Software Defect Prediction using Boosting Techniques

V.Jayaraj, PhD.
Associate Prof.
Bharathidasan University
Trichy

N.Saravana Raman
Research Scholar
Bharathidasan University
Trichy

ABSTRACT
Identification and removal of software defects is tedious and time consuming for software development. Improperly planned projects could have defects and the time spent to spot and fix them requires more than the code development time. A reverse engineering sub-area is identification of modules necessitating re-engineering, focusing on faulty modules prediction based on existing information sources like documentation and source code. Predicting defective module is essential in maintenance and reuse by simplifying system working with information and reusable parts localization. In software defect prediction, predictive models estimation is based on code attributes to assess software modules containing errors likelihood. In this paper, the classification accuracy of Boosting techniques for software defect prediction based on the KC1 dataset is investigated.

Keywords
Software Defect Prediction, KC1 Dataset, Bagging

1. INTRODUCTION
Software engineering involves various aspects of software production, developing and delivering useful, practical and reliable software. Software engineering includes theoretical methodology and tools required for professional software development. Software engineers implement a systematic/organized approach with suitable tools and techniques desirable for software development. System engineering as part of software engineering deals with computer-based system development including hardware, software and process engineering. Software development involves actions called a software process which involves system specification, development constraints and development of software system, validation and evolution.

Identifying and fixing software defects is hard and development teams place effort to find and fix defects. The resulting change in software defects and subsequent change that fixes it are recorded in project's software history records. Software defects can cause problems, ranging from minor to catastrophic glitches leading to loss of life [1]. Finding and removing defects is tedious and time consuming for software development. Improperly planned projects are liable to defects and time spent to spot and fix defects is more than actual development time. Zero defects are not practical. Despite intensive defect testing, many continue to exist, resulting in unpredictable software behaviour, sometimes becoming unusable or catastrophic.

Usually, software solutions are products with known and unknown defects [2], leading to continuous evolving process of software through unknown defect removal and uncovering new defects over time. As new defects are uncovered in this mode, fixing high priority defects is a software development team's job. Code review, unit testing and system testing integration are conventional methods to identify defects. It also includes software defect prevention process which accompanying software evaluation and design processes. Software developer's record defects discovered during evaluation/testing operations. An organized software development team analyzes defects with statistical processes; systems like Six Sigma effectively reduce defects.

Software metrics usually define program complexity, and estimate programming time. Extensive research has been carried out to calculate a module's defects through use of software metrics [3]. This work includes data mining techniques and classification module applications [4] requiring optimization to ensure software reliability which in turn is evaluated using NASA dataset, KC1 dataset specifically for classification and reliability prediction [5].

The aim of this paper is to identify defects based on existing software metrics using data mining techniques and thereby improve software quality which ultimately leads to reducing the software development cost in the developing and maintenance phase. This paper focuses on predicting defective modules using Boosting techniques.

2. RELATED WORKS
Baojun et al [6] assessed CBA2 classification method comparing it to other rule based classification methods for software defect prediction problems. Investigation were done to check rule sets effectiveness generated on data from a software project and also whether it could predict defective software modules in other similar software projects. Application of CBA2 algorithm led to accurate and comprehensible rule sets.

Song et al [7] suggested a general software defect prediction framework supporting unbiased/comprehensive comparison between competing prediction systems. The framework includes scheme evaluation and defect prediction. Scheme evaluation analyzes prediction performance of competing schemes for specific historical data sets. The defect predictor constructs models based on evaluated learning schemes predicting software defects with new data according to a constructed model. To demonstrate the proposed framework's performance, simulations were undertaken on publicly available software defect datasets. Results demonstrated the requirement of various learning schemes for differing datasets (i.e., no scheme dominates) and that small details in conducting evaluations conduct completely reverses findings. The proposed framework is effective and not liable for bias than earlier approaches.

Li et al [8] suggested a sample-based software defect prediction procedure. It is possible to select and test a small percentage of modules for a large software system, and build a defect prediction model to predict defects. Three methods described sample selection: random sampling with conventional machine learners, random sampling with semi-supervised learner and active sampling with active semi-

supervised learner. To ensure active sampling, a new active semi-supervised learning method ACOForest was suggested which could sample modules helpful for learning to build good prediction model. Experiments were conducted on PROMISE datasets, revealed that proposed methods were effective and had the potential of being applicable to industrial practice.

Defect predictor learners were improved by Zhang et al [9] focusing on training sets with defect-rich portions. Defect data CM1, KC1, MC1, PC1, PC3 were separated into components. A projects subset (randomly selected) was tested. Training sets were generated for a Naïve Bayes classifier in two ways. Components, with higher than median number of defective modules were used for training in dense treatment. In standard treatment, any component modules were used for training. Both samples were run against a test set and evaluated using recall, probability of false alarm, and precision. Under sampling and over sampling were performed on defect data additionally. Every method was repeated in a 10-by-10 cross-validation experiment. Prediction models from defect dense components out-performed the standard method, in both under and over sampling. In statistical rankings based on recall, Probability of false alarm, and precision models learned from dense components won 4-5 times other methods, and also lost the least.

Menzies et al [10] proposed a meta-learner framework WHICH that could be customized to various goals. When customized to AUC (effort, pd), WHICH out-performed all data mining methods. Effectiveness of learning defect predictors from static code features was demonstrated and that it did not necessarily hold when studying performance criteria other than AUC (pf, pd). When defect predictors are assessed by criteria like “read less, see more defects” (i.e. AUC (effort, pd) selection of appropriate learner is critical. The study concluded that:

– A learner tuned to “read less, see more defects” performs best.

– A simple manual analysis out-performs standard learners like NB, C4.5, RIPPER. Learners use is depreciated for “read less, see more defects”.

3. METHODOLGY

3.1 KC1 Dataset

KC1 dataset is a NASA Metrics Data Program [11], and it is publicly. KC1 dataset is widely used for verification, and improving predictive software engineering models. KC1 is a C++ system implementing storage management for receipt and processing ground data. The dataset includes McCabe and Halstead features code extractors. The measures are module based.

The defect detectors are assessed as follows:

a = Classifier predicts no defects and module actually has no error.

b= Classifier predicts no defects and module actually has error.

c = Classifier predicts some defects and module actually has no error.

d = Classifier predicts some defects and module actually has error.

The accuracy, probability of detection (pd) or recall, probability of false alarm (pf), precision (prec) and effort is calculated as

$$Accuracy = \frac{(a+d)}{(a+b+c+d)}$$

$$recall = \frac{d}{(b+d)}$$

$$pf = \frac{c}{(a+c)}$$

$$prec = \frac{d}{(c+d)}$$

$$effort = \frac{(c.LOC + d.LOC)}{(TotalLOC)}$$

The KC1 dataset includes 2109 instances and 22 different attributes which are 5 different LOC, 3 McCabe metrics, 12 Halstead metrics, a branch count and 1 goal-field. Attribute information in the dataset is as follows: McCabe's line count of code (LOC), cyclomatic complexity, design complexity, program length, effort, Halstead, total operands, class and so on.

Examples from dataset:

Example 1 - 1.1, 1.4, 1.4, 1.4, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 2,2, 2,2, 1.2, 1.2, 1.2, 1.2, 1.4, false

Example 2 - 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, true

Example 3 - 83, 11, 1, 11, 171, 927.89, 0.04, 23.04, 40.27, 21378.61, 0.31, 1187.7,65, 10,6, 0,18, 25, 107, 64, 21, true.

3.2 Boosting Methods

Boosting [12, 13] works through classification algorithms use sequentially on training data reweighted versions. The predicted final class label is based on weighted majority vote. The initial weights is set at 1/N in Logitboost where N is the number of instances with probability estimate p(xi=0.5). The process is repeated m times and function fitted using least squares regression. LogitBoost is a boosting algorithm casting AdaBoost algorithm into a statistical framework [14]. If AdaBoost is considered a generalized additive model and then applied the cost functional of logistic regression, one derives LogitBoost algorithm.

LogitBoost is a convex optimization, given that an additive form model in the equation given below is searched for.

$$f = \sum_t \alpha_t h_t$$

The logistic loss is minimized by the LogitBoost algorithm as follows:

$$\sum_i \log\left(1 + e^{-y_i f(x_i)}\right)$$

The bagging for classification or regression can be defined as follows: The data is represented in pairs $(X_i, Y_i)(i = 1,...,n)$, where $X_i \in R^d$ denotes the *d*-dimensional predictor variable

and the output is $Y_i \in R$ (regression) or $Y_i \in \{0,1,....,J-1\}$ (classification with classes).

$E[Y \mid X = x]$ is generally the target function of interest for regression or $P\,|\,Y = j \,|\, X = x\,|\,(j = 0,...,J-1)$, the multivariate function, for classification. The function estimator is obtained from a base procedure, is given as follows:

$$\hat{g}(.) = h_n\left((X_1,Y_1),...,(X_n,Y_n)\right)(.) : \mathrm{R}^d \to R$$

where the function $h_n(.)$ defines the estimator as a function of the data.

The Bagging procedure follows the steps given below:

Step 1. Bootstrap sample $(X_1^*,Y_2^*),...,(X_n^*,Y_n^*)$ is constructed by randomly choosing *n* times with replacement from the data $(X_1,Y_2),...,(X_n,Y_n)$.

Step 2. Calculate the bootstrapped estimator $\hat{g}^*(.)$ by:

$$\hat{g}(.) = h_n\left((X^*{}_1,Y^*{}_1),...,(X^*{}_n,Y^*{}_n)\right)(.)$$

.

Step 3. The steps 1 and 2 are repeated *M* times; M value is assigned either 50 or 100, yielding $\hat{g}^{*k}(.)(k = 1,...,M)$. Bagged estimator is given by:

$$\hat{g}_{Bag}(.) = M^{-1}\sum_{k=1}^{M}\hat{g}^{*k}(.)$$

Bagged estimator in theory is given by:

$$\hat{g}_{Bag}(.) = E^*\left[\hat{g}^*(.)\right]$$

The precision and recall are given by the following equations:

$$Accuracy = \frac{tp+tn}{n}$$

$$\mathrm{Pr}ecision = \frac{tp}{tp+fp}$$

$$\mathrm{Re}call = \frac{tp}{tp+fn}$$

Where *tp* is true positives, *fp* is false positives and *fn* is false negatives.

$$F-measure = \frac{2\ X\ precision\ X\ recall}{precision + recall}$$

To evaluate the KC1 Dataset, Boosting with following techniques are used:

- Boosting with decision stump
- Boosting with REPtree
- Boosting with M5

4. RESULTS AND DISCUSSION

The classification algorithms used in this study is Boosting. The software complexity measures such as LOC measure, Cyclomatic complexity, Base Halstead measures and Derived Halstead measures of the KC1 (NASA) dataset are used to classify the software modules. Weka is a machine learning software written in Java. It supports several data mining process such as preprocessing, clustering, classification and so on. All classification in this study is carried out on Weka.

For the performance evaluation of the Boosting technique, 2107 samples from the KC1 Dataset is used, wherein 1391 samples are used as training set and 716 samples are used for testing. Weka was used on KC1 dataset for classification and the result is summarized in Table 1 and Figure 1 and Figure 2.

Table 1: Classification parameters

Technique used	Correctly classified %	Root mean squared error	Mean Absolute error
Boosting with decision stump	86.9483	0.3159	0.1969
Boosting with REPtree	86.5211	0.3233	0.1925
Boosting with M5	**87.3754**	0.3165	0.178

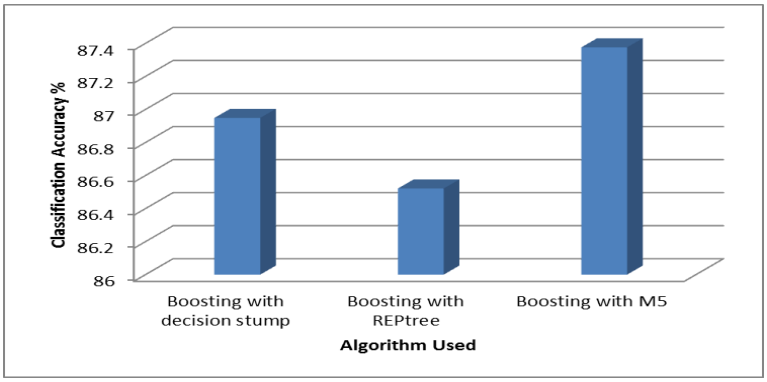

Fig 1: Classification accuracy on KC1 dataset

The mean absolute error is given by

$$Mean\ absolute\ error = \frac{1}{m}\sum_{k=1}^{m}|\ p_i - e_i\ |$$

where

p_i *is the predicted output*

e_i *is the expected output*

m is the number of instances

It is used to measure the deviation of the predicted output with respect to the actual output.

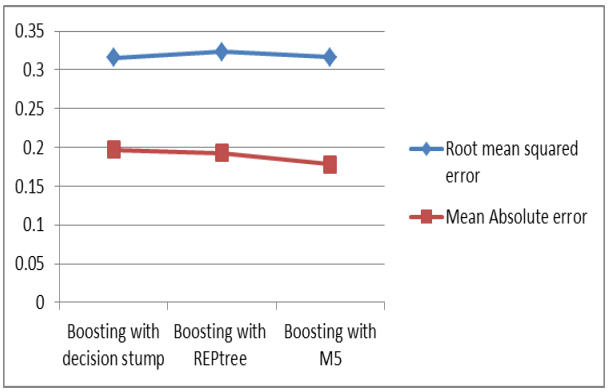

Figure 2: The RMSE and Mean absolute error

It is observed that boosting with M5 provides the best classification accuracy. However from Figure 2, it is observed that the difference between Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) is minimum for Boosting with decision stump which indicates the variance of the error.

5. CONCLUSION

Software defect prediction helps developers in defects identification based on current software metrics with data mining techniques. It is a major requirement for enhancing the quality of the software. This also helps in reducing the software development cost in the development and maintenance phases. The objective of this paper is to investigate the classification performance of various boosting techniques for defect prediction. KC1 dataset was used for evaluation of the boosting algorithms. The static code metrics in the dataset is utilized to predict software defect. Experiments reveal that bagging with decision stump provides the best accuracy of 86.03%. Though Bagging with Random forest achieves minimum difference between Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) which indicate error variance.

6. REFERENCES

[1] Sommerville. I, Software Engineering, 9th Edition, 2010. Pearson.

[2] Zeller, A. (2009). Why programs fail: a guide to systematic debugging. Morgan Kaufmann.

[3] C. Catal, and B. Diri, “A systematic review of software fault prediction studies,” Expert Systems With applications, 36(4):7346-7354, 2009.

[4] Huselius, J.,Andersson, J., Hansson, H.,Punnekkat, S: Automatic Generation and Validation of Models of Legacy Software, 12th IEEE International conference on Embedded and Real-Time Computing Systems and Applications 2006, Sydney, 2006, pp. 342 – 349.

[5] Chia-Chu Chiang,Bayrak, C: Legacy Software Modernization, IEEE International Conference on Systems, Man and Cybernetics, 2006, SMC '06, Taipei, Vol. 2, 2006, pp. 1304-1309.

[6] Baojun, M., Dejaeger, K., Vanthienen, J., & Baesens, B. (2011). Software defect prediction based on association rule classification. *Available at SSRN 1785381*.

[7] Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A general software defect-proneness prediction framework. *Software Engineering, IEEE Transactions on*, *37*(3), 356-370.

[8] Li, M., Zhang, H., Wu, R., & Zhou, Z. H. (2012). Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, *19*(2), 201-230.

[9] Zhang, H., Nelson, A., & Menzies, T. (2010, September). On the value of learning from defect dense components for software defect prediction. In*Proceedings of the 6th International Conference on Predictive Models in Software Engineering* (p. 14). ACM.

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

[11] Shirabad, J. S., & Menzies, T. J. (2005). The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 24.

[12] Jerome Friedman, Trevor Hastie and Robert Tibshirani, “Additive Logistic Regression: A Statistical View of Boosting”, Annals of Statistcs, Vol. 28, No. 2, pp. 337-407, 2000.

[13] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine.(English summary). Ann. Statist, 29(5), 1189-1232.

[14] Friedman, T. Hastie , R. Tibshirani, “Additive Logistic Regression: a Statistical View of Boosting,” Ann. Statist., vol. 28, no. 2, pp.337-407, 1998