT | flow graph's branch count |
| **defects** | Defective, label | Reported defects |

*Table 2: features of datasets [20]*

## 3.2 The concept of hyperparameter tuning

As more and more models are implemented using a singular software like WEKA, the result of every algorithm of each dataset tends to be uniform. The process of changing the variables present within these algorithms is known hyperparameter tuning.

Most of the studies that we came across had mostly similar outcome and those outcomes were a direct result of implementing the algorithm directly through a given software. Using the machine learning libraries of python such as scikit, numpy, pandas etc., one can easily change the perimeter within these set algorithms and give a much more optimized result as shown in Fig. 2.

A cross product of all the hyperparameter can be constructed by the introduction on grid search algorithm that was present in scikit package but mostly used in the projects pertaining to computer vision for the most part. With the introduction of such alternate methods, we can define the all the varying values that we require into any algorithm and expect the best possible result out of all those values.

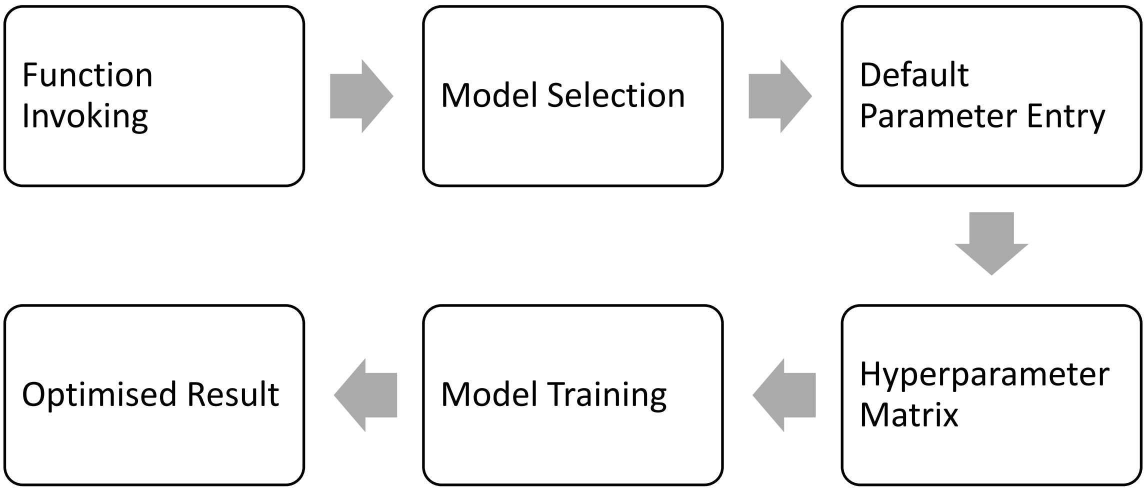

*Fig. 3: Basics of Hyperparameter tuning.*

For the most part the usage of the computer vision algorithm is limited to parameter modification and thus the algorithm retains its originality.

## 3.3 Technique Implementation

The following techniques will be implemented to the dataset sequentially and the result will be recorded. The order of implementation of each model is as follows:

A. Multilayer Perceptron
B. BernoulliNB
C. GaussianNB
D. MultinomialNB
E. LGBM after Hyperparameter tuning

## 3.4 Metrics Calculation

Based on equations

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Negative} + \text{True Negative} + \text{False Positive}}$$

$$\text{F1-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

We can compare the different results from different algorithms when they are implemented on a dataset and those result can then be compared with other research results so that the most viable option can be chosen for the implementation of any algorithm in the future.

## 3.5 Empirical Findings

All the empirical result of the given metrics to the respective dataset will be recorded in a tabular form for easy foresight and quick review. There will be two splits present in the testing of the experimentation to observe the behaviour of the outcomes when datasets are split differently. First set of observations will gather all the results of the datasets with 75% and 25% split between the training and testing of the algorithms, respectively. Second set of outcomes will split the datasets in 80% and 20 between training and testing, respectively.

These dual split results will provide flexibility in comparison to other studies and internal observation in different splits of the datasets.

## 3.6 Comparative analysis

After all the empirical results are obtained, we can compare these results to the latest studies and see which algorithm is efficient for which dataset and publish the results accordingly.

# CHAPTER 4

# EXPRIMENTAL SETUP

Setups help in the faster output of the program as multiple outputs need to be calculated at the same time along with simultaneous processing that needs to take place to gather information from each input and handle it to form a comprehensive data and are presented in this research as a means of full disclosure.

| Name | Description |
|---|---|
| **Central Processing Unit** | AMD 3900x 12 core processor |
| **Random Access Memory** | 32gb 2600 MHz |
| **Graphics Processing Unit** | Nvidia RTX2070super (not detrimental to this research) |
| **Software** | Weka for Multilayer Perceptron, Jupyter Notebooks for other machine learning algorithms including hyperparameter tuning |
| **Language** | Python |
| **Libraries used** | Pandas, numpy, scikit, scipy.io for arff, os, matplotlib.pyplot, seaborn, warnings, tqdm, plotly.offline for iplot, plotly.graph_objs |

*Table 3: Experimental Setup used in this study*

# CHAPTER 5

# EMPIRICAL RESULTS

The core focus is to make a new and improved algorithm that can help in the long run for further research. As there can be problems with a single dimensional checking of the developed model. With that in mind table 4 presents the information when the datasets are processed through a model with 80% and 20% split between training and testing sets, respectively. This is done to check if the testing results improve at a higher training partition.

The second empirical study as shown in table 5 changes this partition to 75% and 25% in training and testing, respectively. There are two benefits to do a dual research random seed-based module testing:

1. A direct comparison within the research can be drawn a legible assumption can be made about the potency of the developed model.

2. As most of the studies are done on a 75% training split, we can have a straightforward comparison of the developed model to the preciously conducted studies and models.

3. Analysis for the outcome for both splits can be regarded in the upcoming section.

4. It is to be noted that all the models are kept constant throughout the outcome collection to affirm uniformity across all datasets.

As a common knowledge we have the luxury of getting the outcomes and verifying them from the confusion matrix provided with the program output. We can compare the results we are getting with the confusion matrices and can then calculate the outcome any new metric that we desire from these values. Therefore, a confusion matrix plays a crucial role in determining the any new metrics and verifying the outcomes that have been calculated in the system automatically as represented in table 4.

| **Class** | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | True Negative (TN) | False Positive (FP) |
| **Actual Positive** | False Negative (FN) | True Positive (TP) |

*Table 4: Confusion Matrix*

**Weighted Results**

The study has been conducted keeping in mind the various empirical evidence that have to be up to the standard to compare to any available paper and to do that, we need to have the results in the form that is incorporating all the intricacies of the given problem with respect to its datasets. The problem that can occur with the initial observation of the results is that the datasets can be taken as an equal quantity and the metrics that are calculated in the empirical evidence can be taken in the absolute value which may provide a skewed result of the provided outcome.

We observe that certain datasets are not even a tenth fraction of some other datasets, and this relation can be followed to all the datasets present in this study. Due to this imbalance, it might not be advisable to calculate the datasets as is and give them a weighted value according to the size of each of them.

This notion will certainly play on the fact that certain datasets that will have a large volume will be given a higher metric value in comparison to another dataset that is miniscule. It is with this idea in mind that one such quantity wiz. accuracy has been taken into consideration as all the major studies referenced in this paper are compared on that metric. To do that, we need to construct a weighted representation of each dataset.

Moreover, as for comparison in difference between the macro and weighted average of the datasets, a subsequent representation is also constructed and can therefore be used if come other study does not wish to follow the concept of weighted results proposed in this study.

We encourage more and more studies to include such metrics in their research and to do that there need to be rigorous metrics focused analysis of the outcome that have come out of the approach.

| Dataset | Number of Entries |
|---|---|
| **CM1** | 498 |
| **JM1** | 7782 |
| **KC1** | 2109 |
| **KC2** | 522 |
| **KC3** | 194 |
| **MC1** | 1988 |
| **MC2** | 125 |
| **MW1** | 253 |
| **PC1** | 1109 |
| **PC2** | 745 |
| **PC3** | 1077 |
| **PC4** | 1458 |

*Table 5: Volume of each dataset*

From the tables below all the information provided in the tables above we can gather that the percentage split of 75% is better or almost equal to the accuracy metrics across all the algorithms' implementation.
We can also gather all the information that is gathered for each dataset in every algorithm at 80% split as well as 75% split in Table 5 and Table 6, respectively.

| | Macro Accuracy average | Weighted Accuracy average |
|---|---|---|
| **Multilayer Perceptron** | 85.33 | 84.57 |
| **BernoulliNB (HT)** | 77.83 | 79.07 |
| **GaussianNB (HT)** | 95.59 | 96.29 |
| **MultinomialNB (HT)** | 85.24 | 92.25 |
| **LGBM(HT)** | 99.07 | 99.32 |

*Table 6: Average accuracy metric comparison for 80% split*

| | **Macro Accuracy average** | **Weighted Accuracy average** |
|---|---|---|
| **Multilayer Perceptron** | 85.31 | 84.85 |
| **BernoulliNB (HT)** | 78.03 | 79.21 |
| **GaussianNB (HT)** | 95.38 | 96.08 |
| **MultinomialNB (HT)** | 91.08 | 92.26 |
| **LGBM (HT)** | 99.43 | 99.44 |

*Table 7: Average accuracy metric comparison for 75% split*

# CHAPTER 6

# COMPARATIVE ANALYSIS

As similar studies we made using the same or similar datasets as used in this study, there needs to be an analysis with those studies as well so that the results can be compared, and an appropriate conclusion can be reached.

Datasets compared in this study for the various techniques used are PC1, CM1, JM1, KC1 and KC2. If we want to apply a weighted or macro accuracy comparison upon the datasets and compare them straightway with the prominent studies, then we need to have an accuracy metric aggregation of these datasets and then provide them for comparison. The subsequent table 10 presents the 75% split accuracy metric for both weighted and macro aggregation. This table can then be used for comparison to other approaches as presented in table 11. As all the other studies in this comparison only use a 75% split for training and 25% split for testing, therefore, we only calculate the aggregate average for this split.

| | **Macro Accuracy average** | **Weighted Accuracy average** |
|---|---|---|
| **Multilayer Perceptron** | 85.93 | 81.9 |
| **BernoulliNB (HT)** | 77.48 | 80.69 |
| **GaussianNB (HT)** | 96.48 | 96.48 |
| **MultinomialNB (HT)** | 93.4 | 92.64 |
| **LGBM(HT)** | 99.62 | 99.39 |

*Table 8: Calculation of metrics in the datasets PC1, CM1, JM1, KC1, and KC2*

Now we can efficiently and correctly compare the observations in this study to the other prominent studies with higher accuracy metric and see how much the new approaches hold good in their respect.

| **Authors** | **Methods** | **Accuracy (%)** | |
|---|---|---|---|
| **G. Abaei and A. Selamat (Abaei, & Selamat, 2014)** | Decision Tree | 79.5 | |
| | Random forest | 81.14 | |
| | Naïve Bayes | 80.42 | |
| | Immunos2 | 80.65 | |
| | CSCA | 80.17 | |
| **A. Chug and A. Dhall (Chug, & Dhall, 2013)** | Decision Tree | 99.36 | |
| | Naïve Bayes | 94.11 | |
| | Random Forest | 99.55 | |
| **M.M. R. Henein et al. (Henein, Shawky, & Abd-El-Hafiz, 2018)** | K-means + ANN | 80.4 | |
| **A.S. Abdou and N.R. Darwish (Abdou, & Darwish, 2018)** | Naïve Bayes | 82.44 | |
| | SVM | 91.74 | |
| | Random forest | 93.4 | |
| | Logistic | 87.4 | |
| | Decision tree | 89.84 | |
| **Safial Islam Ayon (Ayon, 2019)** | FNN | 97.93 | |
| | RNN | 98.39 | |
| | ANN | 98.12 | |
| | DNN | 98.47 | |
| **Proposed Approach (Macro and Weighted respectively)** | | **MA** | **WA** |
| | Multilayer Perceptron | 85.93 | 81.9 |
| | BernoulliNB (HT) | 77.48 | 80.69 |
| | GaussianNB (HT) | 96.48 | 96.48 |
| | MultinomialNB (HT) | 93.4 | 92.64 |
| | LGBM (HT) | 99.62 | 99.39 |

*Table 9: Comparison between other studies and the proposed approach applied in the datasets in this study. (MA- Macro Accuracy, WA- Weighted Accuracy)*

While doing the comparison, it is to be kept in mind that the results that are being compared to need to use the same datasets and which metric needs to be compared with. For uniformity and progression in a linear fashion, this study has focused on comparison for accuracy as the metric of comparison between the different studies. We encourage the future studies to enhance the comparison even further to other metrics to have an extensive analysis of the algorithm on every metric.

# **CHAPTER 7**

# **ANALYSIS**

From all the observations presented in the above table 11 we can see that the newer approaches with hyperparameter tuning are much better in many respects as compared to the techniques that were used in the previous studies. For comparison, in this study itself, we have included a standard, non-hyperparameter tuned version of a model, multiplayer perceptron, which in comparison is better than BernoulliNB model but falls short in comparison to other models. In comparison to other studies, the studies that were done (Chug, & Dhall, 2013) give promising results as well with the accuracy of Decision tree and Random forest techniques but when we get the Light gradient boosting machine in the hyperparameter tuned condition, which in turn is an extension of tree-based training models, we get better accuracy and can therefore suggest these models for a higher accuracy metrics requirement.

Following the studies based on neural networks done in (Ayon, 2019), we tried a similar approach in multilayer perceptron using WEKA libraries and tried to pass the datasets with that model. Although the results hold better when compared (Henein, Shawky, & Abd-El-Hafiz, 2018), but they could not be as good as the results achieved in that study (Ayon, 2019).

As more and more approaches pointed towards a tree-based learning approach as suggested in studies (Abaei, & Selamat, 2014; Chug, & Dhall, 2013; Abdou, & Darwish, 2018), therefore a more tree-based model was the path that was thought to be better. With that approach in consideration, we moved forward with the LGBM and tried to get better and accurate results so that more efficient models are gathered and tested. The part of hyperparameter tuning can massively compliment the calculation of the results by manifold and of the number of iterations is not the issue and there are no processing cycle constraints, it is advisable to use this approach in prediction of datasets in the further study as well.

The other approach in the study uses a specific formula that is used to improve the model as stated in previous sections and within that there is a chance for improvement and the most promising results were shown by GaussianNB among the other naïve bayes approaches. If the condition of using a specific formula-based approach is prevalent or in consideration in any of the study, then these naïve bayes approaches are suggested to them. As seen in the studies (Abaei, & Selamat, 2014; Chug, & Dhall, 2013; Abdou, & Darwish, 2018), the standard naïve bayes approaches were giving results up to 94.11% (Chug, & Dhall, 2013) but this study provides an even better result when the parameters within the standard process are set for utmost accuracy. We must also suggest that all the approach are well within practical range of use but depend more on the type of model that needs to be taken into consideration for any study. The usage or application, in that sense, is the ultimate decider of what technique to apply in what conditions.

More research is certainly encouraged in this study as there can be other caveats that may have been restricted in this study that could have been considered, but due to some limitation, were not contemplated.

Some of the latest studies have also been taken for comparison in this study to have the paper updated and all the nuances are approached with the utmost sincerity but if there are some openings that can be explored further, they can be certainly reflected upon in a subsequent research.

# CHAPTER 8

# OBSERVATIONS

In our study, we observe a split of 80% in training and 20% in testing in table 6. In this division we get the results and compare them to the subsequent results of table 7 which contains training data of 75% with 25% of testing data.

As comparison needs to be accomplished, we also need to compare the results considering the disbalance in the volume of each dataset. To get that we find the size of each dataset as shown in each table 7. When the weight of each dataset is present, then we can compare the outcomes of macro vs the weighted accuracy aggregation of all the datasets in this study. In see that a phenomenon of "over-training may creep in when the models are trained at a higher ratio and the model with 75% training split were similar if not better in some respects that the models in the 80% split. As the models are giving promising results at a 75% split as well, we can say that this split will be more feasible and maybe is the reason that most of the popular studies use this split when doing the training of the models.

Moving forward with the external study comparison, we observe that the studies done and research in this experiment are using 5 major datasets to check the accuracy. Thus, to do a viable comparison, we get the accuracy aggregation metric of those datasets shown in table 10.

Next, we see that although there is not a massive difference in the macro average of the accuracy of a model and the weighted accuracy at higher values but there is certainly a huge difference when accuracy is low in comparison i.e., less than 90%. So, we encourage any future studies to have a separate representation for the concept of weighted aggregation as it gives a clear picture of the metrics that are provided in any model. As many existing and future model may or may not have any one of the aggregations, so we provide both in table 11.

Table 11 also provides a straight contrast between other models that were used in previous studies with means to achieve accuracy. We realize that the naïve bayes models are providing a great machine learning model but fall short in comparison to the tree-based approaches in previous studies as well as this study, which makes this study a linear progression to its predecessors. Also, we see that when the cartesian product of multiple parameters is applied and tested on a particular/existing model i.e., a hyperparameter

version of any model is created, the metrics tend to become high, and this can provide an even better results than the previously applied models and formula-based machine learning approaches.
It seems like with this model we are at the precipice of flawless prediction which may be a misdirection as discussed in the further sections and is the reason why we need more studies in the future. The major takeaway from this research and the motive is to get more people associated and interested in this topic and create a platform for defect prediction discussions and improvements.

# CHAPTER 9

# CONCLUSION

This methodology of hyperparameter tuning has been extensively used in other disciplines to gather the best possible results and therefore gave us the motivation to apply in the highlighted field as well. For comparison we added many relevant studies and construct a large image of the empirical results for any further studies.

In this study, we also touched on the fact of macro and weighted accuracy and the difference in the values, though not significant in terms of difference, but can lead to a massive improvement in our understanding of the subject at hand. It also catered to the imbalance of the size of each dataset and gave us a more holistic view of the different datasets used in this study.

Hyperparameter tuning gave a better result when compared to other prevalent methods and the empirical results gained in most other studies. We know that the methods that may have been applied in this study may have been used in different studies as well but the concept of hyperparameter tuning combined with these techniques certainly showed very promising results.

We have tried to also reduce the threats to this study that may creep in but there may be some of the unavoidable factors that may have hindered the optimal results for this study that we encourage all the future studies to rectify, if possible. There is also the concept of multiple average taken based on distinction in the volume of the datasets that has resulted in another dimension which can be investigated in future studies as an area of exploration and research

Lastly, after all the experimentation is done, we also need to mention the need to do more study on this subject to create more awareness and have much better results in any metric, thus providing us useful information. The purpose of this study which was to try upon the existing techniques and improve on the results that were observed in previous studies has been achieved and the new results can thus be used in the studies that are yet to be done or under development.

# CHAPTER 10

# PROSPECTS

As stated, multiple times in this study, we need to have more interest and discussion in this field of defect prediction as it helps the software industries to save massive resources in terms of both labor and finances. Apart from the industrial impact, in the field of research, prediction has been an ever-growing research area and this study acts as a droplet in that field.

Particularly, as presented in this study, prediction can be improved by working upon the problem areas and improvements that have been discussed in this paper and the novel idea and model development that is yet to come. We not only welcome but encourage young minds as well as veterans to collaborate with us to develop an approach that provides a win-win situation to the industrial as well as research fraternity.

We have done the improvement in the metrics in comparison to the previous studies that we came across but there may be much more studies that may have some compound resultant of development and improvement that may help in developing the study further. There need to be an iterative and a regular improvement to the existing models as presented in this study and we hope that more people help in this endeavour by contributing such and more studies and explore potent prospects

# REFERENCES

[1] Abaei, G., & Selamat, A. (2014). A survey on software fault detection based on different prediction approaches. *Vietnam Journal of Computer Science*, 1(2), 79-95.

[2] Abdou, A. S., & Darwish, N. R. (2018). Early prediction of software defect using ensemble learning: A comparative study. *International Journal of Computer Applications, 179(46),* 29-40.

[3] Agrawal, A., & Menzies, T. (2018, May). Is" Better Data" Better Than" Better Data Miners"?. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (pp. 1050-1061). IEEE.

[4] Aleem, S., Capretz, L. F., & Ahmed, F. (2015). Benchmarking machine learning technologies for software defect detection. arXiv preprint arXiv:1506.07563.

[5] Aquil, M. A. I., & Ishak, W. H. W. (2020). Predicting Software Defects using Machine Learning Techniques. *International Journal, 9(4).*

[6] Ayon, S. I. (2019, May). Neural network based software defect prediction using genetic algorithm and particle swarm optimization. In *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)* (pp. 1-4). IEEE.

[7] Cheng, M., Wu, G., Yuan, M., & Wan, H. (2016). Semi-supervised software defect prediction using task-driven dictionary learning. *Chinese Journal of Electronics, 25(6),* 1089-1096.

[8] Chug, A., & Dhall, S. (2013). Software defect prediction using supervised learning algorithm and unsupervised learning algorithm.

[9] D'Ambros, M., Lanza, M., & Robbes, R. (2010, May). An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (pp. 31-41). IEEE.

[10] Dejaeger, K., Verbraken, T., & Baesens, B. (2012). Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering, 39(2),* 237-257.

[11] Endres, A., & Rombach, H. D. (2003). A handbook of software and systems engineering: Empirical observations, laws, and theories. Pearson Education.

[12] Emam, K. E., & Melo, W. (1999). The Prediction of Defect Classes Using Object-Oriented Design Metrics. Technical report: NRC 43609.

[13] Erturk, E., & Sezer, E. A. (2015). A comparison of some soft computing methods for software fault prediction. Expert systems with applications, 42(4), 1872-1879.

[14] Fagan, M. (2002). Design and code inspections to reduce errors in program development. In *Software pioneers* (pp. 575-607). Springer, Berlin, Heidelberg.

[15] Fu, W., & Menzies, T. (2017, August). Revisiting unsupervised learning for defect prediction. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (pp. 72-83).

[16] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research, 3(Mar),* 1157-1182.

[17] Hall, M. A., & Holmes, G. (2003). Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data engineering, 15(6),* 1437-1447.

[18] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6), 1276-1304.

[19] Hassan, A. E. (2009, May). Predicting faults using the complexity of code changes. In *2009 IEEE 31st international conference on software engineering* (pp. 78-88). IEEE.

[20] He, H., Bai, Y., Garcia, E. A., & Li, S. (2008, June). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (pp. 1322-1328). IEEE.

[21] Henein, M. M., Shawky, D. M., & Abd-El-Hafiz, S. K. (2018). Clustering-based Under-sampling for Software Defect Prediction. In *ICSOFT* (pp. 219-227).

[22] Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T., & Matsumoto, K. I. (2007, September). The effects of over and under sampling on fault-prone module detection. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)* (pp. 196-204). IEEE.

[23] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2012). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering, 39(6),* 757-773.

[24] Kaur, J., & Sandhu, P. S. (2011). A K-Means based approach for prediction of level of severity of faults in software systems. In Proceedings of International Conference on Intelligent Computational Systems (pp. 1897-1901).

[25] Kocaguneli, E., Menzies, T., Bener, A., & Keung, J. W. (2011). Exploiting the essential assumptions of analogy-based effort estimation. *IEEE transactions on software engineering, 38(2),* 425-438.

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

[27] Li, Z., Jing, X. Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. Iet Software, 12(3), 161-175.

[28] Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27, 504-518.

[29] Malhotra, R., & Kamal, S. (2019). An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. Neurocomputing, 343, 120-140.

[30] Malhotra, R., & Khanna, M. (2017). An empirical study for software change prediction using imbalanced data. *Empirical Software Engineering, 22(6),* 2806-2851.

[31] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering, (4),* 308-320.

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

[33] Nagappan, N., Murphy, B., & Basili, V. (2008, May). The influence of organizational structure on software quality. In *2008 ACM/IEEE 30th International Conference on Software Engineering* (pp. 521-530). IEEE.

[34] Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2017). Heterogeneous defect prediction. IEEE Transactions on Software Engineering, 44(9), 874-896.

[35] Nam, J., & Kim, S. (2015, November). Clami: Defect prediction on unlabeled datasets (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 452-463). IEEE.

[36] Prabha, C. L., & Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)* (pp. 728-733). IEEE.

[37] Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., & Riquelme, J. C. (2014, May). Preliminary comparison of techniques for dealing with imbalance in

software defect prediction. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1-10).

[38] Rodríguez, D., Ruiz, R., Riquelme, J. C., & Aguilar–Ruiz, J. S. (2012). Searching for rules to detect defective modules: A subgroup discovery approach. *Information Sciences, 191*, 14-30.

[39] Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2010, November). Predicting faults in high assurance software. In *2010 IEEE 12th international symposium on high assurance systems engineering* (pp. 26-34). IEEE.

[40] Shivaji, S., Whitehead, E. J., Akella, R., & Kim, S. (2012). Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering, 39(4),* 552-569.

[41] Singh, P. D., & Chug, A. (2017, January). Software defect prediction analysis using machine learning algorithms. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence* (pp. 775-781). IEEE.

[42] Tan, X., Peng, X., Pan, S., & Zhao, W. (2011, October). Assessing software quality by program clustering and defect prediction. In 2011 18th working conference on Reverse Engineering (pp. 244-248). IEEE.

[43] Tosun, A., Bener, A., & Kale, R. (2010, July). Ai-based software defect predictors: Applications and benefits in a case study. In *Twenty-Second IAAI Conference*.

[44] Wang, S., Liu, T., & Tan, L. (2016, May). Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 297-308). IEEE.

[45] Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability, 62(2),* 434-443.

[46] Wang, T., Zhang, Z., Jing, X., & Zhang, L. (2016). Multiple kernel ensemble learning for software defect prediction. *Automated Software Engineering*, 23(4), 569-590.

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

[48] Zimmermann, T., & Nagappan, N. (2008, May). Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering* (pp. 531-540).

[49] Zimmermann, T., Premraj, R., & Zeller, A. (2007, May). Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)* (pp. 9-9). IEEE.