International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.2, January 2013 – www.ijais.org

Effect of Ensemble Methods for Software Fault Prediction at Various Metrics Level

Shanthini. A
Research scholar,
Department of Computer Science and Engineering,
Annamalai University,
Annamalai nagar, Tamil Nadu, India

Chandrasekaran.RM
Professor,
Department of Computer Science and Engineering,
Annamalai University,
Annamalai nagar, Tamil Nadu, India

ABSTRACT
Defective modules in software project have a considerable risk. It reduces the software quality. Defective modules decreases customer satisfaction and by increases the development and maintenance costs. In software development life cycle, it is very essential to predict the defective modules in the early stage so as to improve software developers' ability to focus on the quality of the software. Software defect prediction using machine learning algorithms was investigated by many researchers and concluded that classifiers ensemble can effectively improve classification performance than a single classifier. This paper mainly addresses the software fault prediction using ensemble approaches. We conduct a comparative study using WEKA tool for various ensemble methods with perspective of taxonomy. The ensemble methods include Bagging, Boosting, Stacking, and Voting. We also compared these ensemble methods for three different levels of software metrics (Class level, Method level and Package level). Ensemble classifiers were examined for various metrics level datasets. Various ensemble classifiers were examined for three different metrics levels. The experiments were carried out on the datasets such as NASA KC1 method level data set, NASA KC1 class level dataset and Eclipse dataset for package level metrics. The experiments conducted on these three data sets by applying ensemble classification methods to predict defect. The ensemble methods evolved by experiments shows that bagging performs better than other ensemble methods for method level and package level dataset. For class level dataset voting performs better in terms of Area under ROC curve (AUC – ROC).

Keywords
Defect prediction, Classifier Ensemble; Ensemble Methodology.

1. INTRODUCTION
Now-a-days the software systems are getting more and more complex. Due to this complexity, the probability of these software systems having defective modules is getting higher. Software quality assurance is a resource and time-consuming task. Since cost is a vital factor, complete testing of an entire system is not easy. Therefore, identifying which software modules are more likely to be defective can help us allocate limited time and resources effectively. Based on the above reasoning, it is clear that methods are needed to predict, control and improve fault handling in general [13]. The type of methods can be divided into two major classes: methods for prediction of the number of faults in a specific module and methods for identification of fault-prone modules. The first type of methods has been investigated and evaluated, but it has been difficult to develop a valid model, in particular a model which is transferable between projects or organizations. Thus, methods for identification of fault- and failure prone modules and models for fault prediction are a potential way to improve software quality and to reduce cost [11].

Data Mining has become a very useful technique to reduce information overload and improve decision making by extracting and refining useful knowledge through a process of searching for relationships and patterns from the extensive data collected by organization [6] [8]. The extracted information is used to predict, classify, model, and summarize the data being mined. In recent years data mining techniques have been successfully used in software fault detection. The primary objective of this paper is to show that Bagging and Voting had better performance than stacking and boosting.

The rest of this paper is organized as follows. Subsequent sections describe related work. The 3rd section is for the data source. The 4th and 5th sections present details about metrics and machine learning approaches used. Performance evaluation is discussed in the 6th section. Conclusions are given in the 7th section.

2. REVIEW OF RELATED LITERATURE
Considerable research has been performed on software metrics and defect prediction models. In [4], the author has used various machine learning techniques for an intelligent system for the software maintenance prediction and proposed the logistic model Trees (LMT) and Complimentary Naïve Bayes (CNB) algorithms on the basis of Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Accuracy percentage.

Catal *et al.* [5] examined Chidamber-Kemerer metrics suite and some method-level metrics (the McCabe's and Halstead's ones) for a defect model which is based on Artificial Immune Recognition System (AIRS) algorithm. The authors investigated together 84 metrics from the method-level metrics transformation and 10 metrics from the class-level metrics. According to obtained results the authors concluded that the best fault prediction is achieved when CK metrics are used with the lines of code (LOC) metric.

Menzies *et al.* [6] showed that Naive Bayes with logNums filter provides the best performance on NASA datasets for software fault prediction problem. Olague *et al.* [7] found that the complexity metrics have a good performance in distinguishing between fault-prone and not fault-prone classes. In addition, they also found that lesser known metrics

such as SDMC and AMC were better predictors than the commonly used metrics LOC and WMC.

Kanmani *et al.* [8] validated Probabilistic Neural Network (PNN) and Back propagation Neural Network (BPN) using a dataset collected from projects of graduate students, in order to compare their results with results of statistical techniques. According to Kanmani *et al.*'s [8] study, PNN provided better performance.

The use of Machine Learning for the purpose of predicting or estimating software module's fault-proneness is proposed by [11], which views fault-proneness as both a continuous measure and a binary classification task. Using a NASA public dataset, a NN is used to predict the continuous measure while a SVM is used for the classification task. Gondra's [11] experimental results showed that Support Vector Machines provided higher performance than the Artificial Neural Networks for software fault prediction.

3. FAULT PREDICTION DATA SET

Three different data sets used in this research are obtained from the bug database of Eclipse and NASA IV and V Facility Metrics Data Program (MDP). For method level metrics, NASA KC1 method level dataset is used. Public NASA KC1 method level dataset include 21 method-level metrics proposed by Halstead and McCabe. This dataset consists of 2109 instances and 22 attributes.

We use the data associated with the KC1 project for class level metrics. This is a real time project written in C++ consisting of approximately 315,000 LOC. There are 10,878 modules and 145 instances. In addition to the method level metrics, 10 class-level oriented metrics are used.

For package level metrics, the dataset is obtained from bug database of Eclipse 3.0. The dataset lists the number of pre- and post-release defects for every package in the Eclipse 3.0. All data is publicly available and used for defect prediction models. Dataset consists of following attributes. Each case contains the following information:

The **name** attribute indicates the name of the file or package, to which this case corresponds. It can be used to identify the source code and may be needed for additional data collection.

The **pre-release defects** attribute indicates the number of non-trivial defects that were reported in the last six months before release of the project.

The **post-release defects** attribute indicates the number of non-trivial defects that were reported in the first six months after release of the project.

The **complexity metrics** attribute indicates the metrics that are computed for classes or methods are aggregate by using average (avg), maximum (max), and accumulation (sum) to package level.

Structure of abstract syntax tree(s): For each case, we list the size (=number of nodes) of the abstract syntax tree(s) of the package.

4. ENSEMBLE METHODS

4.1. Bagging

Bagging is Bootstrap AGGregatING. The main idea of Bagging is to construct each member of the ensemble from a different training dataset, and to predict the combination by either uniform averaging or voting over class labels [3]. A bootstrap samples N items uniformly at random with replacement. That means each classifier is trained on a sample of examples taken with a replacement from the training set, and each sample size is equal to the size of the original training set. Therefore, Bagging produces a combined model that often performs better than the single model built from the original single training set. Bagging algorithm is shown in Figure 1.

Input :

Data set D= $\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$;

Base learning algorithm B;

Number of learning rounds R.

Process:

for $i = 1,\cdots,R$:

D_i = Bootstrap (D); /* Generate a bootstrap sample from D*/

$h_i = B(D_i)$; /* Train base learner h_t from the bootstrap sample */

end.

Output :

$O(x) = \arg\max\ y \in Y \sum_{t=1}^{T} l(y = h_i(x))$

/* the value of l(a) is 1 if a is true and 0 otherwise*/

Figure 1: Bagging Algorithm

4.2. Boosting

Boosting is another popular ensemble method, and Adaboost is the most well-known of the Boosting family of algorithms which trains models sequentially, with a new model trained at each round [1][6]. Adaboost constructs an ensemble by performing multiple iterations. In this process for each iteration it uses different example weights. The weight of incorrectly classified examples will be increased, so that it ensures misclassification errors for these examples count more heavily in the next iterations. This procedure provides a series of classifiers that complement one another, and the classifiers are combined by voting. Boosting algorithm is shown in the Figure 2.

Input :

Data set *D*= $\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$;

Base learning algorithm *B;*

Number of learning rounds *R.*

Process:

$D_1(j)=1/m$ //Initialize the weight distribution

for $i=1,2,\cdots,R$:

$h_i = B(D, D_1)$; / *Train base learner h_i from *D* using distribution D_1*/

$\varepsilon_i = \Pr_{j=Dj}[h_i(x_j \neq y_j)]$; /*Measure the error of h_i*/

$\alpha_i = \frac{1}{2}\ln\frac{1-\varepsilon_i}{\varepsilon_i}$; //Determine the weight of h_i

$$D_{i+1}(j) = \frac{D_i(j)}{Z_i} \times \begin{cases} \exp(-\alpha_i) if\, h_i(x_j) = y_j \\ \exp(\alpha_i)\, f\, h_i(x_j) \neq y_j \end{cases}$$

// Update the distribution, where Z_i is a

$$= \frac{D_i(j)\exp(-\alpha_i y_j h_i(x_j))}{Z_i}$$

/* Normalization factor with enables D_{i+1} to a distribution*/

end.

Out put :

$O(x) = sign(f(x)) = sign\sum_{i=1}^{R}\alpha_i h_i(x)$

Figure 2: Boosting Algorithm

Input :

Data set *D*= $\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$;

First-level learning algorithms $A_1,\cdots,A_T$;

Second-level learning algorithms A.

Process:

for $i=1,\cdots,T$:

$h_i = A_i(D)$ /* Train a first-level individual leaner h_i by applying the first-level */

end; // learning algorithm A_i to the original data set D

$D' = \phi$; // Generate a new data set

for $j=1,\cdots,m$:

$z_{ji} = h_i(x_j)$ /*Use h_i to classify the training example x_j*/

end;

$D' = D'U\{((Z_j1, Z_i2,\cdots Z_jT), y_j)\}$

end.

$h' = A(D')$. /* Train the second-level learner h' by applying the second-level learning algorithm A to the new set D' */

Output :

$O(x) = h'(h_1(x),\cdots,h_T(x))$

Figure 3: Stacking Algorithm

4.3. Stacking

Stacking is another ensemble learning technique. In Stacking scheme, there are two level models which are set of base models are called level-0, and the meta-model level-1. The level-0 models are constructed from bootstrap samples of a dataset, and then their outputs on a hold-out dataset are used as input to a level-1model. The task of the level-1 model is to combine the set of outputs so as to correctly classify the target, thereby correcting any mistakes made by the level-0 models[11] [3]. Stacking algorithm is shown in Figure 3.

4.4. Voting

Voting is a combining strategy of classifiers. Majority Voting and Weighted Majority Voting are more popular methods of Voting [16]. In Majority Voting, each ensemble member votes for one of the classes. The ensemble predicts the class with the highest number of vote. Weighted Majority Voting makes a weighted sum of the votes of the ensemble members, and weights typically depend on the classifiers confidence in its prediction or error estimates of the classifier.

5. EXPERIMENTS

We used the WEKA machine learning library as the source of algorithms for experimentation. We used bagging, boosting, stacking and voting ensemble algorithms as implemented in WEKA with default parameters. Bagging and boosting are implemented with default classifier in WEKA. The base classifiers used for voting are SVM and Navies Bayes which

are popular in software defect prediction [9]. The combination rule of vote is average of probabilities. The level 1 classifier of stacking is SVM and level 0 classifier used is Navies Bayes. The data sets described above in section 2 is used to test the performance of various ensemble methods. The performances of classifiers are evaluated used RMSE and AUC-ROC as metrics.

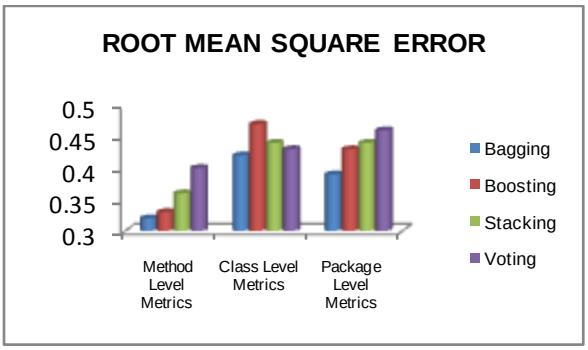

Figure 4: Root Mean Square Error for different level of metrics using various ensembles methods

5.1 Root Mean Square Error

Root Mean square error (RMSE) is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated. It is just the square root of the mean square error as shown in the equation given below.

Assuming that the actual output is *a*, expected output is *c*.

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}}$$

The datasets described in section 2 is being used to test the performance of various ensemble methods. Root Mean Square Error (RMSE) was evaluated using 10 fold cross validation as cross validation is the best technique to get a reliable error estimate. The Root Mean Square Error shown in the Figure 4 reflects the best performance of bagging in terms of classification rate. The error can be reduced to zero as the number of classifiers combined to infinity.

5.2 Area under Curve

The dataset described in section 2 is being to test the performance of various ensemble methods. We adapted ROC curve in our experiment to evaluate the performance of ensemble algorithms. Receiver operative Characteristics (ROC) curve is used as an additional alternative evaluation metric. AUC-ROC is used as a performance metrics (area under ROC curve), an integral of ROC curve with false positive rate as x axis and true positive rate as y axis. If ROC curve is more close to top-left of coordinate, the corresponding classifier must have better generalization ability so that the corresponding AUC will be larger. Therefore, AUC can quantitatively indicate the generalization ability of corresponding classifier. Figure 5, 6 and 7 shows the ROC curves evaluating the performance curve of various classifiers on the KC1 method level dataset, KC1 class level dataset and Eclipse package level dataset. Area under the ROC curve (AUC-ROC) is calculated using trapezoidal method and the result is shown in the Table1. From the ROC curves (AUC-ROC) it is evident that, for method level metrics and package level metrics bagging method gives better performance. For class level metrics voting method performs comparatively better than bagging method.

Table1: Performance of Area under ROC Curve for different metrics

METHODS	AREA UNDER ROC CURVE FOR		
	METHOD LEVEL METRICS	CLASS LEVEL METRICS	PACKAGE LEVEL METRICS
BAGGING	0.809	0.78	0.82
BOOSTING	0.783	0.74	0.78
STACKING	0.79	0.8	0.72
VOTING	0.63	0.82	0.76

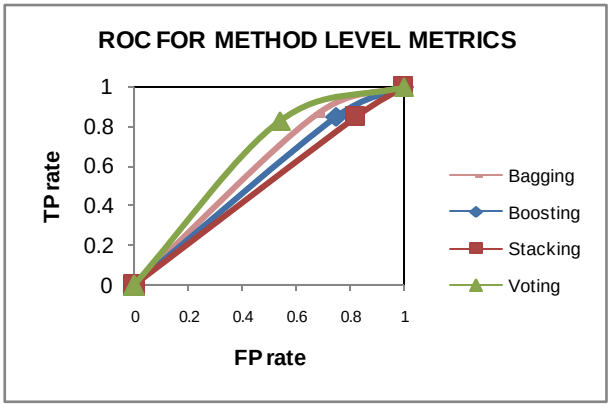

Figure 5: ROC for Method Level Metrics (KC1 method level dataset)

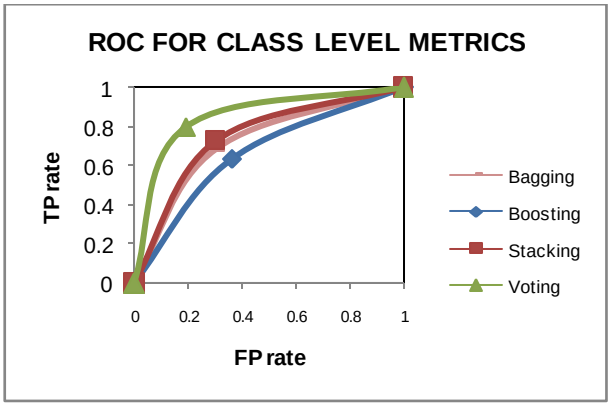

Figure 6: ROC for Class Level Metrics (KC1 class level dataset)

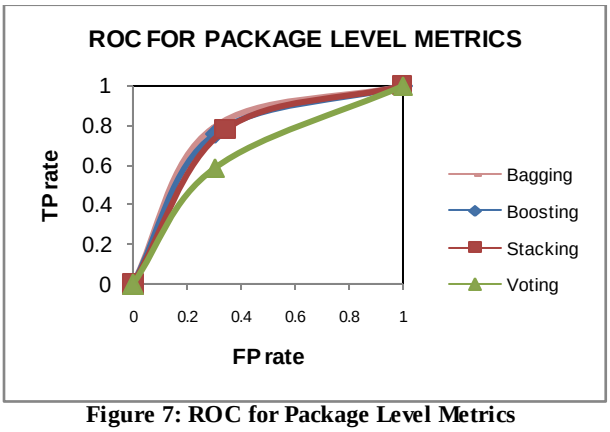

Figure 7: ROC for Package Level Metrics (Eclipse Dataset)

6. CONCLUSION

The goal of our research is to analyze the performance of various classifiers for various metrics level data set on defect prediction .We analyzed the performance of the classifiers using Root Mean Square Error. ROC is also used as an alternative metric. The area under ROC curves (AUC-ROC) is calculated by using the trapezoidal method. From the ROC curve (AUC-ROC) it is evident that, for three different metrics level dataset (KC1 method level dataset, KC1 class level dataset and Eclipse dataset) bagging classifier gives better performance in terms of classification rate. When AUC is used as an evaluation metric, for KC1 method level dataset and Eclipse package level dataset, fault prediction rate is 80% with bagging ensemble which is better when compared with other ensemble classifier methods. For KC1 class level dataset, fault prediction rate is 82% with voting ensemble method using AUC-ROC as a metric. Bagging outperforms other ensemble methods when performance is evaluated using both RMSE and AUC-ROC with an exception that for class level metric, voting performs better than bagging with a negligible difference. Many researchers may apply machine learning methods for constructing the model to predict faulty classes. We plan to replicate our study to predict the models based on other machine learning algorithms such as ensemble using neural networks and genetic algorithms.

7. REFERENCES

[1] I.H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation, Morgan Kaufmann, 2005.

[2] Knab, P., Pinzger, M., and Bernstein, A., 2006. "Predicting defect densities in source code files with decision tree learners," in the 2006 International Workshop on Mining Software Repositories.

[3] H. Zhang and X. Zhang, Comments on "Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Trans. on Software Eng., Vol. 33(9), Sep 2007

[4] Sandhu, Parvinder Singh, Sunil Kumar and Hardeep Singh, 2007 "Intelligence System for Software Maintenance Severity Prediction",Journal of Computer Science, Vol. 3 (5), pp. 281-288.

[5] Catal, C., Diri, B., and Ozumut, B., 2007. "An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software," in 2nd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX.

[6] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1),

[7] Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S., 2007. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on Software Engineering 33 (6), 402– 419.

[8] S.Kanmani, V.R. Uthariaraj, V.Sankaranarayanan, P.Thambidurai, Objected-oriented software fault prediction using neural networks, Information and software Technology 49 (5 (2007)) 483 – 492.

[9] Dr Kadhim M. Breesam, "Metrics for Object Oriented design focusing on class Inheritance metrics", 2nd International conference on dependability of computer system IEEE, 2007.

[10] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, Journal of Systems and Software 81 (5) (2008) 649– 660.

[11] I.Gondra, Applying machine learning to software fault-proneness prediction, Journal of System and Software 81(2) (2008) 186-195.

[12] Amjan Shaik, Dr C.R.K. Reddy, Dr A Damodaran, "Statistical Analysis for Object Oriented Design Software security metrics", International journal of engineering and technology, Vol. 2, pg 1136-1142.2010

[13] Software fault prediction: A literature review and current trends Cagatay Catal, Expert Systems with Applications 38 (2011) 4626 – 4636.

[14] T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Machine Learning, vol. 40, No. 2, pages 139 – 157,

[15] Tsoumakas, G., Katakis, I. and Vlahavas, I. "Effective Voting of Heterogeneous Classifiers", In Proceedings of the 15th European Conference on Machine Learning, Italy, pp 465-476.