SOFTWARE FAULT PREDICTION USING MIXTURE OF EXPERTS

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of degree
of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

by

AMAN OMER

(17535003)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE - 247 667 (INDIA)

May, 2019

CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled **"Software Fault Prediction using Mixture of Experts"** towards the partial fulfilment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering** submitted in the Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Uttrakhand (India) is an authentic record of my own work carried out during the period from July 2018 to May 2019 under the guidance of **Dr. Sandeep Kumar**, Associate Professor, Department of Computer Science and Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other institute.

Date: **(Aman Omer)**

Place: Roorkee **Enroll. No- 17535003**

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: **(Dr. Sandeep Kumar)**

Place: Roorkee Dissertation Supervisor

ACKNOWLEDGEMENTS

I would never have been able to complete my dissertation without the guidance of my supervisor, help from friends, and support from my family and loved ones.

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor Dr. Sandeep Kumar, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, for his invaluable guidance, and encouragement and for sharing his broad knowledge. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. He has been very generous in providing the necessary resources to carry out my research. He is an inspiring teacher, a great adviser, and most importantly a nice person. I would also like to express my sincere appreciation and gratitude towards Dr. Santosh Singh Rathore for his encouragement, consistent support and invaluable suggestions at the time I needed the most.

I am also grateful to the Dept. of Computer Science and Engineering, IIT Roorkee for providing valuable resources to aid my research. Finally, hearty thanks to my parents and siblings, who encouraged me in good times, and motivated me in the bad times, without which this dissertation would not have been possible.

AMAN OMER

ABSTRACT

With increasing applications of software, quality assurance becomes an important phase of software life cycle which makes Software Fault Prediction an essential research topic. Software fault prediction uses existing software metrics, faulty and non-faulty data to predict fault-prone modules. Learning algorithm used for classifying software module plays a vital role hence it also makes the process dependent and vulnerable on single algorithm. To overcome this more than one learning algorithm is being used. This collection of models is called as ensemble. In recent years, many studies have explored different ensemble methods for software fault prediction and it results in significant improvement over individual model. Input space division algorithm for these ensemble techniques are data independent, which certainly affects the model as spatial information could be lost. Training model would perform better if data will be separated depending on the input data. Mixture of Experts (ME) ensemble is a technique which uses soft splitting of the data to train base learners, had been used in various fields such as speech recognition and object detection.

The objective of this study is evaluate the performance of ME with different base learners for Software Fault Prediction. 41 publicly available software project datasets from NASA PROMISE and MDP repositories along with Eclipse project data, are used for simulation. ME with decision tree and multi-layer perceptron as base learners are evaluated along with using Gaussian Mixture Model, an unsupervised technique as a gating function. Performance is measured in terms of accuracy, f1-score, precision and recall. Wilcoxon's statistical test is also performed to evaluate the significant difference of ME. To compare the performance bagging is implemented and results are also compared with individual base model. Results show that while using decision trees as base learners, ME showed improvement in performance and it also performs as good as bagging. When multi-layer perceptron is used as base learner in ME, on average, it shows 7% and 6% improvement in accuracy from individual and bagging model, respectively. Wilcoxon statistical test indicates the significant difference between ME and bagging model for both base learning algorithms.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Software Fault Prediction is the mechanism to predict whether in a software the modules are going to be faulty or non-faulty, before even applying the testing mechanism. In other words, Fault Prediction in Software is a way to find the fault proneness of the software module during the earlier stages of development life cycle process [1]. This prediction has a great role to play in improving the quality of the software as well as reducing the time and efforts needed in the testing phase of the development life cycle of the software. This chapter describes the basic terminologies and brief about Software Fault Prediction mechanism.

1.1 GENERAL CONCEPTS

1.1.1 Software Fault Prediction

The requirement of high quality and maintainable software have increased with the growing complexity and dependency of the software. Software fault prediction is a method for improving the software quality [5]. Fault prediction helps in reducing the efforts for maintenance by giving the prediction of buggy modules beforehand. There are several software metrics proposed in literature for measuring the performance of prediction models [25]. Software fault prediction process is very important in software development and the accurate prediction of faults and the recognition of the area which is most prone to fault occurrence can directly help in reducing the development cost, testing efforts and improves the overall quality of the software. Software a fault is the main concern to be dealt with that affects overall software reliability and correctness. The accurate predictions of the faults empower the software developers to evaluate the overall reliability of the software during the development process. Moreover, the prediction of the accurate location of faults can boost the testing process and allows the developers to focus on the critical modules that may account for the maximum number of faults. Software fault prediction is the prediction whether a software module is faulty or not by using the previous data and some learning models. Thus software fault prediction makes use of the data of previous versions of the software to find out the probability of faults in the upcoming versions of that software based on some characteristics known as metrics, by applying some learning model [2].

As the complexity of the software system are growing continuously, the rate of software failure is also increasing resulting in undesirable behaviour of the system along with poor services and sometimes complete outages. Dealing with software faults is very important task. Faulty modules present in software deteriorates the quality of the software and also increases the overall cost of the software system [31]. Several techniques and processes for providing a high quality software product are included in software quality engineering. Employment of data mining techniques on the software metrics collected during development process, for identifying the potential fault-prone program modules, proved to be an efficient method for improving software quality [11].

Advantages and needs of software fault prediction are listed as follows [1]:

- Delivering a highly dependable system.
- Predicting buggy modules beforehand helps in improving the testing process.
- Improving quality by improving test process.

Software fault prediction becomes important for some software which need much more care regarding testing and cannot afford any type of faults (e.g. medical science, banking, astronomy and finance etc.).

1.1.2 Mixture of Experts

Experts in this ML model is referred to an individual learner model which is expert in its particular section of input space. This model was originally proposed by Jacobs et al. [3] in 1991 as "Adaptive mixture of local experts" which suggested the idea of dividing the input space and use different learners for different input space. ME model relies on the principle of divide and conquer, having three major components [4]:

i Experts which can be either classifiers or regression functions.

ii A gate that provides soft boundaries for input space and introduces those regions where the individual expert results are dependable.

iii A probabilistic model to incorporate the experts and the gate.

Mixture of Experts architecture can be used for solving classification and regression problems of real world applications with some modifications in the architecture these changes are discussed below.

Classification with ME

In the ME architecture, a gate and a set of experts collaborate with each other to break a nonlinear supervised learning problem into smaller linear problems, by separating the input space into a nested set of regions. Whole input space is softly split by gate, and the experts learn the simple parameterized surfaces in these partitions of the regions. There are several methods using which ME model can learn the parameters of both gate and experts surface [4].

Regression with ME

Mixture of experts can also be used to solve the complex regression problem by assigning weights to the result of various regression learners. In the past 20 years, there are various statistical and experimental analyses which had been done on Mixture of Experts model, and numerous amount of researches have been done in the fusion, regression and classification area which shows the suitability of ME in those fields. ME models have shown a better results and found useful in combination with many current classification and regression algorithms because of its flexible and modular structure [4].

1.2 ORGANIZATION OF THESIS

This report is divided into 5 chapters. First chapter concluded the preliminaries and basic concept knowledge that will be needed for understanding this thesis. Second chapter is literature review which includes papers from software fault prediction domain. This chapter highlights the research gaps found in literature and presents the tabular representation for the same. Third chapter contains details about proposed architecture. Fourth chapter shows the experimental result of existing techniques as well as of proposed architecture. Fifth chapter concludes the entire work including the analysis of results and future work that can be done.

CHAPTER 2 LITERATURE REVIEW

Several works have been done till now in predicting whether the software module is faulty or non-faulty, using different classifiers on different datasets. The performances vary on using different classifiers on different set of datasets

2.1 BACKGROUND STUDY

2.1.1 Data Preprocessing

Unavailability of training data and class imbalance problem are most common in dataset of software projects [3] [1] [21] [16]. There are several technique and their combination used in recent past year to tackle these problem. Study [3] presents an iterative approach to overcome the problem of unavailability of data. [3] uses Fuzzy Inference System (FIS) at initial stage of software development when data is unlabeled. Prediction for later versions of software project will be made using FIS model and Artificial Neural Network (ANN). [23], [14] investigates and explore various as well as proposed class sampling techniques in software fault prediction.

2.1.2 Binary class classification

[5] [9] and [5] studies explore the scope of semi-supervised and unsupervised learning techniques in prediction of software faults and suggested that their approach's applicable depending on the software project metric and performance measure. None of the model is generalized for every software project fault data presented publicly. [6] investigates different ML models on different metrics and suggested that multi-layer perceptron results better for all metrics.

2.1.3 Ensemble classifiers

Review papers [1] [28] [29] suggest that ensemble method are improves the performance and produce a reliable prediction framework for software fault prediction. [6] proposes a unique approach of using different set of metrics on different type of base learning algorithm. A conclusion from studies [6], [7], [17], [21] and [11] is that pre-processing data for learning an ensemble model is very important as these studies points out the difference in performance of ensemble approach on using pre-processing techniques.

2.2 TABULAR COMPARISON AND RESEARCH GAPS

This section contains the summary of various researches performed in recent years.

Study	Key Points	Methodology	Advantages	Disadvantages
Data pre-processing				
[7]	A iterative prediction model that begins with no data	ANFIS (Proposed) a combination of ANN and FIS	Also implemented proposed methodology as a tool	Expert is need to gather initial fault information
[8]	A novel active semi-supervised method	DT, LR, NB, CoForest	Proposed approach shows the effective results	Empirical study not exhaustive. No pre-processing technique used
[13]	Investigate the significance of data sampling SFP	DT, 3-layer NN, SVM, RF, KNN	AUC was not influenced with sampling but other metrics shows better results	No comparison with existing studies
[14]	Explores various class imbalance learning methods for SFP	NB RF, AdaBoost	Tabular representation of Optimal Parameters for imbalance learning methods	Validation through more case studies is required
[15]	Proposes to use number of faults to oversample minority class.	NB, Bayes Network, K-NN (k=1,5)	Presented a novel approach for handling class imbalance problem	For acceptance and generality of approach, more studies is required
Binary class classification				
[25]	Semi-supervised learning based on label propagation	FTF, ROCUS, LDS, CMN, GSKLP (proposed)	GSKLP benefits with LS sampling to improve results	Datasets of different domains required to validate proposed approach

Table 1 Tabular representation of past researches

Study	Key Points	Methodology	Advantages	Disadvantages
[9]	Unsupervised Learning using Scaled Dirichlet Distribution	Clustering	Suggests that clustering algorithm needs to be explored for SFP	Conclusions are based on synthetic data
[5]	Connectivity based Unsupervised Classification	RF, LR, SC, LMT, NB	Comparative study is thorough	Pre-processing step is not clear
[27]	Evaluate different ML models on different metrics	LR, NB, MLP	MLP shows better result with all metrics	Pre-processing step is not clear
[10]	Comprehensive evaluation of Bayesian Network (BN) Classifiers	15 different NB classifiers	Augmented NB classifiers and RF produces better results	Proposed H-measure needs to be evaluated on more case studies
[11]	Evaluate high-performance fault predictors	SVM, Probabilistic Neural Network (PNN)	PNN provided best performance for large datasets	Comparative analysis is not complete
[12]	Attempts to improve performance with existing techniques	RF, MLP, NB	Feature selected using BA increases accuracy of ensemble methods	Effect of BA on various ensemble methods needs to be explored
		Ensemble methods		
[13]	Prediction model on multi-metric and multi-type learning models	DT, MLP, NB	New direction for ensemble classifiers in SFP	More metric sets are available on which approach was not tested

Table 1 continued

Study	Key Points	Methodology	Advantages	Disadvantages
[14]	Examine the effects of FS on ELA	Bagging and AdaBoost with DT	Study conclude that FS and DS affects performance positively	Parameters for ELA (i.e., number of predictors) are not discussed
[15]	Just-In-Time Cross-Project prediction model	Voting, Bagging and Joining of traditional models	Encounters several research questions with sufficient proof	Selection of base learning algorithm is not clear
[16]	Investigate algorithms to overcome lack of training data	7 composite algorithm to ensemble traditional methods	Results showed that CODEP of LR effectively handle data unavailability	More empirical validations needed for generality.
[17]	Proposes a two-stage three-way decision based classifier	RF, NB	Results show the efficiency of proposed model	Other ensemble technique such as boosting, stacking, etc. should be used for validation
[29]	Ensemble model which considers class imbalance problem	RF	Proposed a novel approach of ensemble oversampled methods	Evaluation is performed only using RF, hence results cannot be generalized
[18]	Proposed a clustering ensemble framework	K-Means, EM, Particle Swarn Optimization	PSO with Manhattan Similarity measure performs better	Evaluation have been done on only one measure
[21]	New approach to select the best combination of features	SVM, BP-NN, GMCRF (proposed)	Proposed framework shows reliable results with low error rates	Comparative study is limited to few models
[19]	Combine multiple kernel and ensemble learning	SVM, AdaBoost, RF, MEKL (proposed)	Proposed ensemble method (MEKL) produced recall greater in most cases	Under sampling leads to information loss

Table 1 continued

Overall research gaps found during literature survey related to fault prediction in software module are mentioned below. Some of research gaps mentioned were found in those studies.

- In most of the studies, there was a lack in number of software projects used to evaluate to the performance of proposed algorithm, which is a necessary requirement for generalizing the results of proposed method. Also limited performance measure were used for the study.
- Absence of statistical tests and comparative study with past researches puts a question mark on validity of results. Different studies use different performance measure therefore comparing them becomes inconvenient. Also the parameters of proposed approach which were set for generation of results, are not clearly mentioned that leads confusion while following the approach.
- Data splitting technique for ensemble methods used in studies for SFP is not data dependent which is a research gap for future work. Mixture of Experts is a type of ensemble method which uses data dependent technique for splitting data.

2.3 PROBLEM STATEMENT AND OBJECTIVE

To explore the use of Mixture of Experts for Software Fault Prediction.

Objectives of this study are-

- Apply Mixture of Experts, an ensemble method in software fault prediction, with an unsupervised learning algorithm (Gaussian Mixture Model) as gating algorithm and compare DT and MLP learning technique when used as base learners.
- Evaluate the proposed approach on 41 public available datasets collected from standard software engineering repository. And collect the results of four popularly used performance measures.
- To compare the performance results, implement individual model and bagging ensemble method for all datasets. Perform statistical test to note the significance and also compare with previous studies.

CHAPTER 3 PROPOSED APPROACH

In this chapter, the framework of Gaussian Based Mixture of Experts for binary classification of software modules is discussed. Figure 1 shows the architecture of proposed Software Fault Prediction (SFP) model. Model building procedure consists of two parts: 1) Training Gaussian Mixture (GM) Model over unlabelled training data; 2) Building an ensemble of classifiers based on GM model. The main objective is to accurately categorize software modules and reduce the dependency of software project for selecting suitable SFP model. Above figure shows the framework of proposed system. This section contains the details about the flow of data in the framework and shows how each component is working together to achieve final objective i.e. binary classification of software modules as faulty or non-faulty.

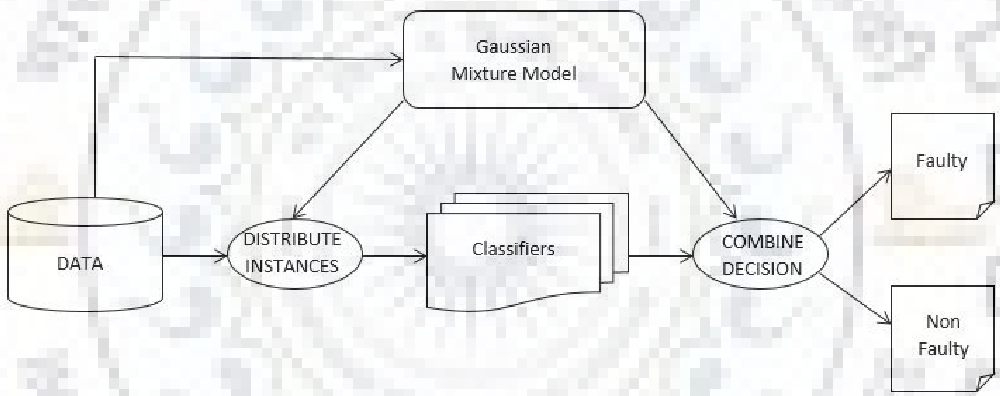


Figure 1 Framework for GME

Data which is at the initial phase is considered to be pre-processed. Pre-processing involves handling missing values, data standardization and class balancing. Details of these methods are discussed later in next chapters. In the proposed system data is firstly give as an input to train Gaussian Mixture (GM) model which is an unsupervised learning technique, that does not need labels of training data. Remaining train data will again be given as input to GM model and output will be a matrix of probabilities. Second step is to train ensemble of classifiers. Remaining train data which has not been given to GM model be used to train the ensemble of classifiers. Probability matrix obtained in the last step will be used to distribute instances to train different classifiers. This distribution of instances is based on the threshold, if the probability of an instance to fall in classifier's subspace is greater than threshold then it will be included for training that classifier. Last step is to make prediction for a new instance. In this stage probability array of an instance will be used to combine the predicted results.

3.1 GAUSSIAN BASED MIXTURE OF EXPERTS

In the first step of the proposed solution for SFP model building, the objective of training GM model is to assign the probability with which each software module belongs to a particular classifier. These assigned probabilities are used, in later stage, to construct training dataset for each expert or classifier. The intention behind using the unsupervised probabilistic classification model rather than strict classification model, is to soft split data among different classifiers.

STEP 1: Training Gaussian Mixture Model

A GM model is a probabilistic mixture model which presumes that all the data points are triggered from a mixture of a finite number of Gaussian distribution with unknown parameters and it has consistently produced state-of-the-art performance in various field of classification, recognition, prediction, etc. [12]. In the first stage of the proposed solution for SFP model building, the objective of training GM model is to assign the probability with which each software module belongs to a particular classifier. These assigned probabilities are used, in later stage, to construct training dataset for each expert or classifier. The intention behind using the unsupervised probabilistic classification model rather than strict classification model, is to soft split data among different classifiers. In training stage, GM model tries to predict unknown parameters of each Gaussian distribution. Since it considers that data points are generated from some Gaussian distribution.

Input: Set of instances without class label $X = \{ x_1, x_2, \ldots x_n \}$ where n is the number of instances in X and number of components k .

Output: 2D Matrix of probability G having n rows and k columns.

Here in X, x_i is a row vector of size m and m is the number of attributes in dataset, k in GME will be set equal to the number of classifiers used to ensemble. Each element in G, $g_{i,j}$ will represent the probability of an instance i to fall into the subspace of classifier k.

The parameters of GM model are updated using multiple iterations of Expectation Maximization (EM) algorithm. In each iteration, EM algorithm updates parameter to maximises the likelihood. GM algorithm is known to a fastest algorithm for learning mixture models. A simple experiment over 2D-blob cluster datasets have been done to understand the working of GM model.

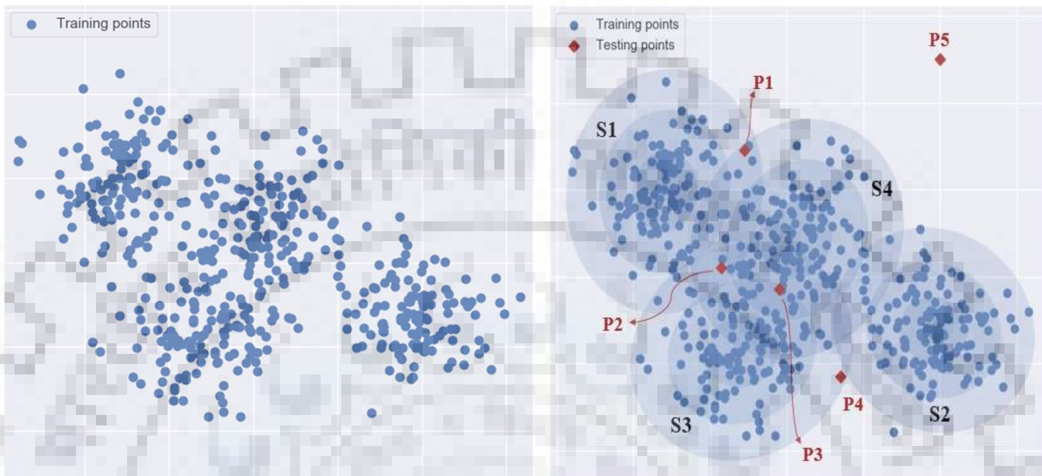


Figure 2 Visualization of data points used in experiment GM model (a) Initial points (b) Trained GM model with testing points

GM model is trained on 2D data points shown in figure 2a. In figure 2b, circle shows the cluster spread after training GM model. Darker circle of each cluster show that density of points in that region is higher that the outer light color circle. Red points in figure 2b shows the testing point. Results are shown in terms of probabilty in table 2. Values which are approximately equal to zero are not shown.

	S1	S2	S3	S4
P1	58%	-	-	41%
P2	27%	-	29%	51%
P3	-	-	12%	85%
P4	-	41%	54%	-
P5	16%	12%	-	71%

Table 2: Results from experiments as probability of points (P) for lying in a subspace(S)

Point P1 which lies on the edge of S1 and S2 have probabilty 58% and 41% respectively, which are approximately equal. Point P2 which lies at the intersection of clusters S1, S3 and S4 but is very near to high probability zone (dark circle) of S4, hence probabilities assigned to it are 27%, 29% and 54%. P3 is in the dark region of S4 but also lies fairly in S4. Here probabilty of point P3 to lie fall in S4 is much larger than S3. Outlier P5 which should not be in any cluster, have fairly large probability of lying in cluster S4.

Another simulation have been done with different shaped 2D cluster data to check the efficiency of GM model.

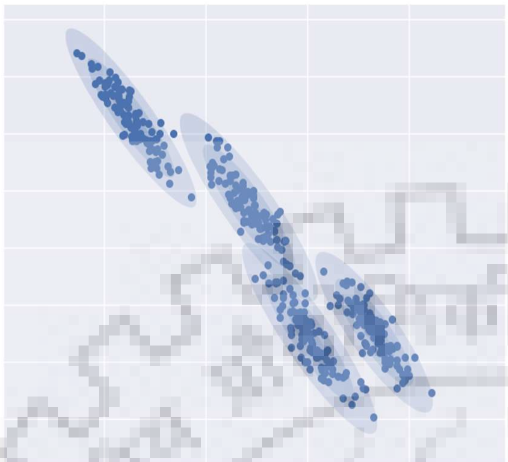


Figure 3(a) Stretched data clusters
n_components = 4

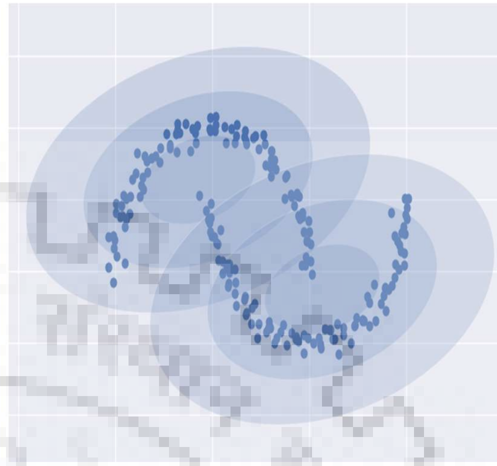


Figure 3(b) Moon shape data clusters
n_components = 2

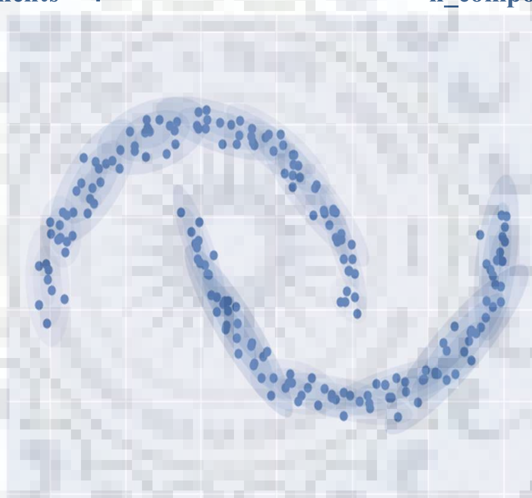


Figure 3(c) Moon shape data clusters
n_components = 16

Figure 3 GM Model cluster on different data shape with different n_components value

Observations about GM model that are made from above experiments are-

- GM model also uses density information to form clusters.
- Probabilty difference is very large when a point lies in dark region (P3 in Figure 2b).
- Outlier point might get higher probability than a point lying on a cluster (P5).
- Clusters of GM model adopts the shape according to data.
- Number of clusters also affects the cluster shape and effectiveness of GM model. Figure 3b and figure 3c shows the difference in cluster shape on changing number of clusters.

STEP 2: Training and testing of Experts

After distribution of instances among different cluster subspaces, machine learning models need be trained. Architecture of training proposed approach (GME) is described using figure 4. It shows the flow for 3 experts with the consideration that GM model has already been trained. Algorithm 1 shows the training procedure for GME. In few words, training algorithm uses output of GM model to soft split the data into subspaces which will be used for training each expert independently. In this step data split using GM model will make sure that no two subspaces will be totally same and hence model trained on two different subspace of data will also be trained differently.

Algorithm 1 Training classifiers of GME

Input: The remaining dataset X'={x_1, x_2,…x_T,…,x_n}, corresponding output labels Y'={y_1, y_2,…y_T,…, y_n}, here $y_i \in$ {F, NF} denotes class label, data selection threshold _DS, number of experts k and trained GM model (G).

Output: Trained experts

1. BEGIN
   For T = 1 to n do
2. Input x_T to G and store the output in g_T={$g_{T,1}$, $g_{T,2}$,…$g_{T,k}$}.
   For j = 1 to k do
3. check $g_{T,j}$ >_DS
      Add x_T and y_T to training subspace of expert j, Sj.
   End For
   End For
   For i = 1 to k do
4. Train expert i using training subspace S_i.
   End For
5. END

Algorithm 2 also uses trained GM model not for splitting but to collect the decision of each expert. Integer value zero is sometimes used in place of class label non-faulty and integer value one for faulty. In most of the cases, X is used to indicate the attributes value of each instance and Y is used to represent a set of corresponding class labels. Figure 4 is drawn to make the data flow and terminology used in algorithms easy to understand.

Algorithm 2 Testing GME

Input: The query or test dataset X"={x_1, x_2,…x_Q,…,x_z}, label prediction threshold (_LP), number of experts k, trained experts (E) and trained GM model (G).

Output: Predicted class labels, Y_P ($|Y_P| = z$).

1. BEGIN

   For Q = 1 to z do

2. Initialize temporary variable (t) with 0
3. Input x_Q to G and store the output in g_Q={$g_{Q,1}$, $g_{Q,2}$,…$g_{Q,k}$}.

   For j = 1 to k do

4. Input x_Q to E_j and store its output in $y_{Q,j}$.
5. Update t, t = t + ($y_{Q,j} \times g_{Q,j}$).

   End For

6. Check t >_LP

   Add class label faulty (F) to Y_P.

7. Otherwise

   Add class label non-faulty (NF) to Y_P.

   End For

8. END

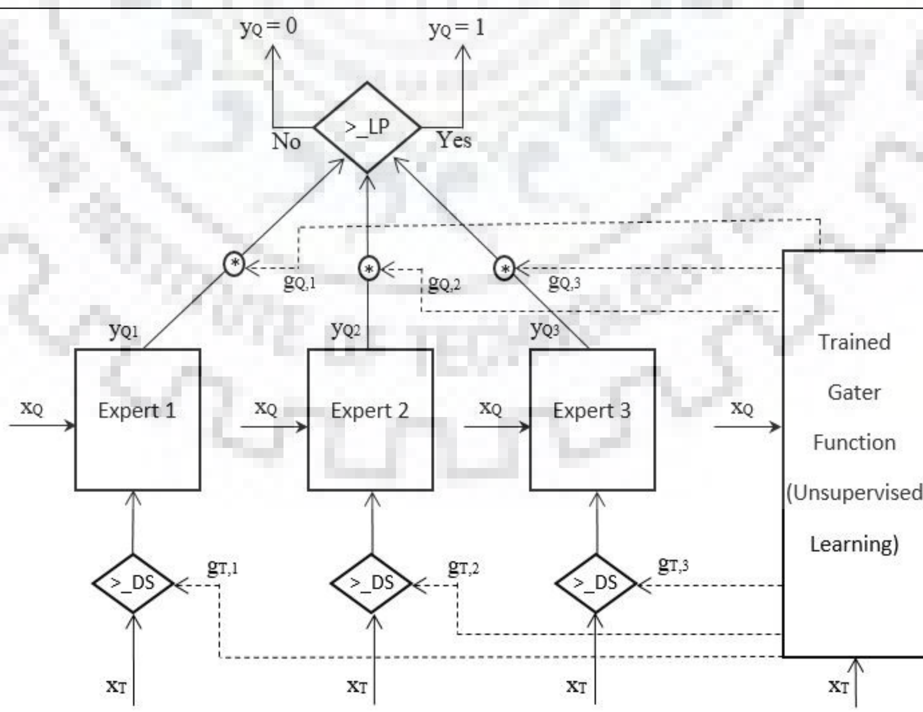


Figure 4 Architecture of GME

3.2 MIXTURE OF LEARNERS

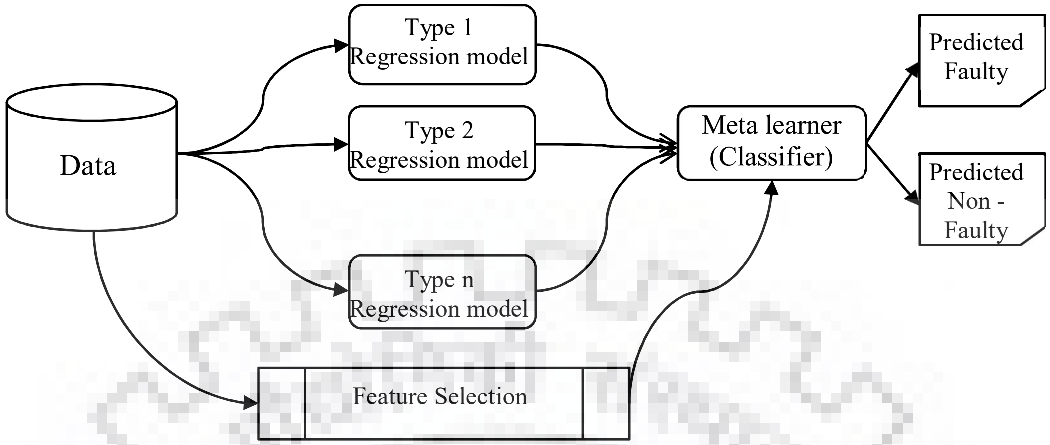


Figure 5 Framework of Mixture of Learners

Mixture of Learner is an ensemble method which uses different learning model at level 1 and these models will perform regression on fault dataset. Note that this technique is a variant of an ensemble method called stacking. In stacking different types of learning models are used at level 1 and meta learner at level 2 will use the results of level 1 models for training. Novelty in approach of Mixture of Learners is, along with results of level 1 models, selected features from input data will also be used to train the meta classifier. This will help meta classifier to make a correlation with input data.

As from the datasets for SFP, it is observed that data is collected on various metrics and these metrics directly impacts the performance of prediction model. So for applying proposed approach, different sets of metrics cane be used for training. At level 1 of proposed approach, regression models are used on classification dataset. At level 1 Gaussian mixture model with n_components as 2 can be used to assign probability to each software module.

Working of proposed approach will start with the training of level 1 regression models on some portion of train dataset and remaining of train dataset and features extracted using technique which have performed best in recent studies, will be used to train level 2 learning algorithm which is a classification algorithm. Model selection for ensemble is an important and essential step in this approach. Level 1 models which uses very different learning strategy like (k-Nearest Neighbours and neural network) should be used. Many application of stacking have used more than 50 models at level 1 which makes a better model but also increases the time complexity.

CHAPTER 4 EXPERIMENTAL DESIGN

To evaluate the effectiveness of proposed approach which is a combination of unsupervised learning (Gaussian Mixture Model) based ensemble of classifiers, the following discussed simulation experiments are performed. This section is organized as, first subsection introduces the benchmark datasets, which are collected from real-world Software projects and are publicly available for research work. Second subsection discusses the performance measures for evaluating the conducted experiments. Later parts contain introduction of classification models and details of experiments. The experiment results are collected based on the performance of 5-fold cross validation.

All the experiments are implemented using libraries of Python programming language on 64-bit Windows operating system over 4GB RAM and Intel i5 processor machine having clock speed @1.75GHz.

4.1 DATA PREPARATION

For obtaining the effectiveness and feasibleness of proposed GME architecture in binary fault classification, total 41 datasets from NASA Metrics Data Program (MDP), NASA PROMISE and Eclipse software engineering repository are used for evaluation. Link are in the reference [20] [21] [22]. These datasets are commonly used for prediction of software modules in many studies discussed in chapter 2. Using those datasets is helpful for the comparative analysis of performance results. In chapter 5, proposed model have been compared with the results of previous studies.

Table 5 contains the overview of datasets used for the simulation. Datasets having very few instances (< 200) are excluded which gives total 41 datasets. In dataset if any class is having lesser number of instances, it is said to be a minority class. From table 5 it can be observed that percentage of minority class in dataset are < 20%, except few datasets (e.g., Equinox, KC2, MC2, etc.). This value is even <10% for some datasets (e.g., camel-1.0, CM1, ivy-2.0, etc.), which shows that it is highly imbalance [2]. On this note, software fault datasets can be said to be imbalance and there is a need of solution to this problem. Because if a model is trained on imbalance data, it is likely to get instances of majority class only on random splitting and will be trained to give the label of majority class to every module, which is not correct but the accuracy of such model will be greater than 80% due to right predictions of majority class.

Name	Instances	Attributes	Missing values	Fault Instances	Minority class %	Instances After Sampling
ant-1.7	745	20	0	93	12.48	1304
camel-1.2	608	20	0	99	16.28	1018
camel-1.4	872	20	0	71	8.14	1602
camel-1.6	965	20	0	101	10.47	1728
CM1	498	21	0	49	9.84	898
eclipse-2.0	6729	199	0	1278	18.99	10902
eclipse-2.1	7888	199	0	1131	14.34	13514
eclipse-3.0	10593	199	0	1579	14.91	18028
Equinox	324	17	0	80	24.69	488
ivy-2.0	352	20	0	28	7.95	648
JDT_Core	997	17	0	138	13.84	1718
jedit-4.3	492	20	0	10	2.03	964
JM1	10880	21	25	2103	19.33	17554
KC1	2109	21	0	326	15.46	3566
KC2	522	21	0	107	20.5	830
KC3	200	40	258	36	18	328
Lucene	691	17	0	51	7.38	1280
MC1	9466	39	0	68	0.72	18796
MC2	127	40	34	44	34.65	166
MW1	264	40	139	27	10.23	474
mylyn	1862	17	0	186	9.99	3352
PC1	1109	21	0	77	6.94	2064
PC2	1585	40	4004	16	1.01	3138
PC3	1125	40	438	140	12.44	1970
PC4	1458	40	0	178	12.21	2560
PC5	17186	39	0	516	3	33340
PDE_UI	1497	17	0	143	9.55	2708
poi-3.0	442	20	0	201	45.48	482
prop-1	18471	21	0	1714	9.28	33514
prop-2	23014	20	0	1677	7.29	42674
prop-3	10274	20	0	923	8.98	18702
prop-4	8718	20	0	577	6.62	16282
prop-5	8516	20	0	939	11.03	15154
prop-6	660	20	0	55	8.33	1210
synapse-1.2	256	20	0	52	20.31	408
velocity-1.6	229	20	0	34	14.85	390
xalan-2.4	723	20	0	79	10.93	1288
xalan-2.5	803	20	0	297	36.99	1012
xalan-2.6	885	20	0	271	30.62	1228
xalan-2.7	909	20	0	660	72.61	1320
xerces-1.4	588	20	0	181	30.78	814

Table 3 Details of selected datasets

With this discussion two things can be concluded. First, class imbalance is problem and needs to be handed in pre-processing step to avoid the incorrect training of model. Second, better accuracy model can sometimes be wrongly trained. So model must be evaluated on more than accuracy. Later section of this chapter discuss the various performance measures used for the evaluation of proposed approach.

In simulation of GME, for handling the class imbalance problem, Synthetic Minority Over-Sampling TEchnique (SMOTE) [1] is used as a step in data preprocessing. SMOTE generates the "synthetic" samples of minority class. In backend, SMOTE uses k-NN to determine "synthetic" samples. Along with data sampling, data cleaning and data standardization are also performed. Data cleaning handle the missing value by either removing all values of instance or by replacing the value with average of that feature. Removing all the values of an instance leads to information loss and when dataset is small (number of instances is less than 200), losing that instance's information will be costly. In that case missing value is replaced with the average value of the attribute.

4.2 CLASSIFICATION MODELS

For comparative analysis and effectiveness of results, two base learning algorithm (DT and MLP) have been applied individually and on two different ensemble methods (Bagging and GME). Results are stored for 4 performance measures (Accuracy, F1-Score, Precision, Recall), as shown in chapter 5. This section contains introduction of various applied classification models with their ensemble technique.

4.2.1 Decision Tree (DT)

First step of classification is to divide the dataset or sample space into two or more homogeneous data sets (or sub-sample sets) based on most significant classifier / differentiator provided implicitly in input variables [12]. It works well with both the discrete, categorical and continuous input and output variables.

In Decision Tree classifier, concept of weighted tree is used, where the internal nodes are marked with featured used for classification and edges of the tree created are marked as trial with dataset weight. Tree leaves are named by categorization. By this way, whole document can be categorized from root till the leaf node is reached, moving through the branches. Learning in decision tree adopts a decision tree classifier, which maps information of an item to conclusions of that items expected value [12].

4.2.2 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) or neural network (NN) is a machine learning algorithm that works on the principle of biological neural network. It consists of series of processing layers interconnected, with each connection possessing some weight. During training, based on the knowledge of domain, it develops a representation that maps input space to output space. MLP uses supervised learning technique called back-propagation to train the network [10].

The working of MLP can be described as follows: For each iteration, the training data are iteratively fed into the neural network and the output obtained is compared with the desired output and error is calculated, which is used to update the hidden layer weights and re-feed the network. The updation is done ensuring that error decreases after each iteration and the output obtained is closer to the desired output.

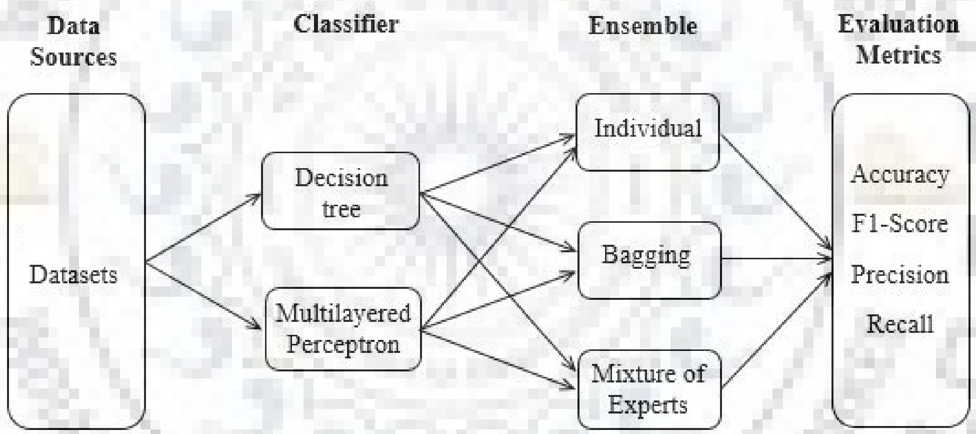


Figure 6 Different schemes for Experimental Analysis

4.2.3 Ensemble Method

With the combination of two base learner and three ensemble method, total six classification technique is generated. DT and MLP on proposed ensemble technique (GME) is compared with the performance from bagging ensemble method and individual model [12].

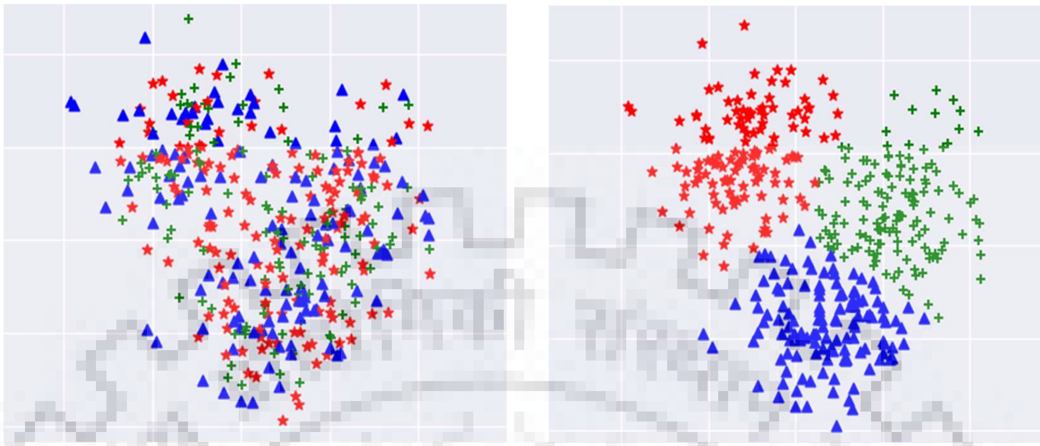
Individual model implements the single learning technique. The objective of the experimental study is to find the improvement in performance on individual model and from existing ensemble technique, bagging [12]. Individual ensemble method implements single learner model for prediction. Figure 6 shows that DT and MLP will be used as individual model.

Bootstrap aggregating, also called bagging, is a meta-learning ensemble technique in machine learning which was designed to enhance the stability and accuracy of individual machine learning model used for classification and regression. Bagging also decreases the variance of individual model and prevent the model from overfitting.

4.3 DIFFERENCE BETWEEN BAGGING AND GME

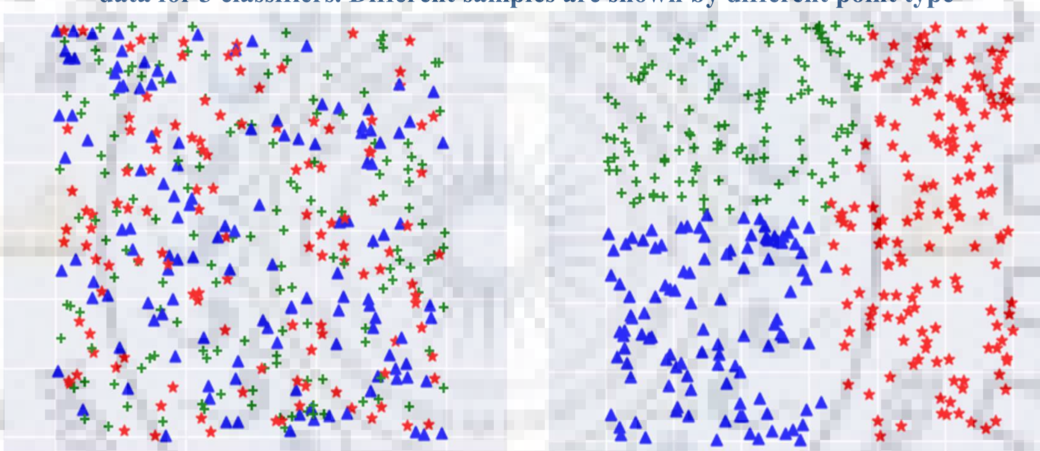
Bagging is described as, for a given training set X of size n, bagging generates k new training sets X_i. Here k is the number of base learners used to ensemble. Each training set X_i will be of size m and it is generated with sampling from X uniformly and with replacement. Some instances may be repeated in each dataset X_i because of using sampling with replacement. Features of X can also be sampled to generate X_i. Hence, number of instance and dimensions of X_i will be less than X. Each X_i represents the X but none of the subset of data X_i will be similar, even when two instances are same, they could have different set of features. These subsets will be used to train k base models independently and for testing this model, query point will be given to each k trained model with sampled features. Next, the result of each model will be combined using voting to provide the final result.

In proposed GME approach for meta learning ensemble technique, the training dataset X will be used to train an unsupervised model (here Gaussian Mixture Model) for k components. This model is termed in ME approach as a gating model. GM model is a probabilistic clustering approach which assigns probability to each data point of falling in a particular cluster. If the probability of lying in a subspace for any data point is greater than threshold value (here, _DS), then it will be considered to be the part of that subspace X_i. Each instance may lie in multiple X_i; hence dataset is known to be softly splitted. These subsets of data will be given as an input to k base leaners to train them independently. For testing of this model, query instance will be given to each base learning model and GM model which output an array of k size. Each element of array will be a probability for query point of lying in the cluster of that model. This probability is used to combine the results of each base learners.



(a) (b)

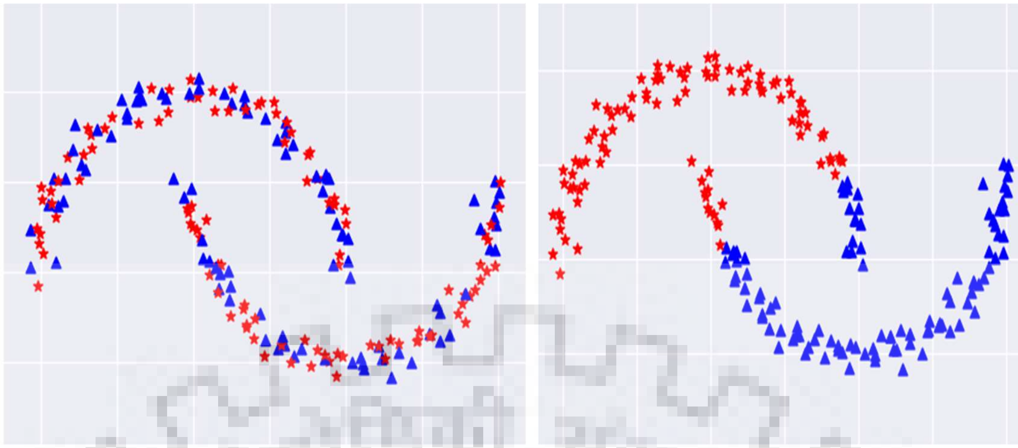
Figure 7 Difference between sampling instances by (a) Bagging and (b) GME on cluster data for 3 classifiers. Different samples are shown by different point type



(a) (b)

Figure 8 Difference between sampling instances by (a) Bagging and (b) GME on random data for 3 classifiers. Different samples are shown by different point type

Figures represents the sampling of instances done in the case of bagging and GME. Figure 1 shows the well separated cluster of data which has been messed up in the case of bagging. GME even for the random spread of data, gives a well formed subset of dataset. Conclusion which can be made on above figure is that, in bagging data is randomly separated which leads to loss of spatial information and subset of data represents the whole dataset, so there can be a case in which model trained on these will be very similar whereas in GME it uses the spatial property of the data for creating the subspace hence model trained on these subspaces will be different from each other. Outlier might affect the training of every base model of bagging but in the case of GME, it is certain that outlier will not affect every model even in the worst case.



(b) (b)

Figure 9 Difference between sampling instances by (a) Bagging and (b) GME on moon shaped data for 3 classifiers. Different samples are shown by different point type

4.4 PERFORMANCE EVALUTION MEASURES

Prediction model gives the predicted class as an output and it is difficult to store this array for comparison from other models. Hence there is a need for some standard performance evaluation measure on which different machine learning techniques can be tested [2]. There are many measures proposed in the theory and some of them are being widely for measuring and comparing the performance of an algorithm. Performance measure column of table 1 shows that accuracy, f1-score, precision and recall have been used in most of the studies.

	Predicted Faulty	**Predicted Non-Faulty**
Faulty Modules	Number of True Positive (TP)	Number of False Negative (FN)
Non-Faulty Modules	Number of False Positive (FP)	Number of True Negative (TN)

Table 4 Confusion matrix for Software Fault Prediction

Above mentioned performance measures will be calculated using confusion matrix. Table shows the confusion matrix and contain basic terminology used to define performance measure. Following discussed measures are as described in table 4.

Accuracy denotes the percentage of correctly classified instances to the total number of instances. *Precision* denotes number of correctly classified faulty instances among the total number of instances classified as faulty. *Recall* indicates the number of correctly classified faulty instances amongst the total number of instances which are faulty. *F1-Score* is the harmonic mean of precision and recall values [2].

F1-score considers both FP and FN, so it is not as easy as accuracy to understand. But f1-score is more useful than accuracy especially if there is uneven distribution of class. A learning scheme is known to be better if accuracy, f1-score, precision and recall values are higher.

To validate the significance of proposed approach and its ranking with respect to other ensemble technique, Wilcoxon's non-parametric test have been applied. It is a statistical hypothesis test which is widely popular and used to compare two related columns (i.e. ensemble methods or learning algorithms) [23]. And second statistical test, Friedman's rank test which is also a non-parametric test is applied to check whether the ranking of multiple columns (i.e. ensemble methods or learning algorithms) is consistent across the dataset [24].

4.5 PARAMETER SETTING

Libraries of python are used in the simulation experiments. Following are details about parameters on which models had been tested.

SMOTE method of class over_sampling in package imblearn, is used for sampling with 5 neighbours to generate synthetic samples.

Cross-Validation: 5-fold cross validation is implemented using KFold method of sklearn library with shuffle set as true that means data will be shuffled before splitting into batches.

Decision Tree: Implemented function DecisionTreeClassifier of library sklearn is used with parameters, splitting criteria as gini index and tree will be extended up to the height until all leaves are pure.

Multilayer Perceptron: Implemented method MLPClassifier of library sklearn is used with initial parameters.

Bagging: BaggingClassifier function of sklearn library is applied. Number of estimators (base learners) is set as 10. Maximum number of samples and features to be drawn from original data is set as 0.7. That shows 70% of total instances and 70% of all features will be used to construct the subset for every model. Bootstrap is set to be true for both samples and features that indicates, an instance or feature will be included in more than one training subset.

GME: Number of experts or estimators is set as same as bagging, 10 for simplicity. The purpose is to evaluate the performance of GME and compare it with individual model and bagging ensemble. So same number of base models are used to ensemble in bagging and GME. Data selection threshold (_DS) and Label Prediction threshold (_LP) is set as 0.2 and 0.6 respectively. These values are concluded from figure 11.

Figure 11 shows that there is minimum deviation in models when _DS and _LS are set as 0.2 and 0.6 respectively. For getting the standard deviation vs data selection threshold graph for different values of label prediction threshold graph, following experiment is conducted.

First, 10 datasets are selected at random and among them, on 5 datasets DT is used as base model and for remaining MLP is used as base model. Second, GME model with number of experts as 10, for different values of _DS and _LP is applied, with 5-fold cross validation. Third, accuracy score of each dataset for every combination of _DS and _LP values is collected. Lastly, calculate standard deviation (SD) of collected accuracy of 10 datasets for every pair of _DS and _LP values. SD is low for _LP=0.6 for almost every value of _DS. And it is minimum when _DS=0.2.

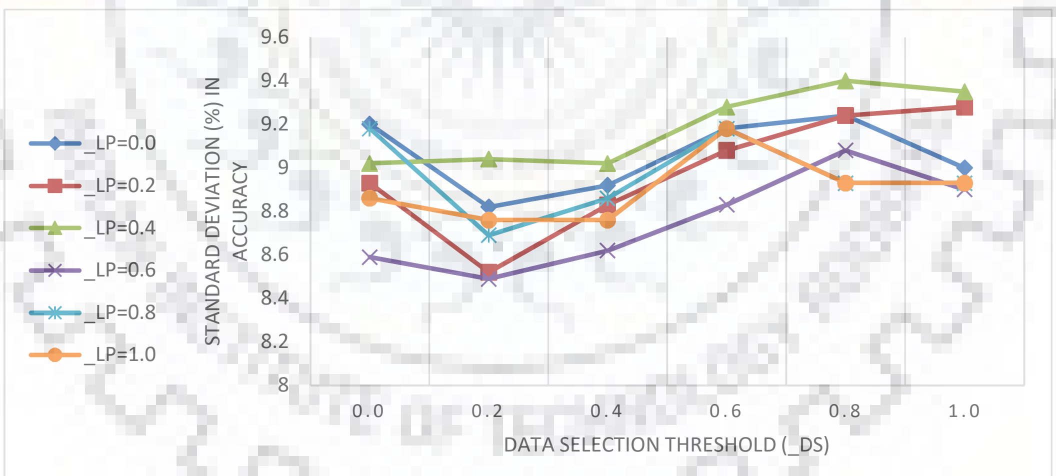


Figure 10 Graph for different values of _DS and _LP on 10 random datasets

CHAPTER 5 RESULTS AND ANALYSIS

In this section, table 5, 6, 7 and 8, show performance results of classification in terms of accuracy, f1-score, precision and recall respectively, when base learning schemes are DT and MLP applied over different aggregation schemes. Every table consists of two sections which contain the result of base expert and for each base expert, there are three subsections that represents the aggregation type. Every result in bagging and GME subsection of DT and MLP is the maximum value obtained by varying the number of experts from 2 to 30, which is the average of 5-fold cross validation. Bold values are the best performances in each row of the tables. Later parts contain discussion about the results and statistical tests which have been performed on the results to evaluate the significance and ranking of model.

5.1 PERFORMANCE COMPARISON

This subsection contains comparison of models on the basis of performance measures. Result of models grouped on the basis of accuracy, f1-score, precision and recall is present in table 5, 6, 7 and 8. Objective of collecting results on different measure is to analyse the model performance independent of other performance measure. Each table contain the results of a performance measure recorded for three different ensemble technique with two different base learner models on 41 datasets. Values which are best for a dataset is highlighted with the bold font.

At the end of each table, to provide the comparative overview of all learning algorithms on different performance measures, average of 41 values for all datasets is mentioned. This helps in concluding the results of that table. For all four performance measure, average value of Multilayer-Perceptron (MLP) when applied as a base model for proposed ensemble approach (GME), is always high. Second conclusion that can be made from the average values present in table 5, 6, 7 and 8 are bagging outperforms individual model when Decision Tree is used as base learner but this is not same for the MLP. GME always perform better than individual model and bagging ensemble model on all four performance measures with both base classifiers. The suffix G in MLP-G and DT-G indicates that proposed Gaussian Mixture approach is used to ensemble those models.

Dataset	DT			MLP		
	Individual	Bagging	GME	Individual	Bagging	GME
ant-1.7	0.8562	0.8931	0.8685	0.8631	0.8554	**0.9285**
camel-1.2	0.6936	0.7222	0.8542	0.8236	0.8118	**0.9054**
camel-1.4	0.7967	0.8356	0.9122	0.9247	0.9153	**0.9567**
camel-1.6	0.8819	0.9167	0.8808	0.8866	0.8860	**0.9271**
CM1	0.9002	0.8936	0.9211	0.8717	0.8553	**0.9473**
eclipse-2.0	0.6571	0.7712	0.7322	0.7760	**0.7884**	0.7871
eclipse-2.1	0.7172	0.8358	0.7897	0.8515	0.8509	**0.8574**
eclipse-3.0	0.6917	0.7722	0.8202	0.826	0.8308	**0.8360**
Equinox	0.8917	**0.9247**	0.7967	0.7459	0.7093	0.8497
ivy-2.0	0.7968	0.8230	0.9447	0.9247	0.9018	**0.9601**
JDT	0.8771	**0.9170**	0.8334	0.7527	0.7584	0.9092
jedit-4.3	0.8652	0.8995	**0.9835**	0.9752	0.9680	0.9773
JM1	0.9721	**0.9762**	0.8334	0.7130	0.7113	0.7384
KC1	0.8312	0.8275	**0.8794**	0.7649	0.7582	0.8212
KC2	0.8769	**0.8777**	0.8675	0.8072	0.8048	0.8494
KC3	0.7643	0.7772	0.8600	0.8756	0.8182	**0.9108**
Lucene	0.8229	0.8699	0.9243	0.8970	0.8548	**0.9571**
MC1	0.8639	0.8725	0.9831	0.9901	0.9862	**0.9911**
MC2	0.6041	0.6170	0.7854	0.7467	0.7269	**0.8185**
MW1	0.8090	0.8090	0.8944	0.9303	0.9191	**0.9551**
mylyn	0.8876	**0.9360**	0.8993	0.8243	0.8159	0.9047
PC1	0.8745	0.8978	0.9423	0.8920	0.8987	**0.9608**
PC2	0.9212	0.9212	0.9787	0.9851	0.9801	**0.9858**
PC3	0.8169	0.8308	0.8867	0.9135	0.9055	**0.9372**
PC4	0.8580	0.8686	0.9230	0.9502	0.9364	**0.9650**
PC5	0.7485	0.7572	0.8044	0.7850	0.7774	**0.8388**
PDE	0.9375	**0.9477**	0.8946	0.8028	0.7922	0.8990
poi-3.0	0.6894	0.7158	0.7336	0.6852	0.6807	**0.7533**
prop-1	0.9188	0.8906	**0.9196**	0.8406	0.8439	0.8726
prop-2	0.7412	0.7450	**0.9312**	0.8899	0.8909	0.9211
prop-3	0.6993	0.7029	0.8391	0.7832	0.7839	**0.8637**
prop-4	0.7080	0.7130	**0.8551**	0.8108	0.8086	0.8322
prop-5	0.7804	0.7969	**0.9323**	0.8782	0.8723	0.9176
prop-6	0.9242	0.9203	0.9845	0.9845	0.9776	**0.9884**
synapse-1.2	0.7918	0.8196	0.8496	0.8296	0.8120	**0.8970**
velocity-1.6	0.8501	0.9097	0.9161	0.9018	0.8606	**0.9302**
xalan-2.7	0.8145	0.8565	0.8321	0.8260	0.7954	**0.8802**
xerces-1.4	0.8115	0.8540	0.8351	0.8493	0.8328	**0.8787**
xalan-2.4	0.8961	0.9041	0.9697	0.9798	0.9697	**0.9814**
xalan-2.5	0.7760	0.7908	0.8933	0.9060	0.8735	**0.9413**
xalan-2.6	0.7945	0.7952	0.8875	0.8829	0.8856	**0.9331**
Average	0.8149	0.8392	0.8798	0.8572	0.8465	**0.9016**

Table 5 Accuracy values

Dataset	DT			MLP		
	Individual	Bagging	GME	Individual	Bagging	GME
ant-1.7	0.8588	0.8938	0.8679	0.8685	0.8633	**0.9312**
camel-1.2	0.7004	0.7404	0.8591	0.8318	0.8192	**0.9137**
camel-1.4	0.8031	0.8348	0.9145	0.9282	0.9189	**0.9579**
camel-1.6	0.8848	0.9163	0.8822	0.8906	0.8916	**0.9298**
CM1	0.9039	0.8989	0.9227	0.8786	0.8639	**0.9501**
eclipse-2.0	0.6591	0.7813	0.7415	0.7905	**0.8002**	0.7991
eclipse-2.1	0.7294	0.8373	0.7920	0.8571	0.8573	**0.8624**
eclipse-3.0	0.6505	0.7762	0.8159	0.8315	0.8366	**0.8419**
Equinox	0.8937	**0.9260**	0.8028	0.7581	0.7165	0.8565
ivy-2.0	0.8054	0.8344	0.9444	0.9289	0.9051	**0.9608**
JDT	0.8818	**0.9198**	0.8398	0.7642	0.7623	0.9150
jedit-4.3	0.8702	0.9026	**0.9841**	0.9756	0.9690	0.9778
JM1	0.9723	**0.9769**	0.8398	0.7209	0.7209	0.7438
KC1	0.8378	0.8347	**0.8798**	0.7763	0.7654	0.8216
KC2	0.8769	**0.8825**	0.8689	0.8029	0.8058	0.8527
KC3	0.7383	0.7569	0.8641	0.8809	0.8183	**0.9168**
Lucene	0.8206	0.8704	0.9257	0.9011	0.8603	**0.9587**
MC1	0.8782	0.8853	0.9834	0.9902	0.9864	**0.9912**
MC2	0.6159	0.6101	0.7812	0.7495	0.7286	**0.8212**
MW1	0.8114	0.8043	0.8980	0.9337	0.9230	**0.9568**
mylyn	0.8910	**0.9368**	0.9010	0.8266	0.8197	0.9071
PC1	0.8756	0.8992	0.9431	0.8941	0.9018	**0.9618**
PC2	0.9262	0.9243	0.9790	0.9854	0.9802	**0.9863**
PC3	0.8237	0.8453	0.8913	0.9187	0.9098	**0.9405**
PC4	0.8691	0.8784	0.9236	0.9510	0.9378	**0.9654**
PC5	0.7568	0.7675	0.8151	0.7940	0.7916	**0.8450**
PDE	0.9377	**0.9485**	0.8974	0.8121	0.7999	0.9038
poi-3.0	0.6779	0.6956	0.6999	0.6556	0.6429	**0.7361**
prop-1	0.9181	0.8891	**0.9188**	0.8381	0.8426	0.8716
prop-2	0.7562	0.7599	**0.9316**	0.8910	0.8914	0.9219
prop-3	0.7147	0.7181	0.8434	0.7928	0.7942	**0.8622**
prop-4	0.6879	0.7000	**0.8529**	0.8055	0.8058	0.8295
prop-5	0.7741	0.7916	**0.9323**	0.8779	0.8729	0.9190
prop-6	0.9274	0.9214	0.9846	0.9849	0.9781	**0.9883**
synapse-1.2	0.7848	0.8170	0.8311	0.8270	0.8055	**0.8977**
velocity-1.6	0.8486	0.9015	0.9115	0.9057	0.8662	**0.9338**
xalan-2.7	0.8114	0.8501	0.8279	0.8268	0.7937	**0.8763**
xerces-1.4	0.8219	0.8652	0.8425	0.8625	0.8455	**0.8881**
xalan-2.4	0.8974	0.9053	0.9703	0.9803	0.9706	**0.9842**
xalan-2.5	0.7522	0.7841	0.8967	0.9084	0.8775	**0.9431**
xalan-2.6	0.8029	0.8004	0.8892	0.8839	0.8880	**0.9351**
Average	0.8158	0.8410	0.8803	0.8605	0.8495	**0.9038**

Table 6 F1-Score values

Figure 12 shows the line graph for accuracy of DT-G and MLP-G on all datasets. It can be easily observed that MLP-G is performing better in almost every case. There are few cases when DT-G accuracy is greater than MLP-G that are JM1, KC1, KC2, prop-1, prop-4, prop-5. All these datasets were collected from NASA repository and have different set of metrics from other datasets. And also their minority class % is very low. Hence it can be concluded that software metrics and minority class % of datasets affects the performance of GME approach and DT-G performs better than when the minority class % is very low.

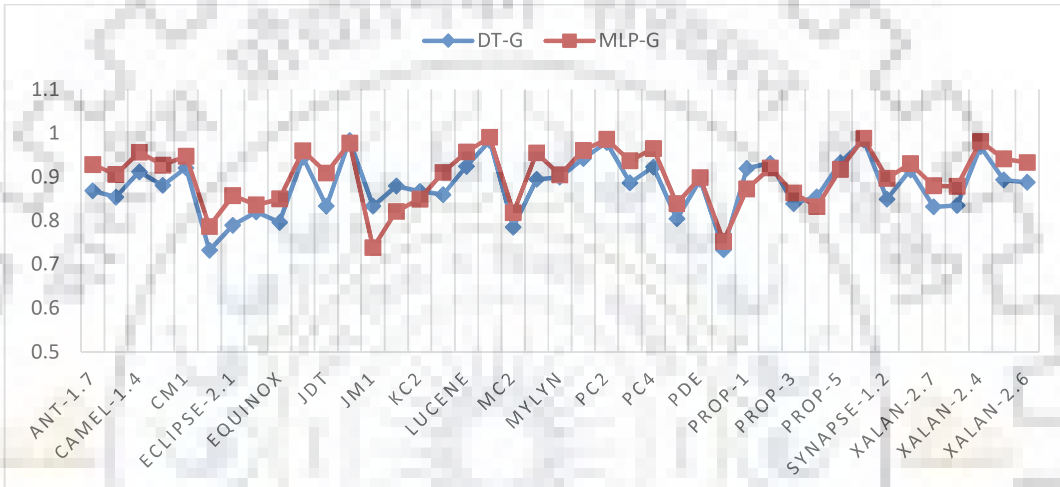


Figure 11 Accuracy graph between DT-G and MLP-G

Graph for F1-Score between DT-G and MLP-G is almost similar and datasets on which DT-G was outperforming MLP-G are same when it comes for f1-score. So same conclusion can be drawn and explanation for underperforming is the same for f1-score as accuracy.

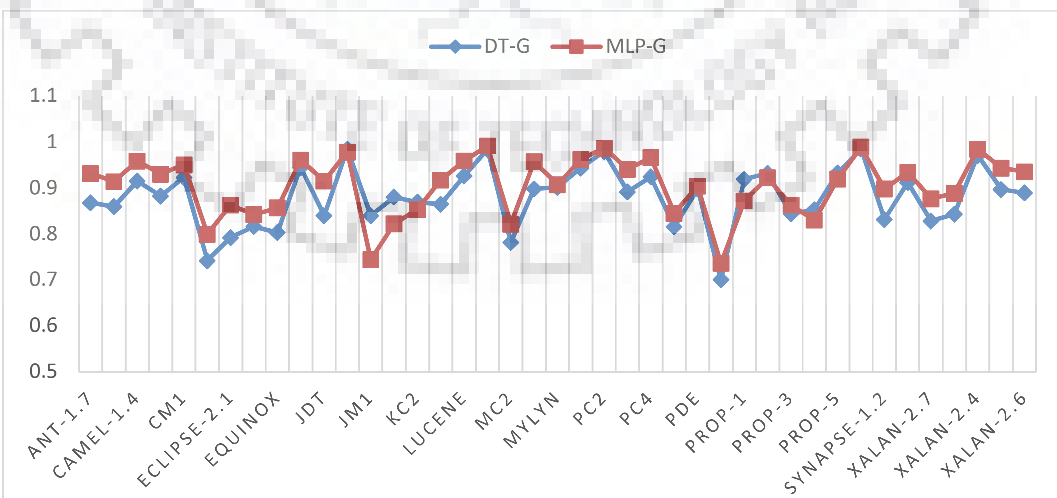


Figure 12 F1-Score graph between DT-G and MLP-G

Dataset	DT			MLP		
	Individual	Bagging	GME	Individual	Bagging	GME
ant-1.7	0.8441	0.8767	0.8537	0.8326	0.8190	**0.8945**
camel-1.2	0.7244	0.7148	0.8605	0.8181	0.8154	**0.8837**
camel-1.4	0.7715	0.8368	0.8936	0.8869	0.8809	**0.9280**
camel-1.6	0.8640	**0.9148**	0.8729	0.8551	0.8527	0.8998
CM1	0.8822	0.8681	0.9093	0.8431	0.8235	**0.9164**
eclipse-2.0	0.6769	0.7711	0.7387	0.7639	**0.7798**	0.7791
eclipse-2.1	0.7029	0.8352	0.7883	0.8308	0.8268	**0.8381**
eclipse-3.0	0.7560	0.7649	**0.8388**	0.8089	0.8112	0.8150
Equinox	0.8829	**0.9147**	0.7849	0.7299	0.7065	0.8161
ivy-2.0	0.7829	0.7888	0.8999	0.8860	0.8719	**0.9438**
JDT	0.8523	**0.8915**	0.8411	0.7447	0.7607	0.8794
jedit-4.3	0.8545	0.8926	**0.9687**	0.9525	0.9400	0.9570
JM1	0.9520	**0.9586**	0.8411	0.7266	0.7225	0.7556
KC1	0.8350	0.8294	**0.8869**	0.7484	0.7510	0.8322
KC2	**0.8863**	0.8573	0.8718	0.8145	0.7961	0.8440
KC3	0.7961	0.8109	0.8617	0.8495	0.7797	**0.8663**
Lucene	0.8320	0.8622	0.8949	0.8651	0.8203	**0.9235**
MC1	0.7948	0.8069	0.9713	0.9805	0.9731	**0.9826**
MC2	0.5590	0.6039	0.7340	0.7151	0.6953	**0.7825**
MW1	0.7952	0.8262	0.8673	0.8766	0.8753	**0.9178**
mylyn	0.8648	**0.9254**	0.8914	0.8144	0.8018	0.8826
PC1	0.8636	0.8845	0.9337	0.8748	0.8715	**0.9358**
PC2	0.8682	0.8868	0.9590	0.9714	0.9612	**0.9730**
PC3	0.8016	0.7818	0.8636	0.8774	0.8748	**0.9014**
PC4	0.7994	0.8100	0.9020	0.9294	0.9100	**0.9433**
PC5	0.7462	0.7501	0.7920	0.7736	0.7571	**0.8285**
PDE	**0.9305**	0.9250	0.8885	0.7794	0.7748	0.8690
poi-3.0	0.6624	**0.7616**	0.6986	0.6740	0.6659	0.7366
prop-1	0.9150	0.8901	**0.9173**	0.8409	0.8395	0.8680
prop-2	0.7154	0.7182	**0.9281**	0.8828	0.8879	0.9142
prop-3	0.6780	0.6822	0.8214	0.7578	0.7567	**0.8604**
prop-4	0.7387	0.7332	**0.8662**	0.8279	0.8175	0.8487
prop-5	0.7943	0.8096	**0.9273**	0.8766	0.8661	0.9119
prop-6	0.8809	0.9065	**0.9829**	0.9705	0.9574	0.9770
synapse-1.2	0.7850	0.8145	0.8391	0.8151	0.7892	**0.8687**
velocity-1.6	0.8518	0.9034	**0.9084**	0.8702	0.8293	0.8947
xalan-2.7	0.8331	0.8966	0.8564	0.8275	0.8028	**0.9113**
xerces-1.4	0.8085	0.8289	0.8367	0.8214	0.8068	**0.8521**
xalan-2.4	0.8853	0.8994	0.9450	0.9618	0.9456	**0.9690**
xalan-2.5	0.8305	0.8055	0.8701	0.8785	0.8453	**0.9101**
xalan-2.6	0.7804	0.7808	0.8780	0.8705	0.8722	**0.9129**
Average	0.8068	0.8297	0.8704	0.8396	0.8277	**0.8835**

Table 7 Precision values

Dataset	DT			MLP		
	Individual	Bagging	GME	Individual	Bagging	GME
ant-1.7	0.8785	0.9131	0.8970	0.9083	0.9149	**0.9770**
camel-1.2	0.7146	0.7689	0.8759	0.8477	0.8262	**0.9589**
camel-1.4	0.8391	0.8339	0.9399	0.9736	0.9610	**0.9900**
camel-1.6	0.9077	0.9189	0.9136	0.9296	0.9348	**0.9652**
CM1	0.9275	0.9338	0.9419	0.9187	0.9104	**0.9960**
eclipse-2.0	0.6432	0.7917	0.7445	0.8191	**0.8219**	0.8205
eclipse-2.1	0.7618	0.8394	0.7959	0.8852	**0.8903**	0.8882
eclipse-3.0	0.5742	0.7944	0.7879	0.8562	0.8637	**0.8707**
Equinox	0.9052	**0.9378**	0.8296	0.7902	0.7340	0.9084
ivy-2.0	0.8393	0.8892	0.9729	0.9767	0.9422	**0.9971**
JDT	0.9140	0.9524	0.8419	0.7858	0.7654	**0.9690**
jedit-4.3	0.8879	0.9138	**1.0000**	**1.0000**	**1.0000**	**1.0000**
JM1	0.9935	**0.9963**	0.8419	0.7157	0.7197	0.7520
KC1	0.8406	0.8402	**0.8781**	0.8081	0.7810	0.8187
KC2	0.8683	**0.9095**	0.8774	0.7936	0.8163	0.8693
KC3	0.6911	0.7151	0.9225	0.9174	0.8665	**0.9825**
Lucene	0.8121	0.8801	0.9574	0.9410	0.9065	**1.0000**
MC1	0.9813	0.9812	0.9979	**1.0000**	**1.0000**	**1.0000**
MC2	0.7060	0.6350	0.8590	0.7982	0.776	**0.9229**
MW1	0.8442	0.7940	0.9640	**1.0000**	0.9780	**1.0000**
mylyn	0.9192	**0.9489**	0.9192	0.8398	0.8385	0.9335
PC1	0.8883	0.9148	0.9628	0.9152	0.9346	**0.9893**
PC2	0.9928	0.9661	**1.0000**	**1.0000**	**1.0000**	**1.0000**
PC3	0.8505	0.9218	0.9296	0.9649	0.949	**0.9884**
PC4	0.9533	0.9597	0.9542	0.9741	0.9677	**0.9906**
PC5	0.7678	0.7869	0.8505	0.8162	0.8304	**0.8689**
PDE	0.9454	**0.9735**	0.9173	0.8483	0.8274	0.9517
poi-3.0	0.7023	0.6514	0.7240	0.6453	0.6232	**0.7474**
prop-1	**0.9212**	0.8881	0.9207	0.8356	0.8459	0.8800
prop-2	0.8021	0.8071	**0.9464**	0.8995	0.8951	0.9323
prop-3	0.7558	0.7584	**0.8725**	0.8318	0.8361	0.8642
prop-4	0.6463	0.6699	**0.8420**	0.7847	0.7945	0.8217
prop-5	0.7551	0.7752	**0.9405**	0.8797	0.8803	0.9375
prop-6	0.9796	0.9380	0.9905	**1.0000**	**1.0000**	**1.0000**
synapse-1.2	0.7961	0.8334	0.9067	0.8409	0.8258	**0.9533**
velocity-1.6	0.8482	0.9020	0.9237	0.9451	0.9087	**0.9850**
xalan-2.7	0.7937	0.8113	0.8142	0.8279	0.7855	**0.8516**
xerces-1.4	0.8369	0.9068	0.8634	0.9096	0.8910	**0.9283**
xalan-2.4	0.9103	0.9114	0.9971	**1.0000**	0.9972	**1.0000**
xalan-2.5	0.6880	0.7725	0.9347	0.9422	0.9140	**0.9807**
xalan-2.6	0.8288	0.8232	0.9013	0.8993	0.9046	**0.9651**
Average	0.8320	0.8575	0.9012	0.8845	0.8746	**0.9331**

Table 8 Recall values

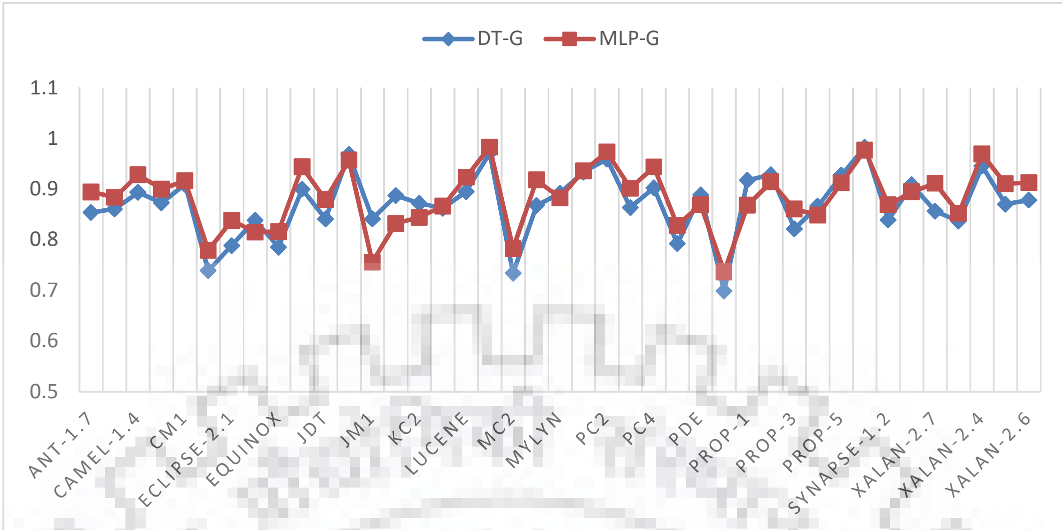


Figure 13 Precision graph between DT-G and MLP-G

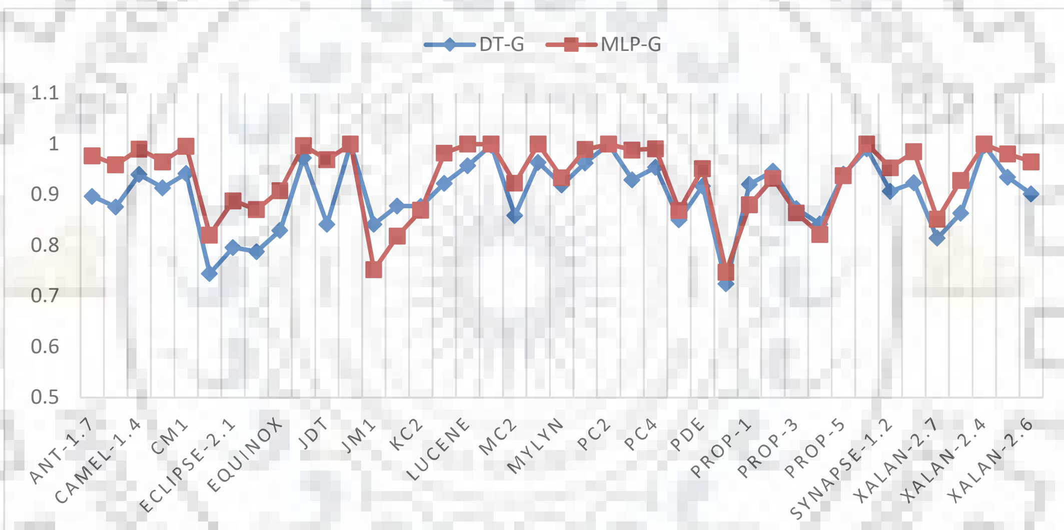


Figure 14 Recall graph between DT-G and MLP-G

Precision value of MLP-G on two datasets is improved and all datasets which were having better accuracy, also have high precision value except eclipse-3.0. Graphs of accuracy, f1-score and precision seems to be very similar. Graph of recall values for MLP-G and DT-G is given in FIG. Recall obtained by MLP-G on various datasets are very high and the difference from recall values of DT-G is very huge. In many cases recall value is one, which means there are no faulty attributes which are labelled as non-faulty. This is a desirable property for a software fault prediction module because faulty modules which goes unattended by software testing teams will increase the time and efforts for removing faulty modules [1].

5.2 STATISTICAL TEST RESULTS

Here are the results of Wilcoxon's statistical test for α=0.5 performed to analyse and compare the effectiveness of Gaussian based Mixture of Experts (GME) and impact analysis of GME when combine with Decision Tree (DT-G) and Multi Layered Perceptron (MLP-G). The proposed approach (GME) is also compared with individual model and bagging ensemble of that model, in terms of four performance measure, results are stored in table 9, 10. And table 11 contain the statistical test's result of both DT and MLP when applied with GME.

In statistical results table, second column (Draw) specify the number of equal results cases. Third column (R^+) and fourth column (R^-) represents the sum of ranks. $P_{wilcoxon}$ indicates the p-value of Wilcoxon's test. If $P_{wilcoxon} < 0.05(\alpha)$, it means that the comparison is significant different [13]. $P_{wilcoxon}$ values which shows significant difference is highlighted in bold font.

The performance results for individual model and GME approach are shown in table 5, 6, 7 and 8. The best performances on each performance measure are highlighted. GME approach obtains better performance measures if compared to individual model, for both base models (DT and MLP). TAB shows the Wilcoxon's test result for the comparison of GME verses individual model in terms of all 4 performance measures with both base learning algorithms.

Methods	Performance Measure	Draw	R^+	R^-	$P_{wilcoxon}$
DT-G Vs DT	Accuracy	0	777	84	**3.67E-06**
MLP-G Vs MLP	Accuracy	0	861	0	**1.26E-08**
DT-G Vs DT	F1-Score	0	775.5	85.5	**4.02E-06**
MLP-G Vs MLP	F1-Score	0	861	0	**1.26E-08**
DT-G Vs DT	Precision	0	781	80	**2.87E-06**
MLP-G Vs MLP	Precision	0	861	0	**1.26E-08**
DT-G Vs DT	Recall	1	737	83	**5.7E-06**
MLP-G Vs MLP	Recall	7	595	0	**1.91E-07**

Table 9 Wilcoxon's test results for the comparison of GME (R+) versus individual model (R-)

Methods	Performance Measure	Draw	R^+	R^-	$P_{wilcoxon}$
DT-G Vs DT-B	Accuracy	0	679	182	**0.000655**
MLP-G Vs MLP-B	Accuracy	0	860	1	**1.36E-08**
DT-G Vs DT-B	F1-Score	0	670	191	**0.000977**
MLP-G Vs MLP-B	F1-Score	0	860	1	**1.36E-08**
DT-G Vs DT-B	Precision	0	679	182	**0.000655**
MLP-G Vs MLP-B	Precision	0	860	1	**1.36E-08**
DT-G Vs DT-B	Recall	0	658	203	**0.001633**
MLP-G Vs MLP-B	Recall	4	700	3	**7.6E-08**

Table 10 Wilcoxon's test results for the comparison of GME (R+) versus bagging (R-)

Methods	Performance Measure	Draw	R^+	R^-	$P_{wilcoxon}$
MLP-G Vs DT-G	Accuracy	0	707	154	**0.000174**
MLP-G Vs DT-G	F1-Score	0	715	146	**0.000117**
MLP-G Vs DT-G	Precision	0	639.5	221.5	**0.003448**
MLP-G Vs DT-G	Recall	2	670	110	**4.8E-05**

Table 11 Wilcoxon's test results for the comparison of MLP (R+) versus DT (R-) with GME

Table 10 evaluate the performance of GME over individual model. $P_{wilcoxon}$ value is less than 0.05 in each case that means that comparison is significantly different. According to the rank values, MLP-G is performing much better than individual model on accuracy, precision and f1-score. Statistical test's results shown in table 10 indicates that GME shows improvement than bagging when MLP is used as a base learner.

Table 11 consists results of statistical tests on each performance measure when DT and MLP models are used for GME. For accuracy score, MLP-G outperforms DT-G and $P_{wilcoxon}$ value indicates the significant difference among both models. F1-score and precision also shows better result with remarkable difference with $P_{wilcoxon}$ value less than 0.05. For recall values the sum of ranks for DT is least even after two draw cases that shows, MLP-G is showing huge improvement than DT-G in recall values.

5.3 COMPARISON WITH PREVIOUS STUDIES

This section is focused to evaluate the proposed algorithm (GME) with studies in recent past which are mentioned in chapter 2. There are various performance measures on which model can be compared but as mentioned in chapter 4, performance of GME is measured in terms of accuracy, precision, recall and f1-score. Study which are made in recent years also used different measures. Tables present in this section uses them to compare with GME. Most of the studies used in this section have used some existing or proposed meta-learning classification technique for binary classification of software modules.

Dataset	**MLP-G**	**DT-G**	[13]	[14]	[9]	[6]	[17]	**[17]**	[11]	[18]	**[21]**
CM1	**0.95**	0.92	0.87	-	-	-	-	0.78	0.90	0.90	0.86
eclipse-2.0	**0.79**	0.73	-	-	-	0.78	0.67	-	-	-	-
eclipse-2.1	**0.86**	0.79	-	-	-	0.82	0.77	-	-	-	-
eclipse-3.0	**0.84**	0.82	-	-	-	0.82	0.77	-	-	-	-
JM1	0.74	**0.83**	0.82	0.82	0.74	-	-	0.79	0.81	-	-
KC1	0.82	**0.88**	0.83	-	-	-	-	0.83	0.87	0.77	0.88
KC2	0.85	**0.87**	-	0.82	-	-	-	0.78	0.84	0.80	-
KC3	**0.91**	0.86	-	-	-	-	-	-	-	-	0.88
MC1	**0.99**	0.98	-	0.98	-	-	-	-	-	-	0.88
MC2	0.82	0.79	0.74	-	-	-	-	-	-	-	**0.86**
MW1	**0.96**	0.89	0.89	0.89	-	-	-	0.81	-	-	0.87
PC1	**0.96**	0.94	0.92	-	0.84	-	-	0.93	0.95	-	0.86
PC2	**0.99**	0.98	-	-	-	-	-	0.13	-	-	0.90
PC3	**0.94**	0.89	0.90	0.87	-	-	-	0.84	-	-	0.86
PC4	**0.97**	0.92	0.90	0.89	-	-	-	0.91	-	-	0.90
PC5	0.84	0.80	-	-	-	-	-	**0.97**	-	-	0.87

Table 12 Comparison of accuracy values with previous studies

Table 12 show that accuracy of GME model is higher in 14 out of 16 cases. For MC2 and PC5, [21] and [7] are performing better respectively. TAB contain the comparative results of f1-score. There were not many studies found which used f1-score for performance measure. Among the studies found, MLP-G is performing better in every case.

Dataset	**MLP-G**	**DT-G**	[8]	[17]
eclipse-2.0	**0.80**	0.74	0.61	0.79
eclipse-2.1	**0.86**	0.79	-	0.86
eclipse-3.0	**0.84**	0.82	0.83	0.84
JDT_Core	**0.92**	0.84	0.79	-
Lucene	**0.96**	0.93	0.85	-
xalan-2.7	**0.94**	0.90	0.70	-

Table 13 Comparison of f1-score values with previous studies

Dataset	MLP-G	DT-G	[8]	[9]
eclipse-2.0	**0.78**	0.73	0.63	-
eclipse-2.1	**0.84**	0.78	-	-
eclipse-3.0	0.81	**0.84**	0.80	-
JDT_Core	**0.88**	0.84	0.79	-
jedit-4.3	**0.98**	0.97	-	-
JM1	0.76	**0.84**	-	0.24
Lucene	**0.93**	0.89	0.83	-
PC1	**0.94**	0.93	-	0.76
xalan-2.7	**0.91**	0.87	0.72	-

Table 14 Comparison of precision values with previous studies

Dataset	MLP-G	DT-G	[8]	[9]	[6]	[17]
CM1	**1.00**	0.94	-	-	-	0.27
eclipse-2.0	**0.82**	0.74	0.59	-	0.58	-
eclipse-2.1	**0.89**	0.80	-	-	0.50	-
eclipse-3.0	**0.87**	0.79	**0.87**	-	0.52	-
JDT_Core	**0.97**	0.84	0.80	-	-	-
JM1	0.75	**0.84**	-	0.16	-	0.38
KC1	0.82	**0.88**	-	-	-	0.52
KC2	0.87	**0.88**	-	-	-	0.66
Lucene	**1.00**	0.96	0.86	-	-	-
MW1	**1.00**	0.96	-	-	-	0.36
PC1	**0.99**	0.96	-	0.12	-	0.39
PC2	**1.00**	**1.00**	-	-	-	0.88
PC3	**0.99**	0.93	-	-	-	0.42
PC4	**0.99**	0.95	-	-	-	0.68
PC5	**0.87**	0.85	-	-	-	0.66
xalan-2.7	**0.98**	0.93	0.68	-	-	-

Table 15 Comparison of recall values with previous studies

MLP-G trains a much precise model. This conclusion can be made from table 14 which shows comparison with two studies and have good lead over them. In terms of recall value, no other study outperforms MLP-G model. [8] gives equal recall results for eclipse-3.0 dataset. Also [21] which was outperforming proposed approach in accuracy, have lesser recall value. With the comparison of each performance measure from previous study it can be said that MLP-G shows significant improvement in results in terms all four performance measures used.

CHAPTER 6 CONCLUSION AND FUTURE WORK

The objective of this work was to evaluate the performance of Mixture of Experts (ME) with Gaussian Mixture Model (GM) as a gating function, in binary classification of software modules. Decision tree and multi-layer perceptron which are most popular algorithm in fault prediction, are used as base learners and results of proposed approach are compared with individual model and bagging ensemble model, in terms of accuracy, f1-score, precision and recall.

For simulation 41 publicly available datasets of real-world software projects are collected from standard software engineering repository, that are, Eclipse, NASA PROMISE and MDP. Python programming language is used to conduct this study and results are stored in terms of four mentioned performance measure. Analysis of collected results and Wilcoxon's statistical test results indicates that ME with gating function as GM (GME) is significantly improving performance from individual model and bagging ensemble technique for both base learning technique (DT and MLP). This statement can be generalized for all performance measures. In later subsection of chapter 5, comparison of performance measure with past studies have been done and it is concluded that GME have shown a remarkable improvement in performance.

Wilcoxon test is performed on DT and ML when combined with GME and it is observed that MLP is more effective than DT when used as base learner in GME. Difference on the recall performance is substantial when MLP is used with GME.

Following are some points which needs to be covered in future work for comprehensive evaluation of performance of GME.

- Extend the experiments for tuning of number of experts or base learners used in GME. This study explores only DT and MLP as experts, more machine learning algorithms is to be inspected in future work.
- Graph shown in FIG suggests that set of metric used in dataset affects the performance of GME and also this study is done without taking feature selection in account, so investigation of appropriate feature selection technique in data pre-processing step of GME is a scope for future work.

- This study proposes a variation of ensemble technique (stacking) in chapter 3, named as Mixture of Learners (MoL). This idea has only theoretically presented, and its applicability in classification or regression problem of software modules is still to be evaluated.

References

[1] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing,* vol. 27, pp. 504-518, 2015.

[2] S. S. Rathore and S. Kumar, “A study on software fault prediction techniques,” *Artificial Intelligence Review,* vol. 51, pp. 255-327, 2019.

[3] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton and others, “Adaptive mixtures of local experts.,” *Neural computation,* vol. 3, pp. 79-87, 1991.

[4] S. E. Yuksel, J. N. Wilson and P. D. Gader, “Twenty years of mixture of experts,” *IEEE transactions on neural networks and learning systems,* vol. 23, pp. 1177-1193, 2012.

[5] F. Zhang, Q. Zheng, Y. Zou and A. E. Hassan, “Cross-project defect prediction using a connectivity-based unsupervised classifier,” in *Proceedings of the 38th International Conference on Software Engineering*, 2016.

[6] Y. A. Alshehri, K. Goseva-Popstojanova, D. G. Dzielski and T. Devine, “Applying Machine Learning to Predict Software Fault Proneness Using Change Metrics, Static Code Metrics, and a Combination of Them,” in *SoutheastCon 2018*, 2018.

[7] Erturk, “Iterative software fault prediction with a hybrid approach,” *Applied Soft Computing,* vol. 49, pp. 1020--1033, 2016.

[8] M. Li, H. Zhang, R. Wu and Z.-H. Zhou, “Sample-based software defect prediction with active and semi-supervised learning,” *Automated Software Engineering,* vol. 19, pp. 201-230, 2012.

[9] B. S. Oboh and N. Bouguila, “Unsupervised learning of finite mixtures using scaled dirichlet distribution and its application to software modules categorization,” in *2017 IEEE International Conference on Industrial Technology (ICIT)*, 2017.

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

[11] H. A. Al-Jamimi and L. Ghouti, "Efficient prediction of software fault proneness modules using support vector machines and probabilistic neural networks," in *2011 Malaysian Conference in Software Engineering*, 2011.

[12] 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.

[13] S. Wang, H. Ping, L. Zelin and others, "An enhanced software defect prediction model with multiple metrics and learners," *International Journal of Industrial and Systems Engineering,* vol. 22, pp. 358-371, 2016.

[14] C. W. Yohannese, T. Li, M. Simfukwe and F. Khurshid, "Ensembles based combined learning for improved software fault prediction: A comparative study," in *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2017.

[15] Fukushima, "An empirical study of just-in-time defect prediction using cross-project models," 2014.

[16] Y. Zhang, D. Lo, X. Xia and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, 2015.

[17] S. Maheshwari and S. Agarwal, "Three-way decision based Defect Prediction for Object Oriented Software," in *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, 2016.

[18] R. A. Coelho, F. RN Guimarães and A. A. A. Esmin, "Applying swarm ensemble clustering technique for fault prediction using software metrics," in *2014 13th International Conference on Machine Learning and Applications*, 2014.

[19] T. Wang, Z. Zhang, X. Jing and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering,* vol. 23, pp. 569-590, 2016.

[20] "Bug prediction dataset," [Online]. Available: http://bug.inf.usi.ch/index.php. [Accessed Jan 2019].

[21] "Eclipse Bug Data," [Online]. Available: https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/. [Accessed April 2019].

[22] "PROMISE Software Engineering Repository," [Online]. Available: http://promise.site.uottawa.ca/SERepository/. [Accessed Jan 2019].

[23] L. H. Son, N. Pritam, M. Khari, R. Kumar, P. T. M. Phuong, P. H. Thong and others, "Empirical Study of Software Defect Prediction: A Systematic Mapping," *Symmetry,* vol. 11, p. 212, 2019.

[24] Y. Peng, G. Kou, G. Wang, W. Wu and Y. Shi, "Ensemble of software defect predictors: an AHP-based evaluation method," *International Journal of Information Technology & Decision Making,* vol. 10, pp. 187-206, 2011.

[25] Z.-W. Zhang, X.-Y. Jing and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Software Engineering,* vol. 24, pp. 47-69, 2017.

[26] P. Singh, "Comprehensive model for software fault prediction," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*, 2017.

[27] D. A. Reynolds, T. F. Quatieri and R. B. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital signal processing,* vol. 10, pp. 19-41, 2000.

[28] L. Li, S. Lessmann and B. Baesens, "Evaluating software defect prediction performance: an updated benchmarking study," *arXiv preprint arXiv:1901.01726,* 2019.

[29] K. E. Bennin, J. Keung, A. Monden, P. Phannachitta and S. Mensah, "The significant effects of data sampling approaches on software defect prioritization and

classification," in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2017.