Multi-Classifier Model for Software Fault Prediction

Pradeep Singh[1] and Shrish Verma[2]
[1]Department of Computer Science and Engineering, National Institute of Technology, Raipur
[2]Department of Electronics and Telecommunication Engineering, National Institute of Technology, Raipur

Abstract: *Prediction of fault prone module prior to testing is an emerging activity for software organizations to allocate targeted resource for development of reliable software. These software fault prediction depend on the quality of fault and related code extracted from previous versions of software. This paper, presents a novel framework by combining multiple expert machine learning systems. The proposed multi-classifier model takes the benefits of best classifiers in deciding the faulty modules of software system with consensus prior to testing. An experimental comparison is performed with various outperformer classifiers in the area of fault prediction. We evaluate our approach on 16 public dataset from promise repository which consists of National Aeronautics and Space Administration(NASA) Metric Data Program (MDP) projects and Turkish software projects. The experimental result shows that our multi classifier approach which is the combination of Support Vector Machine (SVM), Naive Bayes (NB) and Random forest machine significantly improves the performance of software fault prediction.*

Keywords: *Software metrics, software fault prediction, machine learning.*

Received February 7, 2015; accepted September 7, 2015

1. Introduction

As software systems are becoming part of the human life, the developments of these systems are getting more complicated. Development of quality system is desirable and due to the limited budget and the complexity of testing, complete and exhaustive software testing is not possible for generation of nearly fault free software. Hence, to reduce the cost and to generate nearly error free software, identification of fault-prone module prior to system testing can guide the software manager for resource allocation to appropriate modules. The ultimate goal of designing these fault identification system is to predict the best possible fault prone modules.

Generally, different machine learning techniques have been used for fault prediction. But there is no single machine learner is always best [9]. So, for critical application, an effective technique is required to achieve best results and combining the best classifier is one possible solution. The results from the combination of multiple classifiers may generate a better prediction than the individuals concerned. These observation motivated researchers in combining the learners to get decision on consensus of each classifier.

In Software fault identification we have software module with fault data, suppose (x_i, y_i) for i = 1. . .N software modules, with $x_i \in R^d$ and $y_i \in$ {Faulty, Not Faulty}. Our objective is to design an efficient learner S such that $R^d \Rightarrow R^2$. In other words there is an unknown function S which efficiently transforms x to y. Various classifier combination schemes have been devised and it has been experimentally demonstrated that some of them consistently outperforms than a single best classifier. The main benefits of combining classifiers are efficiency and accuracy. Majority voting is a combination strategy of classifiers and widely used by researcher [19]. An extensive study on the possibilities of classifiers input and attribute space is presented in [6, 8]. Various studies have investigated the software fault prediction using different machine learning techniques. These techniques have performed well on some data set but failed to perform on other data set. In order to build a consistent model which can efficiently perform on all different data set ensemble approach is one possible solution. We demonstrate our model on CM1, KC1, PC1, KC2, JM1, MC1, PC2, PC5, PC1, PC3, PC4, CM1, MW1, KC1, KC3, KC4, MC2, AR1, AR3, AR4 and AR5 software fault data. These include software metrics at method level.

Our approach uses software metrics data and its related fault status. We have taken three best performer classifiers reported by the researches of this field [2, 3, 9, 11]. The classifier combination schemes are then compared with the individual experimentally. A surprising outcome of the comparative study is that the combination of multiple machine learners outperforms with other classifier schemes. To explain this empirical finding, we investigate the accuracy and Area Under Curve (AUC) of various schemes to estimation fault prediction capability. We have used t - test at 0.05 level of significance to identify the significant difference between the prediction performance of our model and the other compared models.

The rest of the paper is organized as follows, section 2 describes the related work in the area, section 3 discusses about software project and their corresponding description. Section 4 presents the combination of classifier scheme section 5 provides the results and discusses their comparison with standard machine learners and section 6 offers conclusion with a summary and future work.

2. Related Work

Several papers are presented about machine learning techniques for software fault prediction [17, 18]. Some of these papers discussed methods for fault prediction using size and complexity metrics by applying Naïve Bayes, K-NN, decision tree and Bayesian belief networks [4]. Radjenović *et al.* [12] presented a broad literature survey on software fault prediction based on metrics. When building a software fault prediction model, the software metrics are processed for each module and then associated with number of faults in each module. The machine learners are used to learn the pattern to predict the faults for new modules.

Naïve Bayes (NB) is used by Menizes *et al.* [11] for fault prediction. Area under receiver Operating Characteristic (AUC) as a predictor evaluator is used by Lessmann *et al.* [9] for an extensive study on 10 project data of National Aeronautics and Space Administration (NASA) Metric Data Program (MDP). They evaluated prediction based on Area Under ROC (AUC) using static code attributes. In this paper, we used AUC and accuracy evaluators to evaluate the proposed prediction algorithms.

A group of researchers conducted manual software reviews to find defective modules. They found that approximately 60% of defects can be detected manually [11]. Prediction of faulty module is an efficient technique because the software metrics for any project can be collected easily by processing the source code automatically. On the other hand traditional methods like manual code reviews are labour intensive and only eight to twenty lines of code per minutes can be inspected by humans [15].

In fault prone module prediction the machine learners are trained using metric and fault related to that module.

Arisholma *et al.* [1] evaluated a large Java legacy system project for faulty modules. They reported that process metrics are very useful for fault prediction, and the best model is highly dependent on the performance evaluation parameter. They proposed cost effectiveness measure for assessment of models.

Catal and Diri [2] focus on various issues like selection of metrics, efficient predicator for small data sets based on machine learning such as Random Forests and Artificial Immune Systems. They used Public NASA datasets.

In their study they shown Random Forests provides the best prediction performance for large datasets and Naive Bayes is the best prediction algorithm for small datasets in terms of the Area under Receiver Operating Characteristics Curve (AUC) evaluation parameter. Other author Lessman *et al.* [9] also agrees with the same.

Elish *et al.* [3] empirically evaluated the capability of Support Vector Machine (SVM) on four dataset NASA dataset in predicting defect-prone software modules and compared its prediction performance against eight machine learning and statistical models. Their results indicate that the prediction performance of SVM is generally better than, or at least, is competitive against the compared models.

So we have taken Random forest, NB and Support vector machine.

Turhan and Bener [20] showed that independence assumption in Naive Bayes algorithm is not detrimental with Principal Component Analysis (PCA) pre-processing and they used Probability of Detection (PD), Probability of False alarm (PF) and balance parameters in their study. Numerous software fault prediction approaches have been proposed by various researchers as tree based learners, Logistic Regressions, Naive Bayes, Case-based Reasoning, and random forest.

However, the performance of fault prediction models varies and it depends on the machine learning algorithm used. In order to generate an efficient fault prediction model using software attributes and the module class (i.e., Module is faulty or not), a combination strategy is developed to measure the prediction accuracy of various software's. These software's were developed by NASA and its attribute and fault related datasets are available in PROMISE repository. Table 1 shows a summary of the fault datasets.

Various researcher have investigated the relationship between various feature reduction method and the resulting classification performance on software fault prediction and quality models [10, 11]. Singh and Verma [16] have utilized two different models of machine learning: J48 (a decision tree) and NB algorithm on open source project developed in object oriented language They used the object oriented design metric CK and reported the prediction accuracy of 98.15% and 95.58% respectively. As indicated by Lessmann *et al.* [9] and other researchers that sophisticated techniques are well prepared to cope pre-processing and feature selection through inbuilt regularization facilities. Our model include number of classifiers that are sophisticated approaches, it seems unlikely that pre-processing activities would alter our overall predictive accuracy significantly. So in this investigation pre-processing activity is not used for our model preparation.

3. Experimental Dataset

In order to find a consistent classifier, we propose a combination of classifiers as a solution [7, 8] and conducted extensive comparative study on 16 benchmark dataset from public-domain data sets from the PROMISE repository [13]. Furthermore, we apply state-of-the-art hypothesis testing methods to validate the statistical significance of performance differences among different classification models [13].

The software projects used for this study are CM1, KC1, KC2, KC3, PC1, PC2, PC3, PC4, MC2, MW1, JM1, AR1, AR3, AR4, AR5, and AR6. The data used in this study originates from NASA and Turkish software industry projects. Each data set is consisting of software modules, together with label as fp, for fault prone whereas error-free modules were categorized as nfp (not fault prone). Table 1 shows the brief details of the various projects.

Table 1. Datasets used in this study.

s	Not Faulty Module	Faulty Modules	Total no of Modules in Software	Features	% Defective	Language
CM1	449	49	498	21	9.84	C
KC1	1783	326	2109	21	15.46	C++
KC2	415	107	522	21	20.5	C++
KC3	415	43	458	39	9.39	Java
PC1	1032	77	1109	21	6.94	C
PC2	5566	23	5589	36	0.41	C
PC3	1403	160	1563	37	10.24	C
PC4	1280	178	1458	37	12.21	C
MC2	109	52	161	39	32.3	C++
MW1	372	31	403	37	7.69	C
JM1	8779	2106	10885	21	19.35	C
AR1	112	9	121	29	7.44	C
AR3	55	8	63	29	12.7	C
AR4	87	20	107	29	18.69	C
AR5	28	8	36	29	22.22	C
AR6	86	15	101	29	14.85	C

The software metrics of various projects are of broadly LOC based Halsted metrics, Mc-Cabe metric and few other metrics. Readers can get the details of each project and software metrics used in [11, 13].

4. Development of Predictive Model

To develop a predictive model we have used three best performers as reported by the researches. These are support vector machine, Random forest and NB [2, 3, 9, 14].

SVM: The SVM is a novel machine learning technique based on a statistical learning theory proposed by the Vapnik in 1995 [21]. SVM has gained attention and introduced in various areas like pattern recognition, regression estimation, handwriting identification and object tracing etc., Based on the structural risk minimization concept, it can minimize the probability of misclassifying a previously unseen data. SVMs were originally linear binary classifiers, which allocate the labels +1and−1. The core operation of SVMs is to construct a separating hyper plane (i.e., a decision boundary) on the basis of the properties of the training samples, specifically their distribution in feature space.

Suppose a training sample set, $W=\{(x_i, y_i),\ i=1,\ldots n\}$ an input sample, $x_i \in R^d$, and class lables, $y_i \in \{+1, -1\}$, Where x_i is the feature and y_i is the label of the information class for training case i. The label is +1 or-1, representing class faulty and class not faulty for a linearly separable binary classification problem, the separation hyper plane is:

$$x_i.w + b = 0 \tag{1}$$

Where w is a weight vector and b is a bias. The optimal hyper plane that separates the data into two classes with the decision boundary

$$y_i[(x_i.w) + b] \geq 1 \tag{2}$$

Which minimizes

$$\varphi(w) = \frac{1}{2}||w||^2 = \frac{1}{2}(w.w) \tag{3}$$

Information classes derived from software fault data are not usually totally separated by liner boundaries. Thus constraint shown in Equation (3) cannot be satisfied in practice, so the slack variables, are used to get

$$y_i[(x_{i.}w) + b] \geq 1 - \xi_i \tag{4}$$

Where i=1, 2,…l, Now the problem becomes:

$$\varphi(w,\xi) = \frac{1}{2}(w.w) + c\left[\sum_{i=1}\xi\right], i = 1,\ldots,l \tag{5}$$

Where C is a penalty parameter determined by the user. A large value of C means assign a high penalty to error. Introducing Lagrange multipliers, α_i and using the Karush-Kuhn-Tucker theorem of optimization gives the solution as follows:

$$w = \Sigma \alpha_i y_i x_i \tag{6}$$

Only a few of the α_i coefficients are nonzero. The corresponding x_i values are known as support vectors, and they also define the decision boundary. At the same time, all other training samples with zero α_i values are now rendered irrelevant. Finally, the decision function can be obtained as follows:

$$f(x) = sign\left(\sum_{i=1}^{l} y_i \alpha_i (x_i, x) + b\right) \tag{7}$$

In some case linear hyper plane is unable to separate the class appropriately. In such cases SVM maps the raw input data into a higher dimensional space to improve the seperability between the classes. A transformation, $\Phi\ (x_i)$, maps the data from the input space to a feature space that allows linear separation.

We then seek the optimization separation plane in feature space. An inner product operation is needed in feature space. The inner product operation may be implemented by a certain function (called a kernel function). In SVM, a kernel function $K(x_i,x_j)=\Phi(x_i)$. $\Phi(x_j)$ used to reduce the computational loading. Then the basic form of SVM can be obtained:

$$f(x)=sign\left(\sum_{i=1}^{l} y_i\alpha_i K(x_i,x)+b\right) \quad (8)$$

The common choice for kernel functions is Linear, Polynomial and Gaussian radial basis function. In this experiment we have taken Gaussian radial basis function as kernel function in this study.

- *Random forest.* A random forest uses a large number of individual, unpruned decision trees which are created by randomizing the split at each node of the decision tree. Each tree is likely to be less accurate than a tree created with the exact splits. But, by combining several of these approximate trees in an ensemble, can improve the accuracy. Random forest and NB are well known classifiers so the details are left here.

Classifier combining strategy: Combining multiple classifiers can be performed in various ways. In multi classifier combination method various learners work in parallel.

Let X = {x_1, x2,…,x_p) be a set of input vector in R^p, x_i represents a software module that assigned the software metric at design and $Y=(y_1,y_2)$ in R^2 denotes its binary class labels as output vector. Suppose that the new multiclassifier is the ensemble D = {D_1 ,…,D_L). Each classifier D_i produce support denoted by $d_{i,,j}(x)$ to the hypothesis that x comes from y_i. The larger the support, the more likely the class label is y_i. The output of L classifiers is organized in a decision profile as follows.

$$\begin{array}{l} \\ \text{Classifier 1} \\ \text{Classifier i} \\ \text{Classifier L} \end{array}
\left(\begin{array}{cc} \multicolumn{2}{c}{\text{Class labels}} \\ d_{1,1}(x) & d_{1,2}(x) \\ d_{i,1}(x) & d_{i,2}(x) \\ d_{L,1}(x) & d_{L,2}(x) \end{array}\right)$$

The combiner calculates the support for class y_i using only the ith column of decision profile. In our case there are two classes. These outputs are combined by the way of averaging these using the following typical average estimator.

$$u_j(x)=\frac{1}{L}\sum_{i=1}^{L} d_{i,j}(x) \quad (9)$$

The reason for choosing this average strategy is that this method will work even if the number of the classifiers in the ensemble is even; it is possible to classify the faulty and fault free modules. After calculating the overall support for each class the input x is labeled with the largest average support. The multiclassifier system is first trained using training data. As soon as the all base classifiers are trained the test data of new company is then provided to muticlassifier system to predict the faults of the later project. The combiner calculates the support for faulty and not faulty class for each classifier. The classifier outputs for a particular input are organized as decision profile. Each classifier model outputs the support values indicating the probability of a module belonging to faulty or not faulty classes.

Figure 1 shows the detailed process of our multi - classifier system which is based on average probability.

4.1. Measuring Performance

For two class problems a variety of measures has been proposed and there are four possible cases represented as confusion matrix which gives the TP, FP and FN and TN as shown in Table 2. TP and TN are the correctly identified faulty and not faulty modules. A FP occurs when outcome is predicted yes when it is actually not faulty. A FN occurs when the when the outcome is incorrectly predicted as negative when it is actually positive.

Table 2. Confusion matrix.

		Actual	
Predicted		**Faulty Module**	**Not Faulty Module**
	Faulty Module	TP(True positive)	FP (False positive)
	Not Faulty Module	FN (False negative)	TN (True Negative)

The Accuracy is calculated as the number of correct classifications divided by the total number of classifications:

$$Accuracy=\frac{TP+TN}{(TP+TN+FP+FN)} \quad (10)$$

For the purpose of reliable and stable results in the experiments, K fold cross validation strategy was used. K fold classifier is generally used for classification accuracy measure in this we have to make K partitions and one is used for testing and rest is used for training.

$$\begin{array}{ll} V_1=X_1 & P_1=X_2\cup X_3\cup\ldots\cup X_k \\ V_2=X_2 & P_2=X_1\cup X_3\cup\ldots\cup X_k \\ & \ldots \\ V_k=X_k & P_k=X_1\cup X_2\cup\ldots\cup X_{k-1} \end{array}$$

In the above, V_1, V_2, …V_k are the partitions for testing and P_1, P_2 ,…, P_K are for training. K is typically 10 or 30. K=10 has been used for all our experiments.

To compare the results we have taken the average value of AUC and accuracy in 10 fold cross validation. As it is known and also reported by other research that AUC represents the most informative and objective indicator of predictive accuracy [5, 20].

A better classifier should produce a higher AUC. We have also used AUC for our study.

To evaluate our composite classifier with individual classifier we have used statistical significance of performance using the corrected resampled t-test at 95% confidence level (0.05 level of significance).

The results reported in the 'Significance' columns of Tables 2 and 3 of the corrected resampled t-test. In these columns, *means that there is a significant performance difference between our model and the corresponding model, and o is used if other model has significantly performed better. The algorithm shown below gives the procedure used for model building and evaluation on various datasets.

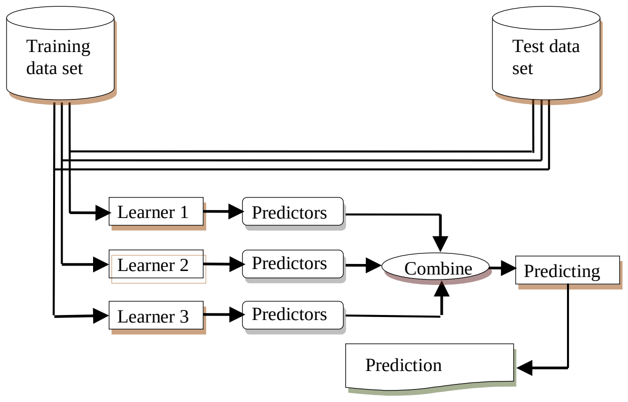

Figure 1. Proposed multi-classifier system for fault prediction.

Algorithm 1: (Detection of faulty modules in software projects)

```
Input: Project= [CM1, KC1, PC1, KC2, JM1, MC1, PC2, PC5, PC1, PC3, PC4, CM1, MW1, KC1, KC3, KC4, AR3 ,MC2, AR1, AR4 and AR5 ]
Output: Fault prediction capability in terms of area Under the Curve, Accuracy
M=10;
N=10;
Meta Learner [Combined Multiple Classifiers (SVM, Naïve Bayes, Random Forest)]
Learner Algorithms =Learner [Metal earner, SVM, Naïve Bayes, Random Forest]
For ALL Projects
For 1=1 to M
R=Randomize the data of Project
T=Generate N parts of Project R
For J =1 to N do
Test=T[j]
Train=T- T[j]
        For Each L Є learner
        Model=Apply L on Train
        Prediction L=Apply model to test
        End for
End for
End for
Prediction=Learner
(Pred_Meta,Pred_NB,Pred_SVM,Pred_RF)
End for
```

5. Result

Table 3 summarizes the results for all sixteen data sets considered in this study by applying 4 learners and 10 accuracies for each case. In the first column we include the data sets, while in the second column we list the correctly classified by our proposed multi expert model which is combination of three classifiers. In the remaining columns of this table we report the result by Naïve Bayes, SVM and Random forest respectively.

Table 3. Comparative result on accuracy values of proposed model with different best learners.

Dataset	Our Model	Naïve Bayes	SVM	RF
CM1	88.34	85.32*	90.16	87.93
KC1	86.06	82.36 *	84.54 *	85.44
KC2	83.72	83.52	79.70 *	82.18
KC3	89.53	85.16*	90.61	89.09
PC1	93.42	89.18 *	93.06	93.24
PC2	99.55	97.30 *	99.59	99.55
PC3	89.64	48.67 *	89.76	89.89
PC4	89.71	87.04*	87.79 *	91.02
MC2	71.40	73.86	67.72	68.93
MW1	92.55	83.87 *	92.31	90.81*
JM1	81.3	80.42 *	80.65 *	81.14
AR1	90.06	85.06	92.56	89.23
AR3	90.71	90.48	87.38	89.05
AR4	85.18	84.27	81.27	85.18
AR5	78.33	84.17	78.33	80.83
AR6	86.18	82.27	85.18	87.18

From the result Table it is evident that our multi classifier system is significantly outperforming than Naïve Bays on 9 out of 16 data set on the basis of t-test at 0.05 level of significance. Our multi- classifier is significantly outperforming on 4 dataset in comparison with the SVM. Also it can be seen from the result that even if the some values are not significant but most of the time it is outperforming than the single classifiers. Also it is found that none of the result for any data set is significantly outperformed by our model. From the graph in Figure 2 we can see the result of multi classifier have the better output in comparison with NB and SVM.

Table 4 summarizes the results for all sixteen data sets considered in this study using area under Receiver Operating Characteristic ROC (AUC). In the first column we include the data sets, while in the second column we list the correctly classified by our proposed multi classifier model.

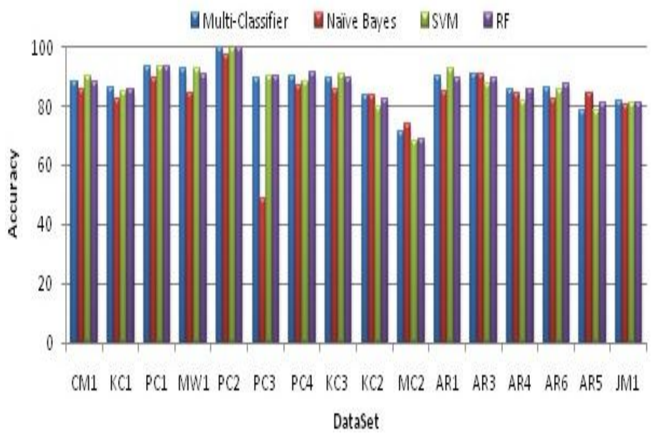

Figure 2. Performance measure using accuracy.

Table 4. Comparative result on AUC values of proposed model with different best learners.

Dataset	Our Model	Naïve Bayes	SVM	RF
CM1	0.77	0.73	0.50 *	0.72
KC1	0.81	0.79	0.50 *	0.79 *
KC2	0.82	0.85	0.50 *	0.80 *
KC3	0.81	0.82	0.50 *	0.76
PC1	0.83	0.71 *	0.50 *	0.81
PC2	0.82	0.82	0.50 *	0.61 *
PC3	0.8	0.77 *	0.50 *	0.79
PC4	0.92	0.84 *	0.50 *	0.93
MC2	0.77	0.71	0.50 *	0.72
MW1	0.78	0.76	0.50 *	0.73
JM1	0.73	0.69 *	0.50 *	0.72 *
AR1	0.85	0.67	0.50 *	0.81
AR3	0.76	0.78	0.50	0.77
AR4	0.85	0.8	0.50 *	0.81
AR5	0.96	0.92	0.50 *	0.85
AR6	0.67	0.66	0.50	0.63

In the remaining columns of this table we report the result by Naïve Bayes, SVM and Random forest respectively. It is evident from the results that our multi-classifier model is significantly outperforming than Naïve bays on 4 data sets. If we see the results our model has significantly outperformed SVM on 14 dataset out of 16 dataset. Also our model has significantly outperformed on 4 dataset in comparison with the RF. It can be seen from the result that even if the values is not significant but most of the time our model is better than the single classifiers.

From the graph in Figure 3 we can see the result of multi-classifier have the better output in comparison with other classifiers. It can be seen that the in AUC the performance of SVM is not at acceptable level. By the graph it is evident that the performance of our model is best and consistent and reliable than the other best model. Our model is consistently outperforming than the other single classifier based models.

Finally we have compared the result of our model with other published result on the same data set.

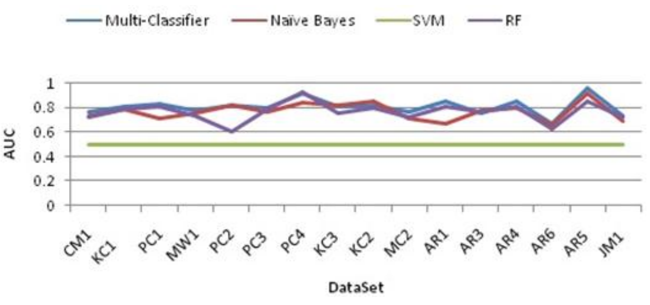

Figure 3. Performance measure using AUC.

Table 5. Comparative study with catal and Diri [2].

		Results by Catal and Diri [2]			
Dataset	**Our Model**	**Naïve Bayes**	**RF**	**AIRS1**	**AIRS2**
CM1	0.77	0.73	0.72	0.55	0.53
KC1	0.81	0.79	0.79	0.60	0.57
KC2	0.82	0.84	0.80	0.56	0.67
PC1	0.83	0.71	0.81	0.55	0.57
JM1	0.73	0.69	0.72	0.55	0.54

Table 5 shows the comparison between results of our model and results reported by Catal and Diri [2]. We have compared with the results of first experiment of which uses same 21 metrics in and five dataset. From the results shown in Table 5 it can be seen that our proposed model is outperforming with other classifiers in AUC. The result of our model is excellent in comparison with Artificial Immune Recognition Systems (AIRS 1 and AIRS2) used by Catal and Diri [2] To summarize, the above experiment results show that the performance of single learner for each dataset is distrustful; this means the prediction performance is unlikely to be good for each dataset. The proposed framework is an improved approach. The prediction performance of the proposed framework is higher than that of all best performers. This indicates that the proposed framework performed consistently better for each data set which intern reduce the time of search of best classifiers for different data sets.

6. Conclusions

A novel and efficient model for software fault prediction has been presented. Sixteen standard fault dataset of software's have been analyzed. The proposed model is combination of Naïve Bayes, SVM and Random forest. The fault prediction capability of our model is compared with previously reported approaches. The proposed model has better prediction capability than SVM, Random forest and Naïve Bayes, AIRS1, AIRS2. It is also clear from the results that the combination of learner can give better result than a single classifier. Identification of which classifier is best fault predicator is tough task. So the proposed model has consistently performed better as it takes the best of all three classifiers. By the results it is evident that the combination of learner not only increases the fault prediction capability as well as it also provides consistent better result for all dataset. It also reduces the search for the best classifier for the different dataset. The accuracy of the proposed method is excellent for all the fault data set in term of AUC and accuracy. Thus the proposed approach is efficient, robust and consistent in case of software fault prediction for various software projects.

References

[1] Arisholma E., Briand L., and Johannessen E., "A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2-17, 2010.

[2] Catal C. and Diri B., "Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction

Problem," *Information Sciences*, vol. 179, no. 8, pp. 1040-1058, 2009.

[3] Elish K. and Elish M., "Predicting Defect-Prone Software Modules Using Support Vector Machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649-660, 2008.

[4] Hall T., Beecham S., Bowes D., Gray D., and Counsell S., "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Transactions* on *Software Engineering*, vol. 38, no. 6, pp. 1276-1304, 2012.

[5] Jiang Y., Cukic B., Menzies T., and Bartlow N., "Comparing Design and Code Metrics for Software Quality Prediction," *in Proceedings of the 4th international Workshop on Predictor Models in Software Engineering*, Leipzig, pp. 11-18, 2008.

[6] Kimura F. and Shridhar M., "Handwritten Numerical Recognition Based on Multiple Algorithms," *Pattern Recognition*, vol. 24, no. 10, pp. 969-983, 1991.

[7] Kittler J., Matas J., Jonsson K., and Sánchez M., "Combining Evidence in Personal Identity Verification Systems," *Pattern Recognition Letters*, vol. 18, no. 9, pp. 845-852, 1997.

[8] Kittler J., Hatef M., Duin R., and Matas J., "On Combining Classifiers," *IEEE Transactions Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226-239, 1998.

[9] Lessmann S., Baesens B., Mues C., and Pietsch S., "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, 2008.

[10] Marchetto A. and Trentini A., "A Framework to Build Guality Models for Web Applications," *The International Arab Journal of Information Technology*, vol. 4, no. 2, pp. 168-176, 2007.

[11] Menzies T., Greenwald J., and Frank A., "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, 2007.

[12] Radjenović D., Heričko M., Torkar R., ans Živkovič A., "Software Fault Prediction Metrics: A Systematic A Systematic Literature Review, *Information and Software Technology*," vol. 55, no. 8, pp. 1397-1418, 2013.

[13] Sayyad S. and Menzies T., "The PROMISE Repository of Software Engineering Databases," *School of Information Technology and Engineering., University of Ottawa*, http://promise.site. uottawa.ca/SERepository, Last Visited, 2005.

[14] Sebastiani F., "Machine Learning in Automated Text Categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1-47, 2002.

[15] Shull F., Basili V., Boehm B., Brown A., Costa P., Lindvall M., Port D., Rus I., Tesoriero R., and Zelkowitz M., "What We Have Learned About Fighting Defect," *in Proceedings of 8th International Software Metrics Symposium*, Ottawa, pp. 249-258, 2002.

[16] Singh P. and Verma S., "Empirical Investigation of Fault Prediction capability of Object Oriented Metrics of Open Source Software," *in Proceedings of 8th International Conference on Computer Science and Software Engineering*, Bangkok, pp. 323-327, 2012.

[17] Singh P. and Verma S., "An Investigation of the Effect of Discretization on Defect Prediction Using Static Measures," *in Proceedings of International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Trivandrum, pp. 837-839, 2009.

[18] Singh P. and Verma S., "Effectiveness Analysis of Consistency Based Feature Selection in Software Fault Prediction," *International Journal of Advancements in Computer Science and Information Technology*, vol. 2, no. 1, pp. 1-9, 2012.

[19] Tumer K. and Ghosh J., "Analysis of Decision Boundaries in Linearly Combined Neural Classifiers," *Pattern Recognition*, vol. 29, no. 2, pp. 341-348, 1996.

[20] Turhan B. and Bener A., "Analysis of Naive Bayes' Assumptions on Software Fault Data: An Empirical Study," *Data Knowledge Engineering*, vol. 68, no. 2, pp. 278-290, 2009.

[21] Vapnik V., *The Nature of Statistical Learning Theory*, Springer Science and Business Media, 2013.

Pradeep Singh is with the Department of Computer Science Engineering as Assistant Professor at National Institute of Technology, Raipur. He has completed his M.Tech. from Motilal Nehru National Institute of Technology (MNNIT) Allahabad, India with specialization in Software Engineering. He has completed his PhD in Computer Science and Engineering from National Institute of Technology Raipur His current research interests include empirical studies on software quality, software fault prediction models, and computational intelligence. He has more than ten years experience in various government academic institutes. He has published over 12 referred articles and served as reviewer of several journals including Knowledge Based System. He is a Member of IEEE, CSI and the ACM.

Shrish Verma is Professor in the department of Electronics and Telecommunication, National Institute of Technology, Raipur. He has completed his Post graduation in Computer Engineering from Indian Institute of Technology, Kharagpur. He has completed his PhD in Engineering from Pt. Ravi Shankar Shukla University Raipur. His area of interest is Image processing, data mining, Software fault prediction models and Software bug classification. He has published over 20 referred articles and served as reviewer of several journals.