Software Defect Prediction Using Minimized Attributes

Md. Habibur Rahman, Sadia Sharmin, Sheikh Muhammad Sarwar, Shah Mostafa Khaled, and Mohammad Shoyaib*

***Abstract*—Software quality estimation requires the identification of the number of defects that exist in a software. A software can be represented by a set of static code attributes and these attributes can be used to determine the defectiveness using simple statistical and machine learning tools. Among those attributes, all are not equally responsible for determining whether a software is defective or not. In this paper, we propose an attribute selection technique to select the most responsible attributes for building a defect prediction model. The model is experimented for both the within project and cross-project defect prediction using NASA Metric datasets and Relink. We observed a significant improvement in accuracy for both the within project and cross-project defect prediction, which proves the effectiveness of our proposed model.**

***Keywords*—software defect prediction; attribute selection; software engineering.**

I. INTRODUCTION

Software quality estimation and maintenance is one of the most important steps in software development life cycle. During the software development, there are always limited resources (i.e. people, time, and budget) and thus project managers face difficulties to allocate resources properly. Finding the software bugs and fixing them before the release is a hard task for a software development organization. Software defect prediction may be done using a statistical or machine learning approach to help finding the defected parts of the software. If the defected parts of the software are identified, it will be helpful for the project managers to assign the available resources properly. Moreover, specific parts of the software can be focused to be tested in spite of testing the whole project reducing the redundant testing time. So, identifying the defected part of the software is a very important aspect of software engineering.

In recent years, software defect prediction using code metrics has grown much attraction in software testing and quality assurance community [1] [2] [3] [4]. So, a good amount of defect prediction dataset such as NASA Metrics Data Program (MDP), PROMISE repository, Apache dataset etc. has been developed to assist the software defect prediction. These historical dataset is used to build a defect prediction model using statistical and machine learning classifier. In feature selection based software defect prediction, a software module is said to be defective for a certain values of those features. But all the features are not equally responsible for the defectiveness. It is observed that a few features are always responsible for the defectiveness of the software module. This condition pushed researchers to develop an approach to identify those features and propose a defect predictor based on those features. There has been a good amount of work on attribute selection technique to improve the software defect prediction performance [5] [6]. Gao et al. evaluated seven different feature ranking techniques using five commonly used classifier algorithms [7]. A Meta learner framework called WHICH has been proposed to change the standard goal to improve the learning capability [8]. Khoshgoftaar et al. studied the feature selection technique considering the learning impact of data sampling on classification models [9]. A combination of multiple feature selection method may improve the overall defect prediction performance. This has been studied and applied by examining 17 different ensembles of feature rankers [10].

Defect prediction requires a reasonable amount of historical data for a project to train the prediction model. But for a new project, sufficient historical

* Corresponding author.

Md. Habibur Rahman, Sadia Sharmin, Sheikh Muhammad Sarwar, Shah Mostafa Khaled, and Mohammad Shoyaib are with the Institute of Information Technology, University of Dhaka. e-mail: bit0216@iit.du.ac.bd, bit0426@iit.du.ac.bd, smsarwar@du.ac.bd, khaled@du.ac.bd and shoyaib@du.ac.bd.

Manuscript received February 10, 2015; revised May 14, 2015.

data may not be available which makes defect prediction within a project difficult. In these circumstances, historical data of the same types of project may help building the prediction model. Considering the scarcity of historical data, cross-project defect prediction techniques have been a great interest for the software defect prediction researchers. In [11], Zimmermann et al. mentioned cross project defect prediction as a serious problem and also concluded that using projects of same domain (i.e. web browser) and different companies (i.e. Mozilla/Google) have a very poor prediction performance. Rahman et al. concludes that cross-project defect prediction gives similar result relatively with within project prediction or at least not worse than that [12]. In [13], Peters et al. assesses the performance of cross-project defect prediction based on peters filter with the Burak filter which is used within company prediction. This shows an improved performance for cross-company defect prediction. Nam et al. [12] applied a state-of-the-art learning approach, which makes the source and target projects similar. They performed a data preprocessing technique and proposed a novel approach called TCA+ by defining five rules to improve the prediction performance. It is observed that the approach has a significant improvement in defect prediction performance.

In this paper we propose a defect prediction technique where we find out the most responsible attributes for the defectiveness of the software code. This task has been conducted for both the within project and cross project defect prediction. In within project defect prediction, we used two measurement scales to calculate the prediction accuracy. So, when we build defect predictor and calculate prediction accuracy, we consider it for both the balance and AUC measurement scales. We have implemented our attribute selection process using both the measurement scales and found improvement for both the cases using five common classifiers in respect of five resultant measurement scales such as balance, precision, recall, f-measure and AUC. In cross project defect prediction, as we know the test and train data come from different project, we have performed our attribute selection process for the training dataset. After getting the best set of attributes, we re-sample our train and test data set only with the best set of attributes and performed defect prediction. We found a reasonable improvement in cross-project defect prediction in respect to a recent work conducted by [14]. The within project defect prediction was conducted with five dataset from NASA MDP repository and the cross project defect prediction was performed by three dataset provided by ReLink.

The rest of the paper is organized as follows. Section II overviews the background and related work for both within project and cross-project defect prediction. Section III describes the methodology of the proposed technique while section IV represents the experimental design. A detailed discussion on the experiment has been given in section IV and section VI concludes the research work.

II. Background and Related Work

In software defect prediction a good number of researches have been conducted by the data mining and machine learning community. All the researches have been accomplished using some of the publicly available datasets and classified by a list of statistical and machine learning classifier. The mostly used dataset are from NASA MDP Repository and PROMISE Data Repository. Sometimes data preprocessing has been needed to make the test and train data similar especially for cross-project defect prediction. Wang et al. conducted research on determining the minimum number of attributes needed for the software defect prediction. They showed that a proper metrics selection technique like threshold based feature selection can sort out the necessary metrics related to defect prediction through eliminating the additional one. Five versions of the proposed selection technique have been experimented creating different size of metric subsets to assess their effectiveness at the time of defect prediction. The versions are specified by five performance metrics- Mutual Information (MI), Kolmogorov-Smirnov (KS), Deviance (DV), Area under the ROC (Receiver Operating Characteristic) Curve (AUC), and Area under the Precision-Recall Curve (PRC). The result of the study reveals that only three metrics are enough to build an effective predictor and the removal of 98.5% metrics enhances the performance of the prediction model. However, the experiments can be extended by using more datasets and classifiers in order to strengthen their classification [4].

In [15], Gray et al. discussed on the publicly available NASA Metrics Data Program datasets and their misuse in automated software defect prediction. In this paper, the authors explained the importance of data analysis before training the classifier and conducted a proper cleaning process for 13 sets of original NASA metrics dataset. Their concern arises due to the presence of identical data both in training and testing sets as a result of data replication. The cleaning method consists of five stages including the deletion of constant and repeated attributes, replacing the missing values, enforcing integrity of domain specific expertise and removing redundant and inconsistent instances. The findings of this experiment reveals that the processed datasets become $6-90\%$ less from their original size after cleaning and it improves the accuracy of defect prediction. According to their study, a possible solution of avoiding repeated data point is to record lower level metrics as it will help to minimize the probability of modules having similar metrics [16].

Gao et al. investigated a feature selection technique for choosing software metrics for defect prediction in order to maintain the efficiency and quality of defect prediction [7]. This paper proposed a technique called hybrid attribute selection approach consisting of both feature ranking and feature subset selection. In this experiment, five feature ranking techniques including Chi-square (CS), Information Gain (IG), Gain Ratio (GR), KolmogorovSmirnov statistic (KS), two forms of the Relief Algorithm (RLF), and Symmetrical Uncertainty (SU) were studied along with four feature subset selection algorithms: Exhaustive Search (ES), Heuristic Search (HS), and Automatic Hybrid Search (AHS) and no subset selection. The hybrid method first categorizes the important attributes and reduces the search space using a search algorithm applied in feature ranking approach. Then it chooses the subsets of metrics through features subset selection approaches. It is found that AHS is better than other search algorithm in the context of choosing attributes and the removal of 85% metrics can enhance the performance of the prediction model in some cases [7]. He et al. investigated the effectiveness of different predictors on a simplified metrics set for predicting defective modules of the software [3]. Also, it finds out the rules of selecting training datasets, classifiers and metrics subsets properly for defect prediction on a project. The study was conducted for both within-project and cross-project defect prediction on the 34 releases of 10 open-source projects of PROMISE repository. The authors proposed a method for simplifying the metric sets by filtering them using feature ranking techniques and analyzed the performance of the predictors with these metric sets. The findings of the experiment shows that the predictors with minimal metric sets provide better result and simple classifiers like Nave Bayes perform well in this context [3]. A micro interaction metrics based software defect prediction has been proposed by Lee et al. in [17].

In recent times, research in cross-project defect prediction has been increased rapidly [11] [13] [14]. In [11], the authors indicated that cross-project defect prediction is a serious challenge and they conducted their defect prediction model on a large scale project. They tried to identify the factors that can influence the accuracy of cross-project defect prediction. In [14], the authors studied to transfer the defect learning between train and test dataset using data normalization techniques. They also proposed five rules to transfer the defect learning and showed acceptable prediction accuracy. In cross-project prediction model, data similarity between the test and train dataset is a big challenge which has been studied by [14] [18] [12].

In our research work, there are two types of challenges to be addressed for both the within project and cross project defect prediction. There have been numerous ways to find the most responsible attributes for the defectiveness of the software. There are such other methods discussed in the above literature review which gives a reasonable improvement in the prediction accuracy. The statistical and machine learning classifiers are always being used in different ways to improve the defect prediction performance. The selection of attributes in parallel with the selection of a good classifier is a big challenge for the defect prediction research community. On the other hand cross-project defect prediction is itself a big challenge in recent times as it is yet to be explored in a broad domain. Data preprocessing is a very nice way to improve the defect prediction result and thus being a preprocessing technique, introducing attribute selection for cross project defect prediction is a challenge to be addressed.

To evaluate the defect prediction performances, a well-accepted metric is proposed in [19] where they used probability of detection (pd) and probability of false alarm (pf). Formal definition for (pd) and (pf) are given in Eq. (1) and (2) where the definition of A, B, C and D can be obtained from the Table 1. Table 1 presents the confusion matrix of a problem where A, B, C, and D denote True Positive, False Positive, False Negative and True Negative respectively.

TABLE 1: Confusion Matrix

	Estimated	
Real	*Defective*	*Non Defective*
Defective	A	C
Non Defective	B	D

Algorithm 1 Defect Prediction Model

```
Input: Set of attributes $\mathcal{A} = a_1, a_2, .........., a_n$
Output: Prediction result and best set of attributes for defect prediction $\overline{\mathcal{A}} = a_1, a_2, .........., a_j$
1: Begin
2: Select Pair wise combinations of attributes from A and store the accuracy of defect prediction for each pair
3: Create a sorted attribute list $\overline{\mathcal{P}}$ from the set of pair-wise combinations based on an accuracy metric
4: Select the candidate attributes list $A^c$ based on their frequency in $\overline{\mathcal{P}}$
5: Find the best set of attributes $\overline{\mathcal{P}}$ for defect prediction from $A^c$
6: End
```

$$pd = \frac{A}{A+C} \tag{1}$$

$$pf = \frac{B}{B+D} \tag{2}$$

These *pd* and *pf* are then combined to present a third performance measure called balance. Balance is used to choose the optimal (*pd*, *pf*) pairs. The equation for computing the balance according to [19] is shown below using equation (3).

$$balance = 1 - \sqrt{\frac{(1-pd)^2 + (0-pf)^2}{2}} \tag{3}$$

As a continuation of the defect prediction research, we propose an attribute selection process to select the best set of attributes to use as the simplified defect prediction data for both the within project and cross-project defect prediction. The considered attributes for our experiments has been given in Table 2. The experimental result shows that our prediction model is effective to improve the defect prediction accuracy.

Algorithm 2 Generating and Selecting Potential Combinations of Pairwise Attributes

```
Input: Set of attributes $A = \{a_1, a_2, ..., a_n\}$
1: set of classes $C = \{defected, notdefected\}$
2: Dataset $D : A \times C$ and Classifier $\gamma$
3: $\delta =$ a threshold value for selecting potential attribute pairs
Output: Sorted combination of paired attributes list $P$
4: Begin
5: $P_{temp} \leftarrow \{(u, v) : u \in Aandu \notin v\}$
6: $B \leftarrow \phi$
7: for each $(u, v) \in P_{temp}$ do
8:     $d_i \leftarrow$all accumulated values from dataset D for attribute pair $(u, v)$
9:     Classify $d_i$ using $\gamma$
10:    Evaluate $pd, pf, balance$
11:    $B_i \leftarrow \{pd, pf, balance\}$
12:    $B \leftarrow B \cup B_i$
13: end for
14: Sort $P_temp$ in decreasing order of balance using $B$
15: Return $\bar{P} \subset P_temp, Where \left|\bar{P}\right| = kand\forall x \in \bar{P}, balance(x) \geq \delta$
16: End
```

III. Methodology

This paper proposes an attribute selection process for software defect prediction considering the mutual effect of all pairs of attributes on the performance of a classifier. The selected attributes will be used for within project and cross project defect prediction. In within project defect prediction, a certain portion of the data will be used for training the model and rest of the data will be used for testing. On the other hand, there will be two different datasets in cross project defect prediction for training and testing. The process of selecting best set of attributes for cross project defect prediction contains the following sequential steps:

1) Generating and selecting potential combinations of paired attributes
2) Selecting candidate attributes
3) Finding the best set of attributes
4) Cross Project defect prediction

Algorithm 1 presents the overall process of the proposed defect prediction method.

1) **Generate and select potential combinations of paired attributes**
   Algorithm 2 gives us an overview of generating the sorted pairwise attribute list based on their corresponding balance. Here, a pair wise attribute list is generated using all the attributes of the dataset. If there are n number of attributes then the number of pairs will be $\frac{n(n-1)}{2}$. All

TABLE 2: Software Code Attributes

No.	**Attribute Name**	**Description**
1	loc blank	Number of blank lines
2	branch count	Number of all possible decision paths
3	call pairs	The depth of the calling of a function
4	loc code and comment	Number of lines of code and comments.
5	loc comments	Number of lines of comments
6	condition count	Number of conditions of a code module
7	cyclomatic complexity	Measure of number of literally independent paths.
8	cyclomatic density	Ratio between cyclomatic complexity and system size
9	decision count	Number of possible decision to be taken of a code
10	decision density	Ratio between total decision count and total modules.
11	design complexity	Amount of interactions between modules in system
12	design density	Ratio of design complexity and system size
13	edge count	Number of edges of a source code control flow graph
14	essential complexity	Degree of a module contains unstructured constructs.
15	essential density	Ratio between essential complexity and system size
16	loc executable	Lines of code responsible for the program execution
17	parameter count	Number of parameter to a function/method
18	halstead content	language-independent measure of algo. complexity.
19	halstead difficulty	Measure the program's ability to be comprehended
20	halstead effort	Estimated mental effort to develop the program
21	halstead error est	Calculates the number of errors in a program
22	halstead length	Total number of operator and operand occurrences
23	halstead level	Ratio between normal and compact implementation
24	halstead program time	proportional to programming effort
25	halstead volume	No. of Bits required to store the abstracted program
26	maintenance severity	How difficult it is to maintain a module.
27	node count	Number of nodes of a programs control flow graph
28	num operands	Total number of operands present
29	num operator	Total number of operators present
30	num unique operands	Number of distinct operands
31	num unique operators	Number of distinct operators
32	number of lines	Total number of lines of a programs source code
33	percent comment	Percentage of comments of a programs source code
34	loc total	Total number of lines of code
35	is defective	defect labels (Y/N, True/False)

the paired attributes are used to calculate the balance using a simple classifier and those are sorted in decreasing order of balance. Then in the final list of potential attributes those pairs are retained who have balance great than or equal to a threshold . This indicates a benchmark value of the predicted balance to filter those attribute pairs which are hardly responsible for the defectiveness of the software. The value can be heuristically selected based on the nature of the dataset and measurement scale. This list will be used to select the candidate attribute list.

2) **Selecting Candidate Attributes** From algorithm 2 we get a sorted pairwise potential at-

Algorithm 3 Selecting Candidate Attributes

```
Input: Sorted paired attributes list $\bar{P} = \{P_1, P_2, ..., P_k\}$
Output: Attribute list $A^+$ of attributes along with their frequencies calculated from $\bar{a}$
 1: Begin
 2: $A^+ \leftarrow \phi$
 3: for each  $P_i \in P$ do
 4:     for each  $a_j \in P_i$ do
 5:         increment frequency of $a_j, f_(a_j)$ by one
 6:         $A^+ \leftarrow A^+ \cup \{a_j, f_{a_j}\}$
 7:     end for
 8: end for
 9: Sort $A^+$ on decreasing order of $f_{a_j}$ return it as $candidateattributelist$
10: End
```

tribute list, which can be further exploited to unearth the final set of attributes for defect prediction. As the list is a result of the combinations of n attributes, there are some attributes, which exist in multiple pairs. In Algorithm 3, we count the frequency of the occurrences of the attributes listed in $\bar{P}$ and generate a list A^+ containing a collection of <attribute, frequency> pairs. Then A^+ is sorted in decreasing order of frequency and used in further processing to find the best set of attributes. For our further usage we define sorted list A^+ as candidate attribute list.

3) **Finding the best set of Attributes** In Algorithm 4, we select the best set of attributes from the candidate attribute list A^+. The best set of attribute is used to calculate the final balance value for a specific dataset using a simple classifier. From the candidate attribute list the first attribute is selected as the final attribute and added in a list named best set of attributes. The desired balance is calculated using this best set of attributes and stored for further experiment. The second attribute from the candidate attribute list is then added in the best set of attributes and again calculate the balance. If the current balance is better than the previous balance, the recently added attribute is kept in the final list or the attribute is discarded. This process continues till the last attribute of the candidate attribute list. Then we get the final set of attributes from the best set of attributes list which is used to calculate the final balance.

4) **Cross project defect prediction**
In Cross-project defect prediction, the most important thing is the training and test dataset.

Algorithm 4 Finding the best set of attributes

```
Input: Set of decreasing order sorted attributes $A^+$
 1: Set of classes $C = \{defected, notdefected\}$
 2: Dataset $D : A \times C$ and Classifier $\gamma$
Output: Best set of attributes $F$ for defect prediction with calculated $pd$, $pf$ and $balance$
 3: Begin
 4: $balance \leftarrow 0$
 5: for each  $i = 1, .. \left|\bar{A^+}\right|$ do
 6:     $F \leftarrow F \cup \{a_i\}$
 7:     $tempBalance \leftarrow balancecalculatedusingFand\gamma$
 8:     if $tempBalance \geq balance$ then
 9:         $balance \leftarrow tempBalance$
10:     else
11:         $F \leftarrow F \backslash \{a_i\}$
12:     end if
13: end for
14: Return the set $F$,and $calculatepd, pf$ and $balanceusingF$
15: End
```

Algorithm 5 Cross Project Defect Prediction

```
Input: Data set $D_1$ and $D_2$ and best set of attributes F
Output: $D_{train}$ and $D_{test}$ with the best set of attributes F and f-measure on $D_{test}$
 1: Begin
 2: Attribute set $AD_1 = a_1, a_2, ..., a_n$ obtained from data set $D_1$
 3: $D_{train} \leftarrow AD_1 \cap$ F (with all instances from $D_1$)
 4: Attribute set $AD_2 = a_1, a_2, ..., a_m$ obtained from data set $D_2$
 5: Select $D_{test} \in AD_2 \cap$ F (with all instances from $D_2$)
 6: Use $D_{test}$ as training instances
 7: Calculate pd and pf using $D_{test}$
 8: Calculate f-measure
 9: End
```

In Algorithm 5 we incorporated our attribute selection technique in cross-project defect prediction. The attribute selection process is used for the train dataset D_{train} and from this we get a best set of attribute list. The train and test dataset D_{train} and D_{test} are then simplified by the selected attributes to make the dimensions of the datasets same. Now, the defect predictor is trained using $D_{train}$and the prediction accuracy is calculated using D_{test}. The algorithm for cross project defect prediction using the best set of attributes is shown in Algorithm 5.

IV. Experimental Design

In our defect prediction model, we used five publicly available datasets from NASA MDP Repository for within project prediction and ReLink datasets for cross project prediction. Most of the researchers developing defect prediction models fall short of datasets

TABLE 3: Dataset Overview

Dataset Name	Software Type	Language	Number of Attributes	Number of Instances	Defected data (%)
CM1	NASA Space Craft Instrument	C	38	498	9.83%
PC3	Flight software for earth orbiting satellite	C	38	1125	12.44%
PC4	Flight software for earth orbiting satellite	C	38	1399	12.72%
MW1	A zero gravity experiment related to combustion	C++	37	403	7.69%
KC3	Storage management for ground data	Java	39	458	9%

TABLE 4: CLASSIFIER INTRODUCTION

Classifier Name	Type	Description
Bayesian Network	Statistical	Consists of a multivariate linear regression model and heuristics to shrink the number of features
Nave Bayes	Machine Learning	One of the simplest classifier based on conditional probability
Logistic Regression	Statistical	Probabilistic statistical regression model, fits data to a logistic curve
Decision Tree	Machine Learning	Decision tree breaks down the dataset into several subsets based on information gain and builds the classification or regression model in a tree structure
Random Forest	Machine Learning	Random forest gives classification decision based on the votes from its individual tress, while each tree is grown based on a set of rules

to compare their prediction result as companies keep their software dataset private. As a consequence, we had to use the publicly available benchmark datasets for our experiments alike other researchers. A brief overview of the datasets is depicted in Table 3. Bayesian network and Nave bayes are widely used classifier and proved to be effective for software defect prediction [20] [21]. In logistic regression binary logistic model is used for the prediction purpose based on one or more predictor attributes which is a good rationale for choosing logistic regression for our defect prediction model [22]. On the other hand decision tree has been studied as a good classifier due to its entropy and information gain which is important for our feature selection technique [23]. A brief introduction of the classifiers is given in Table 4. We have represented our results using balance, f-measure, precision, recall and AUC (Area under ROC Curve) measurement scale. Table 5 gives an introduction to the measurement scales.

During the experimental setup, we prepared pairwise attribute list using the attributes of a dataset. So, if there are n numbers of attributes in a dataset then we have $\frac{n(n-1)}{2}$ number of attribute pairs if we use the classical rule for finding the number of combinations. We then calculated the balance using those attribute pairs by a 10-fold cross validation method. After that we had the pairwise attribute list and their corresponding balances. The list is then sorted in decreasing order of balance. As a result of the combination of the attributes, an attribute exists in several pairs and thus the list is again sorted based on the frequency of the attributes. We named the list as candidate attribute list and the set of final selected attributes was constructed from this list. From the candidate attribute list we picked the first attribute and add it to another list named selected attribute list and calculate balance using attributes from this list. Thus we kept adding attributes from candidate attribute list to selected attribute list unless adding an attribute would not decrease the balance obtained using the already developed selected attribute list. This process continues until the end of the candidate attribute list. Finally, the selected attribute list is used as the final

TABLE 5: PERFORMANCE MEASUREMENT SCALES

Measurement Scale	Description
Balance	$balance = 1 - \sqrt{\frac{(1-pd)^2+(0-pf)^2}{2}}$
F-Measure	$\frac{2 \times precision \times recall}{precision + recall}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
AUC	Area under the ROC (Receiver Operating Characteristic) curve

TABLE 6: Result with balance and AUC filtering

Classifier	Experimental Phase	Measurement Scale				
		Balance	Precision	Recall	F-measure	AUC
Bayesian Network	Before	0.648	0.33	0.621	0.4	0.746
	Balance Filtering	0.672	0.419	0.607	0.442	0.764
	AUC Filtering	0.66	0.445	0.582	0.45	0.803
Naive Bayes	Before	0.547	0.341	0.51	0.341	0.737
	Balance Filtering	0.676	0.432	0.616	0.456	0.756
	AUC Filtering	0.557	0.428	0.408	0.36	0.84
Logistic Regression	Before	0.495	0.48	0.297	0.353	0.705
	Balance Filtering	0.502	0.679	0.297	0.425	0.778
	AUC Filtering	0.413	0.561	0.171	0.312	0.83
Decision Tree	Before	0.527	0.402	0.347	0.371	0.629
	Balance Filtering	0.531	0.569	0.342	0.426	0.702
	AUC Filtering	0.489	0.713	0.282	0.368	0.761
Random Forrest	Before	0.47	0.366	0.26	0.334	0.71
	Balance Filtering	0.619	0.57	0.471	0.499	0.776
	AUC Filtering	0.501	0.48	0.303	0.321	0.806

list of attributes to calculate the balance, which is considered as the measurement value of the defect predictor. The same process is repeated for another test using AUC measurement scale instead of balance. In Cross project defect prediction we perform the attribute selection process for the train dataset and with the selected attributes we prepare the test and train dataset. The training and test dataset is then used for the defect prediction.

V. Experimental Result and Discussion

As mentioned in the experimental design section, we used five datasets from NASA MDP Repository for within project defect prediction and three other datasets from Relink for cross-project defect prediction. We presented the experimental result by averaging the five results of the within project defect dataset based on five classifiers. For within project prediction, we have compared the results between before introducing the attribute selection process and after using the selected attributes. We measured the result of every measurement scale by two type of filtering where the filtering responsible for sorting the attribute pair. Table 6 represents the results for both the balance and AUC filtering. The result gives us a

TABLE 7: RESULT FOR CROSS PROJECT DEFECT PREDICITON

Source => Target	Nam et. al [14]	Measurement Scale				
		f-score	precision	recall	balance	AUC
Safe => Apache	0.64	0.711	0.711	0.711	0.711	0.755
ZXing=>Apache	0.72	0.609	0.719	0.644	0.534	0.638
Apache=>Safe	0.72	0.738	0.832	0.768	0.582	0.739
ZXing=>Safe	0.64	0.699	0.738	0.696	0.701	0.727
Apache=>ZXing	0.49	0.599	0.612	0.697	0.323	0.498
Safe=>ZXing	0.43	0.626	0.628	0.689	0.381	0.641
Average	**0.61**	**0.664**	**0.707**	**0.701**	**0.539**	**0.666**

very good improvement in the prediction performance after using our attribute selection process.

In case of cross project defect prediction we measured the result by calculating the f-measure which is used in [14] to compare the result with TCA (Transfer Component Analysis). The three dataset found in ReLink [21] are Apache, Zxing and Safe and every dataset contains the same number and type of attributes. So, we take the experiment for Safe=>Apache, Zxing=>Apache, Apache=>Safe, Zxing=>Safe, Apache=>Zxing and Safe=>Zxing where the first part of => is for training and the second part is for testing the defect predictor. The result is compared in Table 7 with some other performance measurement scale such as balance, precision, recall and AUC. As shown in the table result improves in four cases comparing with TCA+. For example f-measure for Zxing=>Safe in our approach (0.70) is better than TCA+ (0.64). We have experimented the result with Bayesian Network classifier where the TCA+ authors used logistic regression for their experiment. We observed that Bayesian Network gives a better result in our attribute selection approach.

VI. Conclusion

Attribute selection can significantly improve the performance of a defect prediction model and it has been shown in several researches. Our proposed attribute selection algorithm has been developed concerning the joint influence of paired attributes on the performance of defect prediction model. In this context performance is measured using two classic metrics balance and AUC. Specifically, we show that our proposed method is suitable for cross project defect prediction, which is a serious and challenging issue in the defect prediction community. However, we believe that the accuracy of our method can be improved if we create a dynamic combination of attributes, i.e. forming combination of different number of attributes for our attribute selection process.

References

[1] T. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Information Systems Frontiers*, vol. 16, no. 5, pp. 801–822, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10796-013-9430-0

[2] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, "A comparative study of different strategies for predicting software quality." in *SEKE*. Knowledge Systems Institute Graduate School, 2011, pp. 65–70. [Online]. Available: http://dblp.uni-trier.de/db/conf/seke/seke2011.html#KhoshgoftaarGN11

[3] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[4] H. Wang, T. M. Khoshgoftaar, and N. Seliya, "How many software metrics should be selected for defect prediction?" in *FLAIRS Conference*, 2011.

[5] R. S. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic optimization based feature selection for software defect prediction," *Journal of Software*, vol. 9, no. 5, pp. 1324–1333, 2014.

[6] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.

[7] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579–606, 2011.

[8] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[9] T. M. Khoshgoftaar and K. Gao, "Feature selection with imbalanced data for software defect prediction," in *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*. IEEE, 2009, pp. 235–240.

[10] H. Wang, T. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, Dec 2010, pp. 135–140.

[11] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 91–100.

[12] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[13] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 409–418. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487161

[14] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 382–391.

[15] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the nasa metrics data program data sets for automated software defect prediction," in *Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*. IET, 2011, pp. 96–103.

[16] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *Software Engineering, IEEE Transactions on*, vol. 39, no. 9, pp. 1208–1215, 2013.

[17] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 311–321.

[18] P. He, B. Li, D. Zhang, and Y. Ma, "Simplification of training data for cross-project defect prediction," *arXiv preprint arXiv:1405.0773*, 2014.

[19] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2–13, 2007.

[20] A. Okutan and O. T. Yıldız, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[21] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 15–25.

[22] J. I. Khan, A. U. Gias, M. Siddik, M. Rahman, S. M. Khaled, M. Shoyaib *et al.*, "An attribute selection process for software defect prediction," in *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*. IEEE, 2014, pp. 1–4.

[23] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 61.

Md. Habibur Rahman received his MS degree in Software Engineering (SE) from Institute of Information Technology, University of Dhaka in 2015. He has also completed his bachelor in Software Engineering (SE) from the same institution in 2014. His research interest includes software testing, pattern recognition and machine learning. He has conducted his masters thesis on software defect prediction entitled as "Designing a Method to Select the Responsible Static Code Attributes for Software Defect Prediction".

Sadia Sharmin is currently studying her bachelor in Software Engineering (SE) in Institute of Information Technology, University of Dhaka. She has completed her 6 month internship program from LEADS Corporation Limited from January 1st to June 30th in 2015. Her research interest includes Software Engineering and Pattern Recognition. She is currently involved in research on software defect prediction under the supervision of Dr. Mohammad Shoyaib, an Associate Professor of the same institution.

Sheikh Muhammad Sarwar is currently working as a Lecturer at Institute of Information Technology, University of Dhaka. He completed his B.Sc. and M.Sc. from Department of Computer Science and Engineering, University of Dhaka. His research interest include Information Retrieval, Recommender Systems and Machine Learning.

Shah Mostafa Khaled is currently working as an Assistant Professor at Institute of Information Technology, University of Dhaka. Khaled completed his B.Sc. and M.Sc. from Department of Computer Science and Engineering, University of Dhaka. He completed his second masters in Computer Science from University of Lethbridge, Canada with a thesis titling "Heuristic Algorithms for Wireless Mesh Network Planning" under the supervision of Robert Benkoczi. Theoretical Optimization, Operations Research and Machine Learning are his areas of interest. Khaled is also the coordinator of IIT DU Optimization Research Group.

Mohammad Shoyaib received his MS degree in computer science from the University of Dhaka, Bangladesh, in 2000 and in 2012 he has completed his PhD degree from the department of the computer Engineering, Kyung Hee University, Rep. of Korea. Currently he is a faculty member in the University of Dhaka, Bangladesh. His research interests include pattern recognition and machine learning in different areas of computer vision and image processing. He has also interest in software engineering and bioinformatics.