An Attribute Selection Process for Cross-Project Software Defect Prediction

Md. Habibur Rahman[1], Sadia Sharmin[2], Md. Shariful Islam[1], Shah Mostafa Khaled[1], Sheikh Muhammad Sarwar[1],

[1]Institute of Information Technology, University of Dhaka
[2]Dept. of Computer Sci. and Eng., Islamic University of Technology
bit0216@iit.du.ac.bd, sharmin@iut-dhaka.edu, shariful@iit.du.ac.bd, khaled@du.ac.bd, smsarwar@du.ac.bd

Abstract: Software defect prediction is a key research area in the domain of software quality estimation. Usually, software attributes are used for building a defect prediction model and a specific prediction model can produce positive, negative, or neutral outcomes depending on the characteristics of these attributes. Therefore, choosing an optimal set of attributes for the development of a defect prediction model remains a vital yet relatively unexplored issue. To address this issue, we propose a technique for attribute selection to improve the accuracy of software defect prediction for both within project and cross-project. Experimental results using the data sets from Relink and NASA MDP repository demonstrate the superiority of the proposed algorithm.

***Index Terms*—Software testing, Cross project defect prediction, Software quality.**

1. Introduction

Software quality depends on the identification of the number of defects in software. Proper identification of software defects may help in improving the quality of the software. If the defective part of the software code can be identified, then the software test engineer can take important steps to inspect only that portion of the code. It will also be helpful for a project manager for resource allocation. A software defect prediction model works to identify the error-prone parts of the software. If a defect prediction model works as intended, then software testing time may be reduced, as well as development cost may be decreased. Thus, it will help a company building better quality software with their limited resources.

Software modules contain error-prone code which results in incorrect output for a specific feature. These error-prone codes are good candidates to build a set of software attributes that can help to identify similar errors in other code [1]. Some of the examples of these attributes are cyclomatic complexity, lines of code, conditions count, etc. They represent the characteristics of the software and are responsible for a software module to be defected or non-defected. Thus, defect prediction using software attributes has been a common practice and a reasonable number of defect prediction models have been developed following different machine learning algorithms based on these attributes. These attributes have an impact on the effectiveness and

performance of the defect prediction model. However, all the attributes do not show the same level of importance to describe projects' characteristics. Among them, there are few which show redundant knowledge and some others do not describe the project at all. In recent studies [2], [3], it is also demonstrated that removal of irrelevant and redundant features before constructing the model enhances the performance of defect prediction models. The irrelevant and redundant attributes may be called noisy attributes and the defect prediction model may produce questionable performance due to these noisy attributes [4], [5]. However, it is not an easy task to identify the presence of those noisy attributes using machine learning models [6].

In defect prediction, researchers used several algorithms such as Genaro *et. al.* [7] used logistic regression, Khoshgoftaar *et. al.* [8] used decision tree, Park *et. al.* [9] worked with neural networks, and Menzies *et. al.* [10] introduced Naïve Bayes (NB) for their model. If a proper set of attributes are employed for training, their results will be improved [11]. Thus, attribute selection has been regarded as an important research topic in the software defect prediction domain.

Test data and train data are a must for any kind of machine learning-based prediction technique. When a defect prediction model considers test and train data from the same project, it is called within project defect prediction. To build a within project defect prediction model, the data repository should be large enough to train the model. But some companies do not track defect prediction data and for constructing a new project the necessary historical data may not be available, which makes the prediction model infeasible. So, cross-project defect prediction is necessary if training data are scarce and, in that case, the training data comes from different projects. However, dissimilarity among the distributions of the datasets is a core problem. So, the distributions of the test and train dataset should be made same for effective defect prediction. In cross-project defect prediction, there are several works conducted in recent times. Zimmermann *et al.* [34] mentioned cross-project defect prediction as a serious challenge and concluded that using the projects of the same domain (i.e., web browser) and different companies (i.e. Mozilla/Google) has a very poor prediction performance. Burak *et. al* [15] investigated the relative performance of cross-project and within project defect predictors and concluded that cross-project defect predictors cannot supersede within project defect predictors. They also demonstrated the minimum requirement of data samples for an effective defect predictor. Nam *et al.* introduced transfer learning and proposed TCA+, an extension of TCA (Transfer Component Analysis) to improve the results in cross-project defect prediction [16].

In this paper, we introduce a new approach for selecting the best set of attributes in software defect prediction. Our proposed method conducts two stages of selection. At the first stage, we rank each feature considering their pairwise dependency, and the later stage finds the best set of attributes following a forward search mechanism from the ranking obtained from the first stage. Those selected attributes are then used to construct the defect prediction model. With a selection of appropriate attributes, we achieve better accuracy on benchmark datasets with an appropriate classifier.

For cross-project defect prediction, training and test data comes from different project but should contain the same attributes for both projects. As a result, the set of attributes selected for training the model from one project is further used in testing the model with another project. We observed a reasonable improvement in the performance of cross-project defect prediction model using the selected attributes.

We discuss the related work in the next section. Our proposed methodology is discussed in section 3 and section 4 describes the experimental result. We conclude our paper by suggesting some future work with section 5.

2. Related Literature

There are several attribute selection methods proposed by researchers [17], [18], [19], [20], [21], [22], [23], [24] for general applications. Jong [18] used support vector machines (SVM) to propose feature selection method. Ilczuk *et al.* [19] worked on why the attribute selection is important. Forman [20] showed how multiple filter-based feature ranking works for attribute selection. Software cost and effort estimation was studied in respect of feature selection in [21].

The attribute selection process has been utilized in different kinds of applications for a long time but in defect prediction research it is relatively unexplored. Song *et al.* [25] constructed an attribute selection process by employing the forward selection and backward elimination technique. They found that the effect of applying feature selection technique varies in respect of the learning algorithm and data set. As a continuation of research, Wang *et al.* [26] introduced ensemble feature selection techniques and applied to a total of sixteen datasets. They found that ensembles of only a few numbers of rankers perform well which is better than ensembles of all the rankers. Khoshgoftaar *et al.* worked on four scenarios based on original and sample data to compare the prediction performance in attribute selection and data sampling [12]. Gao *et al.* proposed a hybrid model for attribute selection where feature ranking techniques are adopted for reducing the search space [11]. Romi and Nanna combined genetic algorithm and bagging technique for defect prediction [13]. One was used for attribute selection while the other technique was used for data sampling. Moreover, they have also shown the application of particle swarm optimization to select important features for defect prediction. In [14], the authors presented two feature ranking strategies such as threshold-based techniques and signal-to-noise filter technique for attribute selection.

Traditionally, some researchers used McCabe attributes [57] to build defect prediction models, some others used Halstead complexity measures [58], and lines of code count. However, Menzies *et. al.* [10] mentioned that the learning process for a software defect prediction model is important than the subset of learning dataset.

Usually, quality and size of the training data plays an important role for building any prediction model. Large sample of training data is required for producing an accurate prediction model. However, in software defect prediction, we cannot get

enough training data for a project especially for a new project where historical dataset is not sufficient, or in some cases are not available. In these circumstances, the cross-project defect prediction model is investigated and applied in recent times by most of the researchers in defect prediction domain. Zimmerman *et al.* studied the cross-project defect prediction as a large-scale experiment on data vs. domain vs. process [34]. In that work, the authors expressed the need to understand and evaluate the process, code, and domain before building a prediction model. They also described the necessary factors to identify the project to build the defect predictor, but any specific way of improving the defect prediction performance has not been discussed. Turhan *et al.* studied the relative values of cross-company defect prediction with the company by answering three questions about the usefulness of cross-company defect prediction, local tuning of cross-company data, and generalization of cross-company results [15]. Their approach increases the probability of detection and the probability of false alarm in terms of median. But the increment of false alarm rate decreases the performance of the defect predictor. They tried to reduce the false alarm rate using nearest neighbour filtering. In cross-project defect prediction, the data sampling and normalization is an important task to improve the prediction performance. In [16], the authors presented TCA+, an extension of TCA (Transfer Component Analysis) by using source and target projects from Relink [35] and AEEEM [36]. They showed an improved performance relative with their previous work TCA [37] using logistic regression. They normalized the train and test dataset based on max-min and z-normalization and then defining Dataset Characteristic Vector (DCV) for each project they applied four decision rules to determine the best dataset. Such other work has been conducted by some other researchers [38], [39]. Considering the limited research on cross-project defect prediction we have introduced our attribute selection method to improve the prediction accuracy in this regard.

3. Proposed Method

In this section, we describe our proposed attribute selection process for software defect prediction. The selected attributes will be used to predict the defects both for cross-project and within project. For within project defect prediction, a certain portion of the data will be used for training the model and the remaining data will be used for testing. On the other hand, there will be two different data sets taken from two different projects that are used in cross-project defect prediction, one for training and the other for testing.

The overview of the proposed process to select the best set of attributes for software defect prediction (given in algorithm 1) contains the following sequential steps:

1) Generate and sort the pairwise combination of attributes
2) Select the candidate attributes set
3) Select the final attributes set

The defect prediction model begins by selecting a list of pair wise combination of attributes from the dataset. Here, we consider pairwise combination instead of considering all possible combinations which is NP hard [23]. These pairs of attributes are then used to generate the accuracy of the defect prediction and corresponding accuracy will be stored for further use. The pair wise attribute list is then sorted based on the descending order of the accuracy. Then the candidate attributes list is selected from the sorted list which is used to find the final set of attributes for defect prediction.

Algorithm 1 presents the overall process of the proposed algorithm for defect prediction.

Algorithm 1 *Attribute Selection Model*

Input: Original Set of attributes A = $\{ a_1, a_2 \dots \dots, a_n \}$
Output: Prediction results and best set of attributes $F = \{ a_1, a_2 \dots \dots \dots \dots, a_j \}$
1: Begin
2: Generate Pair wise combinations of attributes (P) from data set
3: Compute the accuracy of defect prediction for each pair
4: Sort the set of pairwise combinations based on accuracy metric
5: Determine the candidate attributes set (A^+) based on their occurrences in P
6: Find the final set of attributes F for from A^+
7: End

1. Generating and selecting pairwise combination of attributes

As all the attributes of software defect dataset are not equally important, we need to select a best set of attributes. To accomplish that, first we generate a list of all possible pair of combination from the given attributes. Let, we have a set of attributes A = $\{ a_1, a_{2,} \dots \dots \dots \dots, a_n \}$. Now, according to the algorithm 2 we generate a pairwise combination of those attributes. For example, the list will contain pair of attributes like $a_1 a_2, a_1 a_3, \dots \dots \ a_{n-1} a_n$. These pairs will then be used individually to generate corresponding defect prediction accuracy (balance, a metric describe in the Experimental Results and Analysis) list. To do this, the dataset will be reconstructed by only the pair of attributes and will be used to classify the defect data using a classifier. This will return a list of accuracy for those pair of attributes. The next task will be to sort the pair wise attribute list based on the balance in a decreasing order. The final list will not contain any balance less than δ, a threshold to filter the balance list.

Our algorithm for the selection of pairwise combination is presented in Algorithm 2

Algorithm 2 *Generating and Selecting the Combinations of Pairwise Attributes*

Input: Original set of attributes $A = \{a_1, a_2, \dots, a_n\}$
Set of classes $C = \{defected, not\ defected\}$
Dataset D: $A \times C$ and Classifier γ
δ = a threshold value for determining potential attribute pairs
Output: **S**orted list of paired attributes list P

1: **Begin**
2: $P_{temp} \leftarrow \{(u, v)$: $u \epsilon A, v \epsilon A$ and $u \neq v\}$
3: $B \leftarrow \emptyset$
4: **for** $each (u, v) \epsilon P_{temp}$
5: $d_i \leftarrow$ *all accumulated values from dataset D for attribute pair* (u, v)
6: *Classify* d_i *using* γ
7: *Evaluate balance*
8: $B_i \leftarrow \{balance\}$
9: $B \leftarrow B \; U \; B_i$
10: **end for**
11: Sort P_{temp} in descending order based on balance using B
12: Return P$\subset P_{temp}$, where$|P|= k$ and $\forall x \in P, balance(x) \geq \delta$
13: **End**

2. Selecting the set of candidate attributes

In this phase, we rank all the attributes based on the frequency of occurrences in the selected pairwise combinations. The attributes which appear more have higher importance. To generate the candidate attribute list, we compute the number of occurrences of each attribute from the pairwise sorted list which we got from Algorithm 2. Then the attributes are sorted in a descending order based on the total occurrence of an attribute. At the top of the candidate attribute list, we get those attributes which are most responsible for better defect prediction accuracy. The candidate attribute selection process is represented in Algorithm 3, which returns the candidate attributes set sorted based on their frequency.

Algorithm 3 Selecting the set of Candidate Attributes

Input: Sorted paired wise combination of attributes P: $ATTRIBUTE \times ATTRIBUTE \times Balance\_VALUE$
Output: Set of decreasing order sorted attributes A^+ with frequency for each attribute
1: Begin
2: $A^+ \leftarrow$ *the list of unique attributes appeard in P*
3: for all $a_i \in A^+$ **do**
4: *Count the frequency* c_i *of each* a_i *appeard in P*
5: $C \leftarrow C \; U \; c_i$
6: **end for**
7: *sort* A^+ *in descendng order based on their frequency in C*
8: End

3. Selecting the final subset of attributes

In the candidate attribute list, we have a ranked list of attributes according to their individual performance. Now to determine a subset of attributes that combinedly perform better for the defect prediction task we proposed to use algorithm 4. Algorithm 4

represents the process of choosing the best attributes for final attribute set where all pairwise combination is evaluated and finally resulted in the subset which achieves the highest *balance*. To select the best set of attributes we get the candidate attribute list A^+ from algorithm 3. Let name the best set of attributes as *F*. Primarily the top ranked attribute is added into *F* and calculate balance using *F* which is stored for further checking. Then the second ranked attribute from the candidate attribute list is added into *F* and again calculate the balance using *F*. Now the current balance is compared with the previously stored balance. If the current balance is better than the previous one, the previous balance is replaced by the current one. If the current balance is lower than the previous one, the last added attribute is discarded from the best set of attribute F. Then the next top ranked attribute is added into *F* and do the same until the last attribute of the candidate attribute list. Thus, we follow a greedy forward search algorithm to find the final subset of features.

Algorithm 4 Selecting the final set of attributes

Input**:** Set of descending order sorted attributes A^+
Set of classes $C = \{defected, not\ defected\}$
Dataset D: $A \times C$ and Classifier γ
Output**:** Final set of attributes *F* for defect prediction with their corresponding balance
1: **Begin**
2: *balance* ← 0
3: **for** *all* $i = 1, \dots, |A^+|$ **do**
5: $F \leftarrow F\ U\ \{a_i\}$
6: $tempBalance \leftarrow balance\ calculated\ using\ F\ and\ \gamma$
7: $\boldsymbol{if}\ (tempBalance > baalnce)\ \boldsymbol{then}$
8: $balance \leftarrow tempBalance$
9: end if
10: **else** $\boldsymbol{F \leftarrow F \setminus \{a_i\}}$
11: **end for**
12: Return the set F, with their corresponding balance
13: ***End***

4. Cross project defect prediction

From algorithm 4, we can identify the best set of attributes for which the accuracy of the within project defect prediction improves significantly. Now, we want to use those best set of attributes for our cross-project defect prediction. In cross project defect prediction, we need a dataset (D_{train}) for training the defect prediction model and another dataset (D_{test}) for testing the model. To accomplish this, the defect prediction model will be trained with the D_{train} and D_{test} will be used to evaluate the performance. Note that, only the selected attributes (obtained from the D_{train}) that match with the test set (D_{test}) is used for performance evaluation.

The algorithm for cross project defect prediction using the best set of attributes is as follow:

Algorithm 5 Defect Prediction for Cross Project

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

4. Experimental Results and Analysis

In this paper, an attribute selection technique is proposed for better prediction of software defects. Most often the researchers of defect prediction domain face problems to find the appropriate data sets for their experiments as several companies use private data sets to build their defect prediction model. So, we cannot compare our result with their prediction model to validate the accuracy. In these circumstances, we had to use the public data sets which help us to verify and validate the prediction model. NASA MDP repository shared the public state-of-the-art data sets for building and testing different prediction models. As a consequence, we have collected the dataset from the publicly available NASA MDP repository to validate the attribute selection process for defect prediction, which was also used by many noted researchers like [10], [40], [41], [42]. We employed ReLink [35] dataset used by [16] for the validation of cross-project defect prediction model.

TABLE 1

NASA DATASET DESCRIPTION

Data Set	Type of Software	# Instances	#Attributes	Defected (%)
CM1	NASA Space Craft Instrument	498	22	9.83%
JM1	Real-time predictive ground system	10885/7782	22	80.65%
PC1	Flight software for earth orbiting satellite	1109	22	93.05%
PC2		1585	37	1.01%
PC3		1125	38	12.44%
PC4		1399	38	12.72%
PC5		17001	39	2.96%
MW1	A zero-gravity experiment related to combustion	403	37	7.69%
KC1	Storage Management	2109	22	15.45%
KC2		522	22	20.49%
KC3		458	39	9%
KC4		125	39	49%
MC1		9466	39	0.7%
MC2	Video guidance system	161	39	32%

TABLE 2

OTHER DEFECT PREDICTION DATA SET DESCRIPTION

Data Set	Type of Software	#Attributes	#Instances	Defected (%)
Eclipse 3.0 (Package)	Eclipse Foundation	198	661	62.78%
Eclipse 2.0		198	6729	38.80%
Jedit	jEdit Project	24	492	2.23%
Ant	Apache Soft. Foundation	24	745	22.15%
Tomcat		22	885	8.98%
Poi		22	492	1.28%
Apache		27	194	50.51%
Velocity		24	229	51.66%
Synapse		24	256	33.59%
Lucene		20	340	59.71%
Xalan		20	886	46.44%
Ivy		24	352	11.36
ar1	Turkish White-goods manufacturer	30	121	7.43%
ar3		30	63	12.70%
ar4		30	107	18.70%
ar5		30	36	22.23%
ar6		30	101	14.85%
JDT.Core	Eclipse	198	939	53.46%
SWT		198	843	24.67%
ZXing	Android Project	27	399	29.57%
Safe		27	56	39.29%

Most frequently used NASA dataset overview has been given in the Table 1. Table 2 gives us an overview of datasets other than the NASA MDP repository. For experimental evaluation, we have used five public data sets obtained from NASA MDP Repository. The number of samples for each data set varies from 200 to 1585. Two different types of experiments are performed for generating the results. In the first experiment, 3 datasets out of 7 are utilized for finding the best attributes and a simple classifier (e.g., Naïve Bayes classifier). Then, the remaining 4 data sets are employed to test the performance based on these selected attributes and the classifier. Repeating this process for ten times, the average output is collected and presented in table 8. Along with the averages, we also incorporate the respective standard deviations [22]. For the second experiment, 90 percent of data are chosen randomly from each data set to identify the best set of attributes with a classifier and the rest 10 percent data are used testing. This process is also repeated ten times and the average results along with their standard deviations are displayed in table 9. In our experiment, the results are generated for different datasets using two classifiers namely Naïve Bayes and Bayesian Network (BN). To show the superiority of the proposed attribute selection process we have chosen these two classifiers, because most of the existing state-of the-arts methods commonly used these classifiers for result comparison. Several metrics namely balance, AUC, f-measure, precision, recall is employed for experimental results.

Table 3 presents the confusion matrix of a problem where TP, FN, FP and TN denote True Positive, False Negative, False Positive and True Negative respectively. To evaluate the performance of defect prediction model, the authors in [10] used two

well-known metrics namely probability of detection (*pd*) or recall and probability of false alarm *(pf).* Formal definition for (*pd*) and *(pf)* are given in Equation (1) and (2).

. **TABLE 3**

CONFUSION MATRIX

	Real Defect	
Predicted Defect	Yes	No
Yes	TP	FP
No	FN	TN

$$pd\ (Recall) = \frac{TP}{TP + FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$pf = \frac{FP}{FP+TN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Several metrics namely balance, f-score, precision, recall is employed for experimental evaluation. These metrics are calculated using the following the equations. (Equation 3, 4 and 5)

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

$$Precision = \frac{TP}{TP+FP} \ldots\ldots (4)$$

$$f{-}score = \frac{2 \times Precision \times recall}{Precision + recall} \ldots\ldots (5)$$

Another evaluation is the AUC (for "Area under the ROC Curve.") that measures the entire two-dimensional area underneath the entire ROC curve. Receiver Operating Characteristic (ROC) curve provides a graphical visualization of the classification results [59].

TABLE 4

CLASSIFICATION RESULT USING COMMON ATTRIBUTES

Dataset	NB (%)			BN (%)		
	Probability of Detection	Probability of False Alarm	balance	Probability of Detection	Probability of False Alarm	balance
CM1	32	8	51.58±1.12	39	20	54.60±1.02
PC3	47	15	61.05±1.51	61	24	67.61±1.32
PC4	35	5	53.90±1.72	73	20	76.24±1.25
KC3	30	10	50±2.01	32	8	51.58±1.36
MW1	62	13	71.60±1.22	50	10	63.94±2.12
Avg:	42	9.42	58.31	53.14	15	64.72

TABLE 5

CLASSIFICATION RESULT USING DATA SPECIFIC ATTRIBUTES

Dataset	NB (%)			BN (%)		
	Probability of Detection	Probability of False Alarm	balance	Probability of Detection	Probability of False Alarm	balance
CM1	37	9	55.00±1.12	72	33	69.39±1.02
PC3	76	33	71.14±1.32	70	27	71.46±1.32
PC4	80	30	74.50±1.53	86	20	82.73±1.12
KC3	46	15	60.37±1.22	44	6	60.17±1.35
MW1	63	14	72.02±2.01	61	14	70.69±2.02
Avg:	57.28	16.71	65.81	68.85	20	72.45

It is observed from Table 4 that when we select common attributes for the combination of all different data sets the performances are not satisfactory. On the other hand, dataset (software) specific attribute selection provides better results (Table 9) even with a simple classifier. To demonstrate the effectiveness our proposed method, we have provided a comparison of our results with the existing state-of-the-art methods on the same data sets. For this comparison, we take *balance* values (using BN) from Table 5. Apart from *balance*, other two metrics *AUC and F-score* results are also provided in Table 6. It is observed from Table 6 that in most of the cases, our proposed algorithm performs better than other methods. This is because, our algorithm can choose the important attributes (e.g., *McCabe Cyclomatic Complexity*, *Design Complexity)* that provide significant information about the defect modules in a software. However, in some datasets, balance is slightly lower than the compared methods as they have generated their results with improved classifiers. However, we focus on showing the performance of attribute selection process with a simple classifier. The use of advanced classifiers might improve the overall performance of the model.

TABLE 6

WITHIN PROJECT RESULT COMPARISON AMONG DIFFERENT METHODS

	Balance			
Dataset	Jing *et al.* [33]	Yao *et al.* [43]	Song *et al.* [44]	Proposed Method
CM1	0.68	0.66	0.70	**0.71**
JM1	0.67	**0.68**	0.59	0.63
PC1	0.77	0.69	0.67	**0.77**
PC2	-	-	0.80	**0.87**
PC3	0.74	0.75	0.71	**0.76**
PC4	0.79	**0.85**	0.82	0.81
PC5	0.88	-	0.90	**0.92**
KC1	0.71	0.72	0.71	**0.71**
KC2	-	0.75	-	**0.78**
KC3	0.68	0.69	0.71	**0.82**
KC4	-	-	0.69	**0.78**
MC1	-	-	0.79	**0.82**
MC2	**0.76**	0.62	0.61	0.71

MW1	**0.80**	0.64	0.66	0.75
AR1	-	-	0.41	
AR3	-	-	0.66	
AR4	-	-	0.68	**0.73**
AR6	-	-	0.49	

	AUC			
Dataset	Issam *et al.* [45]	Yao *et al.* [43]	Ahmet *et al.* [46]	Proposed Method (fold)
CM1	-	0.79	-	**0.81** (20)
KC1	-	0.80	-	**0.83** (20)
KC2		0.82		**0.87** (20)
KC3	**0.86**	0.83	-	0.76
PC1	-	-	-	**0.90** (20)
JM1	-	0.75	-	**0.75** (20)
MC1	**0.98**	-	-	0.96 (20)
MC2	-	0.75	-	**0.79** (20)
MW1	-	0.78	-	**0.80** (20)
PC1		0.87		**0.88** (20)
PC2	**0.95**	-	-	0.94
PC3	-	**0.85**	-	0.83
PC4	0.96	0.94	-	**0.98**
Ant-1.7	**0.86**	-	0.82	0.83
Camel-1.6	**0.80**	-	-	0.72
Synapse	-	-	0.66	**0.79**
Lucene	-	-	0.63	**0.72**
Xalan	-	-	0.62	**0.82**
Ivy	-	-	**0.85**	0.83
Tomcat	-	-	0.77	**0.84**
Poi	-	-	0.85	**0.89**
Jedit	-	-	0.66	0.63
Velocity	-	-	0.68	**0.77**

F-score		
Dataset	Ren *et. al.* [47]	Our Method
ar3	0.569	**0.600**
Ar4	0.474	**0.501**
Ar5	0.625	**0.653**
cm1	0.254	**0.347**
kc1	0.462	0.441
kc2	0.534	**0.593**
kc3	0.412	0.400
mw1	0.371	**0.461**
pc1	0.398	**0.414**

Now the selected attributes are applied for our cross-project defect prediction. In cross-project defect prediction, train and test data set are used from different projects. So, we have taken only those selected attributes and their instances to generate the new dataset. This process is done for both the training and the test data set. It should be kept in mind that the selection process will be run on the training data set. We have evaluated our cross-project defect prediction using the data sets found in ReLink [35] and used for transfer learning [16].

In case of cross project defect prediction, we measured the result by calculating the f-measure which is used in [16] to

compare the result with TCA (Transfer Component Analysis). The three data set found in ReLink [35] are Apache, Zxing and Safe and every data set contains the same number and type of attributes. So, we take the experiment for Safe→Apache, Apache→Safe, Zxing→Apache, Apache→Zxing, Zxing→Safe, and Safe→Zxing where the first portion of → is for training and the second portion 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 BN classifier where the TCA+ authors used logistic regression for their experiment. We observed that BN gives a better result in our attribute selection approach.

TABLE 7

CROSS PROJECT RESULT COMPARISON AMONG DIFFERENT METHODS

Source →Target	Nam *et al.* [16] **f-score**	**Our Method**				
		f-score	**precision**	**recall**	**balance**	**AUC**
Safe → Apache	0.64	0.71	0.71	0.71	0.71	0.75
Apache→Safe	0.72	0.74	0.83	0.77	0.58	0.74
Zinxg→Apache	0.72	0.61	0.72	0.64	0.53	0.64
Apache→Zxing	0.49	0.60	0.61	0.70	0.32	0.50
Zxing→Safe	0.64	0.70	0.74	0.70	0.70	0.73
Safe→Zxing	0.43	0.63	0.63	0.69	0.38	0.64
Avg.	**0.61**	**0.66**	**0.71**	**0.70**	**0.54**	**0.67**

The bold-faced results are better than the compared defect prediction technique and underlined results are for the average accuracy representation. The attribute selection process has been performed by our implementation and after preparing the data set for the testing is performed using Weka [48]. Based on the result from Table 7 we can say that the attribute selection process improves the cross-project defect prediction result. Our proposed algorithm performs better as it selects the important attributes both for within project and cross-project software defect prediction. This is because, we first provide a ranking of the attributes based on their pairwise performance and then select the best set from this ranking.

5. CONCLUSION

Transferring defect knowledge from one project to another is a very complex task for software defect prediction. But if a proper approach can be used then this complex task can be improved for a better defect prediction accuracy. In this paper, we presented an attribute selection process for predicting the software defects in both within project and cross-project domain. Our result indicated that in both domain our approach may produce significantly better result for a specific classifier. However, incorporation of an improved classifier along with our attribute selection method may enhance the overall performance. We will address this issue in our future work. Moreover, there are other meta-heuristic approaches that can also be adopted here for

selecting best set of attributes. we will also address this issue in future.

References

[1] Rawat, S. Mrinal and K. D. Sanjay, “Software defect prediction models for quality improvement: A literature study,” International Journal of Computer Science Issues (IJCSI), vol. 9, no. 5, 2012.

[2] K. Gao, T. Khoshgoftaar and H. Wang, “An empirical investigation of filter attribute selection techniques for software quality classification,” in IEEE International Conference in Information Resue & Integration, 2009.

[3] H. Wang, T. Khoshgoftaar and V. Hulse, “A Comparative Study on threshold-based feature selection techniques,” in IEEE International Conference on Granular Computing, 2010.

[4] T. Wang, L. Weihua, S. Hoabin and L. Zun, “Software Defect Prediction Based on Classification Ensemble,” Journal of Information & Computational Science, vol. 8, no. 16, pp. 4241-4254, 2011.

[5] S. Kim, H. Zhang, R. Wu and L. Gong, “Dealing with Noise in Defect Prediction,” in International conference on Software Engieering (ICSE), 2011.

[6] D. Gray, D. Bowes, N. Davey, Y. Sun and B. Christianson, “Reflections on the Nasa MDP Data Sets,” Software, IET, vol. 6, no. 9, pp. 549-558, 2012.

[7] G. Genaro, “Estimating software fault-proneness for tuning testing activities,” in 22nd International Conference on Software Engineering (ICSE), 2000.

[8] T. Khoshgoftaar and K. Gao, “Feature Selection with Imbalanced Data for Software Defect prediction,” in International Conference on machine Learning and Applications, 2009.

[9] B.-J. Park, S.-K. Oh and W. Pedrycz, “The Design of Polynomial Function-Based Neural Network Predictors for Detection of Software Defects,” Journal of Information Science, vol. 229, pp. 40-57, 2013.

[10] T. Menzies and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” IEEE Transaction on Software Engineering, vol. 33, no. 1, pp. 2-13, 2007.

[11] K. Gao, T. M. Khoshgoftaar, H. Wang and N. Seliya, “Choosing Software metrics for defect prediction: an investigation on feature selection techniques,” in International Conference on Tools with Artificial Intelligence (ICTAI) , Arras, 2011.

[12] T. Khoshgoftaar, K. Gao and N. Seliya, “Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction,” in Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on, Arras, 2010.

[13] R. S. Wahono, S. Nanna and S. Ahmad, “Metaheuristic Optimization based Feature Selection for Software Defect Prediction,” Journal of Software, vol. 9, no. 5, pp. 1324-1333, 2014.

[14] H. Wang, T. M. Khoshgoftaar and A. Napolitano, “A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction,” in ICMLA '10 Proceedings of the 2010 Ninth International Conference on Machine Learning and Applications, Washington, DC, USA, 2010.

[15] T. Burak, T. Menzies, A. B. Bener and J. D. Stefano, “On the relative value of cross-company and within-company data for defect prediction,” Empirical Software Engineering, vol. 14, no. 5, pp. 540-578, 2009.

[16] J. Nam, S. J. Pan and A. Kim, “Transfer Defect Learning,” in International Conference on Software Engineering (ICSE), Piscataway,NJ,USA, 2013.

[17] D. Rodr´ıguez, R. Ruiz, J. Cuadrado-Gallego and J. Aguilar-Ruiz, “Detecting fault modules applying feature selection to classifiers,” in IEEE International Conference on Information Reuse and Integration, 2007.

[18] K. Jong, E. Marchiori, M. Sebag and A. V. D. Vaart, “Feature selection in proteomic pattern data with support vector machines,” in IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 2004.

[19] G. Ilczuk, R. Mlynarski, W. Kargul and A. Wakulicz-Deja, “New Feature Selection Methods for Qualification of the Patients for Cardiac Pacemaker Implementation,” in IEEE International Conference on Computers in Cardiology, 2007.

[20] G. Forman, “An extensive empirical study of feature selection metrics for text classification,” The Journal of machine learning research, vol. 3, pp. 1289-1305, 2003.

[21] Z. Chen, T. Menzies, D. Port and B. Boehm, “Finding the right data for software cost modeling,” Journal of Software, vol. 22, no. 6, pp. 38-46, 2005.

[22] S. Sharmin, M. Shoyaib, A. Ahsan Ali, M. A. Hossain Khan and O. Chae, “Simultaneous feature selection and discretization based on mutual information,” Pattern Recognition, vol. 91, pp. 162-174, 2019.

[23] S. Sharmin, A. Ahsan Ali, H. Khan, M. Asif and M. Shoyaib, “Feature selection and discretization based on mutual information,” in IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR), 2017 .

[24] S. Sharmin, M. R. Arefin, M. A.-A. Wadud, N. Nower and M. Shoyaib, “SAL: An effective method for software defect prediction,” in 18th International Conference on Computer and Information Technology (ICCIT), 2015.

[25] Q. Song, Z. Jia, M. Shepperd, S. Ying and J. Liu, “A general Software defect-proneness prediction framework,” IEEE Transaction on

Software Engineering, vol. 37, no. 3, pp. 356-370, 2011.

[26] H. Wang, T. M. Khoshgoftaar and A. Napolitano, “Software mesuret data reduction using ensemble techniques,” Neurocomputing, vol. 92, pp. 124-132, 2012.

[27] L. Stefan, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framwork and Novel Findings,” IEEE Transaction on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.

[28] S. A. Romi and S. Nanna, “Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction,” International Journal of Software Engineering and Its Application, vol. 7, no. 5, pp. 153-166, 2013.

[29] H. Wang, T. M. Khoshgoftaar and N. Seliya, “How Many Software Metrics Should be Selected for Defect Prediction?,” in Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, 2011.

[30] A. Okutan and O. T. Yildiz, “Software defect prediction using Bayesian networks,” Empirical Software Engineering, vol. 19, no. 1, pp. 154-181, 2014.

[31] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang and A. Bener, “Defect prediction from static code features:,” Automated Software Engineering, vol. 17, no. 4, pp. 375-407, 2010.

[32] 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.

[33] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu and J. Liu, “Dictionary Learning Based Software Defect Prediction,” in International Conference on Software Engineering (ICSE), Hyderabad, India, 2014.

[34] T. N. G. M. Zimmerman, “Cross-Project Defect Prediction,” 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, New York, NY, USA, 2009.

[35] R. Wu, H. Zhang, S. Kim and S.-C. Cheung, “ReLink: Recovering Links between Bugs and Changes,” in Foundations of software engineering, New York, NY, USA, 2011.

[36] M. D’Ambros, M. Lanza and R. Robbes, “An extensive comparison of bug prediction approaches,” in IEEE Working conference on Mining Software Repositories (MSR), Cape Town, 2010.

[37] S. J. Pan, I. W. Tsang, J. T. Kwok and Q. Yang, “Domain Adaptation via Transfer Component Analysis,” IEEE Transactions on Neural Networks, vol. 22, no. 2, pp. 199-210, 2010.

[38] F. Rahman, D. Posnett and P. Devanbu, “Recalling the "imprecision" of cross-project defect prediction,” in 20th International Symposium on the Foundations of Software Engineering, New York, NY, USA, 2012.

[39] F. Peters, T. Menzis and A. Marcus, “Better Cross Company Defect Prediction,” in 10th IEEE Working Conference on Mining Software Repositories (MSR), San Francisco, CA, 2013.

[40] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed frame-work and novel findings,” IEEE Transaction on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.

[41] Y. Jiang, B. Cukic and T. Menzies, “Fault prediction using early lifecycle data,” in 18th IEEE Symposium on Software Reliability, 2007.

[42] H. Zhang, X. Zhang and G. Ming, “Predicting defective software components from code complexity measures,” in 13th Pacific Rim International Symposium on Dependable Computing, 2007.

[43] S. W. a. X. Yao, “Using Class Imbalance Learning for Software Defect Prediction,” IEEE Transaction on Reliability, vol. 62, no. 2, pp. 434-443, 2013.

[44] Q. Song, J. Zihan, M. Shepperd, S. Ying and J. Liu, “A General Software Defect-Proneness Prediction Framework,” IEEE Transaction on Software Engineering, vol. 37, no. 3, pp. 356-370, 2011.

[45] H. L. Issam, A. Mohammad and G. Lahouari, “Software defect prediction using ensemble learning on selected features,” Journal of Information and Software Technology, vol. 58, pp. 388-402, 2015.

[46] O. Ahmet and O. Y. Taner, “Software defect prediction using bayesian network,” Empirical Software Engineering, vol. 19, no. 1, pp. 154-181, 2014.

[47] Ren, Jinsheng, et al. "On software defect prediction using machine learning." Journal of Applied Mathematics 2014 (2014).

[48] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, “The WEKA Data Mining Software: An Update,” SIGKDD Explorations, vol. Volume 11, no. 1, 2009.

[49] IET Digital Library, “Author Guide - IET Research Journals,” [Online]. Available: http://digital-library.theiet.org/journals/author-guide. [Accessed 27 November 2014].

[50] N. S. S. A. RS Wahono, “Metahuristic Optimization based Feature Selection for Software Defect Prediction,” Journal of Software, pp. 1324-1333, 2014.

[51] S. Y. S.-W. Z. S.-S. W. J. L. Xiao-Yuan Jing, “Dictionary Learning Based Software Defect Prediction”.

[52] Nam, Jaechang, Sinno Jialin Pan, and Sunghun Kim. "Transfer defect learning." 2013 35th international conference on software engineering (ICSE). IEEE, 2013.

[53] G. A. D. Canfora, D. p. Massimiliano, R. Oliveto, A. Panichella and S. Panichella, “Multi-Objective Cross-Project Defect Prediction,” in IEEE Sixth International Coference on Software Testing, Verification and Validation, Luembourg, 2013.

[54] Z. He, F. Shu, Y. Yang, M. Li and Q. Wang, “An Investigation on the Feasibility of Cross-Project Defect Prediction,” Automated

Software Engieering, vol. 19, no. 2, pp. 167-199, 2012.

[55] A. Bener and B. Turhan, “Software defect prediction: Heuristics for weighted naive bayes,” in ICSOFT (SE), 2007.

[56] G. Czibula, Z. Marian and I. G. Czibula, “Software defect prediction using relational association rule mining,” Journal of Information Sciences.

[57] T. J. McCabe, "A complexity measure," IEEE Trans. Software Eng., vol. SE-2, pp. 308-320, Dec. 1976.

[58] M. Halsted, Elements of Software Science (Operating and Programming Systems Series). New York, NY, USA: Elsevier, 1977. [Online]. Available: https://dl.acm.org/doi/book/10.5555/540137

[59] T. Fawcett, “An introduction to roc analysis,” Pattern recognition letters, vol. 27, no. 8, pp. 861–874, 2006.