A Novel Approach for Converting N-Dimensional Dataset into Two Dimensions to Improve Accuracy in Software Defect Prediction

Rayhanul Islam[1*], Abdus Satter[2], Atish Kumar Dipongkor[3], Md. Saeed Siddik[4], Kazi Sakib[5]

[1] Institute of Leather Engineering and Technology, University of Dhaka, Dhaka, Bangladesh.
[2,4,5] Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh.
[3] Jashore University of Science and Technology, Jashore, Bangladesh.

* Corresponding author. Email: rayhanul.islam@du.ac.bd
Manuscript submitted February 2, 2020; accepted July 12, 2020.
doi: 10.17706/jsw.15.6.147-162

Abstract: Software defect prediction model is trained using code metrics and historical defect information to identify probable software defects. The accuracy and performance of a prediction model largely depend on the training dataset. In order to provide proper training dataset, it is required to make the dataset clustered with less variabilities using clustering algorithms. However, clustering process is hampered due to multiple attributes of dataset such as Coupling between Objects, Response for Class, Lines of Code, etc. This research will aim to predict software defects through reducing code metrics dimensions to two latent variables. It will finally help the clustering algorithms to group data properly for the defect prediction model. In this paper, the dataset similarities are analyzed by reducing code metrics' attributes into two latent variables based on their impacts to defects. Their impacts to defects can be analyzed using regression analysis because it identifies the relationship among a set of dependent and independent variables. Then, the code metrics are merged into two variables - *PosImpactValue* and *NegImpactValue* based on their positive or negative impact, respectively. As a result, multi-dimensional dataset is mapped into two-dimensional dataset. Plotting those dimensions reduced datasets enable distance-based clustering algorithms to group those datasets based on their similarities. Experiments have been performed on 18 releases of 6 open source software datasets such as jEdit, Ant, Xalan, Synapse, Tomcat and Camel. For comparative analysis, one of the most commonly used dimension reduction techniques named Principle Component Analysis (PCA) and two popular clustering techniques in defect prediction – DBSCAN and WHERE have been used in the experiment. First, the dimensions of the experimental datasets have been reduced using the proposed technique and PCA separately. Then, the reduced datasets have been clustered using DBSCAN and WHERE independently for identifying number of defects accurately. The comparative result analysis shows that the defect prediction models based on the clustering algorithms are more accurate for the dataset reduced by the proposed technique than PCA.

Key words: Software defect prediction, principal component analysis, DBSCAN, WHERE clustering, code metrics' dimension reduction technique, dataset pre-processing.

1. Introduction

Software Defect Prediction (SDP) model identifies probable software defects using code metrics and

knowledge from previous projects [1]. Moreover, it helps practitioners to assess their current project status, and to reduce software development cost by identifying faultiness in advance [2]. To predict defects, many researches have been conducted using different techniques such as Neural Network, Naive Bayes, Regression modelling, Decision tree, etc. [1], [3], [4]. Most of these techniques are trained using the whole software dataset, which contains the entire system's code metrics and historical defect information [1]. Usually, the dataset contains lots of attributes in each row such as Coupling Between Objects (CBO), Number of Public Methods (NPM), Response for a Class (RFC), Weighted Method of a Class (WMC), etc. [5]. The number of those attributes are actually called the dimensions of the dataset. Apart from that, the dataset also contains lots of variabilities because there exists heterogeneity among these attributes. It eventually results in a poor fit for the SDP model.

Researches show that the effectiveness of an SDP model depends on the learning procedure, because better learning increases the performance and accuracy of a model [5]-[8]. Usually, heterogeneity among the code metrics in software datasets [9] hinder the learning procedure of a model. To resolve this, the training datasets can be clustered based on less variability. However, existing clustering techniques suffer when dimensionality of dataset is high. To reduce the dimensionality, several works have been found in the literature based on Principal Component Analysis (PCA) [10] and Factor Analysis [6], [11]. Zimmermenn et al. performed an experiment for predicting defects, which used network analysis of dependency graphs among various pieces of codes [12]. For that purpose, it used PCA to select the best set of attributes by reducing the multicollinearity [11] among the datasets. As PCA sometimes causes loss of information, the failure of PCA might decrease the performance of the technique. Menzies et al. used PCA to reduce the multicollinearity of the dataset for the SDP model [6], [7]. It plotted the dataset considering the greatest variability component in the X-axis and the next component in the Y-axis. It then applied the WHERE clustering [6], [7] algorithm to find similar objects from the dataset. Although, it used only two most significant PCA components for plotting the dataset, it did not clarify whether only two components could describe all the variances of the dataset or not. All these techniques perform well when the dataset is linear. However, these techniques suffer when the dataset is non-linear, which leads the following research question:

How to reduce the dimensionality of software engineering dataset which is usually non-linear?

This paper proposes a source code similarity analysis technique named *SARCM* to reduce the dimensionality of software defect dataset. If dimensionality defined by the attributes of software defect dataset (such as CBO, RFC, LCOM, etc. [13]) are considered as the independent variables and the number of defects is the dependent variable, the technique reduces the dimensions based on the regression coefficient value of the independent variables to the dependent variable. To identify the relationship between the independent and dependent variables, the regression analysis is applied to the available dataset and the coefficient values are calculated. The values determine whether the independent variables are positively or negatively related to the dependent variable. Next, the proportionate impact of each independent variable on the dependent variable is calculated by multiplying the coefficient value to the corresponding independent variable value. Then, the independent variables will be grouped based on their positive or negative impacts on the dependent variable. Finally, positively related values to the dependent variable are summed and assigned to one variable named as *PosImpactValue* and negatively related values are summed to another variable named as *NegImpactValue*. Now, the similarity score between two objects can be measured by the distance where *PosImpactValue* and *NegImpactValue* are considered as *X* and *Y* axis, respectively. This makes the dataset plottable to a two-dimensional plane. As the dimension of the dataset is reduced based on the relation of independent variables to the dependent variable, the defective classes get the higher value of *PosImpactValue* and lower value of *NegImpactValue*.

And similarly, the non-defective classes get the low value of *PosImpactValue* and high value of *NegImpactValue*. Thus, the technique transforms N-dimensional dataset into 2D plane by placing similar objects closer to each other.

To assess the proposed technique, an experiment has been performed on different versions of six renowned open source software which are jEdit, Ant, Xalan, Camel, Synapse, and Tomcat from the Promise Repository [9]. The technique reduces the dimensions of the dataset into two dimensions which helps distance-based clustering algorithms. The clustering algorithms such as DBSCAN [14] and WHERE clustering [6], [7] have been applied to the dimension reduced dataset because WHERE has already been used in software defect prediction [6], [7] DBSCAN has been used here because it is yet to implement in software defect prediction. Then, the linear regression model has been applied to each cluster to find predicted defects. To compare the results of the dimension reduction approach, these two clustering approaches were also applied to dimension reduced dataset by PCA. Finally, the results are compared to show how the dimension reduction technique influences the clustering, and the clustering effects the defect prediction model.

Results show that the proposed dimension reduction technique can successfully assign new values to each entry based on the significance of the independent variables to the dependent variable. As a result, both DBSCAN [14] and WHERE clustering techniques [6], [7]. using SARCM performs better than PCA based DBSCAN and WHERE in the defect prediction, because PCA may lose information. Experimental results show that the used SDP model outperforms the existing technique in 14 datasets out of 18, clustered by the DBSCAN [14] and WHERE [6] clustering techniques where dimensions are reduced by SARCM.

2. Related Work

Training the prediction model by similar dataset improves its performance and accuracy [1]. The similarity of the dataset can be obtained by applying clustering algorithms on it. It is found that dimensions of the dataset hamper the clustering process. Many researchers suggest to reduce dimensions for clustering algorithms [6]-[8]. In this section, the widely used dimension reduction techniques along with clustering techniques in SDP are described.

2.1. Dimension Reduction in Software Defect Prediction

Dimension Reduction techniques reduce unimportant and insignificant features from a dataset. Although, the dataset having more features contains lots of information, it is difficult to extract desired information from more features. As a result, the machine learning or statistical models cannot draw conclusions from more features. So, it is needed to reduce the dimensions of the dataset by preserving all the variances for the machine learning or statistical models. Many researches in software defect prediction have used Dimension Reduction techniques. Some of those important researches are outlined below.

Nagappan et al. proposed a failure prediction model by investigating the relationship between failure-prone software entities and their complexity measures [11], [15]. It used linear regression analysis as the prediction model for identifying failure prone components. It performed an empirical study on five Microsoft software systems. It showed that multicollinearity existed among the complexity metrics and there was no single set of metrics that could act as the best defect predictor. To overcome the multicollinearity problem, it used PCA to select minimum numbers of metrics for which the cumulative variance was greater than 96%. After selecting the best set of metrics, it used these to identify the relationship between complexity metrics and failure-proneness. Results showed that the complexity metrics can predict post release defects. Although, the prediction model using PCA worked well, but this technique might select lots of features until the cumulative frequencies was greater than 96%, which might

hinder the training process.

Zimmermen *et al*. proposed a defect prediction model using network analysis of dependency graphs among various pieces of code [12]. It used multiple linear regression analysis as the prediction model for predicting the critical binaries. It used PCA to reduce the multicollinearity [11] and selected only those principal components, for which the cumulative variance was greater than 95%. To show the effectiveness of the proposed method, it performed an experiment on Windows Server 2003, and the results showed that complexity metrics and network measures could predict 30% and 60% of these critical binaries, respectively. As it used PCA, it might lose a small amount of information and might select a set of metrics for which the cumulative variance was greater than 95%.

Ceylan *et al*. proposed a defect prediction model using Decision Tree, Multi-Layer Perceptron and radial basis functions to predict the number of defects per module or function [16]. To remove multicollinearity by eliminating the correlations among the attributes, it used PCA to the dataset. The experiment had been carried out on some real-life software projects collected from three big software companies in Turkey. Results showed that the proposed prediction model improved the performance approximately 32.61% for Company-A and 60% for the other two companies. As it used PCA, it also inherited the same problems mentioned above.

Menzies *et al*. proposed a software defect prediction model that learnt from software clusters with similar characteristics to resolve the variabilities [6], [7]. It performed clustering of the source code using WHERE clustering technique that considered only the code metrics and learning treatment using pairs of neighboring clusters. To perform the WHERE clustering in the dataset, the dimensions of the dataset were needed to be reduced. It used PCA to reduce the dimensions of the dataset. It plotted the dataset considering the most variability component in X-axis and the next component in Y-axis. Then, it applied the WHERE clustering to find similar objects. The downside was the consideration of only two PCA's components to plot the dataset without taking into account others. It did not also mention that whether these two components could describe all the variances or not. As a result, the clustering algorithms considering only two PCA's components might not group the dataset based on their similarity properly.

For the software defect prediction, sometimes the dataset needs to be divided into multiple clusters based on their similarity to train the SDP model properly. As the dataset contains multiple dimensions [5]-[8], the distance-based clustering cannot perform well until the dimension is reduced. The existing dimension reduction techniques such as PCA can reduce the dimensions, but taking two components into account for plotting into two-dimensional plane by avoiding others might cause a great loss of information. So, further researches are needed to represent the whole dataset in a meaningful way.

2.2. Used Clustering Techniques in Defect Prediction

The clustering algorithm in defect prediction divides the whole software dataset into clusters based on its different properties, so that the prediction model can get more accurate dataset for training purposes. The software engineering datasets always contain lots of variability such as heterogeneity among the code metrics [1], [3]. These variabilities cause the poor fit of machine learning algorithms or statistical inferences to the dataset [1]. If the variabilities among the datasets can be minimized, it will increase the probability of fitting the data to SDP models.

Zimmermann *et al*. proposed a technique to predict defect at the design time by considering call dependencies, data dependencies and Windows specific dependencies such as shared registry entries [17]. It used Support Vector Machine (SVM) to predict the post release defects at design time. To perform the experiments, it collected the dependencies of all binaries such as executable files, for example, COM, EXE, etc. and dynamic-link files such as DLL for Windows Server 2003. It concluded that the software defect

proneness could be predicted using the dependencies among all binaries.

To resolve the variabilities among the dataset, Menzies *et al.* proposed a software defect prediction model that learnt from software clusters with similar characteristics [6], [7]. It showed learning from clusters was better than learning from the entire system because it might falsify the data used by the prediction model. It performed clustering using WHERE technique that considered the code metrics and learning treatment using pairs of neighboring clusters. It advised that empirical software engineering should focus on ways to find the best local lessons for groups of related projects because global context was often obsolete for particular local contexts in the defect prediction. This premise also showed the importance of using the clustering to provide the accurate dataset for training SDP model.

Scaniello *et al.* [5] proposed a defect prediction model to predict defects using Step-Wise Linear Regression (SWLR) that used clustering of the source code rather than the entire system. It considered references between methods and attributes to form clusters among the related classes using BorderFlow algorithm. The BorderFlow clustering algorithm performed clustering by maximizing the flow from the border to center and minimizing the flow from border to outside of the cluster. Then, it applied the SWLR model on each cluster and produced better results than other models that perform prediction considering the entire system. It focused on clustering using source code whereas Menzies *et al*. [6], [7] focused on clustering using code metrics. It formed clusters considering only related classes which meant it used coupling information among the classes to form clusters. So, the other code metrics' impacts were needed to analyze for defect prediction.

Rayhanul *et al*. proposed a defect prediction model for Java based software to predict defects, where clustering was performed using package information of a project, named as Package Based Clustering (PBC) technique [18], [19]. PBC used package information because thousands of related classes and interfaces are organized by placing these into packages in Java based software. As a result, PBC produces clusters based on the related classes and interfaces that eventually improves the performance of the SDP model. It then validated the clusters using the number of selected code metrics before applying prediction model. It used linear regression model as the prediction model, and performed an experiment on jEdit-3.2 to evaluate the proposed PBC. Results showed that it outperformed the entire system and BorderFlow algorithm in some cases.

In a nutshell, a general overview of defect prediction using clustering emphasizes on the clustering of source code to improve the training process. All of the above discussed clustering algorithms use software code metrics, source code dependencies or code similarities, etc. to group source codes. Some approaches use PCA to reduce the dimension of the dataset before applying the different clustering algorithms [5]-[7]. However, none of those methods work perfectly in all Promise Repository's datasets [9].

3. Proposed Methodology

The existing literature supports that there exists a relationship between software code metrics and defect proneness [1]-[3]. The appropriate software code metrics for defect prediction are CBO, LCOM, RFC, WMC, DIT, LOC, NOC and NPM [13]. Those are also considered as the dimension of the dataset and independent variables for the SDP model. So, if the dimension reduction technique reduces the dimensions based on the coefficient values of the independent variables, the produced latent variable will be based on the independent variables' significance to the dependent variables [1], [19].

The coefficient value of one independent variable to the dependent variable can be measured using regression analysis, because it estimates the relationships among variables [19]. In this context, if CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC are considered as the independent variables, and defect as dependent variable, the regression equation will be as follows-

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c \quad (1)$$

Here,

- Y is the number of defects in an Object Oriented class
- $x_{1...}\, x_n$ are the independent variables, which are CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC
- $b_{1...}\, b_n$ are the coefficient values of independent variables respectively
- c is the value of Y when all independent variables are 0.

The coefficient values ($b_{1...}\, b_n$) represent the linear association between the independent and dependent variables [19]. The coefficient values can be positive or negative based on the impact of the independent variables to the dependent variable [19]. A coefficient value –

- greater than 0 indicates that two variables are positively related.
- less than 0 indicates that two variables are negatively related.
- 0 indicates that there is no linear relationship between the two variables.

Table 1. The Coefficient Values of the Selected Independent Variables for Ant-1.6

Independent variable	Coefficient value	Latent Variable
CBO	-0.0004216989	*NegImpactValue*
DIT	-0.0568333506	
WMC	-0.0722705159	
LOC	0.0004402161	*PosImpactValue*
NPM	0.0527224790	
LCOM	0.0002719006	
RFC	0.0256817433	
NOC	0.0025542023	

To reduce the dimension of the dataset, the coefficient values are then multiplied to the corresponding variable values. Now the software engineering dataset's attributes have positive and negative values which represent their impacts to defects. Then the positive and negative values are summed to produce two latent variables, named as *PosImpactValue* and *NegImpactValue* respectively for each row in the dataset. The *PosImpactValue* and *NegImpactValue* are calculated using Equation (2) and (3) respectively. As a result, the final dataset contains only two values for each entry. As an example, the impact analysis of code metrics to software defects for Ant-1.6 is given in Table 1. This table shows that CBO, DIT and WMC have negative coefficients and produce *NegImpactValue*, and the remaining code metrics have positive coefficients and produce *PosImpactValue*. It is needed to mention that the impact of code metrics may change from dataset to dataset. Although, CBO, DIT and WMC have negative impacts in Ant-1.6, those may have positive impacts in other projects and this is also true for the other code metrics. The whole procedure to reduce the dimensions of a dataset is given in Algorithm 1.

$$PosImpactValue = \sum b_i x_i \quad (2)$$

Here, b_i is positive.

$$NegImpactValue = \sum b_i x_i \quad (3)$$

Here, b_i is negative.

In the very beginning of Algorithm 1, a regression equation such as Equation (1) is selected and applied to the existing available dataset. It finds the regression coefficients b_i using Lines 1-3. These regression coefficient values might be positive or negative depending on their impacts to the dependent variable. This algorithm produces the new value for each entry based on these significances. For that purpose, it traverses each row and selects each variable in the row. Then it multiplies the variable value with its coefficient value using Lines 4-9. The used dataset now contains only positive and negative values which actually mean their impacts to the number of defects.

Algorithm 1: Dimension Reduction Algorithm

Require: dataset D, Coefficient Value b, Independent Variable Value x, Number of software defects Y, *PosImpactValue* P and *NegImpactValue* N

1: Select a linear regression equation, $Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c$
2: Apply the selected equation on the available D
3: Compute b for each independent variable, x_i.
4: **for each** x_i row in D **do**
5: **for each** in row do
6: Select b for the selected x_i
7: $x_i \leftarrow b_i * x_i$
8: **end for**
9: **end for**
10: **For each** row in D do
11: Set value for P=0, N=0
12: **for each** x_i in a row do
13: **if** $x_i > 0$ then
14: $P \leftarrow P + x_i$
15: **else**
16: N=N+ x_i
17: N=N*(-1)
18: **end if**
19: **end for**
20: **end for**
21: Set value P, N for the selected entry.

After getting some positive and negative significance, this algorithm combines those values into *PosImpactValue* and *NegImpactValue*. To do so, it first initializes *PosImpactValue* and *NegImpactValue* to 0 using Line 11. Then, it iterates again the dataset and selects each variable in the dataset's row. After that, it identifies their significance using Line 13. If the value is greater than 0, it adds this variable to the *PosImpactValue* using Line 14. Otherwise, it adds to *NegImpactValue* applying Line 16. To remove negative sign from *NegImpactValue*, it is then multiplied by -1 using Line 17. Finally, it assigns *PosImpactValue* and *NegImpactValue* values to the corresponding dataset's row.

4. Experimental Setup

In this section, the focus will be kept on how the proposed SARCM is performed in comparison with the PCA in the context of software defect prediction, including the implementation details, code metrics selection and data collection.

4.1. Code Metrics Selection

The software code metric is a quantitative measure to define the quality of software. These metrics are used in software defect prediction to improve software qualities. Among all the metrics, the method level metric is widely used in structured and Object-Oriented Programing (OOP), and the class level code metric is only used for OOP [1], [3]. According to some research work [1], [13] all of the above metrics' properties are not influential for defect prediction. Furthermore, these studies also show the following eight code metrics which are influential in software defect prediction [1], [13]. Those selected code metrics are Weighted Methods per Classes (WMC), Depth of Inheritance Tree (DIT), Number of Children (NoC), Coupling between Objects (CBO), Response for Classes (RFC), Lack of Cohesion in Methods (LCOM), Number of Public Method (NPM) and Lines of Code (LOC).

4.2. Dataset Collection

The proposed dimension reduction technique for software defect prediction has been experimented on different versions of six open source software projects. These datasets are Ant 1.3-1.7, jEdit 4.2-4.3, Xalan 2.4-2.7, Camel 1.0-1.7, Synapse 1.2, and Tomcat. All of these defect datasets have been collected from the Promise Repository [9]. These datasets contain the corresponding software code metrics and defect information which are usually used by the prediction model to predict the defect for future releases [3].

4.3. Measuring the Similarity of Objects

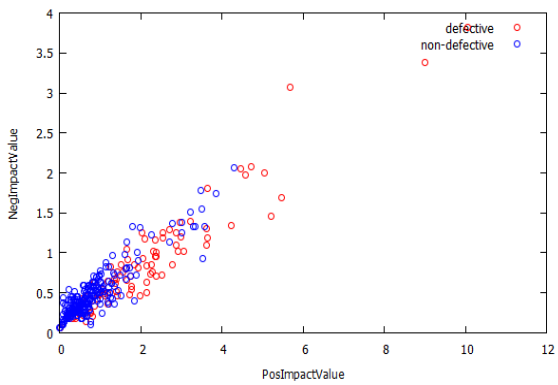

Fig. 1. The distribution of OO classes in the dimension reduced dataset by SARCM for Ant-1.6.

To measure the similarity among the objects based on their defectiveness, the dimension reduced dataset is plotted considering the *PosImpactValue* as X-axis and *NegImpactValue* as Y-axis. In the plane, similar classes become adjacent and dissimilar classes become farther from each other because *PosImpactValue* is high for defective objects and *NegImpactValue* is high for low defective objects. So, the similarity among the objects can be analyzed using their Cartesian distances in the two-dimensional plane.

As an example, the distribution of defective and non-defective objects of the dimension reduced dataset for Ant-1.6 project is shown in Fig. 1. However, the detailed results for all the experimental datasets have been described in Section V. In Fig. 1, the defective and non-defective objects are represented by the red and blue circle, respectively. Here, the density of the red circle is high in the area, nearest to X-axis, because high defective objects usually have high value of CBO, WMC, etc. That makes the high value of *PosImpactValue* and low value of *NegImpactValue* for those objects. Contrary, non-defective objects have

low value of *PosImpactValue* and high value of *NegImpactValue*. It is now clear that the position of each object in Fig. 1 corroborates the assumption that the similar objects get closer values in the dimension reduced dataset, and objects are distributed based on their defects.

In terms of OOP objects distribution, there exists a significant difference between Fig. 1 and Fig. 2. In Fig. 1, different Object Oriented (OO) classes create cumulative straight line because SARCM technique reduces the dimension based on the cumulative impact of the independent variables to the dependent variable. On the contrary, in Fig. 2, all OO classes form a line, parallel to the X-axis because the values of the most significant principal component are closer to each other.

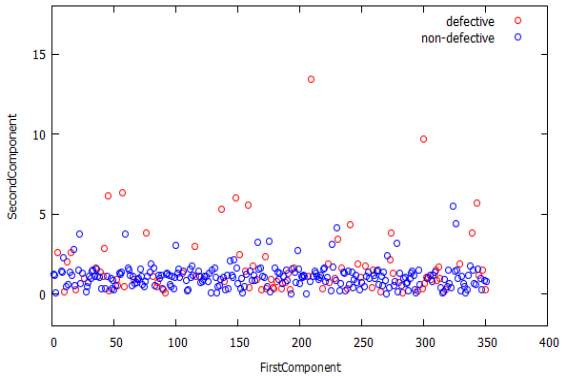

Fig. 2. The distribution of OO classes in the dimension reduced dataset by PCA for Ant-1.6.

As the new dimension reduced dataset contains closer value for similar objects, it can be divided into clusters considering distance measure. For that purpose, the density-based clustering algorithms such as DBSCAN and WHERE clustering can be better options [6]. It is also necessary to mention that DBSCAN is not the only solution to perform clustering. Any distance-based clustering algorithm, for example, k-means, can be applied to find clusters from the dimension reduced dataset.

4.4. Validation of the SDP

As the success of the linear regression model depends on the training dataset, after reducing the dimensions, the total dataset is divided into two sets such as trainset and testset. In this paper, like Juban *et al*. [20], 80% of data is used to train the prediction model and the remaining is used to assess its performance and accuracy.

To evaluate the quality of the SDP model, the Mean (M), Median (Md) and Standard Deviation (StD) of Absolute Residuals (AR) are computed. The AR value, widely used in the performance measure of the linear regression model [5], is the difference between predicted defects and actual defects of a particular OO class. It shows that the smaller the AR value is, the better the SDP model will be [5].

To compare *SARCM* with PCA in terms of defect prediction accuracy, the *error* is computed using Equation (4) [5]. The *error* value, in between -1 and +1, shows whether *SARCM* is better or worse in the context of software defect prediction [5]. For a software release, negative value of *error* indicates *SARCM* outperforms *PCA* technique, whereas a positive value indicates the opposite [5].

$$error = \frac{MAR(SARCM) - MAR(PCA)}{StDAR(PCA)} \tag{4}$$

where

- MAR(SARCM) is the mean value of the AR using SARCM
- MAR(PCA) is the mean value the AR using PCA
- StDAR(PCA) is the standard deviation of the AR using PCA

5. Result Analysis

The results of the selected SDP model using different clustering approaches are compared using the MAR, MdAR and StDAR of the Absolute Residuals (AR). To evaluate the performance of SARCM compared to PCA, two clustering techniques which are DBSCAN and WHERE are applied separately to the dimension reduced dataset. For each dimension reduced dataset by SARCM, those clustering techniques divide the whole dataset into clusters based on the similarity of the OO classes, because SARCM reduces the dimensions in a way where the similar objects get closer. For the dimension reduced dataset by PCA, the above-mentioned clustering methods also divide the dataset into multiple clusters based on the variances among elements, because PCA reduces the dimensions in such a way so that the selected principal components can describe all variances. Then, the SDP model uses those clusters of datasets for training purposes to increase the accuracy and performance. Finally, the accuracy and performance of the SDP model are measured by taking the residual value of predicted and actual defects.

The whole analysis is divided into two phases. In phase I, the comparison of AR values is analyzed by computing the MAR, MdAR and StDAR values. In phase II, the error values are calculated to determine which technique performs well in software defect prediction.

Phase I: The Comparison of Absolute Residual Values

The Absolute Residual is the difference between actual and predicted defects. It determines how better the SDP model is. Usually, the low value of AR shows the high performance and accuracy of the SDP model, and on the contrary, the high value determines the low performance and accuracy [5]. In this paper, the descriptive statistics of the ARs are summarized in Table 2 to compare the impact of the proposed SARCM technique on the SDP model. This table illustrates the comparison of MAR, MdAR, StDAR values of DBSCAN and WHERE clustering techniques using the dimension reduced dataset by SARCM and the PCA, respectively. The descriptions of DBSCAN using SARCM (S-DBSCAN) vs. DBSCAN using PCA (P-DBSCAN), and WHERE using SARCM (S-WHERE) vs. WHERE using PCA (P-WHERE) are given in Table 2.

S-DBSCAN vs. P-DBSCAN

In Table 2, the second and third column represent the AR values of S-DBSCAN and P-DBSCAN, respectively. It shows that MAR values are minimum for S-DBSCAN compared to P-DBSCAN. It means that the SDP model considering S-DBSCAN produces smaller AR values compared to P-DBSCAN. Which indicates that the SDP model considering S-DBSCAN has better performance and accuracy than P-DBSCAN.

The SDP has poor performance in Ant-1.7, Synapse-1.2 and Xalan-2.4 because DBSCAN clustering algorithm cannot divide those datasets properly. The detail inspection in those datasets shows that those contain low number of objects that lead small number of objects in a cluster. As a result, SARCM cannot perform well in those datasets, and finally that results low prediction accuracy in the SDP model. In a nutshell, the SDP model considering S-DBSCAN outperforms P-DBSCAN in 15 datasets out of 18, illustrated in Figure 3, which indicates that S-DBSCAN is better than P-DBSCAN in software defect prediction.

S-WHERE vs. P-WHERE:

The comparison of WHERE clustering method is illustrated in Table 2 using column S-WHERE and P-WHERE. It shows that the mean value of AR is also minimum in S-WHERE compared to P-WHERE. Figure 4 shows S-WHERE performs better than P-WHERE in 14 datasets out of 17, which is an indication of better prediction accuracy of SDP model when SARCM is taken into account in WHERE clustering approach.

This proposed technique has poor performance in Ant-1.4, jEdit-4.2, jEdit-3.2 and Xalan-2.5. Among these, the MAR value is high for Ant-1.4 which means that it largely has low performance in this dataset. The detail inspection in those dataset gives that those contain low number of objects. As a result, SARCM cannot perform well, which results in inappropriate clustering of the dataset, and finally affects the

accuracy and performance of the SDP model. In essence, it can be concluded that the SDP model considering S-WHERE outperforms P-WHERE in 14 datasets out of 17 datasets, listed in Table 2.

Phase 2: The Comparison of Error Values

The error values show how much better or worse the SDP model is, compared to others. It is calculated from MAR and StDAR values using Equation (4). Usually, the negative value shows high accuracy and positive value shows low accuracy of the SDP model [5]. The error values of DBSCAN are calculated considering S-DBSCAN and P-DBSCAN by applying Equation (4). In the same way and using the same equation, the error values for WHERE clustering are also calculated considering S-WHERE and P-WHERE. Table 3 summarizes all error values for the DBSCAN and WHERE clustering techniques.

Figure 5 summarizes the error values considering DBSCAN. In this figure, all points under the horizontal line represent the software releases, for those the SDP model considering S-DBSCAN has better prediction accuracy. Here, 14 software releases have negative error values, so the SDP model has better accuracy in those datasets. In this figure, only 3 datasets which are Ant-1.7, Xalan-2.4 and Synapse-1.2 have positive error values, so the model considering S-DBSCAN has poor performance in those datasets because of the inappropriate reachability distance measure for DBSCAN. As a result, the DBSCAN cannot properly group the software dataset which actually results low prediction accuracy of the SDP model. Finally, it can be concluded that S-DBSCAN works well in 14 datasets.

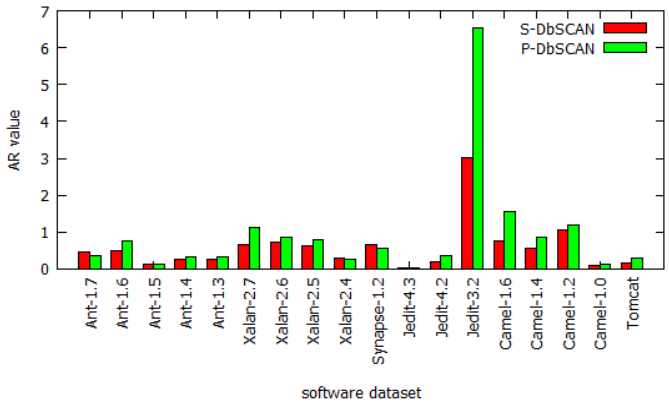

Fig. 3. The AR mean values of S-DBSCAN and P-DBSCAN.

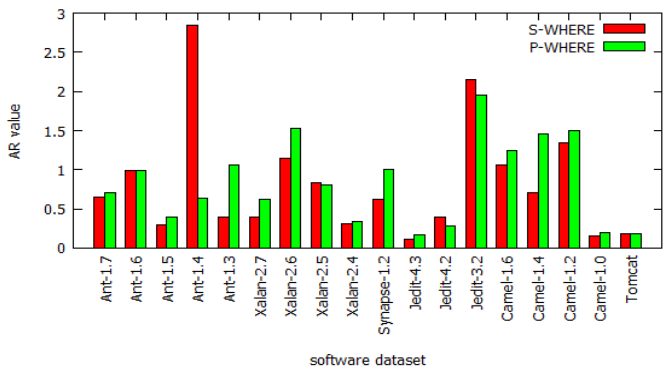

Fig. 4. The AR mean values of S-WHERE and P-WHERE.

In the same way, the error values considering the WHERE clustering technique are summarized in Fig. 6. In this Fig. 14 software releases reside below the horizontal line. As said earlier, the points below the line represent better prediction performance and accuracy, so SDP model using S-WHERE has better accuracy in 14 datasets compared to P-WHERE. Here, only 3 datasets reside above the horizontal line. So,

the WHERE clustering algorithm considering SARCM performs better than PCA in the 14 datasets. It has poor performance in Ant-1.4, because it contains lower number of objects, as a result, clustering technique produces clusters with small number of objects which actually results in low prediction accuracy of the SDP model. For Xalan-2.5 and jEdit-4.2, the error values are close to zero, so it can be negligible. Eventually, it is coherent that the SDP model using S-WHERE performs better than P-WHERE.

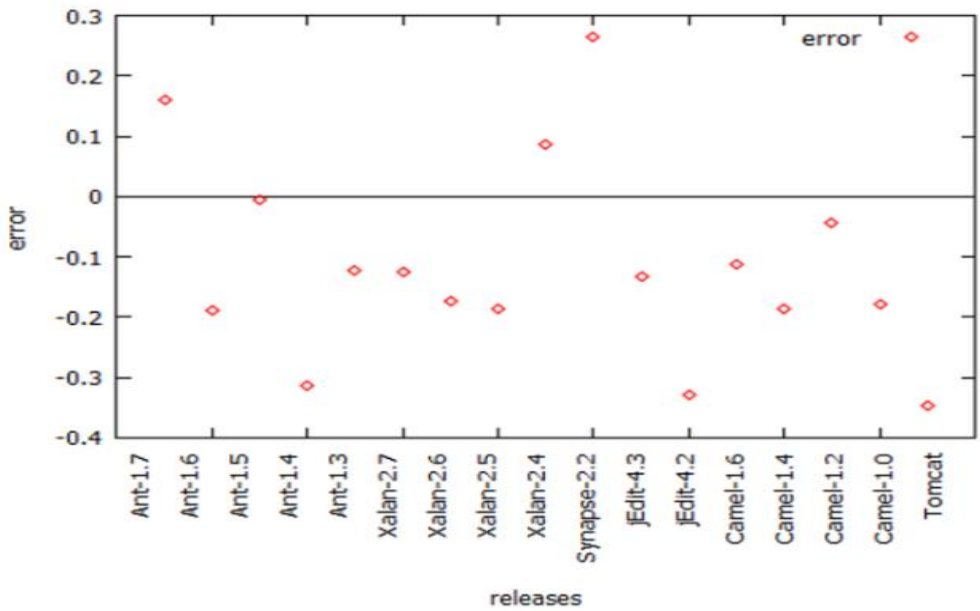

Fig. 5. The distribution of error values for DBSCAN.

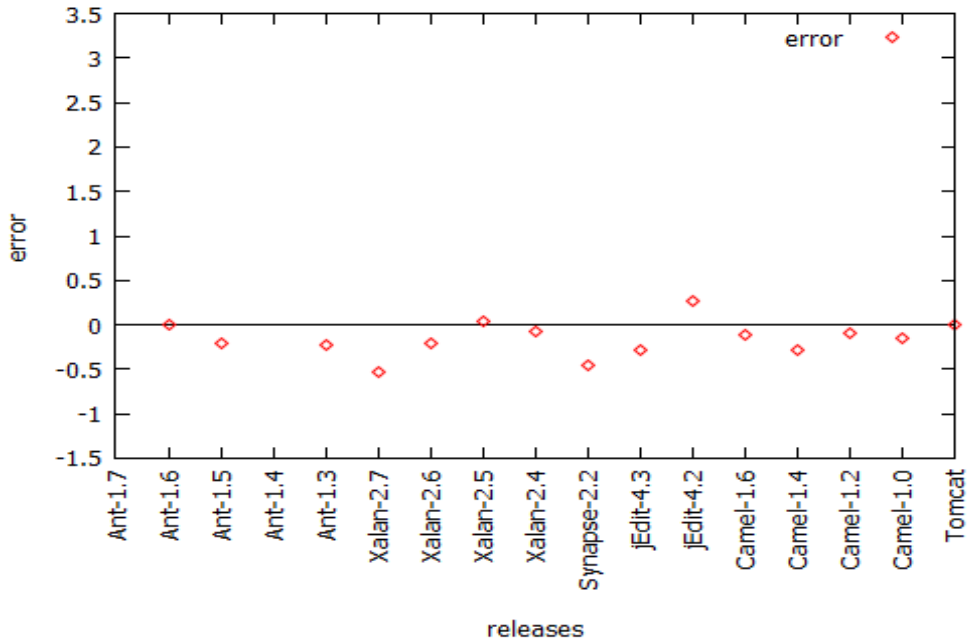

Fig. 6. The distribution of error values for WHERE.

Table 2. The Mean, Median and Standard Deviation Value of AR

Dataset	S-DBSCAN	P-DBSCAN	S-WHERE	P-WHERE

	MAR	MdAR	StDev	MAR	MdAR	StDev	MAR	MdAR	StDev	MAR	MdAR	StDev
Ant-1.7	0.46	0.24	0.54	0.37	0.18	0.54	0.65	0.48	0.60	0.71	0.30	1.07
Ant-1.6	0.48	0.29	0.59	0.76	0.27	1.48	0.99	1.02	0.70	0.99	0.63	0.98
Ant-1.5	0.13	0.10	0.12	0.13	0.08	0.14	0.30	0.05	0.67	0.40	0.14	0.47
Ant-1.4	0.27	0.19	0.22	0.32	0.29	0.18	2.85	0.17	7.68	0.64	0.45	0.57
Ant-1.3	0.25	0.15	0.31	0.31	0.08	0.49	0.39	0.17	0.50	1.06	0.26	2.82
Xalan-2.7	0.664	0.57	0.40	1.11	0.37	3.6	0.38	0.26	0.37	0.62	0.62	0.45
Xalan-2.6	0.72	0.57	0.54	0.87	0.62	0.88	1.14	0.94	0.93	1.52	0.89	1.85
Xalan-2.5	0.62	0.53	0.51	0.78	0.61	0.87	0.83	0.56	0.88	0.80	0.51	0.81
Xalan-2.4	0.28	0.12	0.45	0.25	0.13	0.33	0.30	0.16	0.44	0.33	0.15	0.47
Synapse-1.2	0.67	0.37	0.69	0.55	0.46	0.45	0.61	0.38	0.62	1.0	0.86	0.86
jEdit-4.3	0.02	0.02	0.02	0.04	0.01	0.12	0.10	0.04	0.15	0.16	0.08	0.19
jEdit-4.2	0.21	0.158	0.202	0.349	0.313	0.415	0.397	0.288	0.35	0.28	0.13	0.41
jEdit-3.2	3.01	1.97	4.34	6.53	2.23	10.02	2.15	1.34	3.23	1.94	1.33	2.04
Camel-1.6	0.748	0.322	1.035	1.549	0.49	7.064	1.065	0.403	1.34	1.23	0.64	1.5
Camel-1.4	0.69	0.27	1.34	0.86	0.51	0.90	0.71	0.23	1.33	1.45	0.42	2.58
Camel-1.2	1.11	0.53	1.27	1.20	0.62	2.21	1.34	0.70	1.41	1.50	1.00	1.73
Camel-1.0	0.089	0.034	0.200	0.145	0.023	0.313	0.149	0.041	0.303	0.198	0.183	0.327
Tomcat	0.149	0.046	0.241	0.290	0.093	0.406	0.189	0.033	0.303	0.189	0.126	0.274

Table 3. The Error Values of all Datasets for DBSCAN and WHERE

Dataset	Error		Dataset	Error	
	DBSCAN	WHERE		DBSCAN	WHERE
Ant-1.7	16%	-6%	Synapse-2.2	26.5%	-44.63%
Ant-1.6	-18.87%	0.135%	jEdit-4.3	-13.23%	-29.27%
Ant-1.5	-0.576%	-20.58%	jEdit-4.2	-32.85%	26.82%
Ant-1.4	-31.31%	383%	Camel-1.6	-11.3%	-11.55%
Ant-1.3	-12.38%	-23.4%	Camel-1.4	-18.72%	-28.546%
Xalan-2.7	-12.52%	-52.23%	Camel-1.2	-4.42%	-8.82%
Xalan-2.6	-17.40%	-20.29%	Camel-1.0	-17.76%	-14.8%
Xalan-2.5	-18.65%	4.7%	Tomcat	-34.7%	-0.25%
Xalan-2.4	8.6%	-6.7%			

The SDP model has better prediction accuracy when the clustering algorithms use the dimension reduced dataset by SARCM, because it reduces the dimension based on the significance of different software metrics (such as CBO, RFC, LCOM, WMC, DIT, NPM, LOC and NOC) to the number of software defects. As a result, the dimension reduced dataset gets closer value for similar objects and SDP model

gets similar dataset for learning procedure. In principle, it can be stated that SARCM can significantly improve the accuracy of SDP model which results lower AR values compared to PCA.

6. Conclusions

In this paper a technique has been proposed for reducing code metrics' attributes for software engineering dataset through the relationship of code metrics to defects. It analyzes the relationship of code metrics such as CBO, RFC, LCOM, WMC, etc. to defect by applying regression analysis on the available datasets. Then, the dimensions of dataset are reduced to two variables which are *PosImpactValue* and *NegImpactValue*, based on those positive and negative impacts, respectively. This dimension reduced dataset is plotted on the two-dimensional plane considering *PosImpactValue* as X-axis and *NegImpactValue* as Y-axis for the distance-based clustering approaches. It has been observed, in the two-dimensional plane, the similar objects situate adjacent to each other and dissimilar objects reside far from each other.

For the experimental analysis, the DBSCAN and WHERE clustering techniques have been applied to dimension reduced dataset to group those into clusters based on their distance measure. Finally, the SDP model has been applied to 18 releases of 6 open source software systems. During the experiment, the SDP model learnt from the clusters of the dataset by DBSCAN and WHERE clustering techniques where dimensions were reduced by SARCM. Then, the same SDP model was inherited and implemented on those datasets, dimensions reduced by PCA. Eventually, results are compared based on the absolute residual values, and the results show that the SDP model performs well in the 14 datasets out of 17 datasets when dimensions were reduced by SARCM. In future, a software tool will be developed based on SARCM along with different popular clustering techniques to predict software defects easily.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Rayhanul Islam conducted the research and wrote the paper; Abdus Satter and Atish Kumar Dipongkor provided active feedback on the drafts for revision; Saeed Siddik helped a lot while formulating the research question and analyzing the existing works; Prof. Dr. Kazi Sakib supervised the workflow of the research; all authors had approved the final version.

References

[1] Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications, 36(4)*, 7346-7354.

[2] Dallal, J. A. (2018). Predicting object-oriented class fault-proneness: A replication study. *Journal of Software, 13(5)*, 269-276.

[3] Liu, W. (2019). Software defect data mining: A survey of severity analysis. *Journal of Software*, *14(10)*, 457-478.

[4] Neela, K. N., Asif, S. A., Ami, A. S., & Gias, A. U. (2017). Modeling software defects as anomalies: A case study on promise repository. *Journal of Software, 12(10)*, 759-772.

[5] Scanniello, G., Gravino, C., Marcus, A., & Menzies, T. (2013). Class level fault prediction using software clustering. *Proceedings of the 28th International Conference on Automated Software Engineering (ASE)*.

[6] Menzies, T., Butcher, A. D. C., Marcus, A., Layman, L., Shull, F., & Turhan, B. (2013). Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*.

[7] Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., & Cok, D. (2011). Local vs. global models for effort estimation and defect prediction. *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering Proceedings*.

[8] Bettenburg, M. N. A. E. H. N. (2012). Think locally, act globally: Improving defect and effort prediction models. *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*.

[9] Menzies, T., Caglayan, B., He, Z., Kocaguneli, E., Krall, J., Peters, F., & Turhan, B. (2020). The PROMISE repository of empirical software engineering data. Retrieved from http://promisedata.googlecode.com

[10] Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics,* 433-459.

[11] Nagappan, N., & Ball, T. (2007). Using software dependencies and churn metrics to predict field failures: An empirical case study. *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*.

[12] Zimmermann, T., & Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. *Proceedings of the 30th International Conference on Software Engineering*.

[13] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*.

[14] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise.

[15] Nagappan, T. B. A. Z. N. (2006). Mining metrics to predict component failures. *Proceedings of the 28th International Conference on Software Engineering*.

[16] Ceylan, E., Kutlubay, F. O., & Bener, A. B. (2006). Software defect identification using machine learning techniques. *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*.

[17] Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. *Proceedings of the International Workshop on Predictor Models in Software Engineering.*

[18] Islam, R., & Sakib, K. (2017). A package-based clustering approach to enhance the accuracy and performance of software defect prediction. *International Journal of Software Engineering, Technology and Applications,* 2(1), 1-2.

[19] Draper, N. R., & Smith, H. (1981). *Applied Regression Analysis*, New York: John Wiley and Sons.

[20] Juban, J., & Siebert, G. N. K. N. (2007). Probabilistic short-term wind power forecasting for the optimal management of wind generation. *Power Tech,* 683-688.

Copyright © 2020 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).

Rayhanul Islam is currently working as a lecturer at the Institute of Leather Engineering and Technology (ILET), University of Dhaka, Dhaka, Bangladesh. He has completed his master of science in software engineering (MSSE) with the thesis in software source code's dimension reduction for defect prediction from Information Technology (IIT), University of Dhaka. He also worked as an associate software engineer at a renowned software company named KAZ Software Ltd. His research interests include software engineering, machine learning, and data mining.

Abdus Satter is a lecturer at the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. He pursued his master of science in software engineering (MSSE) and the bachelor of science in software engineering (BSSE) from the same institution with the top score in his class. His core areas of interest are software repository mining, software engineering, web technologies, systems and security. He has numerous awards in various national and international software and programming competitions, hackathons project showcasings.

Atish Kumar Dipongkor is a faculty member of Computer Science and Engineering at Jashore University of Science and Technology (JUST), Bangladesh. He has earned his master of science in software engineering (MSSE) from the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. Before joining just as a lecturer, he has worked as a senior software engineer in a multinational IT organization (Brain Station 23 Ltd.). His core areas of interest are code smell, refactoring, system architecture design, web technologies, and bangla text processing.

Saeed Siddik has been working on software testing and software analysis research, where he experimented how software are developed and tested efficiently. He has completed his M.Sc. in software engineering, including the highest marked thesis dissertation on software test case prioritization from IIT, University of Dhaka. He was the first research student of IITDU optimization research group, where he was working on software design migration to enhance modularity and manageability. He is a member of IEEE, SIGSOFT, and group adviser of IEEE CS SB at University of Dhaka.

Kazi Sakib is a professor at the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. He received his Ph.D. in computer science at the School of Computer Science and Information Technology, RMIT University. His research interests include software engineering, cloud computing, software testing, software maintenance, etc. He is an author of a great deal of research studies published at national and international journals as well as conference proceedings.