Using Source Code and Process Metrics for Defect Prediction — A Case Study of Three Algorithms and Dimensionality Reduction

Wenjing Han, Chung-Horng Lung, Samuel Ajila*
Department of Systems and Computer Engineering, Carleton University Ottawa, Ontario, Canada.

* Corresponding author. Tel.: 1-613-520-2600 ext. 2673; email: ajila@sce.carleton.ca
Manuscript submitted May 13, 2016; accepted July10, 2016.
doi: 10.17706/jsw.11.9.883-902

Abstract: Software defect prediction is very important in helping the software development team allocate test resource efficiently and better understand the root cause of defects. Furthermore, it can help find the reason why a project is failure-prone. This paper applies binary classification in predicting if a software component has a bug by using three widely used algorithms in machine learning: Random Forest (RF), Neural Networks (NN), and Support Vector Machine (SVM). The paper investigates the applications of these algorithms to the challenging issue of predicting defects in software components. Thus, this paper combines source code metrics and process metrics as indicators for the Eclipse environment using the aforementioned three algorithms for a sample of weekly Eclipse features. In addition, this paper deals with the complex issue of data dimension and our results confirm the predictive capabilities of using data dimension reduction techniques such as Variable Importance (VI) and PCA. In our case the results of using only two features (NBD_max and Pre-defects) are comparable to the results of using 61 features. Furthermore, we evaluates the performance of the three algorithms vis-à-vis the data and both Neural Network and Random Forest turned out to have the best fit.

Key words: Software defect prediction, data analysis, eclipse, machine learning techniques.

1. Introduction

Software defect is a critical issue in software engineering, because its correct prediction and analysis can be utilized for decision management regarding resource allocation for software testing or formal verification. Moreover, software defect prediction can be used for assessing the final software product quality and estimating if contractual quality standards or those imposed by customer satisfaction are met. Within the software engineering community, the research on defect prediction has a long tradition [1]-[3] [4]. More importantly, it also creates potential research opportunities for incorporating the techniques in data science with software defects count and predictors.

Our research approach includes the application of machine learning techniques in predicting the occurrence of defects in software components. The paper aims to answer the following three questions: Firstly, what metrics are effective defect predictors? In this case, we focus on identifying specific metrics by grouping source code metrics and process metrics that are effective for defect prediction. Secondly, how accurate are these machine learning models and which machine learning models are more effective for defect prediction, either quantitatively or qualitatively? Thirdly, what is the impact data dimension

reduction on the predictive results?

Furthermore the work in this paper used a large and up-to-date data set from the open source Eclipse project (www.eclipse.org). The Eclipse project is popular in the open source community for software development and the data for the Eclipse bug information are readily available.

Our approach in this paper involves pre-processing and analyzing data sets that contain 102,675 records, then adopting different dimension reduction techniques, such as evaluating variable importance and applying Principal Component Analysis (PCA). The evaluation of variable importance allows for the reduction of data size and saves computational cost, and also identifies key features for defect prediction. Furthermore, we apply PCA for dimension reduction and compare its quality of result with other algorithms that do not use dimension reduction. Also, in order to achieve high accuracy of classification, we use three algorithms – Random Forest (RF), Neural Networks (NN), and Support Vector Machine (SVM) – to train the data. Following that, we use confusion matrix and resample to compare algorithm performance, and use Receiver Operating Characteristic (ROC) curve to describe the result of probabilistic classifiers.

The main contributions of this research are as follows: Firstly, we identify the key features of code metrics and process metrics for defect prediction. Secondly, we evaluate the performance of three algorithms, which can serve as a reference for choosing an effective algorithm for a specific project. Thirdly, we utilize different dimension reduction techniques (evaluating variable importance and PCA) and compare the quality of their results.

The rest of the paper is organized as follows: Section 2 presents the background information and related work. Section 3 discusses the methodology and approach. Section IV presents the results of data training using the three algorithms. Section 5 analyses and discusses the experimental results. Lastly, Section 6 concludes the paper and outlines directions for future work.

2. Background and Related Work

2.1. Neural Networks, Support Vector Machine, and Random Forest

This section briefly describes the concepts and definitions of the three machine learning algorithms used in this research work. The related works are also presented.

2.1.1. Neural networks

A neural network (NN) is a two-stage regression or classification model, typically represented by a network diagram [5]. Several variants of neural network classifier (algorithm) exist, some of which are; feed-forward, back-propagation, time delay and error correction neural network classifier. NN are popular due to their flexibility and adaptive capabilities, making them ideal algorithms for non-linear optimization. Russell (1993) [6] detailed the four main steps of a neural network algorithm:

- Processing units denoted by uj, with each uj possessing a certain activation level aj (t) at any point time t.
- Weighted interconnections between different processing units, which are responsible for determining how the activation of one unit leads to an input for a corresponding unit.
- An activation rule which operates on a set of input signals at a specific unit to produce a new output signal and activation.
- A learning rule, specifying how to adjust the weights for a given input/output pair. This one is optional.

The starting values in NN for weights are chosen to be random values near zero so that the model starts out nearly linear, and becomes nonlinear as the weights increase. Use of exact zero weights lead to zero derivatives and perfect symmetry, and the algorithm never moves. Conversely, starting with large weights often leads to poor solutions

Often neural networks have too many weights and will over-fit the data at the global minimum. Therefore,

an early stopping rule is used to avoid overfitting by training the model only for a while and stopping well before we approach the global minimum.

Scaling of inputs can have a large effect on the quality of the final solution. Standardizing all inputs to have a mean zero and standard deviation ensures all inputs are treated equally in the regularization process, and allows one to choose a meaningful range for the random starting weights

Having too many hidden units is better than too few as with few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data [6]. Typically the number of hidden units is in the range of 5 to 100, with the number increasing with the number of inputs and training instances. Choice of the number of hidden layers is guided by the background knowledge and experimentation.

Learning rate is the amount by which the weights are updated. Usually, it should not be too small or too large as because if the rate is too small, the prediction model will need more time to converge and if it is too large; it might oscillate around the local optima.

For the purpose of this work, we chose a feed-forward, single hidden layer artificial neural network as the first model. The number of units of hidden layer is one of the parameters for the training process. The training process runs 4 times and chooses the best value of hidden units for the best performance.

2.1.2. Support vector machine

Support Vector Machine (SVM) is a powerful and flexible type of supervised learning model, used for classification and regression analysis. It has been applied to learning algorithms that analyze data and recognize patterns. It has the advantage of reducing problems of overfitting or local minima. In addition, it is based on structural risk minimization as opposed to the empirical risk minimization of neural networks.

Given a set of samples for training purpose [7], a SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

A SVM model represents the samples as points and maps those points into a new transformed space, so that separate sample categories are divided by a clear gap that is as wide as possible. New samples are then mapped into that same space and predicted to belong to a category based on the side of the gap they fall in.

SVMs also include a regularization parameter that controls how much the regression line can adapt to the data with smaller values resulting in more linear (i.e., flat) surfaces. This parameter is generally referred to as the "cost".

There is a significant penalty for having samples within a certain margin to the "decision boundary". However, the decision boundary can be tuned with the cost parameter. It is important to tune the cost parameter for specific datasets, as well as any kernel parameters, because of the problem of over-fitting the training data.

2.1.3. Random forest

The Random Forest (RF) algorithm is one of the most fundamental and simplest classification and regression methods. It was first proposed by Ho in 1995 [8], [9]. RF operates by building a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees [10]. Most importantly, random decision forests avoid or minimize the problem of over-fitting to the training set. RF can be used to rank the importance of variables in regression and classification problem. The first step is to fit a random forest to the data set and the error for each data point is stored and averaged over the entire forest. The RF training algorithm generally applies the technique of bootstrap aggregation to tree learning. Given a training set $S = s_1, s_2, \ldots, s_n$ with responses $R = r_1, r_2, .., r_n$, the bootstrap aggregation repeatedly (M times) selects a random sample replacement of the training set and fits trees to these samples:

For m = 1, …, M

*Sample [*with *replacement] n training examples from S, R; call these* S_m, R_m.

Train *a regression tree (or a decision tree)* d_m *on* S_m, R_m.
End for

When the training is completed, predictions for unseen samples *s'* can be made by averaging the prediction for all the regression trees on *s'* or by taking the majority vote if it is decision trees.

The bootstrapping method gives better model performance because of decreasing variance of the model without increasing bias.

2.2. Related Work

This sub-section summarizes the related works on defects prediction that are relevant to this paper. Nagappan and Ball [4] studied the influence of code churn (that is, the amount of change to the system) on the defect density using the Windows Server 2003 data. They demonstrated that relative code churn is a better predictor than absolute code churn.

Ajila and Dumitrescu [11] investigated the use of code delta, code churn, and rate of change to understand software product line evolution. These researchers did not use machine learning algorithms, but the results (among others) showed that there is a relationship between code churn and software product complexity. The research is a longitudinal study of change processes. It links changes in the product line architecture of a large telecommunications equipment supplier with the company's customers, inner context, and eight line card products over six-year period. There are three important time related constructs in the study: the time it takes to develop a new product line release; the frequency in which a metric is collected; and the frequency at which financial results and metrics related to the customer layer are collected and made available. Data collection has been organized by product release.

Moser *et al.* [12] used metrics that include code churn, past bugs, and refactoring to predict the presence or absence of bug in Eclipse data repository. They used three common machine learners: logistic regression, Naïve Bayes, and decision trees and proposed a cost-sensitive classification method to investigate the Eclipse data.

Kim *et al.* [13] investigated seven open-source systems and utilized the *Top Ten List* approach to analyze properties such as most recent/frequent changes and defects. However, they assumed that faults occur in bursts. These changes were extracted from the SCM logs of Apache, PostgreSQL, Subversion, Mozilla, JEdit, Columba, and Eclipse.

Nagappan and Ball [14] used a static analysis tool to calculate the pre-release defect density of Windows Server 2003. In addition, Nagappan et al. [15] predicted post-release defects at the module level on five Microsoft systems by utilizing a group of code metrics. They found that some metrics were useful to predict defects in some specific projects, but not helpful for some projects. It is difficult to find metrics that fit all projects.

Likewise, Kamei *et al.* [16] confirmed that process metrics which were extracted from the version control system or the bug database perform better than software system product metrics when they reviewed common findings in predicting defects by using effort-aware performance measurements.

Zimmermann *et al*. [17] used a number of code metrics on Eclipse. Furthermore, the authors in [18] solved the issue of cross-project defect prediction by building a prediction model from a project and applying the model to another project. They called it cross project prediction and proved that applying the model from projects in the same domain or with the same process does not achieve accurate prediction results. Further, they identified more important factors that impact cross-project predictors' performance.

Now, what then is the difference between this research work and previous research works in the domain? All previous research works sited above either used code metrics or process metrics to measure defects. In contrast, our primary goal is not to analyze prediction accuracy for a single set of predictor variables

instead, we focus on a thorough comparative analysis between the product (code), process, and a combined (code and process metrics) approach for defect prediction. More specifically, we combine these metrics and utilize variable importance and PCA to evaluate the effectiveness of the metrics. Next, we apply feature reduction and choose top two features to predict defects of Eclipse project to compare the results.

3. Methodology and Approach

This section presents the Eclipse datasets and the different data preprocessing procedures. The preprocessing includes variable importance, correlation of features, dimension reduction using PCA, and data classification quality.

3.1. Eclipse Bug Datasets

The Eclipse data consists of six files in total – one file for each level (files, packages) and release (2.0, 2.1, and 3.0). It involves the number of pre-release defects that were reported in the last six months before a release and the number of post-release defects that were reported in the first six months after a release. As we mentioned before, the complexity metrics which are computed for each case are important features in our work. These metrics that are computed for classes or methods are aggregated by using average (avg), maximum (max), and accumulation (sum) at the file and package level. Another type of features is Change Metrics which is described in [19], and they are TotalPeopleInBurst, MaximumCodeBurstLate, NumberOfChanges, MaxPeopleInBurst, TotalBurstSizeLate, NumberCodeBurstsLate, NumberOfChangesLate, NumberOfChangesEarly, MaxChurnInBurst, MaximumCodeBurstEarly, NumberCodeBurstsEarly, TimeFirstBurst, TotalChurnInBurst, ChurnTotal, MaximumCodeBurst, NumberOfConsecutiveChangesEarly, NumberConsecutiveChangesLate, TotalBurstSizeEarly, TotalBurstSize, TimeMaxBurst, NumberConsecutiveChanges, TimeLastBurst, NumberCodeBursts, and PeopleTotal.

In this paper, three versions of data which were collected weekly for the package level are integrated into one data file. All the complexity metrics and change metrics are also integrated. The total data size is 267,670 KB. The metrics from the structure of abstract syntax tree(s) are not used, since they are not common to other projects. Instead, the complexity metrics are used, as they are clear to express a package's complexity.

3.2. Data Analysis and Procedure

This paper applies a sequence of steps to analyze the Eclipse bug datasets and they are outlined as follows.

Data Preprocessing with Variable Importance – Starting with 62 variables (61 features, and 1 dependent class i.e. Number of Defects), the first step of our approach deals with preprocessing the dataset. Those 61 features include pre-release defects, complexity metrics and change metrics.

Various features have different ranges of values. We first normalize each feature in order to avoid over-fitting. A K-Fold Cross Validation technique is used with the number of created folds equals 10 (K = 10) and the number of alternative versions set to 2 (time = 2). The reason why we choose ten folders and two repeat times is that it's a compromise between efficiency and time.

Next, a dimension reduction technique is utilized to assess the validity of the 61 features. Out of two possible dimension reduction techniques, feature selection and feature extraction, we use feature selection, specifically via a regression algorithm known as General Linear Model (GLM). From GLM, we can evaluate the variable importance (VI) of each feature. For most classification models, each feature used as a predictor will have a separate VI for each class (the exceptions are classification trees, bagged trees and boosted trees). Fig. 1 shows the top 20 features based on VI out of 61 features using the R statistics tool [17][18]. The original R output for the 20 variables is given in fig. 2.

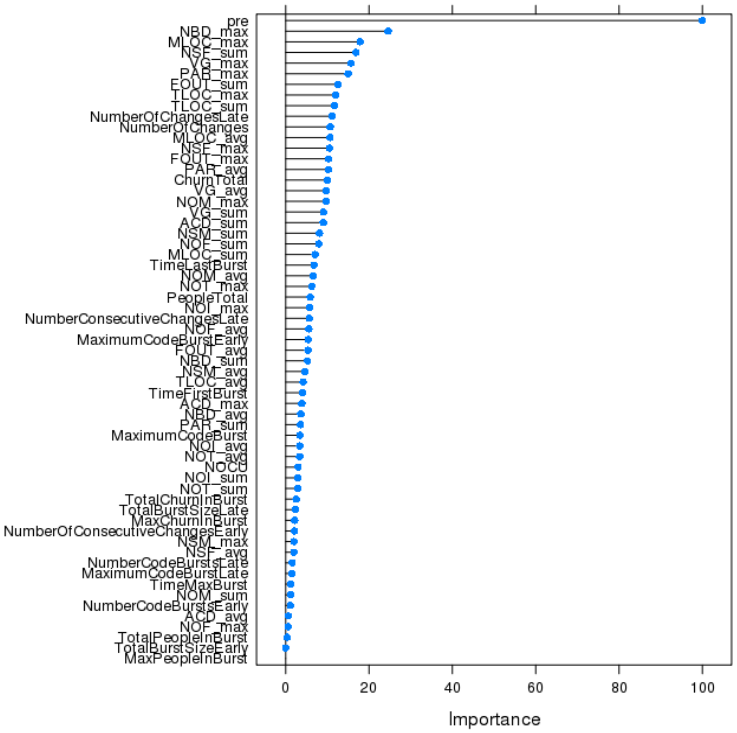

Fig. 1. Variable importance results of the 61 features in the recursive feature selection algorithm.

From the evaluation of VI (cf. Fig. 1 and 2), the top two features, i.e., Pre (pre-release bug) and NBD_max (max number of Nested block depth), are chosen as predictors for training and testing phases in the three algorithms. Pre represents the number of pre-released defects and NBD_max is the max value of nested block depth. The reason for choosing the top 2 predictors is because they not only have high VI values, but represented both process metrics and code metrics categories. As will be shown in Section IV, the final results also demonstrate that they have a significant impact on predicting defects. Furthermore, we choose the two features in order to reduce the size of the data because it will take a lot time to train the data. In addition, dimension reduction using feature selection is necessary for performance. Furthermore, the correlations (cf. fig. 3) between the top 13 features show that the pre and NBD_max are not highly correlated hence the choice of these variables.

```
glm variable importance

  only 20 most important variables shown (out of 61)

                    Overall
pre                 100.000
NBD_max              24.590
MLOC_max             17.882
NSF_sum              16.860
VG_max               15.675
PAR_max              14.980
FOUT_sum             12.497
TLOC_max             11.979
TLOC_sum             11.709
NumberOfChangesLate  11.150
NumberOfChanges      10.683
MLOC_avg             10.647
NSF_max              10.506
FOUT_max             10.287
PAR_avg              10.241
ChurnTotal           10.015
VG_avg                9.733
NOM_max               9.671
VG_sum                9.069
ACD_sum               9.009
```

Fig. 2. R output of the variable importance using GLM algorithm.

Analysis of the Correlation of Features – *The second step in the data preprocessing is the correlation analysis. Correlation analysis of the features can help better understand data and find best predictors among the features.*

To analyze the correlation of features, a correlation plot is generated using the R data analysis tool [20] (fig. 3). The correlation plot for 11 predicators and a class is displayed in Fig. 3. These predictors are the first 11 features generated when evaluating the variable importance (VI) in Fig.1. The color blue indicates positive correlation, and the color red indicates negative correlation. The correlation is stronger when the color is darker (or larger circle size and value).

Fig. 3 shows that features (columns) 2-4, i.e., Pre (pre-release bug), NBD_max (max number of Nested block depth), FOUT_sum (sum of method calls (fan out)), and TLOC_sum (total lines of code) have a strong positive correlation with the class – Number of defects (the 1st column). Currently, it is labeled with a binary number (0 or 1) in the dataset. Therefore, these features are considered to have a high impact on the defect-prone component. There is no negative correlation among these features. Further, VG_max (6th column) (max number of McCabe cyclomatic complexity) and MLOC_max (3rd column) (max Method lines of code) have a strong positive correlation with a coefficient of 0.9. Likewise, TLOC_sum and FOUT_sum also have a strong positive correlation of 0.98. The correlation result reveals that it is highly likely that redundancy exists in these features.

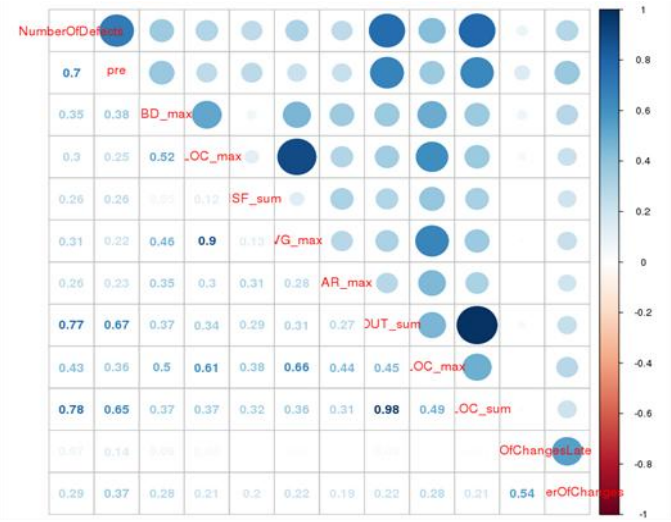

Fig. 3. Correlation matrix for top 11 features and class.

There are also some feature pairs that are loosely correlated or have almost no correlation (e.g., Pre (2nd column) and NBD_max (3rd column). Hence, it has a small absolute correlation value 0.38. The features Pre and NBD_max are not strongly correlated, which indicates that the number of pre-release defects and the max number of nested block depth are not very closely related.

Dimension Reduction Using PCA (Principal Component Analysis) **–** In addition to using variable importance (VI) to choose the two most important variable, we apply PCA to our data to avoid information loss and to compare the result of using VI to PCA in order to see if there is any significant difference during the experiments. In general, high dimensional dataset is a significant challenge for data analysis if the volume is large. Therefore, dimension reduction techniques can reduce data size and save substantial efforts during the training phase. With dimension reduction, the original data is mapped from a high-dimensional space to a lower dimensional space. This step is not mandatory. However this step will not only simplify the analysis and reduce computational time, but also can eliminate redundant information if some features are closely related. In section V, we will compare the quality of the results of applying PCA

and VI. In general, PCA is a linear dimension reduction technique which reduces the number of features (or dimensions) in a dataset. This is done by finding a few principal components (PCs) with large variances in the transformed new space. PCs are orthogonal and are generated by a series of linear combinations of the variables [21]. Fig. 4 below shows the result of the PCA model using GLM algorithm.

```
Call:  NULL

Coefficients:
(Intercept)          PC1          PC2          PC3          PC4          PC5
    0.21011      0.41210     -0.01327      0.05204     -0.14240      0.07062
        PC6          PC7          PC8          PC9         PC10         PC11
    0.18474     -0.23715      0.10021     -0.04348     -0.17915     -0.18184
       PC12         PC13         PC14         PC15         PC16         PC17
   -0.12663      0.01667     -0.24819      0.35707     -0.34697      0.14371
       PC18         PC19         PC20         PC21         PC22         PC23
   -0.11080      0.16131     -0.12307      0.04375      0.25871     -1.20001
       PC24         PC25         PC26         PC27
    0.73011      0.23460      0.71400     -0.50819

Degrees of Freedom: 102674 Total (i.e. Null);  102647 Residual
Null Deviance:      142300
Residual Deviance: 101700       AIC: 101800
```

Fig. 4. R Output of the PCA model using GLM algorithm.

In this experiment, we set the threshold of PCA as 0.95, since it is not too high to lose significant information and it is not too low to include noise. Consequently, 27 transformed features are selected during this process.

Evaluation of Classification Quality – In this paper, we adopted binary classification, which means that we predict whether a component/file has a defect or not. The content of Table 1 shows the confusion matrix which is generally recommended for assessing the quality of binary classification. The confusion matrix, sensitivity as shown in Eq (1), and specificity shown in Eq (2) are mainly used as classification measures [22].

The ROC (Receiver operating characteristic) curve and AUC (Area under Curve) [23] are well known graphical techniques that are useful for evaluating the quality of classification models. Moreover, $\mathbf{A_{ROC}}$ is defined as the area under the ROC curve. And in our case, if $\mathbf{A_{ROC}}$ is closer to 1, it means a better result for the related algorithms. The ROC graph is sometimes called the *sensitivity vs. (1 − specificity)* plot. Each point in the ROC space represents a prediction result or instance of a confusion matrix (see Table.1). Moreover, the best possible prediction approach would yield a point in the upper left corner or coordinate (0, 1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives) [23]. In this research work, we choose the value of $\mathbf{A_{ROC}}$ to measure the quality of results of different machine learning models. $\mathbf{A_{ROC}}$ is calculated using the R tool [20] [21]. $\mathbf{A_{ROC}}$ is selected as the measurement of quality, because it incorporates sensitivity and specificity and we need to investigate both true positive and true negative.

Table 1. Confusion Matrix

		Defects are Observed	
		TRUE	FALSE
Predicted Condition	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

$$\text{sensitivity (or recall)} = \frac{TP}{TP+FN} \quad (1)$$

$$\text{specificity} = \frac{TN}{FP+TN} \quad (2)$$

4. Data Training

This section presents steps taken and results of data training with emphasis on the three algorithms: NN, SVM, and RF. The tuning parameter is used to construct the data model during data training. To start with, we utilize linear classification model (i.e. GLM) to train the entire data of 61 features. The summary of the training process is presented in Appendix A1. The cross validation is used to avoid overfitting and repeat cross validation to get maximize efficiency.

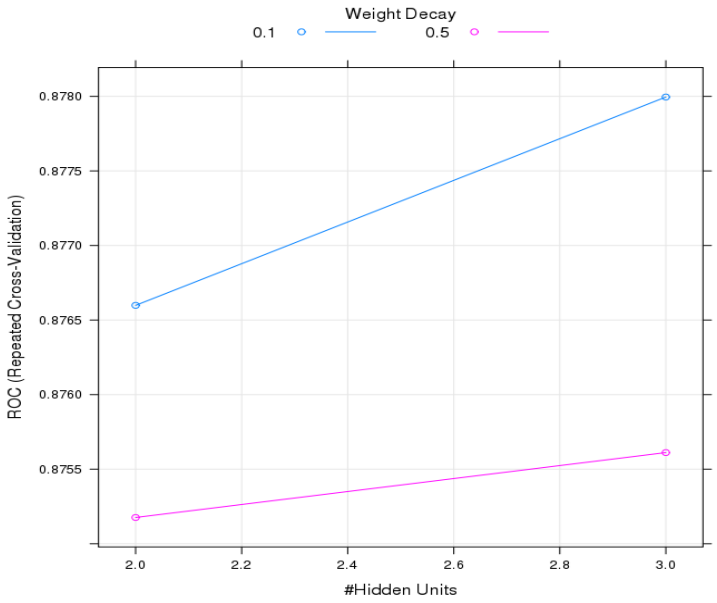

Fig. 5. NN training result visualization.

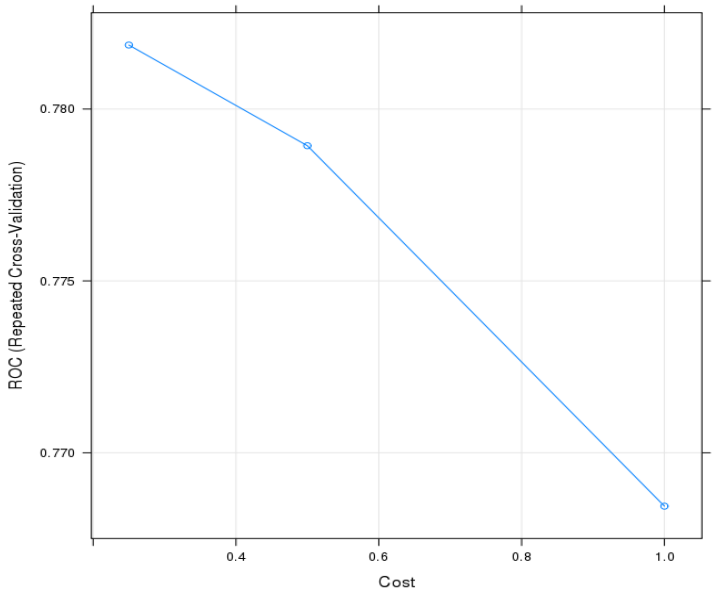

Fig. 6. SVM training result visualization (R output).

4.1. Neural Networks (NN) Training

For the Neural Network algorithm, 102,675 samples were used. Cross-validation resampling was also

used, as previously stated, with 10 folds and 2 repetitions spread across multiple iterations of the *decay* and *size* parameters in the algorithm's iterative process and via the utilization of a ROC. We finally decided a size value of 3 and a decay value of 0.1 which correspond to a highest $\mathbf{A_{ROC}}$ value of 0.878 (cf. fig. 5). Fig. 5 shows the NN training result visualization. The R output for the NN training data is shown in Appendix A2.

4.2. Support Vector Machine (SVM) Training

Like the NN algorithm, there are 102675 samples, 232 predictors, and 2 classes corresponding to a binary classification. A cross validation (CV) resampling technique was used as well as an iterative process based on one turning parameter and cost. The other turning parameter, sigma, was held constant at the value 132. The R visualization of the SVM training result is given in fig. 6. The SVM training result is shown in Appendix A3. Based on fig. 6 (and Appendix A3), the optimization across the various parameter values relied on the highest value of ROC (i.e. 0.782) in order to select the optimal cost which is 0.25.

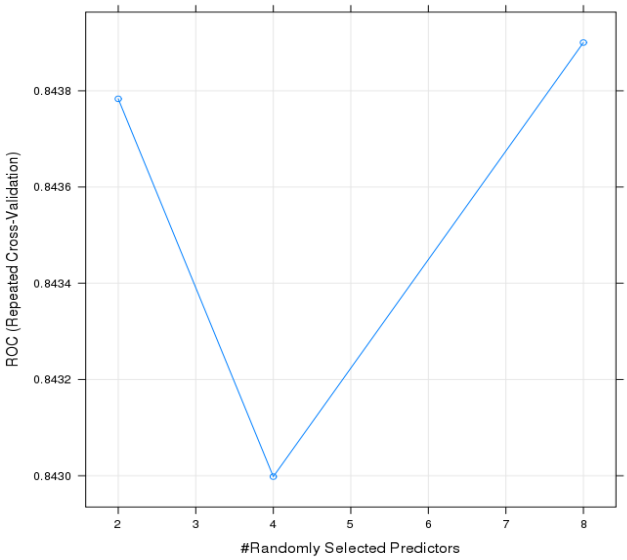

Fig. 7. RF training result visualization.

4.3. Random Forest (RF) Training

The same procedure was used for the RF model training (i.e. 102675 samples, 232 predictors, 2 classes, and CV resampling). Fig. 7 shows the relationship between the different values of ROC and the number of randomly (#Randomly) selected predictors. The highest point corresponds to 8 and a ROC value just shy of 0.844. This illustrates an uneven relationship between the ROC and number of randomly selected predictor parameters. The R output of the RF training result is presented in Appendix A4.

4.4. Summary of Data Training

Finally, we train the result of PCA output (section IIIB) using GLM model with 27 topmost transformed features. The ROC value for this process is 0.835. Note that we used the same data set that is, 102675 samples, 232 predictors, and 2 classes (binary classification). The out of GLM result is presented in Appendix A5.

In this section we have discussed how data are trained using five models – Linear Regression, Neural Networks, Support Vector Machine, Random Forests, and PCA. All these trained models will be used in the data testing and the final results. In addition, the trained model for the linear regression data set that includes all the features will be used as control experiment for NN, SVM, and RF in section V. PCA data test will also be used for comparison.

In conclusion, we hypothesis as follows:

- In terms of sensitivity, the Support Vector Machine (SVM) model will perform slightly better than Neural Networks (NN) and Random Forest (RF).
- In terms of specificity, the RF model will perform slightly better than NN and SVM
- In general and using AROC (i.e. sensitivity vs (1 – specificity)), NN and RF will perform better than SVM.
- There is little or no statistical difference between using a reduced number of features compared to using all the features in a data set.

5. Results and Analysis

5.1. Experiment Setup

The platform used for the experiments has the following configuration: 16 (CPUs) AMD Opteron Processor 4386 (3.1 GHz with 8 cores), 32 GB of memory, and 1770 GB of storage, running on 64-bit CentOS Linux 6.0 operating system.

As stated in Section III, the first step is data pre-processing from which and based on VI evaluation, Pre and NBD_max are chosen as predictors to predict defects of Eclipse project, since both features are the 2 topmost features with stronger VI than other features, as shown in figures 1 and 2.

The second and third steps are training and testing data. The predictive results using the three algorithms, NN, SVM, and RF, are depicted in TABLE 2. The main comparative metric used in this paper is A_{ROC}, as explained in Section III.B. This table shows the results for the three machine learning models using only two features – Pre and NBD_max. In addition, TABLE 2 also shows the results of using PCA with 27 transformed features, and linear regression using all the features. The linear regression result is used as a control to compare the three algorithms and PCA. Note that all the outputs of R statistics in relation to TABLE 2 are given in Appendix B (B1 to B5).

5.2. Experiment Results Analysis

The result of using GLM linear regression with 61 features serves as a benchmark to compare other results, as all features are used for the evaluation. The predicative efficiency of GLM with 61 features shows that the sensitivity and specificity values are 0.7642 and 0.8744, respectively, and A_{ROC} value equals 0.888 (cf. TABLE 2, Appendix B1). The result of using NN with Pre and NBD_max has an A_{ROC} value of 0.878 which is comparable to (0.888) using GLM with 61 features. The results indicate that the features Pre and NBD_max have a significant impact on the defect-prone component, which confirms our decision to select Pre and NBD_max as the main features for defect prediction in this research work.

The GLM with PCA features, by contrast, shows that the A_{ROC} value is 0.835. Note that, due to information loss, the result of GLM with PCA feature reduction may not be as good as some other algorithms. However, the computational time can be lower using feature reduction. The time savings can be evident for a large volume of data.

Table 2. The Result of Classification Models

	Sensitivity	Specificity	A_{ROC}
Linear Regression (with 61 features)	0.7642	0.8744	0.888
Neural Network (with *Pre* & *NBD_max* features)	0.7778	0.8273	0.878
Support Vector Machine (with *Pre* & *NBD_max* features)	0.7988	0.7852	0.782

Random Forest (with *Pre* & *NBD_max* features)	0.7717	0.8634	0.844
PCA (with 27 transformed features)	0.7091	0.8006	0.835

In comparing the performance of the three machine learning models, the predictive capabilities of the NN and RF algorithms are preferable to that of SVM algorithm. The reason is primarily due to the predictive specificity value that each technique generates. The $\mathbf{A_{ROC}}$ results obtained from NN and RF are 0.878 and 0.844, respectively, which are superior to SVM's 0.782. The result indicates that NN and RF techniques are more suitable for the experimental data. On the other hand, the SVM model has a sensitivity of 0.7988 in comparison to 0.7778 and 0.7717 obtained from the NN and RF, respectively. The sensitivity value obtained from SVM is even higher than that of the GLM.

The sensitivity value reflects True Positive. A higher sensitivity value implies that the corresponding component may be more vulnerable. As a result, project managers may be interested in finding the components that are most vulnerable to defects in order to facilitate resource allocation to areas that are in need of more resources and management of software verification. In this case, SVM model would be a good candidate, as it generates higher sensitivity value than other models. However, in this paper and judging by the $\mathbf{A_{ROC}}$ values in TABLE 2, NN and RF are better, because $\mathbf{A_{ROC}}$ is more generic and comprehensive because it incorporates both sensitivity and specificity.

Furthermore, Fig.8 is generated with techniques for making inferential comparisons using resampling [17] [18]. This has the effect of using the same resampling data sets for the NN, SVM, and RF algorithms. Fig. 8 plots the ROC of the three algorithms used in this study. Effectively, we have paired estimates of performance. It is clear that the NN and RF algorithms perform more effectively than SVM due to the higher A_{ROC} values generated from those two algorithms.

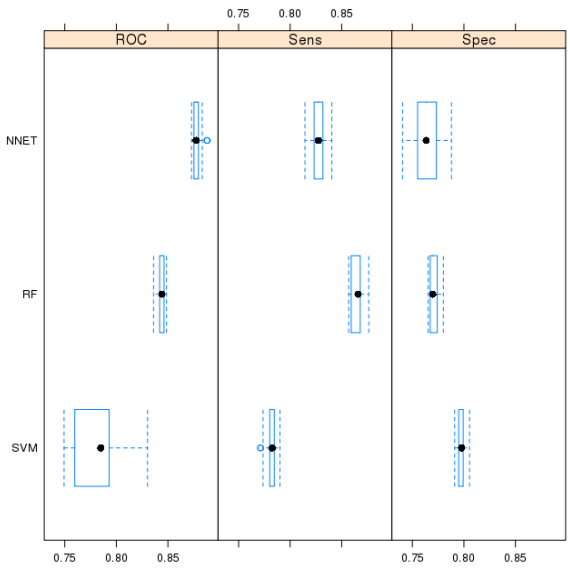

Fig. 8. Plot of comparing NN, RF, and SVM.

We also use the probabilistic classifier with Pre and NBD_max using the machine learning models for evaluating prediction efficiency. Probabilistic classifiers [19] assign a score or a probability to each sample. A probabilistic classifier is a function f: X -> [0, 1] that maps each sample x to a real number f(x). Normally, a threshold *t* is chosen for which the samples with f(x) ≥ *t* are considered positive. This implies that each pair of a probabilistic classifier and threshold *t* defines a binary classifier. In this research work, we set *t* to be 0.5.

Figs. 9, 10, and 11 depict the ROC curve for NN, RF and SVM respectively. The graph and A_{ROC} are generated using the R tool. The AROC result for NN is better than that of RF and SVM. Specifically, A_{ROC} for NN is 0.8796, whereas A_{ROC} for RF is 0.8392 and A_{ROC} for SVM is 0.8729. Note that the result of SVM is very close to that of the NN using the probabilistic classifier.

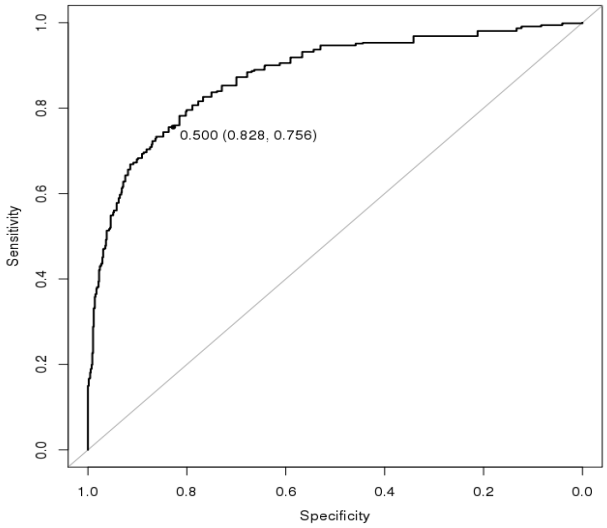

Fig. 9. The ROC curve for neural network for probabilistic classifier.

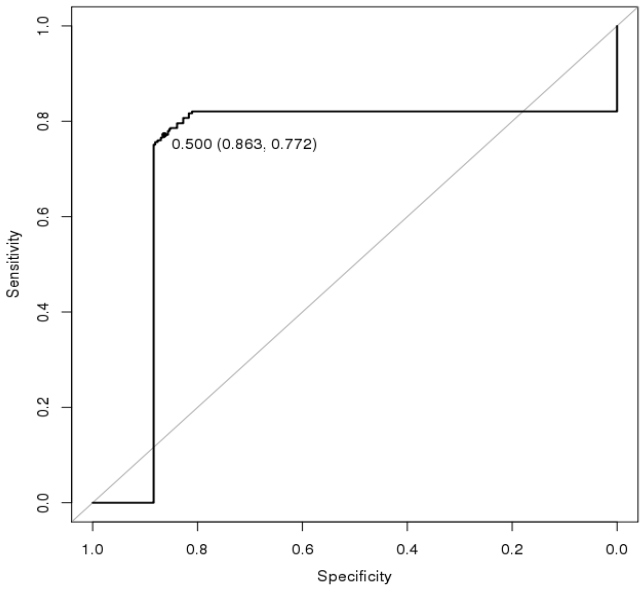

Fig. 10. The ROC curve for random forest for probability classifier.

Although we have shown that using the ROC curve (cf. figs. 9, 10, and 11) that individually, the three models (i.e. NN, SVM, and RG) are effective both for training and testing the Eclipse data. In what follows, we want to further demonstrate the effectiveness of the three models combined using a *Lift Curve over Probability*. Lift curve is a measure of the performance of a targeting model at predicting or classifying cases as having an enhanced response vis-à-vis the sample population measured against a random targeting model. In this case our targeting models are NN, SVM, and RF. A targeting model is doing a good job if the response within the target is better than the average for the population as a whole. In plain language, lift curve shows the total number of events captured by a model over a given number of samples. It is shown from fig. 12 that the three models perform better than the average for the population. In general, the models

captured close to 80% sample (i.e. % sample tested – horizontal axis) compared to the population which is roughly 60% (i.e. % sample found - vertical axis).

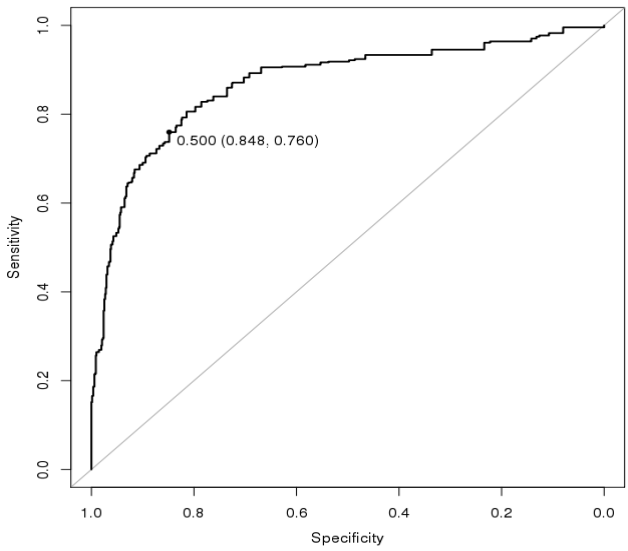

Fig. 11. The ROC curve for SVM for probabilistic classifier.

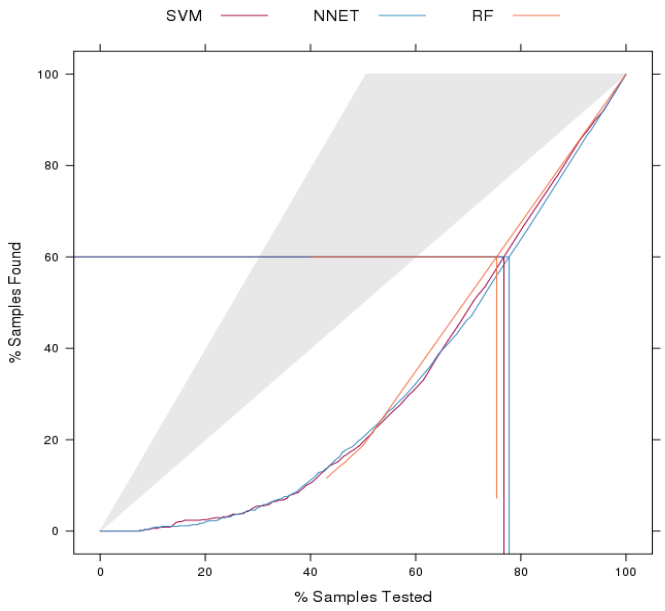

Fig. 12. Lift curve over probability for three models - NN, SVM, and RF.

6. Conclusions and Future Work

Effective defect prediction can facilitate software project management and resource allocation. This paper investigated defect prediction for the Eclipse project. The main objective of this paper was to identify the key features which may have a significant impact on identifying defect-prone components or files in a specific software project and to determine which algorithm will be the best fit.

The results obtained from our experiments showed that Pre and NBD_MAX were key features for classifying defect-prone components. This means that if the source code is complex and there are pre-release defects in one component, there is a high possibility to have post-release defects in this component. For our investigation, we also achieved comparable results with only two features (compared

to using all the 61 features), which can reduce data size and hence reduce computational time.

Among the three machine learning models investigated in this paper, NN possesses the best fit and predictive capabilities as judged by its A_{ROC} value (0.878) in comparison to SVM and RF. On the other hand, SVM resulted in highest sensitivity value, which may be useful in identifying defect-prone components. Furthermore, we used PCA for dimension reduction. PCA may lose some information. Therefore, the result of GLM with PCA may not be as good as other better algorithms. But using feature reduction can improve performance which could be useful if the volume or number of features is high.

Although our aim in this research is not to accept or reject any hypothesis but we can conclude by accepting the hypotheses given as conclusion to section IV within the limit of the dataset used in this research. The only source of bias in this research is that it is applied to one dataset that is, the Eclipse bug datasets. Will there be different results using a different dataset or combination of datasets – this is an open research question.

A more effective approach to defect prediction is to continuous and dynamic application of various techniques to project releases and evaluate the results and use these results as a feedback into the analysis. This approach warrants further investigation and research. In any case, the complexity of such an approach is yet to be investigated.

Appendices

Appendix A

Appendix A1 - General Linear Regression Training Result (*Source:* R output)

```
> lm
Generalized Linear Model

102675 samples
   232 predictors
     2 classes: '0', '1'

Pre-processing: re-scaling to [0, 1]
Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 92407, 92407, 92408, 92407, 92407, 92407, ...

Resampling results

  ROC    Sens   Spec   ROC SD   Sens SD  Spec SD
  0.888  0.871  0.759  0.00192  0.00467  0.00674
```

Appendix A2 - Neural Networks Training Result (*Source:* R output)

```
Neural Network

102675 samples
   232 predictors
     2 classes: '0', '1'

Pre-processing: re-scaling to [0, 1]
Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 92407, 92407, 92407, 92408, 92407, 92408, .

Resampling results across tuning parameters:

  decay  size  ROC    Sens   Spec   ROC SD   Sens SD  Spec SD
  0.1    2     0.877  0.83   0.767  0.00367  0.00707  0.0129
  0.1    3     0.878  0.828  0.764  0.00348  0.00674  0.0127
  0.5    2     0.875  0.846  0.738  0.00348  0.0051   0.00689
  0.5    3     0.876  0.845  0.74   0.00355  0.005    0.00675

ROC was used to select the optimal model using  the largest value.
The final values used for the model were size = 3 and decay = 0.1.
```

Appendix A3 - SVM Training Result (*Source:* R output)

```
Support Vector Machines with Radial Basis Function Kernel

102675 samples
   232 predictors
     2 classes: '0', '1'

Pre-processing: centered, scaled
Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 92408, 92408, 92407, 92407, 92407, 92407, ...

Resampling results across tuning parameters:

  C     ROC    Sens   Spec   ROC SD  Sens SD  Spec SD
  0.25  0.782  0.783  0.797  0.0238  0.00497  0.00361
  0.5   0.779  0.783  0.797  0.0231  0.00497  0.00361
  1     0.768  0.783  0.797  0.0163  0.00497  0.00361

Tuning parameter 'sigma' was held constant at a value of 132.4306
ROC was used to select the optimal model using  the largest value.
The final values used for the model were sigma = 132 and C = 0.25.
```

Appendix A4 - Random Forest Training Result (*Source:* R output)

```
Random Forest

102675 samples
   232 predictors
     2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 2 times)

Summary of sample sizes: 92407, 92408, 92408, 92407, 92409, 92407, ...

Resampling results across tuning parameters:

  mtry  ROC    Sens   Spec   ROC SD   Sens SD  Spec SD
  2     0.844  0.865  0.771  0.00315  0.0054   0.00478
  4     0.843  0.864  0.772  0.0029   0.00593  0.00428
  8     0.844  0.865  0.771  0.00309  0.00599  0.00449

ROC was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 8.
```

Appendix A5 - R output of PCA Training Result (*Source*: R output)

```
Generalized Linear Model

102675 samples
   232 predictors
     2 classes: '0', '1'

Pre-processing: principal component signal extraction, scaled, centered
Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 102675, 102675, 102675, 102675, 102675, 102675, ...

Resampling results

  ROC    Sens   Spec   ROC SD   Sens SD  Spec SD
  0.835  0.805  0.706  0.00307  0.00653  0.00461
```

Appendix B

Appendix B1 - General Linear Regression Predictive Result with 62 Indicators (*Source:* R output)

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 15136  3988
         1  2175 12926

               Accuracy : 0.8199
                 95% CI : (0.8158, 0.824)
    No Information Rate : 0.5058
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6394
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7642
            Specificity : 0.8744
         Pos Pred Value : 0.8560
         Neg Pred Value : 0.7915
             Prevalence : 0.4942
         Detection Rate : 0.3777
   Detection Prevalence : 0.4412
      Balanced Accuracy : 0.8193

       'Positive' Class : 1
```

Appendix B2 - Neural Network Predictive Result Visualization (*Source:* R output)

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 14322  3758
         1  2989 13156

               Accuracy : 0.8029
                 95% CI : (0.7986, 0.8071)
    No Information Rate : 0.5058
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6055
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7778
            Specificity : 0.8273
         Pos Pred Value : 0.8149
         Neg Pred Value : 0.7921
             Prevalence : 0.4942
         Detection Rate : 0.3844
   Detection Prevalence : 0.4717
      Balanced Accuracy : 0.8026

       'Positive' Class : 1
```

Appendix B3 - Support Vector Machine Predictive Result (*Source:* R output)

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 13593  3403
         1  3718 13511

               Accuracy : 0.7919
                 95% CI : (0.7876, 0.7962)
    No Information Rate : 0.5058
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5839
 Mcnemar's Test P-Value : 0.0001984

            Sensitivity : 0.7988
            Specificity : 0.7852
         Pos Pred Value : 0.7842
         Neg Pred Value : 0.7998
             Prevalence : 0.4942
         Detection Rate : 0.3948
   Detection Prevalence : 0.5034
      Balanced Accuracy : 0.7920

       'Positive' Class : 1
```

Appendix B4 - Random Forest Predictive Result (*Source:* R output)

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 14946  3861
         1  2365 13053

               Accuracy : 0.8181
                 95% CI : (0.814, 0.8222)
    No Information Rate : 0.5058
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6358
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7717
            Specificity : 0.8634
         Pos Pred Value : 0.8466
         Neg Pred Value : 0.7947
             Prevalence : 0.4942
         Detection Rate : 0.3814
   Detection Prevalence : 0.4505
      Balanced Accuracy : 0.8176

       'Positive' Class : 1
```

Appendix B5 - Predictive Result with 27 Indicators from PCA (*Source:* R output)

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 13851  4924
         1  3449 12001

               Accuracy : 0.7554
                 95% CI : (0.7508, 0.7599)
    No Information Rate : 0.5055
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5102
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7091
            Specificity : 0.8006
         Pos Pred Value : 0.7768
         Neg Pred Value : 0.7377
             Prevalence : 0.4945
         Detection Rate : 0.3507
   Detection Prevalence : 0.4514
      Balanced Accuracy : 0.7549

       'Positive' Class : 1
```

Appendix B5 - Three Models (NN, SVM, and RF) Resample ROC Result (*Source:* R output)

```
Call:
summary.resamples(object = cvValues)

Models: NNET, SVM, RF
Number of resamples: 40

ROC
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
NNET 0.8729  0.8755 0.8774 0.8780  0.8796 0.8883    0
SVM  0.7493  0.7600 0.7850 0.7819  0.7926 0.8302    0
RF   0.8361  0.8421 0.8441 0.8439  0.8462 0.8487    0

Sens
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
NNET 0.8144  0.8234 0.8275 0.8279  0.8314 0.8404    0
SVM  0.7712  0.7804 0.7826 0.7826  0.7844 0.7901    0
RF   0.8568  0.8593 0.8659 0.8651  0.8679 0.8763    0

Spec
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
NNET 0.7404  0.7551 0.7634 0.7644  0.7733 0.7880    0
SVM  0.7911  0.7954 0.7978 0.7972  0.7992 0.8054    0
RF   0.7653  0.7674 0.7697 0.7709  0.7736 0.7801    0
```

Acknowledgment

This research is partly sponsored by two research grants (Engage and Discovery grants) from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, *22(10)*, 267-271.

[2] Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2007). Using developer information as a factor for fault prediction. *Proceedings of the* 3*rd International Workshop on Predictor Models in Software Engineering*.

[3] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, *32(11)*, 1-12.

[4] Nagappan, N., & Ball, T. (2005b). Use of relative code churn measures to predict system defect density. *Proceedings of the 27th International Conference on Software Engineering* (pp. 284–292).

[5] Trevor, H., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. New York: Springer.

[6] Russell, I. (1993). Neural networks. *UMAP Journal*. Retrieved from: http://uhaweb.hartford.edu/compsci/nnintro.pdf

[7] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20(3)*, 273-297.

[8] Ho, T. K. (1995). Random decision forests (PDF). *Proceedings of the 3rd International Conference on Document Analysis and Recognition*.

[9] Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20(8)*, 832–844.

[10] Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning. Springer, Berlin: Springer Series in Statistics.

[11] Ajila, S. A., & Dumitrescu, R. T. (2007). Experimental use of code delta, code churn, and rate of change to understand software product line evolution. *The Journal of Systems and Software*, *80(1)* , 74–91.

[12] Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proceedings of the 30th Int'l Conf. on Software Eng*., 181-190.

[13] Kim, S., Zimmermann, T., Whitehead, J., & Zeller, A. (2007) Predicting faults from cached history. *Proceedings of the 29th International Conference on Software Engineering* (pp. 489–498).

[14] Nagappan, N., & Ball, T. (2005a). Static analysis tools as early indicators of pre-release defect density. *Proceedings of the 27th International Conference on Software Engineering*, 580–586

[15] Nagappan, N., Ball, T., & Zeller, A. (2006). Mining metrics to predict component failures. *Proceedings of the 28th International Conference on Software Engineering* (pp. 452–461).

[16] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K-i., Adams, B., & Hassan, A. E. (2010). Revisiting common bug prediction findings using effort aware models. *Proceedings of the 26th IEEE International Conference on Software Maintenance* (pp. 1–10).

[17] Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. *Proceedings of the 3rd International Workshop on Predictive models in Software Engineering* (pp. 9–15).

[18] Zimmermann, T., Nagappan, N., Gall H., Giger E., & Murphy B. (2009). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *Proceedings of the 7th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (pp. 91–100).

[19] Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., & Murphy, B. (2010). Change bursts as defect predictors. *Software Reliability Engineering*, 309-318.

[20] R, the comprehensive r archive network. Retrieved from: http://www.r-project.org/
[21] R, the r caret package. Retrieved from: https://github.com/topepo/caret/ [Accessed July 2 2015].
[22] Fodor, I. K., (2002). A survey of dimension reduction techniques. *Technical Report UCRL-ID-148494,* Lawrence Livermore National Laboratory.
[23] Vuk, M., & Curk, T. (2006). ROC curve, lift chart and calibration plot. *Metodolǒski Zvezki*, *3(1)*, pp.89-108.

Wenjing Han received her B.S. degree in computer science and technology from Northeastern University, China and M.Eng degree in system and computer engineering specialization in data science from Carleton University, Ottawa, Canada. She was a software engineer at Oracle Corporation from 2009 to 2014. Currently she is a software engineer at Filecatalyst, Ottawa, Canada.

Chung-Horng Lung received the B.S. degree in computer science and engineering from Chung-Yuan Christian University, Taiwan and the M.S. and Ph.D. degrees in computer science and engineering from Arizona State University. He was with Nortel Networks from 1995 to 2001. In September 2001, he joined the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is now a professor. His research interests include software engineering and communication networks.

Samuel A. Ajila is currently an associate professor of engineering at the Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada. He received B.Sc. (Hons.) degree in computer science from University of Ibadan, Ibadan, Nigeria and a Ph.D. degree in computer engineering specializing in software engineering and knowledge-based systems from LORIA, Université Henri Poincaré – Nancy I, Nancy, France. His research interests are in the fields of software engineering, cloud computing, and big data analytics and technology management.