# A Comparison of Naïve Bayes and Random Forest for Software Defect Prediction

Yan Naung Soe, Khine Khine Oo
*University of Computer Studies, Yangon*
*yannaungsoe@ucsy.edu.mm, khinekhineoo@ucys.edu.mm*

## Abstract

*The software defect can cause the unnecessary effects on the software such as cost and quality. The prediction of the software defect can be useful for the developing of good quality software. For the prediction, the PROMISE public dataset will be used and Random Forest (RF) algorithm and Naïve Bayes algorithm (NB) will be applied with the RAPIDMINER machine learning tool. This paper will compare the performance evaluation upon the different number of trees in RF and NB. As the results, the accuracy will be slightly increased if the number of trees will be more in RF. The maximum accuracy is up to 99.59 for RF and 97.12 for NB. The minimum accuracy is 87.13 RF and 45.87 for NB. Another comparison is based on AUC and all of the results show that RF algorithm is more accurate than Naïve Bayes algorithm for this defect prediction.*

***Keywords**: Software Defect, Defect Prediction, Random Forest, Naïve Bayes*

## 1. Introduction

A defect is an error or a bug one of the software or application. In the software developing process, the programmer can make mistakes or error. These mistakes or errors mean that there are flaws in the software that are also called defects.

A software defect or bug is a condition in a software product which does not meet a software requirement or end-user expectations. In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.

The goal of software quality control, and in a broader sense, quality management in general, is to remove quality problems in the software [6]. These problems are referred to by various names such as bugs, faults, errors, or defects to name a few. As software engineers, it is needed to find and correct as many errors as possible before the customer and/or end user encounter them. It is needed to avoid because defects may make software bugs. It is important to detect errors and defects. Figure 1 shows the software can be damaged due to the defects.

Detection of defects is the types of software testing. Most of the software defects detection researches are using data mining algorithm and tools. There are many approaches to detect such as static analysis and dynamic analysis. Static analysis examines code in the absence of input data and without running the code [1]. It can detect potential software defect, runtime errors (dereferencing a null pointer) and logical inconsistencies. Dynamic analysis is the analysis of computer software that is performed with executing programs built from that software on a real or virtual processor.

Researchers focus on the development process and final product to investigate the reasons of software defects and to detect as early as possible. But, the prediction of software defects is still needed for software development life cycle. Most defect prediction techniques used in planning which rely on the historical data. Defect prediction techniques are varying in the types of data which are based on the product characteristics or only defects data.

In the paper, the Predict Or Models In Software Engineering (PROMISE), public datasets are used for prediction of software defect. For the prediction, RAPIDMINER tool is used to apply the random forest algorithm and naïve bayes algorithm. The performance evaluation of these algorithms will show by using the selected datasets. And, this paper will show the comparison of performance which is varying the number of trees upon the random forest algorithm and naïve bayes.

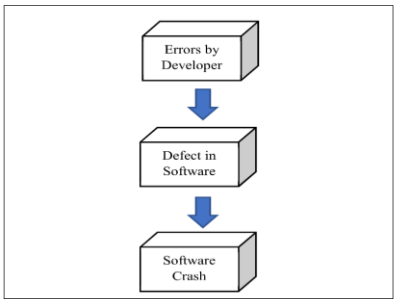

**Figure 1. Defects Make Software Crash**

This paper is organized as follows. In section 2, it will introduce about cost and impact of software defects. Section 3 discuss about the related work of the software defect prediction. In section 4, methodology of the prediction is presented, including with the usable datasets, random forest algorithm, naïve bayes and performance measurement mechanisms. The experimental results are discussed in section 5. Finally, this paper will conclude in section 6.

## 2. Cost and Impact of Software Defects

If software process improvement is considered, a quality problem that is propagated from one process framework activity (e.g., modeling) to another (e.g., construction) can also be called a "defect" [6]. The cost of defects can be measured by the impact of the defects and when these are found. Earlier the defect is found lesser is the cost of defect. A defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix. It is quite often the case that defects detected at a very late stage, depending on how serious they are, are not corrected because the cost of doing so is too expensive. Figure 2 shows that the cost effective of software defect depends on processing time. If the finding of software defects will take more time, the software cost may be more expensive.

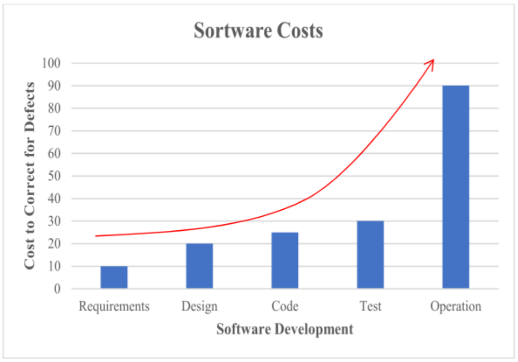

**Figure 2. Defects vs Processing Time for Effectiveness of Software Cost**

A number of industry studies indicate that design activities introduce between 50 and 65 percent of all errors (and ultimately, all defects) during the software process. However, review techniques have been shown to be up to 75 percent effective in uncovering design flaws [6]. By detecting and removing a large percentage of these errors, the prediction process substantially reduces the cost of subsequent activities in the software process.

## 3. Related Works

Most of the software defect detection is based on the datasets by using machine learning techniques. And most of the classification processing used WEKA tools for generation the result.

Selvaraj et al. used Support Vector Machine (SVM) for software defects prediction. KC1 dataset is used for this prediction and it is C++ system implementing storage management for ground data receipt/processing [7]. Their research can detect the defects and defect detectors are calculate as classifier predicts no defects and the module has no error, classifier predicts no defects and the module has error, classifier predicts some defects and the module has no error, classifier predicts some defects and the module has error. But it can't solve complex problems. The accuracy of this work is 86.05 % for the KC1 dataset which is used 1391 samples as training set and 716 samples for testing. The results are carried out by WEKA tool.

Ma et al. proposed software defect prediction based on association rule classification. This paper assesses that the use of such a classification method based on association rule, CBA2 and compares it to other rule-based classification methods. And then, it investigates whether rule sets generated on data from one software project can be used to predict defective software modules [3]. Their finding is CBA2 algorithm results in both accurate and comprehensible rule sets. Comparisons are based on the area under the receiver operating characteristics curve and AUC represents the most informative indicator of predictive accuracy within the field of software defect prediction. Twelve datasets are used for detection, one-third of each dataset is used for testing and the remaining part of each dataset is for training with implementing using WEKA. The accuracy of each process got around 69 to 99% but CBA2 classifier would yield a lower performance then a rule set induced on the same data set.

Song et al. proposed software defect association mining and defect correction effort prediction [8]. They used association rule mining based methods to predict defect associations and defect correction effort and their proposed methods to the SEL defect data consisting of more than 200 projects over more than 15 years. And, the results are compared the defect correction effort prediction method with other types of methods—PART, C4.5, and Naive Bayes. NASA SEL datasets are used for defect, defect isolation effort, the defect correction effort dataset. The 37.36 percent of cases in the defect data set contain only one defect (will not be correctly predicted) and; resulted in a total of 60 sets of rules, which consist of more than 1,000 rules. They found that higher support and confidence levels may not result in higher prediction accuracy. This work got the maximum accuracy 96.06% minimum support 20% and minimum confidence 30% and the defect association prediction, the minimum accuracy is 95.38 percent, and the false negative rate is just 2.84 percent.

Most of the existing software defect prediction systems are limited in performance analysis. Some are used to discuss their proposed method and compare the existing techniques. Some are only compared the overall performance of the machine learning techniques.

## 4. Methodology

This work will use the public datasets that are downloaded and these data are applied with random forest algorithm. And then, the accuracy is calculated with the outcomes by using performance measuring mechanisms.

### 4.1. Dataset

The public datasets (software defect prediction) are getting from other researchers [2], these datasets contain (ar1, ar3, ar4, ar5, ar6, kc1, kc2, kc3, pc1, pc2, pc3 and pc4). These datasets are created by NASA, then the NASA metrics data program. The name of this package is PROMISE Software Engineering Repository dataset that make publicly available in order to encourage repeatable, verifiable, refutable, and improvable predictive models of software engineering.

As an example, KC datasets are a "C++" system implementing storage management for receiving and processing ground data that comes from McCabe and Halstead features extractors of source code. These features were defined in the 70s in an attempt to objectively characterize code features that are associated with software quality. The nature of association is under dispute. The McCabe and Halstead measures are "module"-based where a "module" is the smallest unit of functionality. In C or Smalltalk, "modules" would be called "function" or "method" respectively. KC1 dataset contains 21 feature attributes and on class label attributes that include defects and non-defects. There are 2109 records containing 326 defects and 1783 non-defects data.

PC datasets are the data from C functions that are flight software for earth orbiting satellite. AR datasets are software defect prediction data (implemented in C) that are from a Turkish white-goods manufacturer. Each of the datasets is summarized in Table 1.

**Table 1.PROMISE Datasets**

| # | Attributes | Records | Defects | Non-Defects |
|---|---|---|---|---|
| ar1 | 29 | 121 | 9 | 112 |
| ar3 | 29 | 63 | 8 | 55 |
| ar4 | 29 | 107 | 20 | 87 |
| ar5 | 29 | 36 | 8 | 28 |
| ar6 | 29 | 101 | 15 | 86 |
| kc1 | 21 | 2109 | 326 | 1783 |
| kc2 | 21 | 522 | 107 | 415 |
| kc3 | 21 | 458 | 43 | 415 |
| pc1 | 21 | 1109 | 77 | 1032 |
| pc2 | 36 | 5589 | 23 | 5566 |
| pc3 | 37 | 1563 | 160 | 1403 |
| pc4 | 37 | 1458 | 178 | 1280 |

### 4.2. Random Forest

Random forest algorithm is a supervised classification algorithm that creates the forest with a number of trees.

The pseudo code for random forest algorithm can split into two stages [5].

- Random forest creation
- Prediction from the created random forest classifier

#### 4.2.1. Creation

The following is the pseudo code of the process flow of random forest.

1. Randomly select “k” features from total “m” features. (where k << m)
2. Among the “k” features, calculate the node “d” using the best split point.
3. Split the node into child nodes using the best split.
4. Repeat 1 to 3 steps until “1” number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

The beginning of random forest algorithm starts with randomly selecting “k” features out of total “m” features. In the next stage, these are being used the randomly selected “k” features to find the root node by using the best split approach. After that, it will be calculated the “d” nodes using the same best split approach. This will repeat the first 3 stages until to form the tree with a root node and having the target as the leaf node. Finally, it is repeated 1 to 4 stages to create "n" randomly created trees. This randomly created trees to form the random forest.

#### 4.2.2. Prediction

To perform prediction using the trained random forest algorithm uses the below pseudo code.

1. Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target).
2. Calculate the votes for each predicted target.
3. Consider the high voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm it will be needed to pass the test features through the rules of each randomly created the trees. Each random forest will predict different target (outcome) for the same test feature. Then by considering each predicted target votes will be calculated. Then the final random forest returns as the predicted target.

### 4.3. Naïve Bayes

The Naive Bayes classifier applies to learning tasks where each instance ***x*** is described by a conjunction of attribute values and where the target function f*(x)*can take on any value from some finite set V. A set of training examples ofthe target function is provided, and a new instance is presented, described by thetuple of attribute values *($a_l$,$a_2$ ...$a_n$)*. The learner is asked to predict the target value, or classification, for this new instance. The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. The assumption is that given the target value of the instance, the probability of observing the conjunction $a_l$,$a_2$ ...$a_n$is just the product of the probabilities for the individual attributes.

The naive Bayes learning method involves a learning step in which the various P($v_j$) and P($a_i$|$v_j$)terms are estimated, based on their frequencies over the training data. The set of these estimates corresponds to the learned hypothesis. This hypothesis is then used to classify each new instance by applying the rule. Whenever the naive Bayes assumption of conditional independence is satisfied, this naive Bayes classification $v_{NB}$ is identical to the MAP classification [4].

### 4.4. Performance Evaluation

The evaluation performance of the prediction of the software defects as the following criteria such as accuracy, classification error and AUC.

- True Positive (TP): Number of defect correctly classify.
- False Positive (FP): Number of non-defect correctly classify.
- True Negative (TN): Number of defect wrongly classify.
- False Negative (FN): Number of non-defect wrongly classify.
- Accuracy (Acc): Percentage of correctly identified for prediction.
  Acc = (TP+TN)/(TP+TN+FP+FN) (1)
- Classification Error (Err): Percentage of incorrectly identified for prediction.
  Err = (FP+FN)/(TP+TN+FP+FN) (2)

The Area under the curve (AUC) is a performance metrics for the binary classifiers. By comparing the ROC curves with the area under the curve, or AUC, it captures the extent to which the curve is up in the northwest corner. The higher AUC is the better result. Normal threshold for two classes is 0.5 and the algorithm classify upon the true and false. In this paper, AUC values will be calculated based on positive class (true).

## 5. Experimental Result

The experimental results are compared with three categories which are based on the number of trees such as 10, 100 and 1000 trees in random forest. Table 2 and 3 show the prediction accuracy and error. In Table 2, all of the accuracy is the same in ar1, ar4, kc3 and pc2. Other results show that most of the prediction accuracy is slightly increase when the number of trees will be more. Using 10 trees for random forest, some of results are stable but most are slightly decrease accuracy result. If the number of tree will increase up to 100, most of accuracy results are slightly increase. If the number of tree will create up to 1000, there is no difference upon the accuracy results. As the result show that the usage of around 100 trees is more suitable for the prediction of the software defects by using the PROMISE datasets. In the datasets, there are 20 percent of records are the defects classes and others are non-defects records.

**Table 2. Accuracy of Defect Prediction (Acc)**

| # | Number of Trees | | |
|---|---|---|---|
| | **10** | **$10^2$** | **$10^3$** |
| ar1 | 92.56 | 92.56 | 92.56 |
| ar3 | 95.24 | 95.24 | 93.65 |
| ar4 | 87.85 | 87.85 | 87.85 |
| ar5 | 88.89 | 97.22 | 97.22 |
| ar6 | 88.12 | 87.13 | 87.13 |
| kc1 | 84.54 | 84.64 | 84.59 |
| kc2 | 85.06 | 85.25 | 85.25 |
| kc3 | 90.61 | 90.61 | 90.61 |
| pc1 | 93.06 | 93.15 | 93.15 |
| pc2 | 99.59 | 99.59 | 99.59 |
| pc3 | 89.76 | 89.76 | 89.76 |
| pc4 | 87.79 | 87.86 | 87.93 |

Table 3 shows the classification error for software defect prediction based on RF algorithm. The minimum error rate is 0.41 and the maximum error rate is only 15.44. These error rates are acceptable rate in prediction. If it will be applied by using feature selections method, it can get more acceptable the error rate.

**Table 3. Classification Error of Defect Prediction (Err)**

| # | Number of Trees | | |
|---|---|---|---|
| | **10** | **$10^2$** | **$10^3$** |
| ar1 | 7.44 | 7.44 | 7.44 |
| ar3 | 4.76 | 4.76 | 6.35 |
| ar4 | 12.15 | 12.15 | 12.15 |
| ar5 | 11.11 | 2.78 | 2.78 |
| ar6 | 11.88 | 12.87 | 12.87 |
| kc1 | 15.46 | 15.36 | 15.41 |
| kc2 | 14.94 | 14.75 | 14.75 |
| kc3 | 9.39 | 9.39 | 9.39 |
| pc1 | 6.94 | 6.85 | 6.85 |
| pc2 | 0.41 | 0.41 | 0.41 |
| pc3 | 10.24 | 10.24 | 10.24 |
| pc4 | 12.21 | 12.14 | 12.07 |

Another comparison is based on the area under the receiver operating characteristics curve (AUC). The AUC represents the most informative indicator of predictive accuracy within the field of software defect prediction. In the Table 4, AUC values will be calculated based on positive class (true). Most of the results show that the values will be increased if random forest will create more trees. By summarizing these three tables, the prediction of defects is more suitable for using around 100 trees in the forest.

**Table 4. AUC Performance of Defect Prediction**

| # | Number of Trees | | |
|---|---|---|---|
| | **10** | **$10^2$** | **$10^3$** |
| ar1 | 0.61 | 0.891 | 0.897 |
| ar3 | 0.873 | 0.918 | 0.973 |
| ar4 | 0.849 | 0.939 | 0.928 |
| ar5 | 0.973 | 0.982 | 0.982 |
| ar6 | 0.658 | 0.853 | 0.858 |
| kc1 | 0.595 | 0.716 | 0.763 |
| kc2 | 0.856 | 0.86 | 0.862 |
| kc3 | 0.643 | 0.854 | 0.871 |
| pc1 | 0.588 | 0.643 | 0.659 |
| pc2 | 0.566 | 0.565 | 0.629 |
| pc3 | 0.517 | 0.777 | 0.791 |
| pc4 | 0.616 | 0.849 | 0.878 |

**Table 5.Accuracy Comparison of Naïve Bayes and Random Forest (100 trees)**

| # | Naïve Bayes | Random Forest |
|---|---|---|
| ar1 | 80.17 | 92.56 |
| ar3 | 87.30 | 95.24 |
| ar4 | 85.98 | 87.85 |
| ar5 | 86.11 | 97.22 |
| ar6 | 87.13 | 87.13 |
| kc1 | 82.50 | 84.64 |
| kc2 | 83.72 | 85.25 |
| kc3 | 84.72 | 90.61 |
| pc1 | 89.00 | 93.15 |
| pc2 | 97.12 | 99.59 |
| pc3 | 45.87 | 89.76 |
| pc4 | 87.93 | 87.86 |

The comparative results of defect prediction based on Naïve Bayes and Random Forest algorithms is shown in Table 5 and Figure 3.Almost the results of Random Forest are more accurate than Naïve Bayes especially in one of dataset such as pc3.In only one dataset (pc4), the prediction result of Naïve Bayes is more accurate than Random Forest. But, it is only 0.07 percentage difference that the result is based on one hundred trees. If it is used one thousand trees on Random Forest, there is no difference accuracy with these two algorithms for this dataset.

**Table 6. Prediction Results using Naïve Bayes for p1, pc2, pc3 and pc4 datasets**

| Prediction | | Non Defect | Defect | Accuracy |
|---|---|---|---|---|
| pc1 | Non Defect | 964 | 52 | 89.00% |
| | Defect | 68 | 23 | |
| pc2 | Non Defect | 5423 | 13 | 97.12% |
| | Defect | 143 | 10 | |
| pc3 | Non Defect | 575 | 18 | 45.87% |
| | Defect | 828 | 142 | |
| pc4 | Non Defect | 1210 | 106 | 87.93% |
| | Defect | 70 | 72 | |

Table 6 shows the prediction results for pc datasets that are predicted by Naïve Bayes. Most of the accuracy of datasets is acceptable but the accuracy of p3 dataset is significantly low. This algorithm can't effectively classify for pc3 dataset because it wrongly classifies 828 out of 1403 non-defects records and it can correctly classify 575 records on non-defects data. The prediction accuracy of this dataset has only 45.87% due to the value of some records are too distinct with others. But, this algorithm can significantly classify on other datasets and the prediction rates are between 80% and 97%. As all comparison of detection rates for these two algorithms, Random Forest algorithm is more stable than Naïve Bayes for software defect prediction with these datasets.

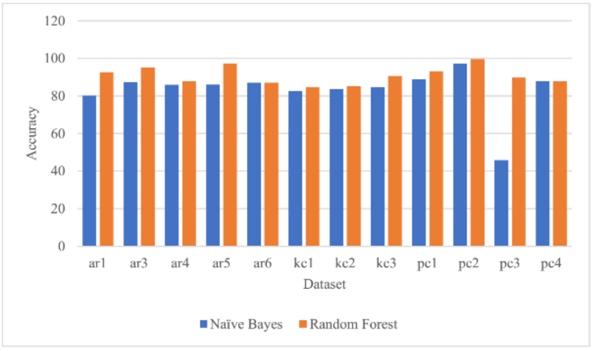

**Figure 3. Accuracy Comparison of Naïve Bayes and Random Forest**

## 6. Conclusions

Most of the software defect prediction works are based on the datasets by using machine learning techniques. In this paper, the prediction results are based on PROMISE public datasets by using Naïve Bayes and Random Forest algorithm. The results show that Random Forest that uses around hundred trees is more accurate and stable than Naïve Bayes. Random forest algorithm is more stable for this defect prediction to get the high accuracy. In the future, it will be analyzed the prediction for software defects by using another datasets and machine learning algorithms for getting the most stable in software defect prediction.

## References

[1] N. Ayewah, D. Hovemeyer, D. J. David Morgenthaler, J., Penix, and W. Pugh, *"Using Static Analysis to Find Bugs"*, IEEE Software, ISSN: 0740-7459, 19 August 2008.

[2] G. Boetticher, T. Menzies, and T. Ostrand, *"PROMISE Repository of Empirical Software Engineering Data Repository"*, West Virginia University, Department of Computer Science, 2007.

[3] B. Ma, K. Dejaeger, J. Vanthienen, and B. Baesens, *"Software defect prediction based on association rule classification"*, Department of Decision Sciences and Information Management, K. U. Leuven, B-3000, Leuven, Belgium, 2011.

[4] T. M. Mitchell, *"Machine Learning"*, ISBN: 0070428077, McGraw-Hill Science/Engineering/Math, March, 1997.

[5] S. Polamuri, *"How the Random Forest Algorithm Works in Machine Learning"*, May 2017, [Online] available: http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/

[6] R. S. Pressman, *"Software Engineering – A Practitioner's Approach"*, 7th Edition, ISBN 978–0–07–337597–7, 2010.

[7] P. A. Selvaraj, and P. Thangaraj, *"Support Vector Machine for Software Defect Prediction"*, International Journal of Engineering & Technology Research, ISSN Online: 2347-4904, 2013.

[8] Q. Song, M. Shepperd, M. Cartwright, M., and C. Mair, *"Software Defect Association Mining and Defect Correction Effort Prediction"*, IEEE Transactions on Software Engineering, DOI: 10.1109/TSE.2006.1599417, 2006.