ANALYZING EFFECT OF FEATURE SELECTION IN SOFTWARE FAULT DETECTION

Shamse Tasnim Cynthia

ID: 2016–1–60–113

Md. Golam Rasul

ID: 2016–1–60–080

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh

December, 2019

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of name of our supervisor, Professor, Department of Computer Science and engineering, East West University. We also declare that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

.....

(Dr. Shamim H Ripon)

.....

(Shamse Tasnim Cynthia)
ID: 2016-1-60-113

.....

(Md. Golam Rasul)
ID: 2016-1-60-080

Abstract

The quality of software is enormously affected by the faults associated with it. Detection of faults at a proper stage in software development is a challenging task and plays a vital role in the quality of the software. Machine learning is now a days a commonly used technique for fault detection and prediction. However, the effectiveness of the fault detection mechanism is impacted by the number of attributes presented in the dataset. This paper thoroughly gives the importance to compare between different machine learning approaches and by observing their performances we can conclude which models perform better to detect fault in the selected software modules and investigates the effect of various feature selection techniques on software fault classification by using NASA's some benchmark publicly available datasets. Various metrics are used to analyze the performance of the feature selection and classification techniques. The experiment discovers that some particular classifiers can detect the presence of the faults more effectively and by selecting the best features and solving the class imbalance problem can ensure better quality of the software.

Acknowledgment

All praise, gratitude and thanks are due to omnipotent, omnipresent and omniscient Allah, who has created to conduct the research work successfully.

The authors would like to express their sincere appreciation, deepest sense of gratitude and immense indebtedness to their honorable thesis supervisor **Dr. Shamim H Ripon,** Professor, Department of Computer Science and Engineering, East West University, Bangladesh, for their systematic planning, painstaking and scholastic guidance, encouragement, inestimable help, valuable suggestions, loving care and gratuitous labor and all kinds of support in conducting and successfully completing the research work and in the preparation of the manuscript.

The authors would like to express their heartiest gratitude and profound respect to **Dr. Taskeed Jabid**, Chairperson & Associate Professor, Department of Computer Science and Engineering, East West University, Bangladesh, for his kind co-operation during the study period and helping us in various steps of this work.

The authors would like to extend their heart-felt thanks and deepest appreciation of gratitude to all the teachers of Department of Computer Science and Engineering, East West University, Bangladesh, for their guidance, valuable suggestions, compassionate help and continuous encouragement throughout the period of research work.

Last but not least, authors feel heartiest indebtedness to their beloved parents, brother, sisters for their patient inspiration, sacrifice, blessing and never ending encouragement.

Shamse Tasnim Cynthia

December, 2019

Md. Golam Rasul

December, 2019

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this era, we cannot think of our life without software. In every aspect of life, we have software to facilitate our everyday battles and to accelerate our works. So fault in software can emerge some big and complex issues in the banking, medical, industrial sectors if there exist any defects in the software, the outcome is uncountable which can even cost individuals' lives and huge financial loss [1]. This makes the software system more complex than before and some of the software system have to be delivered with least or non-negligible number of faults possible. With the increasing demand of large and complex software in various sectors, the probability of having software defects has been increased and traditional quality assurance methods are not sufficient to overcome all software defects in such huge systems So detecting faults in the software has turned into a genuine subject to consider. Defect prediction helps in identifying the vulnerabilities in the project plan in terms of lack of resources, improperly defined timelines, predictable defects etc. it can help organizations to fetch huge profits without getting delayed on schedules planned or overrun on estimates of budget. It helps in modifying the parameters in order to meet the schedule variations.

1.1 Problems and Motivation

Faults can be defined as a basic flaw in a product framework. Faults are often generated for misunderstanding, lacking knowledge in the working area or even for the deadlines. The process of software testing was mainly introduced to examine if there are any defects remain in the software. Through numerous steps and techniques, the tester tries to detect the faults. Often the defects are found in the last stage of SDLC. The early stage fault prediction [2] can cause less effort and money however when faults are distinguished at the last stage it requires high repair, cost and the quality of the software reduces. Several studies reveal that 80% of the software faults occur from 20% of the modules and defect dree portion covers the rest half of the module [3]. Software faults demand the rework process that has negative impact on for Software Quality Assurance [4]. Ability in detecting software faults mostly assist software developers in the testing

phase about maintaining software standards [5]. So predicting software faults at the early stage of software development can support the development of more efficient and reliable software within the stipulated limited time and cost [6].

There are numerous perspectives which may prompt software defects in its software life cycle, such as Software requirements, software design, software coding and software testing and so on. However, software defects which are delivered during the phases of software requirements, software design and software testing, will at last expressed in software Source Codes. Therefore, detecting software defect by software source codes is the most well-known method to predict software defect [7]. The most frequently metric that researcher used to predict the software fault in software source codes is Size and Complexity metrics. Size and complexity metrics is an conceptual articulation of software source codes complexity, such as line of code, cyclomatic complexity and design complexity and so on [8]. Accurate prediction of faulty software modules enormously helps in lessening testing effort, by helping software testers to focus on the flawed modules. As we are probably aware that software testing is a very exorbitant and timing consuming activity. In software development process testing usually requires 40% of the whole project schedule [9]. There is no intermediate technique to quantify the fault proneness [8], [9]. Based on software metrics, faulty software modules are detected using software defect prediction module. Unfortunately, there is no generic technique for estimating software modules as faulty or non-faulty [10].

For detecting fault in the software, Machine learning techniques have been broadly utilized. Machine learning is the scientific study of algorithms and statistical models. This is the science of getting computers to operate and generate results without explicit instructions. A set of training data and testing data is required for the machines to make prediction or decisions and gives results without instructions given to it. For this, software researchers have found that defect prediction using machine learning approaches is practical and useful [11]. As this technique automatically predicts of possible faulty environment, software developers can fix those infirmities in early stage of developing a software and put emphasize on software's main development. In [12] various types of fault have been identified and the preventive approach was applied using the Learn-to rank algorithm. For increasing the performance, the back-propagation technique is attached with LRA approach for enhancing its performance. In this paper, we apply different data mining approaches and after analyzing certain results, the fault prediction of software is improved. This not only saves the time but also reduces software's infrastructure cost.

Feature selection is one of the most significant techniques for any kind of data analysis. Features are mainly referred as the attributes which are given in a dataset and have strong correlation with the class attribute [13]. A random dataset containing lots of features needs to extract the most important features for better analysis otherwise the result which comes from classification, prediction or regression may not be the expected one. All the features in a dataset are always not the important or relevant to the classification. They may contain irrelevant, duplicate

and useless data that will not participate in any of the prediction or classification system rather, these type of data can consume extra processing time and affect the quality of the analysis result. So the main aim of feature selection in a dataset is to select the essential features which will help to improve the fruitfulness of a model. The feature selection techniques not only increase the accuracy and efficiency of a classifier but also decreases the chance of overfitting, reduces the dimensionality and eliminates noise [14] [15]. In addition, it can seek out useful information deliberately and lessens the effect of variance in the result. Proper selection of features can help researchers looking for the exact fault in the model. Again when the most essential features are selected, the reduces dimensionality of a dataset boosts the performance of some algorithms, delivers more accurate results in the less amount of time. Most of the feature selection techniques extract features that ranges from sub-optimal to near optimal solutions [16]. By ranking the features with different score, feature selection techniques reach to near optimal solutions.

Another significant issue that should be tended to for making fault proneness prediction process increasingly successful is to manage imbalanced datasets. Normally data in reality is imbalanced. Class unevenness implies that dataset contains countless examples for a specific class then different classes exist in that dataset. The class imbalance issue is deceptive and ever present in the dataset for fault proneness prediction because of the way that number of defective modules cases is not exactly number of non-faulty modules instances. The class imbalance issue commonly happens in classification problem. Because of irregularity classes the vast majority of datasets are exceptionally slanted toward a particular class of occurrences. In such cases, due to imbalanced dataset the productivity of fault detection is gravely trouble. In this manner, there is a need to adjust the dataset so as to improve the productivity of prediction model. Outlier is a perception that seems, by all accounts, to be digress from different examples of test in which it happens for example a perception that is conflicting with the rest of datasets. Presence of outliers in the datasets utilized for software defects surrenders frequently sway the presentation of defect of prediction models.

1.2 Objectives

Our objectives in this project are as follows,

- To compare the accuracies of different machine learning algorithms and providing a comparative examination of these algorithms
- To distinguish the actualities which are causing software being defective or not. The work exhibited in the paper also aims at observing which attributes are responsible more in the classification of software fault prediction.
- To consider the significance of feature selection in prediction and classifying software faults, the work aims at investigating the effect of various feature selection techniques upon the performance of various classification algorithm.

- To apply several feature selection techniques to some used fault prediction datasets and then apply classification and predictive techniques on the datasets having only those features selected earlier.

To find the solution of the problem of class imbalance problem is also taken care of as we need to balance the dataset in order to make our model more efficient and the comparison between the balanced and imbalanced class has also been shown.

1.3 Contribution

We have made the following contribution in this project:

- We have collected a publicly available dataset, applied multiple classification models and rule induction techniques
- We have observed which algorithm performs better by calculating accuracy and which attributes are necessary for generating rules.
- For each dataset, all the feature selection techniques are applied and relevant features are selected for each type of technique.
- Classification algorithms are then applied to the selected features obtained from each feature selection techniques.
- For comparative analysis, experiment has also been conducted considering all the features in the dataset and then compare the result with that of the selected features.

For balancing the dataset, we have applied SMOTE and comparative analysis represented the improvement of the evaluation metrics after balancing the datasets

1.4 Outline

The report is organized as follows:

- Chapter 2 gives a literature review of software fault detection, feature selection and class imbalance problem of these datasets.
- Chapter 3 gives a brief overview of the datasets which are being used in the experiments.
- Chapter 4 gives a model of our proposed works and illustrated it in a diagram.
- Chapter 5 gives a brief description of the result analysis and the comparison between the results.
- Chapter 6 gives a brief description of the class imbalance problem and how this problem can be solved.

- Chapter 7 gives an overview and the description of the tool that we have implemented to make the experiment easy and effective.
- Finally, in Chapter 8 we give a summary of this thesis and outline our future plans:

1.5 Publications

The following international conference papers have been published and presented from the project:

1. Shamse Tasnim Cynthia, Md. Golam Rasul and Shamim Ripon, *Effect of Feature Selection in Software Fault Detection*, 13th Multi- disciplinary International Conference on Artificial Intelligence, November 17-19, 2019. Kuala Lumpur, MALAYSIA, 2019. Springer International Publishing, LNAI 11909. https://doi.org/10.1007/978-3-030-33709-4_5
2. Shamse Tasnim Cynthia and Shamim H Ripon, *Predicting and Classifying Software Faults: A Data Mining Approach*, 7th International Conference on Computer and Communications Management (ICCCM 2019) Bangkok, Thailand, July 27-29, 2019. (ACM Indexed). https://doi.org/10.1145/3348445.3348453.

Chapter 2

Literature Review

2.1 Fault Detection Analysis

For predicting defects in software, various techniques have been developed such as linear regression, discriminate analysis, decision trees, neural networks etc. For software fault prediction different data mining classification techniques have been surveyed by Yuan Chen, et.al [17]. A new model based on Bayesian network and PRM to predict the software defect and manage have been proposed by them. Their Nu Phyu [18] reviewed on various classification techniques such as decision tree induction, Bayesian networks, k-nearest neighbor classifier, case based reasoning, genetic algorithm and fuzzy logic techniques. Finding of the results is not satisfactory on which is the best classifier. A set of interacting methods to large numbers of makers has been produced by several of the classification methods caused a potential risk picking up randomly associated markers.

Issam H et.al [19] have proposed a classifier called two variant ensemble learning classifier which shows that greedy forward selection performs better comparatively than correlation forward selection. For the multiple datasets, further they proposed a model called APE with greedy forward selection to generate higher AUC measures. The results shown stronger robustness to redundant and irrelevant features. For imbalanced datasets R , enqing Li and Shihai Wang [20] predicted defects. On imbalanced datasets of NASA's MDP C4.5, SVM, KNN, Logisitc regression, Naïve Bayes, Adaboost and smooth boost models were tested. Smooth boost found to be the best defect predictor when compared to others when results were found out.

2.2 Feature Selection

Oinbao Song et. al [21] proposed a framework model to follow-up the MGF on defect prediction using Scheme evaluation and defect prediction for feature selection. For comparing the performance, only three algorithms Naive Base, J4.8 and OneR were used. ROCUS for software defect prediction has been used by Jiang et al. [22]. For detecting the software fault two vital issues were addressed by the authors. They proposed a disagreement-based semi-supervised learning method to exploit the abundant unlabeled data but higher misclassification rate is the limitation for this technique for better prediction.

Recently two types of techniques called evolutionary and swarm intelligence evolved and therefore not much of research could have been done on these techniques w.r.t. defect prediction. Relationship of software quality and defects with some mining techniques like Logistic Regression, C4.5, Association Rule Mining, Random Forest, Naïve Bayes, Artificial Neural Network, Fuzzy Programming and Genetic algorithm to conclude that data mining techniques helps eliminate vestigial defects has been studied by Arun Singh et.al. [23]. Different techniques about Hybrid combinations have also evolved in years and have almost always performed better that the original technique itself. A new model defined by him et.al. [24] by combining advantages of Particle Swarm Optimization and SVM: P_SVM and applied on the JM1 dataset of NASA, with 10CV. The result of the comparison with Back-propagation neural network, SVM and Genetic Algorithm SVM (GA-SVM) showed that P-SVM had maximum accuracy. NB, MLP and Votinf Feature intervals (VFI) classification of 5 datasets and results show that combination of these classifiers has higher probability of fault detection specifically for embedded systems which has been used by Atac Deniz Oral et. al. [25].

By selecting important attributes Using five classifiers: IBK, KStar, LWL, Random Tree and Random Forest, Misha Kakkar et al. [1] tried to build a framework. For evaluating the performance of these classifiers the values of accuracy and ROC was used. ChiSquaredAttributeEval and CorrelationAttributeEval ranked the attributes based on their individual evaluation and Attributes selection was done through CfxSubsetEval evaluator. Different feature selection techniques have been combined in a hybrid feature selection approach which has been introduced by Lina Jia [26]. Chi Square, Information Gain and Pearson Correlation Coefficient techniques are used. For finding the correlation among the attributes Qiao Yu et al. [27] proposed a feature selection approach on the basis of similarity measurement for software defect prediction. Feature list is generated in descending order by updating the feature weights and by sorting them according to their rankings. Finally, on the selected dataset for the detection of faults, K-nearest model classifier is applied.

A feature selection framework named MICHAC which stands for Maximal Information Coefficient with Hierarchical Agglomerative Clustering was proposed by Zhou Xu et al. [28].

This framework fetches one feature from each feature subset groups to eradicate the irrelevant features. For evaluating the performance of the model built with selected featured datasets, three different classifiers and four performance metrics were used. Bat-Based Search Algorithm was used by Dyana Rashid Ibrahim et al. [4] in their work for the feature selection purpose

2.3 Class Imbalance Problem

Romi Satria Wahono et al. in their research [16] gave priority on imbalance nature of the NASA dataset on software defect prediction and for feature selection. To predict the defects M. Anbu et al. in their research [29] used Genetic Algorithm which has been used onl Firefly algorithm for feature selection and classifiers like Support Vector Machine, Naïve Bayes and K- nearest neighbor.

Considering the fact of class imbalance problem present in the datasets, a big number of defect prediction models have been introduced. For selecting an appropriate attributes and a technique for dealing with imbalanced class, T. M. Khoshgoftaar et. al. [30] proposed a process that includes a technique. They concluded that selection of appropriate metrics is very vital. For sampling minority class examples, L. Pelayo and S. Dick used SMOTE technique. After analysis of the result by applying SMOTE resampling, an improvement of 23% in average geometric mean of classification accuracy [31]. To handle the software fault prediction problem with highly imbalanced datasets, Z. Li and M. Reformat used SimBoost machine learning method. Their proposed methods reasonably reduce the effect of imbalanced datasets after experiment the result; however, the prediction for balanced dataset was not accurate. they proposed fuzzy label for classification [8], In order to deal with this issue. S. Lessman et.al. investigated multiple classification models over 10 public domain software datasets from NASA MDP repository. the significance of particular classification may possibly be less supposed as there could be no noteworthy differences detected in the performance of top 17 classifiers [32] was specified by their experimental result.

However, in our work we have tried to apply five feature selection processes on five different datasets to select the most relevant and essential features and five classifiers are also used in prediction of the software defects along with solving the class imbalance problem by applying five sampling techniques.

Chapter 3

Dataset Overview

3.1 Dataset Overview

We have used NASA MDP dataset for our research. This dataset is a set of 96 datasets among them 13 datasets have been provided by NASA [33] and we have taken 8 datasets to do experiment. These datasets have been used commonly for software fault prediction though some preprocessing of these data are needed for suitable and errorless defect prediction [34]. Table 3.1 shows the name of the selected datasets, total sample number, the total number of defective and not defective values of each dataset and the language used on them. The attributes in these datasets are mostly of numerical values except for the class attribute which is a polynomial one.

TABLE 3.1: Dataset Overview

Dataset	Total Sample	Defective	Not defective	Language Used
CM1	344	42	302	C
KC1	2096	325	1771	C++
JM1	7782	1672	6110	C
PC1	759	61	698	C
KC3	200	36	164	Java
MW1	264	27	237	C
PC4	1399	178	1221	C
PC2	1585	16	1569	C

The software presented in each data is from different projects of NASA [20]. Table 3.2 shows the source of the datasets.

TABLE 3.2: Source of The Dataset

Dataset	Notes
CM1	spacecraft instrument
KC1	storage management for receiving/ processing ground data
JM1	a real time predictive ground system
PC1	an earth orbiting satellite
KC3	storage management for ground data
MW1	zero gravity experiment related to combustion
PC4	flight software for earth orbiting satellite
PC2	flight software for earth orbiting satellite

The datasets consisted of several and great numbers of attributes. These attributes mainly ensure the quality of metric data program. Each data set is consisted of system or subsystem that represents static code metrics and each module are comprised of fault data. a function, procedure or method is referred as the module. The static code metrics record includes lines-of-codes (LOC) count, Halstead and McCabe based measures. The form of error count metric is taken by the primary fault data to calculate the number of error reports which was issued for each module bua a bug tracking system reportedly. The details which are given at the original NASA MDP Repository, it is not clear exactly how these error reports were mapped back to the individual modules. However, it was stated that, “if a module is changed due to an error report (as opposed to a change request) then it receives a one up count. It cannot receive more than a one up for a given error report”. It was also stated that “the error count metric describes the number of changes due to errors”. The source code explaining the origination for these data sets is wholely closed source, which makes the validation of data integrity more difficult. A huge amount of researches has been conducted over the last decade containing these facts. Fig 3.1 illustrated the description of the dataset attributes.

group	metrics	description or formula
code	PARAMETER_COUNT	Number of parameters to a given module
	NUM_OPERATORS:N1	The number of operators contained in a module
	NUM_OPERANDS:N2	The number of operands contained in a module
	NUM_UNIQUE_OPERATORS:μ_1	The number of unique operators contained in a module
	NUM_UNIQUE_OPERANDS:μ_2	The number of unique operands contained in a module
	HALSTEAD_CONTENT:μ	The halstead length content of a module $\mu = \mu_1 + \mu_2$
	HALSTEAD_LENGTH:N	The halstead length metric of a module $N = N_1 + N_2$
	HALSTEAD_LEVEL:L	The halstead level metric of a module $L = \frac{(2*\mu_2)}{\mu_1 * N_2}$
	HALSTEAD_DIFFICULTY:D	The halstead difficulty metric of a module $D = \frac{1}{L}$
	HALSTEAD_VOLUME:V	The halstead volume metric of a module $V = N * \log_2(\mu_1 + \mu_2)$
	HALSTEAD_EFFORT:E	The halstead effort metric of a module $E = \frac{V}{L}$
	HALSTEAD_PROG_TIME: T	The halstead programming time metric of a module $T = \frac{E}{18}$
	HALSTEAD_ERROR_EST: B	The halstead error estimate metric of a module $B = \frac{E^{2/3}}{1000}$
	NUMBER_OF_LINES	Number of lines in a module
	LOC_BLANK	The number of blank lines in a module
	LOC_CODE_AND_COMMENT:NCSLOC	The number of lines which contain both code and comment in a module
	LOC_COMMENTS	The number of lines of comments in a module
	LOC_EXECUTABLE	The number of lines of executable code for a module (not blank or comment)
	PERCENT_COMMENTS	Percentage of the code that is comments
	LOC_TOTAL	The total number of lines for a given module
design	EDGE_COUNT:e	Number of edges found in a given module control from one module to another
	NODE_COUNT:n	Number of nodes found in a given module
	BRANCH_COUNT	Branch count metrics
	CALL_PAIRS	Number of calls to other functions in a module
	CONDITION_COUNT	Number of conditionals in a given module
	CYCLOMATIC_COMPLEXITY: v(G)	The cyclomatic complexity of a module $v(G) = e - n + 2$
	DECISION_COUNT	Number of decision points in a given module
	DECISION_DENSITY	$Condition\_count/Decision\_count$
	DESIGN_COMPLEXITY:iv(G)	The design complexity of a module
	DESIGN_DENSITY	Design density is calculated as: $\frac{iv(G)}{v(G)}$
	ESSENTIAL_COMPLEXITY:ev(G)	The essential complexity of a module
	ESSENTIAL_DENSITY	Essential density is calculated as: $\frac{(ev(G)-1)}{(v(G)-1)}$
	MAINTENANCE_SEVERITY	Maintenance Severity is calculated as: $\frac{ev(G)}{v(G)}$
	MODIFIED_CONDITION_COUNT	The effect of a condition affect a decision outcome by varying that condition only
	MULTIPLE_CONDITION_COUNT	Number of multiple conditions that exist within a module
	PATHOLOGICAL_COMPLEXITY	A measure of the degree to which a module contains extremely unstructured constructs
others	NORMALIZED_CYLOMATIC_COMPLEXITY	$\frac{v(G)}{NUMBER\_OF\_LINES}$
	GLOBAL_DATA_COMPLEXITY:gdv(G)	the ratio of cyclomatic complexity of a module's structure to its parameter_count
	GLOBAL_DATA_DENSITY	Global Data density is calculated as: $\frac{gdv(G)}{v(G)}$
	CYCLOMATIC_DENSITY	$\frac{v(G)}{NCSLOC}$

Fig. 3.1: Description of the attributes

Chapter 4

Proposed Model

Considering the significance of early fault prediction in software, our work aims to investigate the proper classification, identification of relevant attributes and solving the class imbalance problem. Fig 4.1 shows the complete picture of our proposed model we have done in our project. To do so firstly several classifiers are taken in order to classify the defects in the software. The classification is tested using cross validation and without cross validation. Five types of classifiers are taken, then the model is constructed and by evaluating the model, the results are measured and performance of different classifiers are compared to see which classifier can give us the best result.

The relevant feature selection is one of the major tasks to do. So we took 5 datasets and applied 5 feature selection techniques. The top 10 and top 20 features are selected which are turned into different subset of the actual dataset. After applying the selection processes five classifiers are implemented on the top 10 and top 20 datasets to check which subset of the dataset works better as there might be chances that the relevant features are not considered in any dataset. Then with the help of performance metrics the results are compared to check if the selected features are actually the important one.

The class imbalance problem of NASA dataset in another important issue that needed to be solved. In order to do that three sampling techniques are taken to solve the class imbalance problem. After applying these three techniques, on classifier is applied on each of the balanced dataset and the result contained which technique worked better for sampling the dataset and can improve the efficiency of the model to detect the software defects accurately.

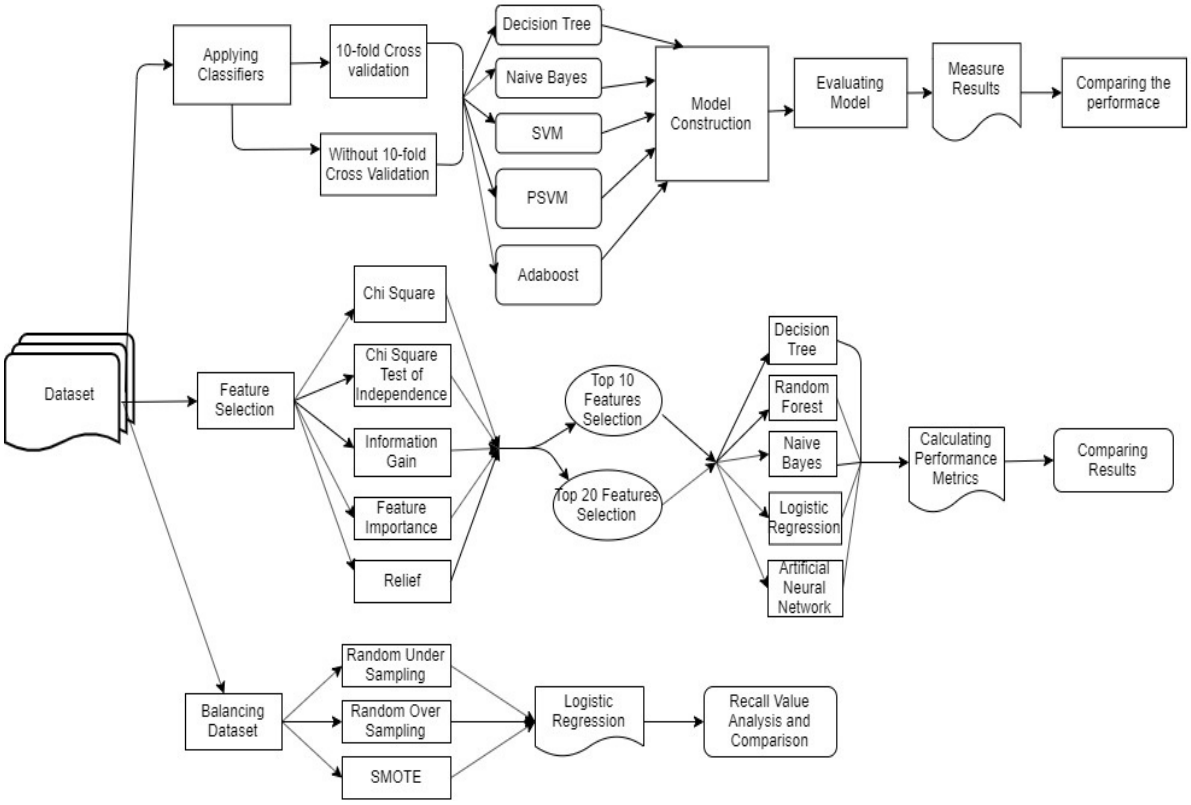

Fig. 4.1: Proposed model of our experiment

4.1 Algorithms and Techniques

There are several algorithms and techniques are applied to achieve our expected outcome.

4.1.1 Algorithms Used

For implementing the proposed methodology, we have used five classifiers [35]. They are: a) Decision Tree, b) Naïve Bayes, c) Support Vector Machine, d) Support Vector Machine(PSO), e) Adaboost [36], f) Logistic Regression, g) Random Forest and h) Artificial Neural Network. Some details of these classifiers are described below:

Decision Tree: Decision Tree [37] is a tree-like collection of nodes plan to produce a decision on values associating with a class or an estimate of a numerical target value. Rules are generated for each attribute. In our research, we have used the information gain criterion. The maximal depth of the tree is 10, confidence is 0.1, minimal gain 0.01 and minimal leaf size is 2.

Naïve Bayes: Naïve Bayes [38] is a machine learning algorithm. It is probabilistic and used for classifying tasks. When the value of the label attribute is given, Naïve Bayes assumes that the value of any attribute is independent of the value of any other attribute.

Support Vector Machine: Support Vector Machine [39] is a supervised learning model.

It is an associated learning algorithm that inspects data which are used for classification and regression analysis [40]. The parameters are: dot kernel type, C parameter is 0.00 and convergence epsilon is 0.001.

Support Vector Machine(PSO): This is Support Vector Machine learner which uses Particle Swarm Optimization [41] for optimizing a problem by repeatedly trying to improve a solution with regard to a given measure of quality. The radial kernel type is used here. C parameter is 0.0, kernel gamma is 1.0, inertia weight is 0.1, the local best weight is 1.0 and the global best weight is 1.0.

Adaboost: Adaboost(Adaptive Boost) is a boosting algorithm. It is a nested operator and it has subprocess. The subprocess consists of a learner and given an example set, it generates a model. It emphasizes on classification problems and tries to convert weak classifiers into a strong one. We have used 10 iterations for our method.

Logisitc Regression: Logistic regression [42] is a linear classifier which is probabilistic. The parameters are a weight matrix w and a bias b. With enabling the system to estimate categorical results it takes help of a group of variables which are independent. The equation for weight is updated according to the value and with this the average cost value is calculated.

Random Forest: Random forest [43] is an ensemble classifier which is being used to utilize a particular number of classifiers to work together so that they can identify class labels for instances which are unlabeled. The high accuracy value of this approach proved its superiority and effectiveness of the with imbalanced dataset. To resolve class imbalance problem, this classifier provides several techniques.

Artificial Neural Network: An Artificial Neural Network [44] is an engineering approach which is of biological neuron. Many inputs and one output are associated with it. Simple processing elements consists which is large in number basically consists the ANN. These elements are interconnected with each other and they are layered also.

4.1.2 Feature Selection Techniques

The **Chi-Square test** is introduced by Karl Pearson (1900) which is a statistical hypothesis test that determines the goodness of fit between a set of observed and expected values [45, p. 1]. It is a nonparametric test that is used for testing the hypothesis of no association between two or more groups, population or criteria and to test how well the observed distribution of data fits with the distribution that is expected [46].

The formula for Chi-square is:

$$X_c^2 = \sum_{1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where X_c^2 = Chi-square test, c = degrees of freedom, O = observed value(s), E = expected value(s).

As mentioned already, Karl Pearson (1904) introduced **Chi-square test of independence** which is used to detect if there is a significant relationship between two nominal (categorical) variables [45]. Each category's frequency for one nominal variable is compared with the categories of another nominal variable. Each row of the data in a contingency table represents a category for one variable and each column represents a category for other variable [47].
The formula for Chi-square test of independence is:

$$X^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{\left(O_{ij}-E_{ij}\right)^2}{E_{ij}}$$

Where X^2 = Chi-square test of independence, O = observed value(s), E = expected value(s), r = number of rows, c = number of columns.

Information Gain is a measure of the change of entropy which reduces the uncertainty of the result. Entropy gives the measure of impurity of the classes. The value of the entropy should be less for getting the best output. When a node in a decision tree is used for partitioning the training instances into smaller subsets, the value of the entropy changes. Information gain specifies the importance of an attribute and decides the ordering of the attributes in the nodes of a Decision Tree.

$$Gain(T, x) = Entropy(T) - Entropy(T, x)$$

Relief is a feature selection algorithm which uses a statistical method and avoids heuristic research. The algorithm inspired by instance-based learning. It needs linear time for the number of given features and the number of training instances regardless of the target concept to be learned.

From given training data, sample size, and a threshold of relevancy, Relief finds those features that are statistically relevant to the target concept. Relief collects the total number of triplets of an instance, its Near-hit instance and Near-miss instance. Euclidian distance is used for selecting Near-hit and Near-miss. A routine is also called by Relief to update feature weight vector for every triplet and finds the average feature weight vector Relevance (of all the features to the target concept) and those features whose average weight is above the given threshold are selected by Relief [48].

Feature Importance returns a score for each feature and based on that score, the features which have higher score get more privilege towards the output variable. It uses ensembles of decision trees which computes the relative importance of each attribute.

We have used extra tree classifier (ETE) which randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting. One great advantage of extra tree classifier over Decision Tree (DT) and Random Forest (RF) is the lower variance whether DT and RF have higher variance. Extra Trees like RF, builds multiple trees and splits node using random subsets of feature but the key difference with RF is the randomness comes from the random splits of all observations.

Chapter 5

Result Analysis

In this section, we investigate the performance of the proposed model based on the algorithms and techniques which are considered to apply.

5.1 Classification Accuracy

For software fault prediction, it is not enough to obtain which model gives higher accuracy. So it is needed to correctly classify the defective and not defective classes for ensuring whether a software is going to be fault-prone or not as classification [49] of the important data can distinguish categories or classes. That is why we have tried to compare the classification models to observe which works better [50]. Different types of classification models have been used throughout the researches in different times [51].

For comparing among the machine learning algorithms [52], accuracy is the most famous one. It is calculated by summation of true positive and true negative divided by total instances. Table 5.1 and Table 5.2 shows the comparison between accuracy calculated with 10-fold cross validation and without cross validation. It clearly shows that decision tree and Adaboost models give the best results recording highest accuracy of 98.55% and 100% from dataset CM1 respectively.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

TABLE 5.1: Accuracy of the classifiers without applying cross validation

Dataset	Decision Tree	Naïve Bayes	SVM	PSVM	Adaboost
CM1	98.55	82.85	87.79	97.67	100.00
KC1	91.08	82.44	85.73	83.73	91.08
JM1	80.84	78.24	78.71	94.35	80.84
PC1	95.65	88.67	92.23	96.84	95.65
KC3	97.50	80.00	83.50	92.00	97.50
MW1	98.48	81.44	89.77	96.97	98.48

TABLE 5.2: Accuracy of the classifiers after applying cross validation

Dataset	Decision Tree	Naïve Bayes	SVM	PSVM	Adaboost
CM1	81.93	82.51	87.46	79.51	83.65
KC1	81.78	82.06	85.12	62.04	81.78
JM1	78.45	78.27	66.95	78.53	78.45
PC1	87.61	88.27	91.97	91.44	89.59
KC3	75.00	79.00	81.50	53.00	78.50
MW1	87.09	81.03	89.72	84.33	87.85

Our main goal is to detect those attributes which are causing a software being faulty. In order to do that the weights of the attributes are calculated and in Rapidminer, the weight operator calculates which attributes are relevant to the class attribute and generates weight for those attributes based on information gain. It was found that the attributes which have higher weight values cause the classes being defective.

In the CM1 dataset, the highest weighted attribute is LOC_COMMENTS. When the graph between LOC_COMMENT and defective class (Y) is generated, it is limpidly shown in Fig 5.1, that whenever the value of LOC_COMMENT increases the Defective (Y) value additionally increases. The probability of getting not defective class decreases as the LOC_COMMENT value increases and after a certain number, it stops completely.

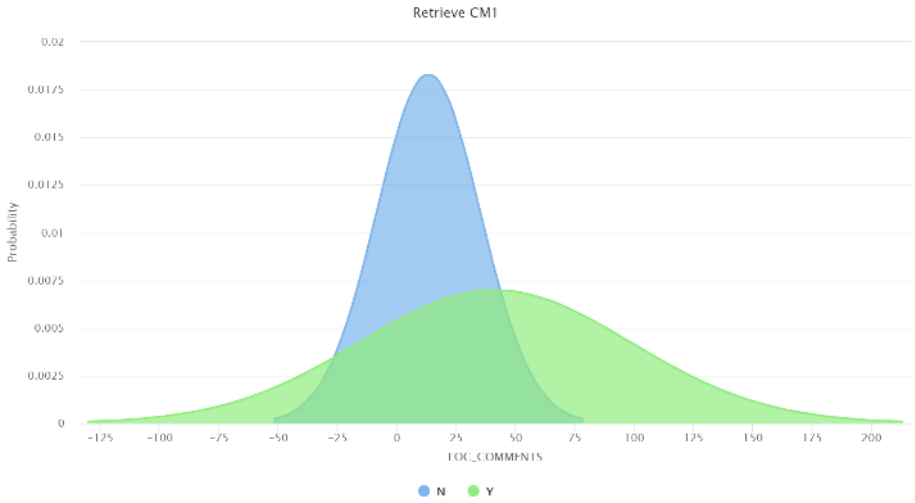

Fig. 5.1: Probability of Classes vs LOC_COMMENTS

Here TP = True Positive, TN = True Negative, FP = False Positive and FN = False Negative. The weight value of LOC_COMMENT for the CM1 dataset was 0.075. Again for MW1 dataset, the highest weighted attribute is NODE_COUNT. The graph in Fig 5.2 shows the probability of classification into defective and not defective is related to mostly with this attribute. We can also see that the lower value of NODE_COUNT attribute causes the software not being so faulty whereas the higher values tend to do the opposite.

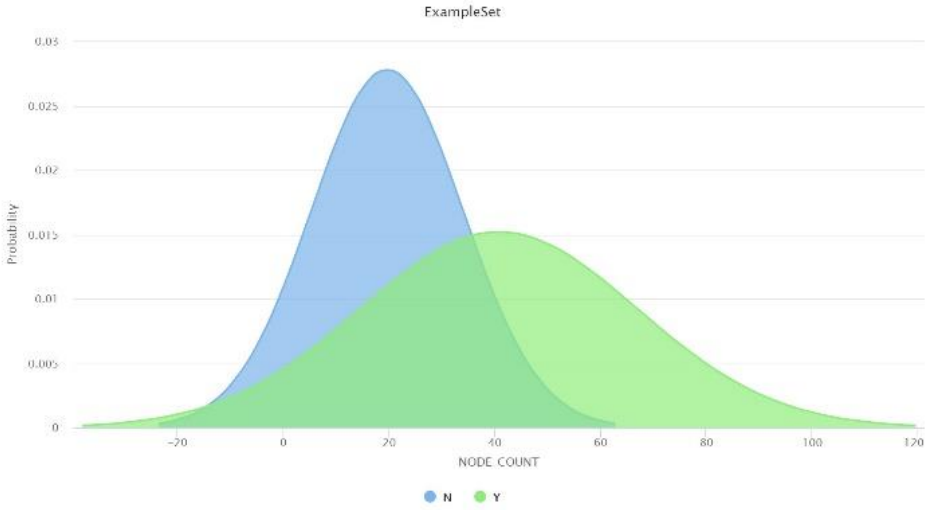

Fig. 5.2: Probability of Classes vs NODE_COUNT

Different cases can occur when the lowest presence of an attribute can make a software being faultier. In PC4 dataset, the highest weighted attribute is LOC_CODE_AND_COMMENT. Fig 5.3 can show us that the minimum values of this attribute makes the greater probability and highest presence of this software actually causes the software for not being faulty

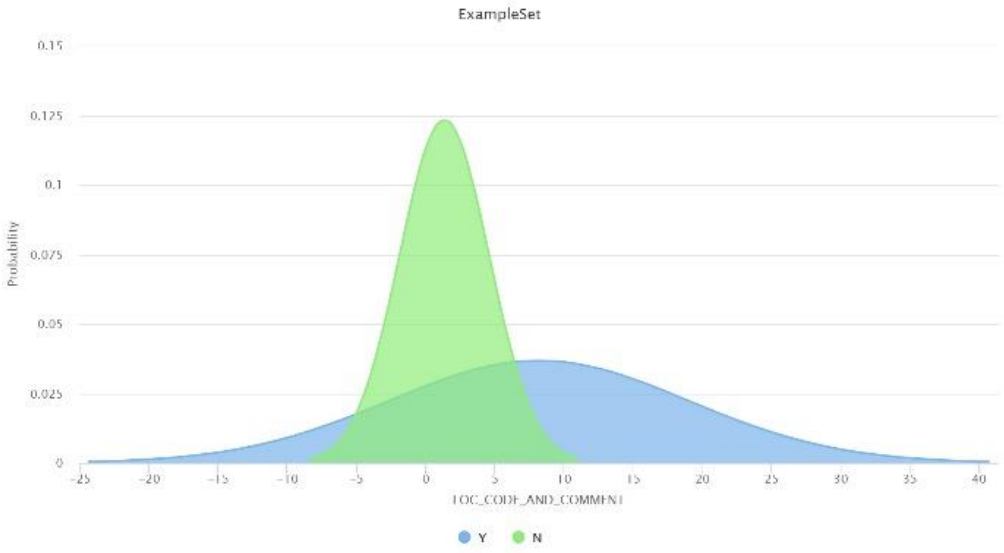

Fig. 5.3: Probability of Classes vs LOC_CODE_AND_COMMENTS

A manual calculation between this attribute and the class attribute in Table 5.3 shows the ratio of being defective and not defective with respect to the incrementing number of LOC_COMMENTS. For example, when the LOC_COMMENT ranges from 1 to 10, the classification ratio is N:Y = 189:7 which concludes that when the number of line of comments are less, the module is more likely to be faultless. Again when the LOC_COMMENT value increases the class is more likely to be defective.

TABLE 5.3: Attributes Statistics

Defective (N)	Defective(Y)	LOC_COMMENT value Range
189	7	1-10
65	12	11-20
28	7	21-30
47	14	31-100
3	2	101-339

5.2 Applying Features Selection Techniques

We have evaluated the performance of our five feature selection processes using the True Positive Rate, True Negative Rate and Accuracy. These metrics help us to examine whether the methods can correctly and efficiently recognize the optimized features and show us the effects of feature selection in the classification [53].

For this purpose, we have used RapidMiner version 9.3 software which provides platform for data science analysis. For calculating accuracy, TPR and TNR values with the classifiers, this software has been used. The classifiers produced confusion matrix containing four parts. True positive, true negative, false positive and false negative. True positive rate or Sensitivity is the result where the positive class is correctly predicted by the model.

$$TPR = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Similarly, true negative rate or Specificity is the result where the negative class is correctly predicted by the model.

$$TNR = \frac{True\ Negative}{True\ Negative + False\ positive}$$

Classification accuracy is the fraction of prediction to see whether the model works right.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

True positive rate, true negative rate and accuracy these three metrics need to be higher for better prediction. We have calculated all these three metrics on the five datasets with five classifiers to see the performance evaluation before and after the five feature selection techniques have been applied. The classifiers used here are: Decision Tree [37], Random Forest [54], Naïve Bayes [38], Logistic Regression [55] and Artificial Neural Network [56]. Table:5.4,5.5,5.6,5.7,5.8 show the results obtained after selecting relevant features and compares the classification metrics with each of the classifiers' predicted results.

5.2.1 Relief Test

TABLE 5.4: Feature selection by Relief test and performance evaluation

Data set	Decision Tree			Random Forest			Naïve Bayes			Logistic Regression			ANN		
	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy
CM1	0.00	1.00	88%	0.38	1.00	92%	0.29	0.90	83%	0.26	0.98	89%	0.10	0.99	88%
KC3	0.56	1.00	83%	0.61	1.00	93%	0.39	0.89	81%	0.36	0.96	86%	0.14	0.98	83%
PC2	0.38	1.00	99%	0.88	1.00	99%	0.31	0.96	95%	0.13	1.00	99%	0.00	1.00	99%
PC4	0.35	0.99	91%	0.23	1.00	91%	0.56	0.86	82%	0.38	0.98	90%	0.48	0.98	92%
MW1	0.00	1.00	90%	0.63	1.00	96%	0.56	0.85	82%	0.40	0.98	91%	0.30	0.98	91%

Here, Table 5.4 illustrates the True Positive Rate(TPR), the True Negative Rate(TNR) and

the Accuracy of the datasets after running the Relief test with some classification techniques. This table shows the results when 20 features have been selected. Again, we have selected 10 features by the same process and the results were more likely similar to this table result or a bit improved result was shown. For example, in PC2 dataset with 20 features, the TPR, TNR and accuracy values when applied Naïve Bayes are 0.31, 0.96 and 95% respectively and with the 10 features, the result is 0.96, 0.25 and 96% respectively. Comparing both the results, we have seen that for Relief test, dataset with 10 most relevant features performed better than the dataset composed of 20 features.

5.2.2 Chi Square Test

TABLE 5.5: Feature selection by Chi Square test and performance evaluation

Data set	Decision Tree			Random Forest			Naïve Bayes			Logistic Regression			ANN		
	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy
CM1	0.43	0.99	92%	0.33	1.00	92%	0.29	0.90	83%	0.24	0.97	87%	0.09	0.99	88%
KC3	0.06	1.00	83%	0.67	1.00	94%	0.36	0.91	81%	0.31	0.96	84%	0.11	0.98	83%
PC2	0.25	1.00	99%	0.88	1.00	99%	0.19	0.97	96%	0.19	0.99	99%	0.00	1.00	99%
PC4	0.29	0.99	91%	0.25	1.00	90%	0.26	0.94	85%	0.39	0.98	91%	0.41	0.98	91%
MW1	0.11	1.00	91%	0.67	1.00	97%	0.56	0.85	82%	0.33	0.99	92%	0.44	0.99	93%

Table 5.5 illustrates the TPR, TNR and Accuracy values of different datasets where features are selected through Chi Square test. We have also calculated values for these dataset with 10 selected features. The dataset CM1 has TPR = 0.43, TNR = 0.99 and accuracy value = 92% by applying Decision Tree classification while with 10 features those values are TPR = 0.28, TNR = 1.00 and accuracy = 91%. Comparing both the results, we have seen that for Chi Square test, dataset with 20 most relevant features performed better than the dataset composed of 10 features.

5.2.3 Information Gain Test

Table 5.6 represents the TPR, TNR and Accuracy values of different datasets where features are selected through Information Gain test. We have also calculated values for these dataset with 10 selected features. The dataset KC3 has TPR = 0.64, TNR = 0.99 and accuracy value = 93% by applying Random Forest classification algorithm while with 10 features those values are TPR = 0.67, TNR = 1.00 and accuracy = 94%. Comparing both the results, we have seen that for Information Gain test, dataset with 20 most relevant features performed almost same as the dataset composed of 10 features.

TABLE 5.6: Feature selection by Information Gain test and performance evaluation

Data set	Decision Tree			Random Forest			Naïve Bayes			Logistic Regression			ANN		
	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy
CM1	0.38	0.99	92%	0.36	1.00	92%	0.36	0.90	84%	0.26	0.99	90%	0.17	0.99	90%
KC3	0.50	0.99	90%	0.64	0.99	93%	0.34	0.91	82%	0.44	0.97	88%	0.31	1.00	87%
PC2	0.25	1.00	99%	0.69	1.00	99%	0.19	0.97	96%	0.19	1.00	99%	0.00	1.00	99%
PC4	0.30	0.99	90%	0.26	1.00	90%	0.54	0.92	88%	0.43	0.99	91%	0.51	0.98	92%
MW1	0.11	1.00	91%	0.59	1.00	96%	0.59	0.86	83%	0.33	0.99	93%	0.44	0.99	94%

5.2.4 Feature Importance

Table 5.7 illustrates the TPR, TNR and Accuracy values of different datasets where features are selected through Feature Importance test. We have also calculated values for these dataset with 10 selected features. The dataset MW1 has TPR = 0.41, TNR = 0.98 and accuracy value = 93% by applying Artificial Neural Network classification algorithm while with 10 features those values are TPR = 0.33, TNR = 0.99 and precision = 93%. Comparing both the results, we have seen that for Feature Importance test, dataset with 20 most relevant features performed almost the same as the dataset composed of 10 features

TABLE 5.7: Feature selection by Feature Importance and performance evaluation

Data set	Decision Tree			Random Forest			Naïve Bayes			Logistic Regression			ANN		
	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy
CM1	0.43	0.99	92%	0.38	1.00	92%	0.38	0.89	83%	0.26	0.98	90%	0.14	0.98	88%
KC3	0.06	1.00	83%	0.67	1.00	94%	0.36	0.91	82%	0.42	0.96	87%	0.22	0.98	85%
PC2	0.31	1.00	99%	0.88	1.00	99%	0.19	0.97	96%	0.25	0.99	99%	0.00	1.00	99%
PC4	0.30	0.99	91%	0.30	1.00	91%	0.30	0.94	85%	0.40	0.98	91%	0.43	0.98	91%
MW1	0.11	1.00	92%	0.67	1.00	97%	0.56	0.86	83%	0.33	0.99	92%	0.41	0.98	93%

5.2.5 Chi Square Test of Independence

Table 5.8 shows the TPR, TNR and Accuracy values of different datasets where features are selected through Chi Square Test of Independence. We have also calculated values for these

dataset with 10 selected features. The dataset PC4 has TPR = 0.29, TNR = 0.99 and accuracy value = 91% by applying Logistic Regression classification algorithm while with 10 features those values are TPR = 0.28, TNR = 0.97 and precision = 85%. Comparing both the results, we have seen that for Chi Square Test of Independence, dataset with 20 most relevant features performed better than the dataset composed of 10 features.

TABLE 5.8: Feature selection by Chi Square Test of Independence and performance evaluation

Dataset	**Decision Tree**			**Random Forest**			**Naïve Bayes**			**Logistic Regression**			**ANN**		
	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy	TPR	TNR	Accuracy
CM1	0.43	0.99	92%	0.33	1.00	92%	0.29	0.90	83%	0.24	0.97	89%	0.09	0.99	88%
KC3	0.06	1.00	83%	0.67	1.00	94%	0.36	0.91	81%	0.31	0.96	84%	0.11	0.98	82%
PC2	0.25	1.00	99%	0.88	1.00	99%	0.19	0.97	96%	0.19	0.99	99%	0	1	99%
PC4	0.29	0.99	91%	0.25	1.00	90%	0.26	0.94	85%	0.39	0.98	91%	0.41	0.98	91%
MW1	0.11	1.00	91%	0.63	1.00	96%	0.51	0.99	83%	0.52	0.87	93%	0.44	0.98	93%

The aim of feature selection for detecting the faults in software was to find out the features which are more important and relevant to the target class and discard the features which are less important or the correlation between them and the target class is less enough to compute the classification without them. NSA dataset consisting 13 datasets each has many numbers of attributes. So if we can find the more important features, the computational time can be reduced, the resources can be less used and the classification efficiency can be increased. We have tried 5 selection processes to select features from the datasets and we took 20 most important features for the classification process.

When applied in decision tree, considering all features to calculate the confusion matrix in CM1 dataset, we found that the algorithm could only predict for N values but no Y values. The Not defective class only has the class precision value. But when applied Chi Square Test, Decision Tree could predict both the Not defective class and Defective class (TP = 18, TN = 300). For Information Gain process the numbers are TP = 16, TN = 299, for Feature Importance process the numbers are TP = 18, TN = 299, for chi square test of independence test the number are TP = 18, TN = 200 but for relief process the numbers are TP = 0, TN = 302. So it can be shown that among the five processes relief could not predict the Defective class like the other classes.

Again for KC3 dataset, the feature selection process had less effect compared to the result which was calculated considering all the features. The TPR and TNR values were usually the same or a little different from the main dataset calculation. For the PC2 dataset, the total number of defective class is very less, so the algorithms could not work better in the classification. The Naïve

Bayes and Logistic Regression algorithms can detect the defective classes in a better way. The datasets with all features and the datasets with the selected features show almost the same result in both cases. In the PC4 dataset, the features selected by Relief process worked better for Naïve Bayes and Decision Tree algorithms and Chi Square test of independence performed better for Random Forest algorithm compared to the result computed when all the features were present. For example, the TP and TN values calculated with all the features are 69 and 1158 respectively but with 20 selected features through Relief process the TP and TN values become 100 and 1049 respectively. Lastly, for the MW1 dataset, all the algorithms with all the features and with the selected features performed almost the same.

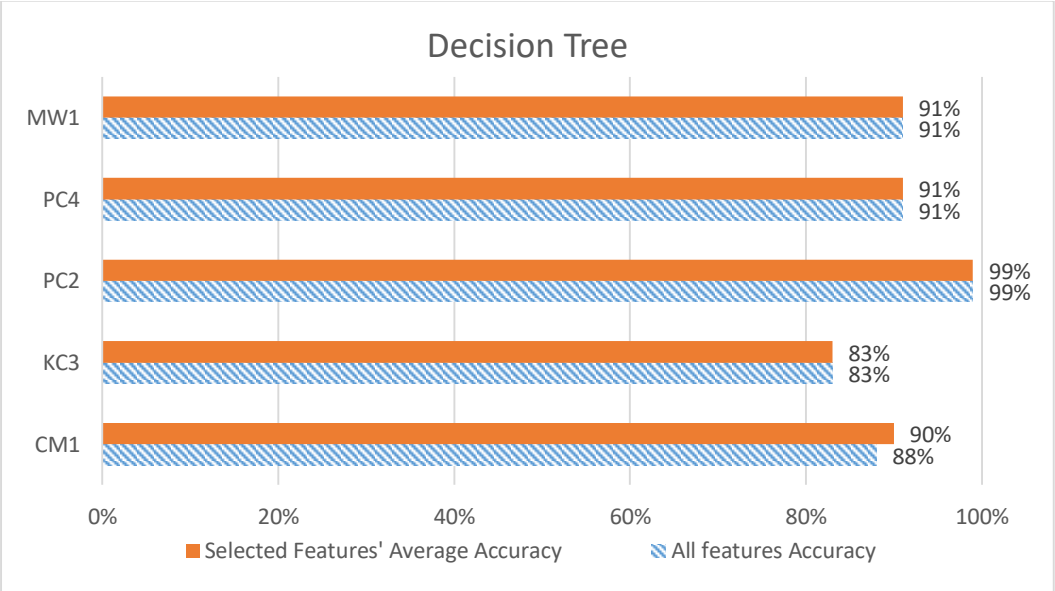

Fig. 5.4: Accuracy of different datasets before and after applying feature selection

In Fig 5.4 the accuracy of the different dataset is shown for the decision tree algorithm. Each dataset is tested with all the features and also with the subsets of selected features' datasets. We have taken the average value of the accuracies when the features are selected with five different processes. It can be showed that the accuracy which was calculated with all the features is almost same and absolutely same in some of the cases. It goes same for the other four algorithms we have applied on our datasets.

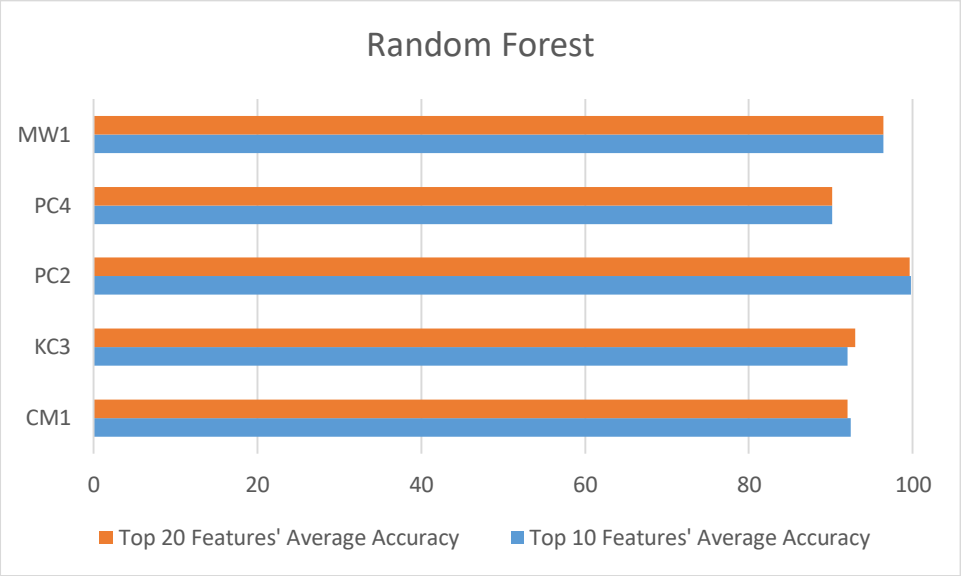

Fig. 5.5: Accuracy comparison between 10 features' average accuracy and 20 features' average accuracy

Similarly, Fig 5.5 illustrates the accuracy comparison between 10 features' average accuracy and 20 features' average accuracy when random forest classifier is applied for the defect prediction. Here we can see that for dataset MW1, PC4 and PC2 the accuracy in both subsets of the datasets gives same or almost the same accuracy values. Whereas, on dataset KC3 the classifier gives better result for subset with top 20 features and on dataset CM1 the classifier gives better result for subset containing top 10 features. For other classifiers, while compared the accuracy with subsets containing 10 features, it has been observed that some classifiers worked better on 10 features' subsets and some worked better on 20 features' subsets but the difference between these accuracy values is quite negligible. This can conclude that the selected features are the most relevant ones to predict the classes more efficiently. The obtained result shows without taking all the features for calculation, the top 10 or top 20 features can be taken and get the best result

5.3 Recall Value Comparison

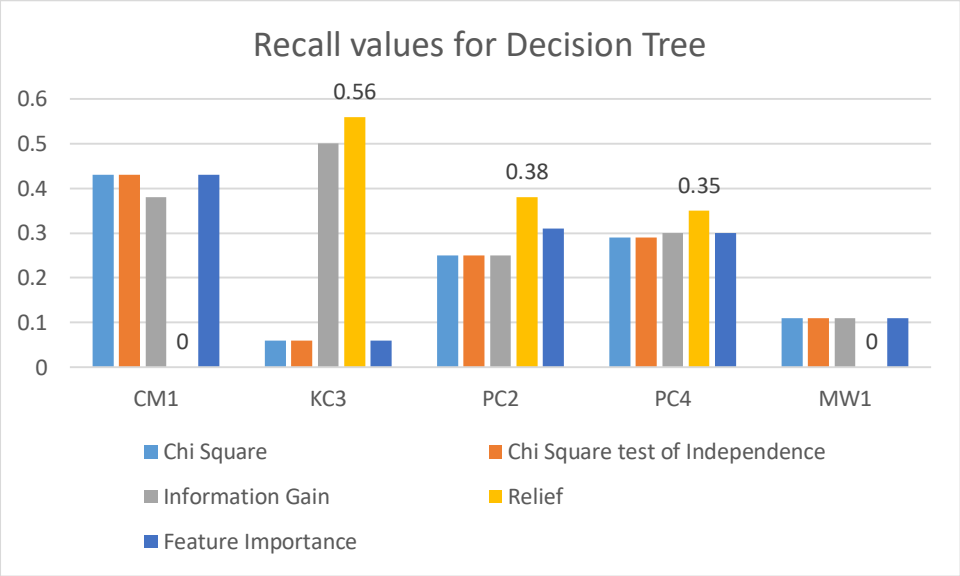

Fig. 5.6: Recall values for Decision Tree

In Fig 5.6 the recall values for each of the feature selection techniques are shown after applying Decision Tree. We can see that the Relief test worked best in case of dataset KC3, PC2 and PC4 and the recall values are 0.56, 0.38 and 0.35 respectively.

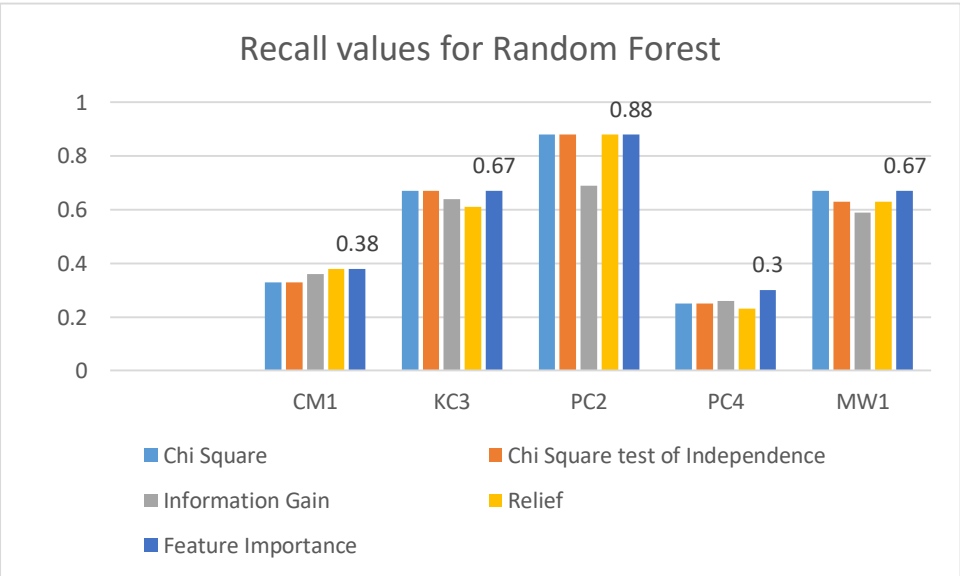

Fig. 5.7: Recall values for Random Forest

In Fig 5.7 the recall values for each of the feature selection techniques are shown after applying Random Forest. We can see that the Feature Importance test worked best in case of all datasets and the recall values are 0.38, 0.67, 0.88, 0.3 and 0.67 respectively.

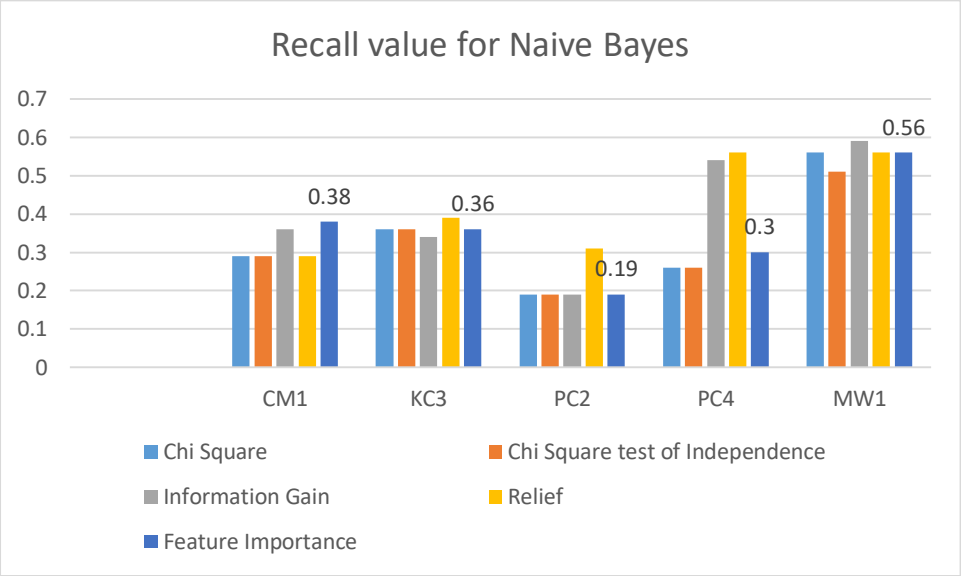

Fig. 5.8: Recall values for Naïve Bayes

In Fig 5.8 the recall values for each of the feature selection techniques are shown after applying Naïve Bayes. We can see that the Feature Importance worked best for CM1 dataset, Relief worked best for KC3, PC2 and PC4 datasets and Information Gain worked best for MW1 dataset.

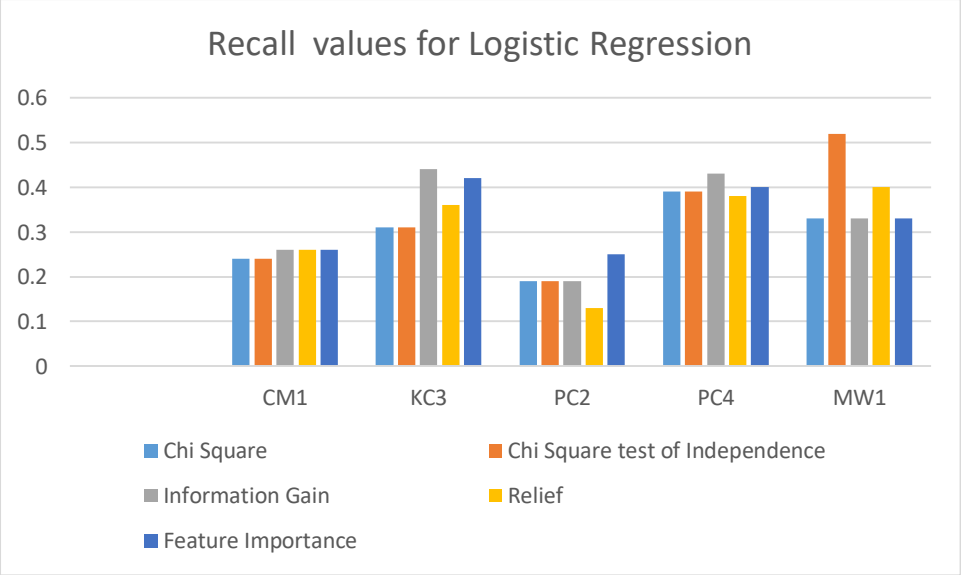

Fig. 5.9: Recall values for Logistic Regression

In Fig 5.9 the recall values for each of the feature selection techniques are shown after applying Logistic Regression. We can see that the Information Gain worked best for CM1, KC3 and PC4 datasets, Feature Importance worked best for PC2 and Chi Square Test of Independence worked best for MW1 dataset.

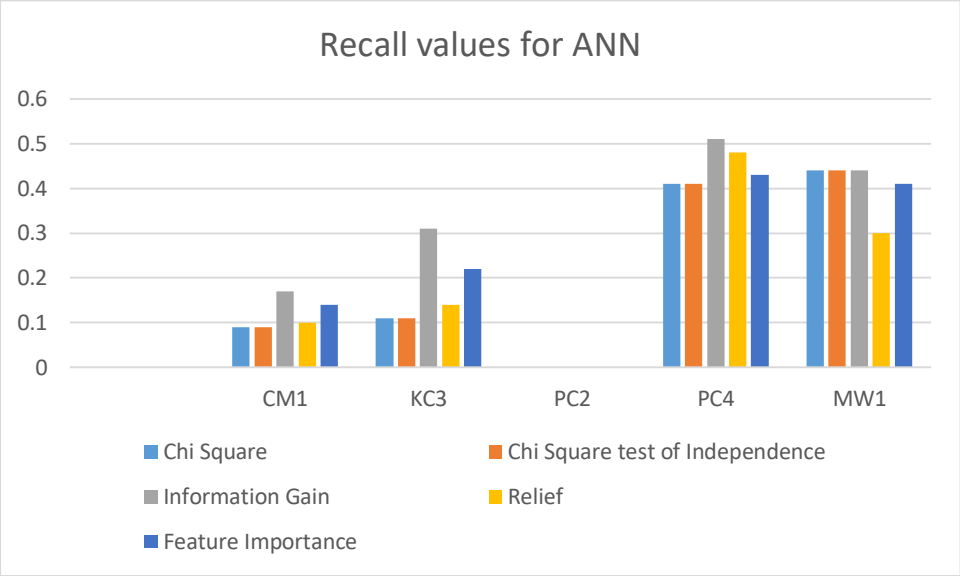

Fig. 5.10: Recall values for Artificial Neural Network

In Fig 5.10 the recall values for each of the feature selection techniques are shown after applying Artificial Neural Network. We can see that the Information Gain worked best for CM1, KC3, PC4 and MW1 datasets. We can also notice here that in case of feature selection techniques ANN classifier could not detect any classes at all.

5.4 Rules Generation

5.4.1 Rules generation for selected attributes without applying feature selection

The association rules [57] are also generated for a specific dataset to witness the attributes' effect on the target values. The rules are basically generated for improving the existing patterns by correctly identifying the important patterns. If the important patterns cannot be generated, the useless patterns get maximized and the prediction of the targeted class gets hampered. The problem which is common to all that is the prediction models which are used to predict the class attributes is not easily understandable to human [58]. Software developers cannot quickly and easily get the fact why a selected module is faulty so by generating rules, it can be easier to the developers to understand better why a certain module is showing a negative result or vice versa. In our selected dataset, each one contains a great number of attributes So we have used the Rapidminer weight calculation. For the experiment, JM1 has been chosen and it was found that LOC_TOTAL, LOC_EXECUTABLE, LOC_BLANK have higher weight values. So association rules are generated based on these attributes and the confidence and support values are also used. These two parameters identify the most important relationships. The confidence value is set to 0.95 for generating the following rules. Following is Table 5.9 where the association rules are shown along with their confidence values.

TABLE 5.9: Rules Generation

$\boldsymbol{R1}$: $\big((LOC_{TOTAL} <= 40.5 \wedge LOC_{TOTAL} >= 24.0) \wedge (LOC_{BLANK} >= 3.5 \wedge LOC_{BLANK} <= 7.5) \wedge LOC_{EXECUTABLE} <= 2.5\big) \rightarrow (CLASS: N)$

$\boldsymbol{R2}$: $\big((LOC_{TOTAL} <= 40.5 \wedge LOC_{TOTAL} >= 12.5) \wedge (LOC_{BLANK} >= 3.5 \wedge LOC_{BLANK} <= 7.5) \wedge LOC_{EXECUTABLE} >= 2.5\big) \rightarrow (CLASS: Y)$

$\boldsymbol{R3}$: $\big((LOC_{TOTAL} <= 40.5 \wedge LOC_{TOTAL} >= 6.5) \wedge (LOC_{BLANK} <= 0.5) \wedge LOC_{EXECUTABLE} <= 2.5\big) \rightarrow (CLass: N)$

$\boldsymbol{R4}$: $\big((LOC_{TOTAL} <= 34 \wedge LOC_{TOTAL} >= 24.0) \wedge (LOC_{BLANK} >= 4.5 \wedge LOC_{BLANK} <= 7.5) \wedge LOC_{EXECUTABLE} <= 22.5\big) \rightarrow (CLASS: Y)$

$\boldsymbol{R5}$: $\big((LOC_{TOTAL} >= 0 \wedge LOC_{TOTAL} <= 24.5) \wedge (LOC_{EXECUTABLE} >= 0 \wedge LOC_{EXECUTABLE} <= 28.5) \wedge LOC_{BLANK} <= 1.5\big) \rightarrow (CLASS: Y)$

$\boldsymbol{R6}$: $\big((LOC_{TOTAL} >= 40.5 \wedge LOC_{TOTAL} \leq 67.5) \wedge (LOC_{EXECUTABLE} >= 0 \wedge LOC_{EXECUTABLE} <= 6.5) \wedge LOC_{BLANK} <= 1.5\big) \rightarrow (CLASS: N)$

$\boldsymbol{R7}$: $\big((LOC_{TOTAL} >= 77) \wedge (LOC_{EXECUTABLE} >= 0 \wedge LOC_{EXECUTABLE} <= 6.5) \wedge LOC_{BLANK} <= 1.5\big) \rightarrow (CLASS: N)$

$\boldsymbol{R8}$: $\big((LOC_{TOTAL} <= 77.0 \wedge LOC_{TOTAL} >= 70.5) \wedge (LOC_{BLANK} >= 0 \wedge LOC_{BLANK} <= 3.5) \wedge LOC_{EXECUTABLE} <= 56.5\big) \rightarrow (CLASS: Y)$

$\boldsymbol{R9}$: $\big((LOC_{TOTAL} >= 0 \wedge LOC_{TOTAL} <= 40.5) \wedge (LOC_{BLANK} >= 0 \wedge LOC_{BLANK} <= 7.5) \wedge LOC_{EXECUTABLE} <= 6.5\big) \rightarrow (CLASS: N)$

$\boldsymbol{R10}$: $\big((LOC_{TOTAL} >= 40.5 \wedge LOC_{TOTAL} <= 77.5) \wedge (LOC_{BLANK} >= 0 \wedge LOC_{BLANK} <= 1.5) \wedge LOC_{EXECUTABLE} <= 6.5\big) \rightarrow (CLASS: N)$

$\boldsymbol{R11}$: $\big((LOC_{TOTAL} >= 0 \wedge LOC_{TOTAL} <= 40.5) \wedge (LOC_{BLANK} >= 0 \wedge LOC_{BLANK} <= 7.5) \wedge LOC_{EXECUTABLE} <= 6.5\big) \rightarrow (CLASS: N)$

We have also used another operator of that software which is w-predictive apriori [17] which is class implementing the predictive apriori algorithm to mine association. The values of selected three attributes with the class attribute have been converted to binomial form from numerical form. After then the predictive apriori operator has been applied and this implementation gave some rules

TABLE 5.10: Generating rules using Apriori

1. $LOC\_EXECUTABLE = false \rightarrow LOC\_BLANK = false$
2. $LOC\_EXECUTABLE = false \rightarrow LOC\_BLANK = false \wedge label\{Y, N\} = Y$
3. $LOC\_BLANK = false \wedge label\{Y, N\} = Y \rightarrow LOC\_EXECUTABLE = false$
4. $LOC\_BLANK = false \rightarrow label\{Y, N\} = Y$
5. $label\{Y, N\} = Y \rightarrow LOC\_BLANK = false$
6. $LOC\_BLANK = false \rightarrow LOC\_EXECUTABLE = false$
7. $LOC\_BLANK = false \rightarrow LOC\_EXECUTABLE = false \wedge label\{Y, N\} = Y$

8. $label\{Y, N\} = Y \rightarrow LOC\_BLANK = false \wedge LOC\_EXECUTABLE = false$

5.4.2 Rules generation for selected attributes after applying feature selection

Again, we have mined the following rules for "CM1" dataset after applying feature selection techniques on Decision Tree classifier for best ten attributes. After building the decision tree classifier, we converted it into equivalent set of rules. For generating the rules, we traced each path in the decision tree from root to leaf node. Though we have used 5 classifiers, 5 feature selection techniques and 5 datasets, there can be 125 combinations possible. It is hard to mine the rules for all 125 combinations. That is why we randomly picked one combination and generated all the rules which are really significant for our experiment. We have found out total 17 rules which have big impact on "defective" class for the described scenario.

1. $(LOC\_EXECUTABLE \leq 68.0 \wedge LOC\_COMMENTS \leq 6.0 \wedge LOC\_TOTAL \leq 54.0 \wedge HALSTEAD\_EFFORT \leq 1230.0 \wedge HALSTEAD\_VOLUME > 153.0) \rightarrow defective$
2. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS <= 6.0 ∧ LOC_TOTAL <= 54.0 ∧ HALSTEAD_VOLUME <= 1262.0) -> defective
3. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS <= 8.0 ∧ HALSTEAD_EFFORT > 5332.0) -> defective
4. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS <= 8.0 ∧ LOC_COMMENTS > 16.0 ∧ NUMBER_OF_LINES <= 57.0 ∧ NUM_OPERANDS <= 28.0) -> defective
5. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS <= 8.0 ∧ LOC_COMMENTS > 16.0 ∧ NUMBER_OF_LINES <= 57.0 ∧ NUM_OPERANDS> > 28.0 ∧ NUMBER_OF_LINES <= 50.0) -> defective
6. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS > 8.0 ∧ HALSTEAD_EFFORT <= 8944.0) -> defective
7. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS > 8.0 ∧ HALSTEAD_EFFORT > 8944.0 ∧ NUM_OPERANDS > 46.0 ∧ NUMBER_OF_LINES <= 71.0) -> defective
8. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME <= 2049.0 ∧ NUM_OPERATORS <= 94.0 ∧ NUM_OPERATORS <= 72.0 ∧ LOC_COMMENTS > 8.0 ∧ HALSTEAD_EFFORT > 8944.0 ∧ NUM_OPERANDS > 46.0 ∧ NUMBER_OF_LINES > 71.0 ∧ LOC_EXECUTABLE <= 28.0) -> defective
9. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS <= 30.0 ∧ HALSTEAD_PROG_TIME > 2049.0 ∧ HALSTEAD_EFFORT <= 49109.0) -> defective

10. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES <= 100.0 ∧ LOC_COMMENTS > 30.0 ∧ LOC_COMMENTS <= 34.0 ∧ NUMBER_OF_LINES <= 92.0) -> defective
11. (LOC_EXECUTABLE <= 68.0 ∧ LOC_COMMENTS > 6.0 ∧ NUMBER_OF_LINES <= 104.0 ∧ NUMBER_OF_LINES > 100.0) -> defective
12. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ HALSTEAD_LENGTH <= 233.0) -> defective
13. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ LOC_TOTAL > 80.0) -> defective
14. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ LOC_COMMENTS <= 60.0 ∧ HALSTEAD_EFFORT <= 91115.0) -> defective
15. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ LOC_COMMENTS > 60.0 ∧ NUM_OPERANDS <= 261.0 ∧ HALSTEAD_LENGTH > 444.0) -> defective
16. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ LOC_COMMENTS > 60.0 ∧ NUM_OPERANDS > 261.0 ∧ HALSTEAD_LENGTH > 444.0) -> defective
17. (LOC_EXECUTABLE > 68.0 ∧ LOC_TOTAL <= 100.0 ∧ LOC_COMMENTS > 60.0 ∧ NUM_OPERANDS > 261.0 ∧ LOC_COMMENTS > 168.0 ∧ HALSTEAD_PROG_TIME <= 59812.0) -> defective.

Chapter 6

Managing Class Imbalance Problem

6.1 Class Imbalance

Managing the class imbalance issue of our dataset was one of the significant issues to consider. This is a reality that number of flawed modules in a dataset are in minority when contrasted with number of non-faulty dataset. Accordingly, when the datasets are prepared applying classifier with imbalanced dataset, the classifier will in general disregard the minor classes focusing on major classes. The classifiers will in general produce high predictive accuracy over the majority class, however poor predictive accuracy over the minority class. It has been seen that managing class imbalance issue extraordinarily improve the precision rate of prediction model whereas maintain Pf, balance and accuracy. Recall is measuring of defective software modules correctly predict as defective among the modules classified as defective. Recall is low if the number of defective modules predicted correctly is small in number or large number of defective modules is predicted as non-defective. So it is a huge assessment measure for basic framework where it is increasingly critical to accurately anticipate which software part is defective with the goal that all the more testing has been done so as to make the product item fault free. If a software module that is actually defective but because of poor recall rate predicted as non-defective would be less tested and therefore there is a chance that error may occur when software is deployed in real space. In this manner, recall is a fundamental assessment measurement for the critical real time software. We have also compared our result in order to endorse the efficiency of proposed approach.

Dealing with imbalanced datasets includes various strategies such as improving classification algorithms or balancing classes in the training dataset before providing the data as input to the data mining algorithm. We have considered three techniques used for balancing the classes. The main idea of sampling classes is to either increasing the samples of the minority class or decreasing the number of instances for both the class.

6.2 Techniques to sample dataset

6.2.1 Random Under Sampling

The aim of this technique is to balance the class distribution by randomly reducing majority class examples. When instances of two different classes are very close to each other, the instances of the majority class are eliminated to increase the space between two classes. This helps in classification process. Various experimental results show that random under sampling significantly improves classification performance in comparison to not using the any data sampling [59]. The advantages of random over sampling technique are it can help the runtime of the model and solve the memory problems by reducing the number of training data samples when training data set is enormous. But it can discard useful information about the data itself which could be necessary for building rule based classifiers such as Random Forests. Again, the sample chosen by random under sampling may be a biased sample. And it will not be an accurate representation of the population in that case.

6.2.2 Random Over Sampling

Just like random under sampling, random oversampling has been performed. It is a rationalized technique proposed to handle imbalanced data sets using exclusive safe-level based synthetic sample creation [60]. But in this case, taking any help from majority class, the instances corresponding to minority class are increased by replicating them up to a constant degree. Here the number of instances assigned to the majority class are not decreased. Unlike under sampling, this method leads to no information loss. In fact, it increases the likelihood of overfitting since it replicates the minority class events. A visual representation of these two sampling techniques is illustrated in Fig 6.1.

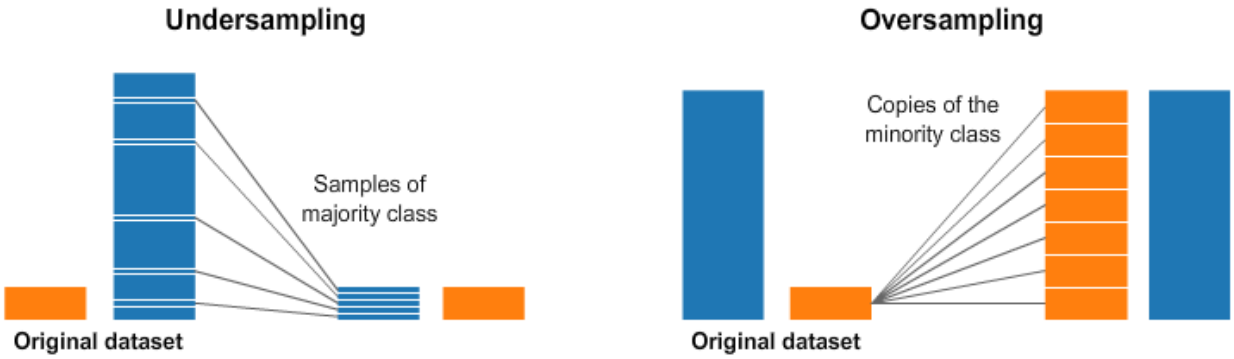

Fig. 6.1: Visual representation of sampling techniques

6.2.3 SMOTE

SMOTE stands for Synthetic Minority Over-Sampling Technique. This over sampling method creates example which are synthetic. It does not over sample by replacements. The minority class is over sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any or all of the k – minority class nearest neighbors. Depending upon the amount of over sampling required, neighbors from the k nearest neighbors are randomly chosen. It synthesizes the instances randomly which are minor. Its selected nearest neighbors ignores the close majority instances [61]. The heart of SMOTE is the construction of minority classes. The intuition behind the construction algorithm is simple. It has been shown that to a machine learning algorithm, the newly constructed instances are not exact copies and thus it softens the decision boundary and thereby helping the algorithm to approximate the hypothesis more accurately. The advantages of SMOTE are it alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances. It does not cause the loss of information. It's simple to implement and interpret. But while generating synthetic examples, SMOTE does not take into consideration neighboring examples can be from other classes. The overlapping of classes can be increased through this and the additional noise can be introduced.

We have applied all these three techniques and the comparison among them is shown in **Error! Reference source not found.**.

TABLE 6.1: Comparison among three sampling techniques

Dataset	Defective detection without resampling	Over Sampling	Under Sampling	SMOTE
CM1	9%	47%	63%	100%
KC3	16%	76%	76%	85%
MW1	27%	46%	46%	100%
PC2	0%	50%	67%	89%
PC4	25%	84%	87%	94%

Here we have applied Logistic Regression in all of the five datasets and the split ratio is 60:40. The results represents that SMOTE works better when classifying the defective class than other techniques because by using SMOTE recall is increased at the cost of precision. It can be seen that the sampling techniques used by the under sampling of the major class and over sampling of the minor class does not affect the classification of major classes as all classifiers can classify the major classes [61]. However, the effect of these three sampling techniques can be observed in classification of minor classes. The recall rate of the minor classes is found to increase. It is visible that before sampling the dataset the recall rate of the classifiers was below 30% but when the sampling techniques are implemented the recall rate increased and the classifiers can detect the

defects more accurately. For example, in the CM1 dataset, before sampling the classifier could detect 9% of the total defects. But after sampling the dataset with three of the sampling techniques it showed increase in the result and when applied SMOTE the classifier can detect defects in the dataset by 100%. This is certainly very positive improvement for detecting faults in software.

So the results can show that by addressing the issue of class imbalance problem the recall rate of defect predictor has been greatly improves as class imbalance significantly affect the performance of defect predictors. It can also be observed that the probability of prediction would be affected a bit when using balanced datasets. So the balanced dataset when applied different classifiers and evaluation metrics can perform better in terms of detecting faults in software.

Chapter 7

Tool Implementation

7.1 Tool Description

We have built a Graphical User Interface (GUI) for our model. This interface has been built using PyQT framework. It consists of five classifiers (Decision Tree, Random Forest, Naïve Bayes, Logistic Regression, Neural Network) and five feature selection techniques (Information Gain, Relief, Chi Square, Chi square Test of Independence, Feature Importance.

We can manually select the dataset by clicking “Input Dataset”. After selecting a dataset, a classifier and/or a feature selection technique, we can calculate the Accuracy, Confusion Matrix and Area Under Curve (AUC). We can also view the ROC curve for the corresponding classifier and technique.

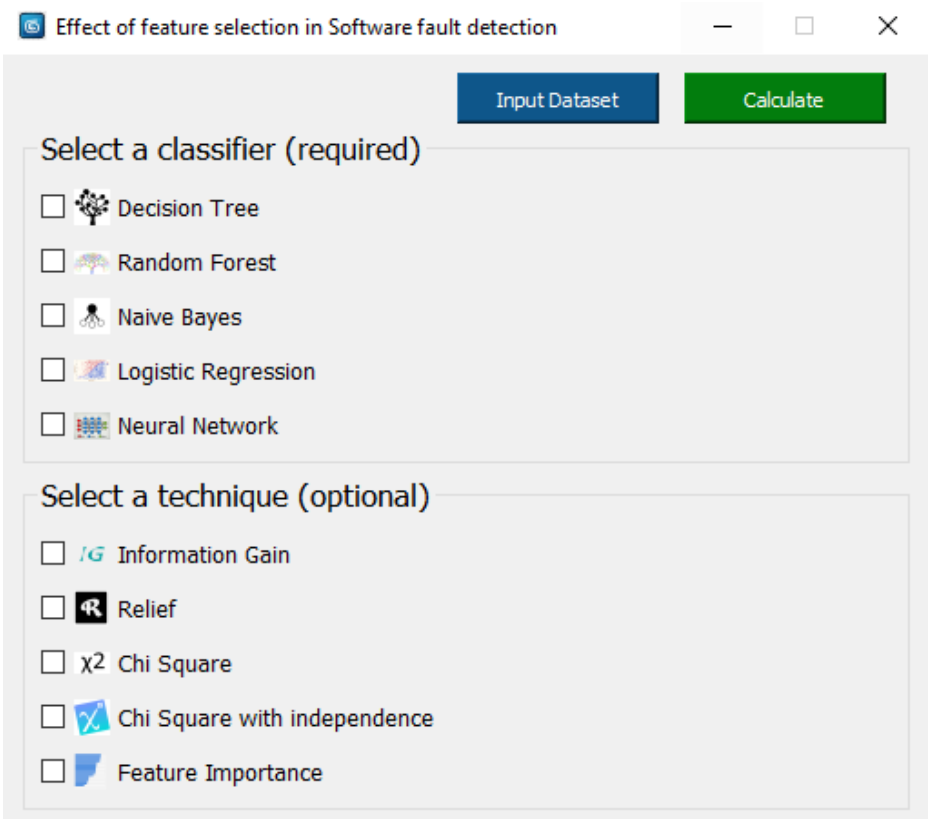

Fig. 7.1: GUI of "Effect of feature selection in software fault detection

Now step by step, we will see the demonstration and the functionality of our GUI.

Step 1 (Selecting a classifier and a technique)

For example, we have selected the Random Forest classifier and Feature Importance. We can select the other options also. Maximum 25 combinations are possible to execute for this scenario.

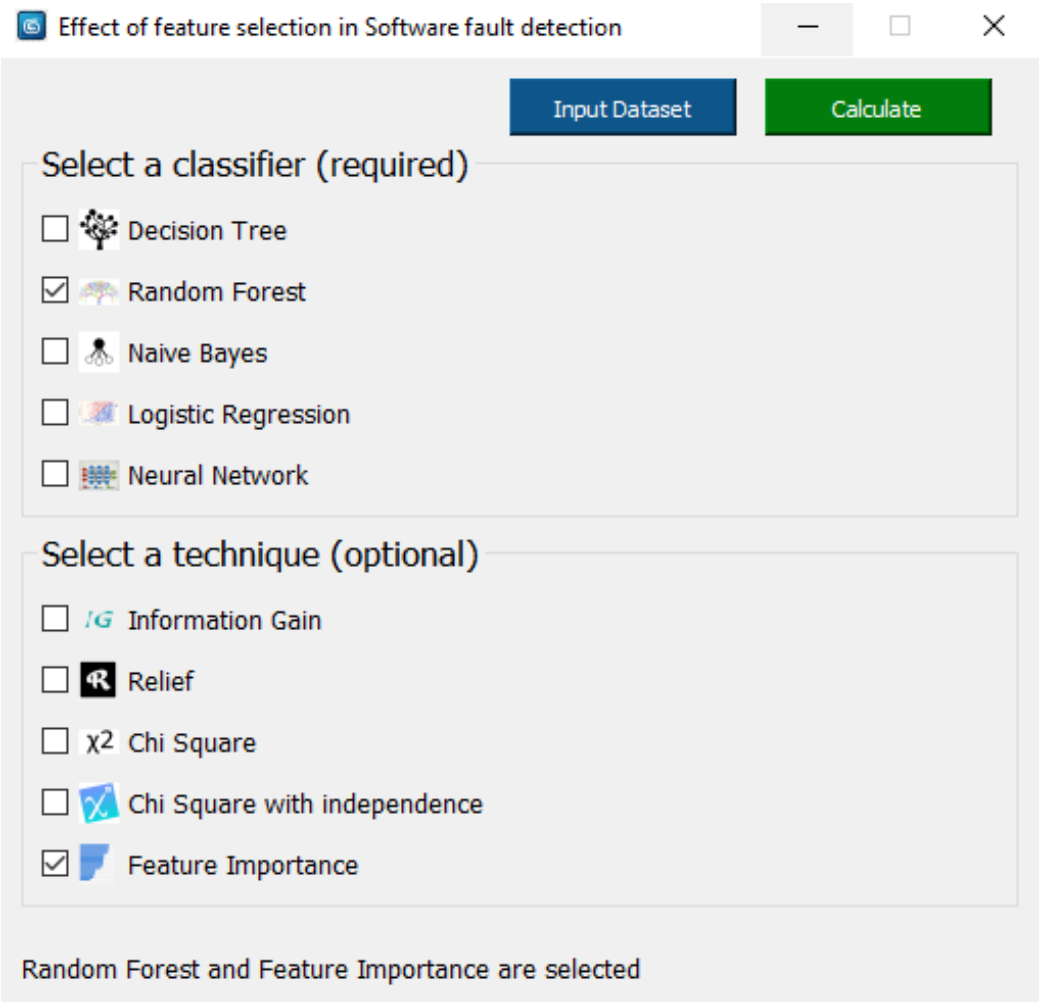

Fig. 7.2: Selecting a classifier and a technique

Step 2 (Selecting a dataset and see the dataset overview)

We can manually select any of the dataset. We have selected MW1.xls for this scenario. After selection the dataset a new window called Dataset Overview will pop up and from this window we can see the dataset's basic properties. For graphical view of the dataset we should hit the Graphical View button.

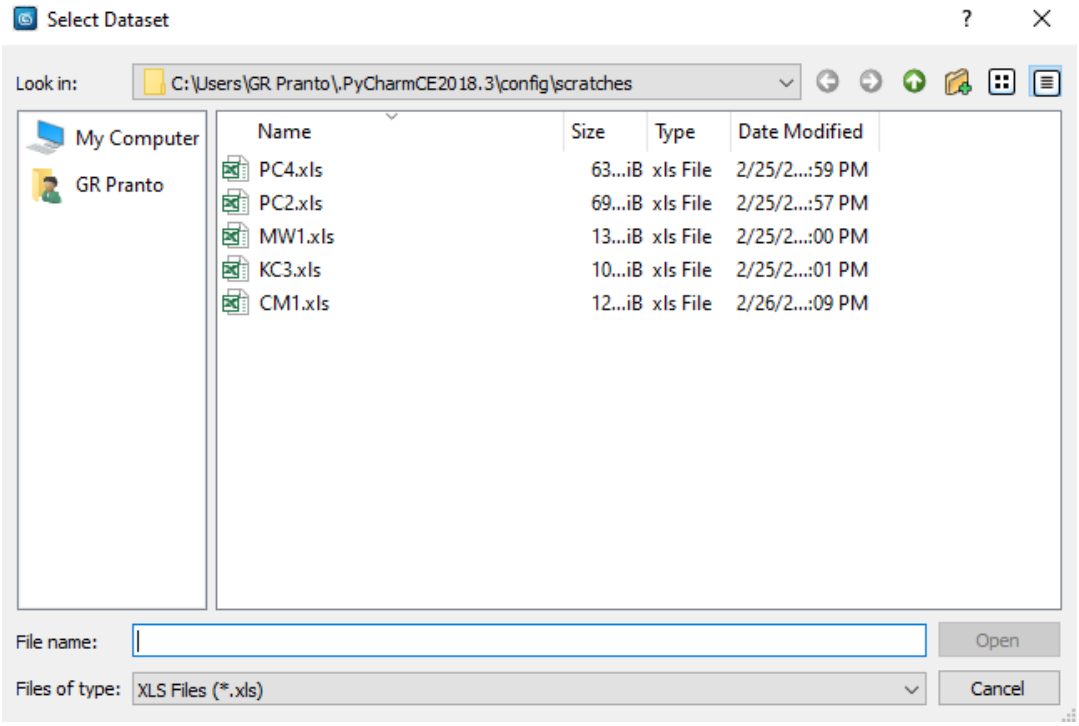

Fig. 7.3: Selecting a dataset

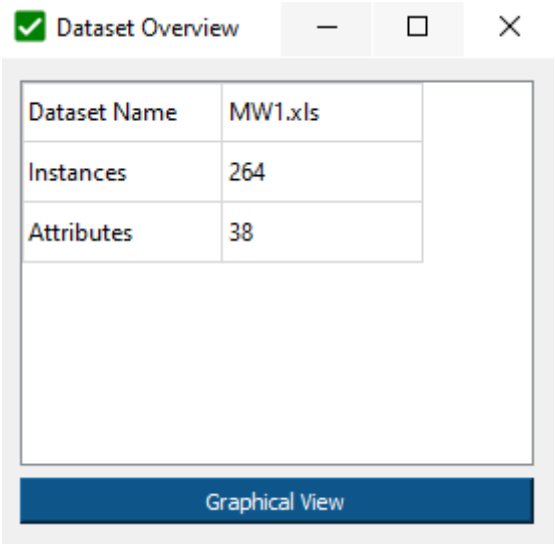

Fig. 7.4: Dataset Statistics

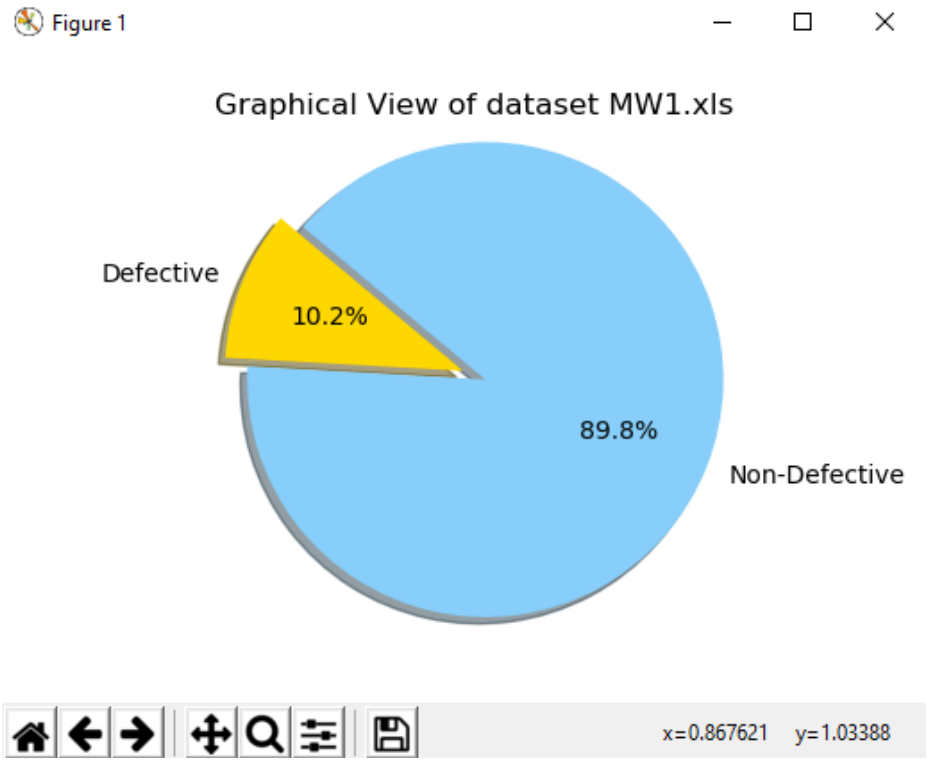

Fig.7.5: Graphical representation of the dataset

Step 3 (Result)

After doing all the previous steps, it's time for seeing the result. Result window only pop up if we have done the previous steps correctly, otherwise it will not. In the result section we will see the Accuracy, Confusion Matrix, Area Under Curve (AUC). For viewing the ROC curve we should hit the View Roc Curve button.

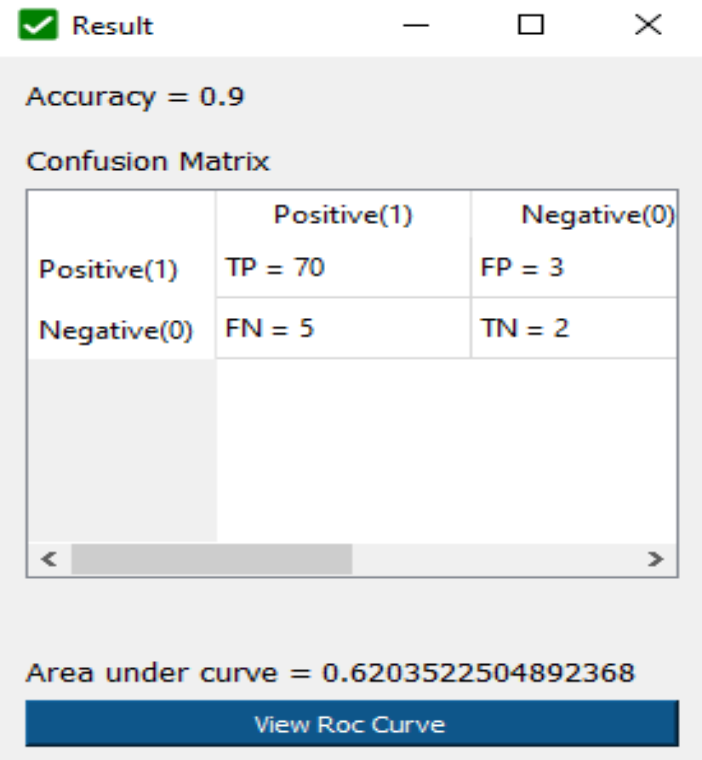

Fig. 7.6: Representation of accuracy, confusion matrix and AUC

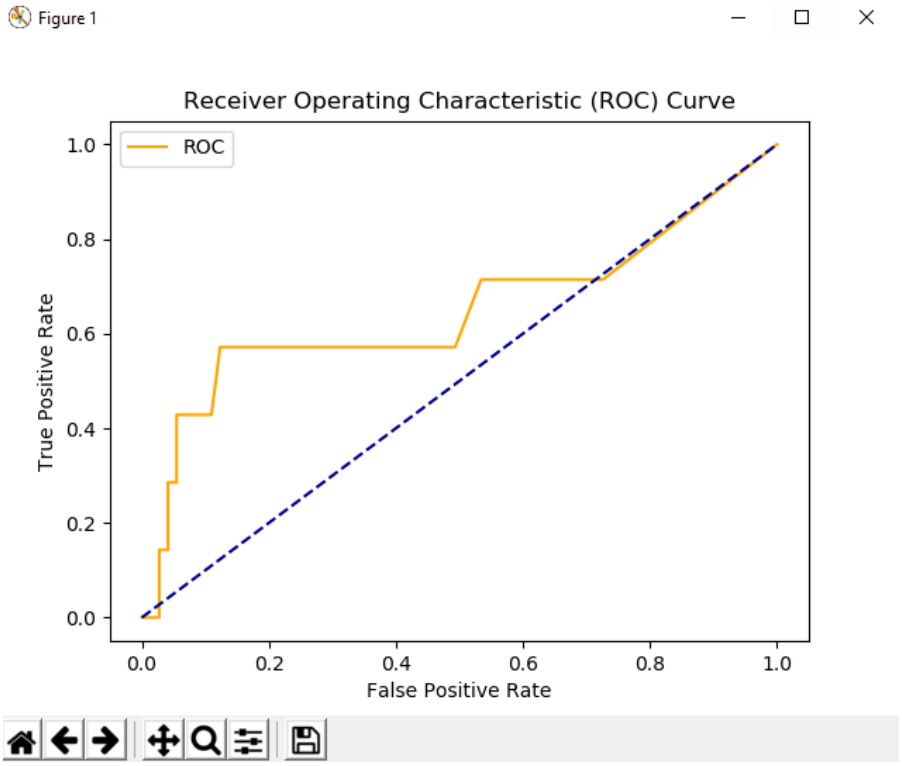

Fig. 7.7: Display of ROC curve

Chapter 8

Conclusion

8.1 Summary

Software fault prediction improves software quality, reliability and prevents from future losses. From one of our experiments we have found that the models applied for calculating accuracies can detect the probability of a software's being faulty before it actually causes some fatal issues. The main importance was given on detecting the defective classes so all the experiments were done accordingly. The effects of the attributes on different dataset shows us that by increasing or decreasing those attributes' values the defective or non-defective classes can be distinguished easily. The improvement or the modification of these classifiers and an extensive comparison among them can definitely bring better results. Moreover, including more metrics in the learning step is one conceivable way to deal with the increment of the accuracy and precision.

Another significant experiment result shows that proper selection of relevant features in a large dataset can immensely improve the performance of classifiers and significantly reduces the training time. Among the various feature techniques, our experiment shows the effect of feature selection of only five approaches. Five search-based classifiers are applied here for our experiments. The experimental results reveal that after feature selection the performance of the classifiers are almost similar to that of without feature selection. Experiments have been conducted by considering both 10 and 20 features from the datasets. The variation among the obtained results are not significant. Such result implies that feature selection approaches do not compromise the performance of the classifiers while taking less time and resource during the experiments.

The improvement in the classifiers defect detection model after applying dataset sampling techniques is another significant result for our project. The imbalanced ratio of the class makes the classifier to give poor classification result and most of the time this fails to detect the most important class which needs to be detected first. That is why balancing of the classification is important and with the three techniques applied in our dataset we can conclude that the classifier worked better and more effectively.

8.2 Future Work

Future works can be done on the other datasets for mining important results from them. Combination of other learning algorithms or other hybrid algorithms can be built to apply in these datasets. This can show us whether the other learning approaches can work more effectively on the dataset. It is mentioned earlier that only a subset of feature selection techniques has been considered in this work. For a better comprehension of the proposed approach, our future plan is to consider both filter and wrapper based feature selection techniques in our experiment. The rough set can be applied on the dataset to identify the relevant features. For the class imbalance problem, the other techniques are thought of applying in the dataset and with the comparison and analysis of the experiment, more important facts can be drawn from the result

Bibliography

[1] M. Kakkar and S. Jain, "Feature selection in software defect prediction: A comparative study," in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, 2016, pp. 658–663.

[2] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 237–257, Feb. 2013.

[3] A. H. Yousef, "Extracting software static defect models using data mining," *Ain Shams Eng. J.*, vol. 6, no. 1, pp. 133–144, Mar. 2015.

[4] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software Defect Prediction using Feature Selection and Random Forest Algorithm," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, 2017, pp. 252–257.

[5] E. A. Felix and S. P. Lee, "Integrated Approach to Software Defect Prediction," *IEEE Access*, vol. 5, pp. 21524–21547, 2017.

[6] A. Nugroho, M. R. V. Chaudron, and E. Arisholm, "Assessing UML design metrics for predicting fault-prone classes in a Java system," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 21–30.

[7] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Software defect prediction using static code metrics underestimates defect-proneness," in *Proceedings of the International Joint Conference on Neural Networks*, 2010.

[8] Z. Li and M. Reformat, "A practical method for the software fault-prediction," in *2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007*, 2007, pp. 659–666.

[9] P. Singh and S. Verma, "An investigation of the effect of discretization on defect prediction using static measures," in *ACT 2009 - International Conference on Advances in Computing, Control and Telecommunication Technologies*, 2009, pp. 837–839.

[10] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *Proceedings - 9th International Conference on Machine Learning and Applications, ICMLA 2010*, 2010, pp.

135–140.

[11] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.

[12] J. Singh and S. Sharma, “Fault detection technique for test cases in software engineering,” *Int. J. Eng. Technol.*, vol. 7, no. 1, p. 53, Jan. 2018.

[13] X. Chen, Y. Shen, Z. Cui, and X. Ju, “Applying Feature Selection to Software Defect Prediction Using Multi-objective Optimization,” in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, 2017, pp. 54–59.

[14] L. Rokach, “Ensemble-based classifiers,” *Artif. Intell. Rev.*, vol. 33, no. 1–2, pp. 1–39, Feb. 2010.

[15] A. K. Jakhar and K. Rajnish, “Software Fault Prediction with Data Mining Techniques by Using Feature Selection Based Models,” *Int. J. Electr. Eng. Informatics*, vol. 10, no. 3, pp. 447–465, Sep. 2018.

[16] R. S. Wahono and N. S. Herman, “Genetic Feature Selection for Software Defect Prediction,” *Adv. Sci. Lett.*, vol. 20, no. 1, pp. 239–244, Jan. 2014.

[17] Y. Chen, X. H. Shen, P. Du, and B. Ge, “Research on software defect prediction based on data mining,” in *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010*, 2010, vol. 1, pp. 563–567.

[18] “(PDF) Enhance Rule Based Detection for Software Fault Prone Modules.” [Online]. Available: https://www.researchgate.net/publication/252063255_Enhance_Rule_Based_Detection_for_Software_Fault_Prone_Modules. [Accessed: 20-Nov-2019].

[19] I. H. Laradji, M. Alshayeb, and L. Ghouti, “Software defect prediction using ensemble learning on selected features,” *Inf. Softw. Technol.*, vol. 58, pp. 388–402, Feb. 2015.

[20] R. Li and S. Wang, “An Empirical Study for Software Fault-Proneness Prediction with Ensemble Learning Models on Imbalanced Data Sets,” *J. Softw.*, vol. 9, no. 3, Mar. 2014.

[21] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A General Software Defect-Proneness Prediction Framework,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, May 2011.

[22] Y. Jiang, M. Li, and Z.-H. Zhou, “Software Defect Detection with Rocus,” *J. Comput. Sci. Technol.*, vol. 26, no. 2, pp. 328–342, Mar. 2011.

[23] A. Singh and R. Singh, “Assuring software quality using data mining methodology: A literature study,” in *Proceedings of the 2013 International Conference on Information Systems and Computer Networks, ISCON 2013*, 2013, pp. 108–113.

[24] C. He, J. Xing, R. Zhu, J. Li, Q. Yang, and L. Xie, “A new model for software defect prediction using Particle Swarm Optimization and support vector machine,” in *2013 25th Chinese Control and Decision Conference, CCDC 2013*, 2013, pp. 4106–4110.

[25] A. D. Oral and A. B. Bener, “Defect prediction for embedded software,” in *22nd International Symposium on Computer and Information Sciences, ISCIS 2007 - Proceedings*, 2007, pp. 346–351.

[26] L. Jia, “A Hybrid Feature Selection Method for Software Defect Prediction,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 394, no. 3, p. 032035, Aug. 2018.

[27] Q. Yu, S. Jiang, R. Wang, and H. Wang, “A feature selection approach based on a similarity measure for software defect prediction,” *Front. Inf. Technol. Electron. Eng.*, vol. 18, no. 11, pp. 1744–1753, Nov. 2017.

[28] Z. Xu, J. Xuan, J. Liu, and X. Cui, “MICHAC: Defect Prediction via Feature Selection Based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 370–381.

[29] M. Anbu and G. S. Anandha Mala, “Feature selection using firefly algorithm in software defect prediction,” *Cluster Comput.*, pp. 1–10, Oct. 2017.

[30] T. M. Khoshgoftaar, K. Gao, and N. Seliya, “Attribute selection and imbalanced data: Problems in software defect prediction,” in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2010, vol. 1, pp. 137–144.

[31] L. Pelayo and S. Dick, “Applying novel resampling strategies to software defect prediction,” in *Annual Conference of the North American Fuzzy Information Processing Society - NAFIPS*, 2007, pp. 69–72.

[32] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” in *IEEE Transactions on Software Engineering*, 2008, vol. 34, no. 4, pp. 485–496.

[33] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data Quality : Some Comments on the NASA Software Defect Data Sets,” vol. 2010, no. 9, pp. 1–13, 2013.

[34] D. Gray, D. Bowes, N. Davey, Yi Sun, and B. Christianson, “The misuse of the NASA Metrics Data Program data sets for automated software defect prediction,” in *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 2011, pp. 96–103.

[35] R. G. Ramani, S. V. Kumar, and S. G. Jacob, “Predicting fault-prone software modules using feature selection and classification through data mining algorithms,” in *2012 IEEE International Conference on Computational Intelligence and Computing Research*, 2012, pp. 1–4.

[36] Q. Taylor, C. G. Carrier, and C. D. Knutson, “Applications of data mining in software engineering,” *Int. J. Data Anal. Tech. Strateg.*, vol. 2, no. 3, p. 243, 2010.

[37] J. R. Quinlan and J. R., “Induction of Decision Trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[38] G. I. Webb, E. Keogh, R. Miikkulainen, R. Miikkulainen, and M. Sebag, “Naïve Bayes,” in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2011, pp. 713–714.

[39] N. Deng, Y. Tian, and C. Zhang, *Support vector machines : optimization based theory, algorithms, and extensions*. CRC Press, Taylor & Francis Group, 2013.

[40] K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support

vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, May 2008.

[41] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Comput.*, vol. 22, no. 2, pp. 387–408, Jan. 2018.

[42] P. Rao and J. Manikandan, "Design and evaluation of logistic regression model for pattern recognition systems," in *2016 IEEE Annual India Conference, INDICON 2016*, 2017.

[43] A. S. More and D. P. Rana, "Review of random forest classification techniques to resolve data imbalance," in *Proceedings - 1st International Conference on Intelligent Systems and Information Management, ICISIM 2017*, 2017, vol. 2017-January, pp. 72–78.

[44] M. Mishra and M. Srivastava, "A view of Artificial Neural Network," in *2014 International Conference on Advances in Engineering and Technology Research, ICAETR 2014*, 2014.

[45] T. F. Crack, "A Note on Karl Pearson's 1900 Chi-Squared Test: Two Derivations of the Asymptotic Distribution, and Uses in Goodness of Fit and Contingency Tests of Independence, and a Comparison with the Exact Sample Variance Chi-Square Result," *SSRN Electron. J.*, Nov. 2018.

[46] R. Singhal and R. Rana, "Chi-square test and its application in hypothesis testing," *J. Pract. Cardiovasc. Sci.*, vol. 1, no. 1, p. 69, 2015.

[47] M. L. McHugh, "The Chi-square test of independence," *Biochem. Medica*, pp. 143–149, 2013.

[48] K. Kira and L. A. Rendell, "A practical approach to feature selection," *Proc. ninth Int. Work. Mach. Learn.*, pp. 249–256, 1992.

[49] R. Sathyaraj and S. Prabu, "An Approach for Software Fault Prediction to Measure the Quality of Different Prediction Methodologies using Software Metrics," *Indian J. Sci. Technol.*, vol. 8, no. 35, Dec. 2015.

[50] G. J. Pai and J. Bechta Dugan, "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 675–686, Oct. 2007.

[51] S. S. Rathore and S. Kumar, "Predicting Number of Faults in Software System using Genetic Programming," *Procedia Comput. Sci.*, vol. 62, pp. 303–311, Jan. 2015.

[52] G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," *Vietnam J. Comput. Sci.*, vol. 1, no. 2, pp. 79–95, May 2014.

[53] S. Agarwal and D. Tomar, "A Feature Selection Based Model for Software Defect Prediction," *Int. J. Adv. Sci. Technol.*, vol. 65, pp. 39–58, 2014.

[54] K. Fawagreh, M. M. Gaber, and E. Elyan, "Random forests: from early developments to recent advancements," *Syst. Sci. Control Eng.*, vol. 2, no. 1, pp. 602–609, Dec. 2014.

[55] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll, "An Introduction to Logistic Regression Analysis and Reporting," *J. Educ. Res.*, vol. 96, no. 1, pp. 3–14, Sep. 2002.

[56] M. Mishra and M. Srivastava, "A view of Artificial Neural Network," in *2014 International Conference on Advances in Engineering & Technology Research (ICAETR -*

2014), 2014, pp. 1–3.

[57] M. Baojun, K. Dejaeger, J. Vanthienen, and B. Baesens, "Software Defect Prediction Based on Association Rule Classification," *SSRN Electron. J.*, 2011.

[58] T. Watanabe, A. Monden, Y. Kamei, and S. Morisaki, "Identifying recurring association rules in software defect prediction," in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, 2016, pp. 1–6.

[59] J. Prusa, T. M. Khoshgoftaar, D. J. DIttman, and A. Napolitano, "Using Random Undersampling to Alleviate Class Imbalance on Tweet Sentiment Data," in *Proceedings - 2015 IEEE 16th International Conference on Information Reuse and Integration, IRI 2015*, 2015, pp. 197–202.

[60] S. S. Patil and S. P. Sonavane, "Improved classification of large imbalanced data sets using rationalized technique: Updated Class Purity Maximization Over_Sampling Technique (UCPMOT)," *J. Big Data*, vol. 4, no. 1, p. 49, Dec. 2017.

[61] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5476 LNAI, pp. 475–482.

Appendix A

List of Publications

International Conference Papers

1. Shamse Tasnim Cynthia, Md. Golam Rasul and Shamim Ripon, *Effect of Feature Selection in Software Fault Detection*, 13th Multi- disciplinary International Conference on Artificial Intelligence, November 17-19, 2019. Kuala Lumpur, MALAYSIA, 2019. Springer International Publishing, LNAI 11909. https://doi.org/10.1007/978-3-030-33709-4_5.

   Includes Chapter 1, 2, 3, 4, 5(5.1, 5,4).

2. Shamse Tasnim Cynthia and Shamim H Ripon, *Predicting and Classifying Software Faults: A Data Mining Approach*, 7th International Conference on Computer and Communications Management (ICCCM 2019) Bangkok, Thailand, July 27-29, 2019. (ACM Indexed). https://doi.org/10.1145/3348445.3348453.

   Includes Chapter 1, 2, 3, 4, 5 (5.2, 5.3).