TOWARDS PREDICTING SOFTWARE DEFECTS WITH CLUSTERING TECHNIQUES

Waheeda Almayyan

Computer Information Department, Collage of Business Studies, PAAET, Kuwait

ABSTRACT

The purpose of software defect prediction is to improve the quality of a software project by building a predictive model to decide whether a software module is or is not fault prone. In recent years, much research in using machine learning techniques in this topic has been performed. Our aim was to evaluate the performance of clustering techniques with feature selection schemes to address the problem of software defect prediction problem. We analysed the National Aeronautics and Space Administration (NASA) dataset benchmarks using three clustering algorithms: (1) Farthest First, (2) X-Means, and (3) self-organizing map (SOM). In order to evaluate different feature selection algorithms, this article presents a comparative analysis involving software defects prediction based on Bat, Cuckoo, Grey Wolf Optimizer (GWO), and particle swarm optimizer (PSO). The results obtained with the proposed clustering models enabled us to build an efficient predictive model with a satisfactory detection rate and acceptable number of features.

KEYWORDS

Software defect prediction, Data mining, Machine learning, Clustering, Feature selection.

1. INTRODUCTION

Information technology companies need to develop and build high-quality software, which can be a challenging process since it leads to extremely high computational complexity [1]. Yet, such concerns can be eliminated, especially if we test the new software modules through learning from defect data [2]. Therefore, the software defect prediction process becomes an essential part of improving software reliability and predicting the potential defects during the early stages of any software development lifecycle.

During building projects, an evaluation of software life-cycle activities, such as performance analysis and functional tests accompanied by the measurement of metrics, is highly recommended to decide at which point to apply quality assurance techniques [3]. Goodman defines software metrics as: “The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products” [4].

Software metric tools have been applied to defect prediction to help in improving the quality of software project management [5]. Therefore, in any information system, every software module is depicted as a set of metrics values and contains binary fault-proneness class label information. The resulting metric values are used to build predictive models to label the module as to whether it is fault-prone or not.

Throughout the past decade, data mining and machine learning methods have been widely recognized as solutions for a number of classification problems [6]. However, high dimensionality threatens the modelling process as most real-time datasets usually include irrelevant and redundant features. The high dimensionality problem has been subjected to numerous studies supporting the claim that it leads to extensive computational cost and degradation of the performance of certain specific models [7]. Therefore, a variety of feature selection methods were recommended to exclude irrelevant and unnecessary features.

The concept of clustering addresses the discovery of natural groupings in datasets [8]. The essence of clustering techniques is to analyse data even in the absence of class label. In case of analysing software quality, defective and non-defective software modules will have similar software metrics and accordingly will likely form clusters. So, instead of inspecting and labelling software modules one at a time, software engineers can assign all of the modules with similar in quality label to the same cluster.

Selecting a suitable clustering algorithm for a software defects prediction problem is worth investigating. This paper concerns implementation of clustering techniques for predicting software defects. In order to evaluate different feature selection algorithms, this article presents a comparative analysis involving the process of combining several swarm-based algorithms with the clustering method to finding optimal solutions. The performance of these algorithms is compared to each other by computational simulation results considering the National Aeronautics and Space Administration (NASA) data repository.

The main contribution of this article is the implementation of a bio-inspired feature selection-based clustering model for predicting software faults. We conducted a comparative analysis on the impact of applying several swarm-based feature selection methods to the defect prediction. We investigated deployment of multiple clustering techniques in an attempt to identify a collection of crucially needed software design processes.

This paper is organized in several sections. The next section, Section II, discusses the related work. Section III explains the proposed algorithm. Section IV reflects the results and findings of the experiments, and Section V concludes this study.

2. RELATED WORK

For more than a decade, software defect prediction has been recognized an important research topic in software engineering. Software defect prediction has attracted the attention of scholars in knowledge discovery and data mining fields. Many scholars have considered numerous machine learning algorithms to tackle the classification problems related to software defect prediction [9–18].

In [9], researchers studied and explored the data gathered from 27 moderated-size software sets using six classification models: (1) principal component analysis (PCA), (2) discriminant analysis, (3) logistic regression (LR), (4) holographic networks, (5) logical classification, and (6) layered neural networks. The classification models were evaluated with respect to misclassification rate, predictive validity, verification cost, and achieved quality. Results indicate that no model satisfied the classification criterion in discriminating the software defects. Researchers in [10], built a model to predict defect-prone software modules using support vector machine (SVM) using four NASA datasets, namely PC1, CM1, KC1, and KC3. In addition, the performance was compared with eight statistical and machine learning models. The researchers concluded that SVM outperformed the other techniques. In a study by [11], researchers performed a comparative experimental study of the effectiveness of artificial neural networks

(ANNs) and SVMs in the dataset obtained from NASA dataset. The performance was compared with the SVM Gaussian kernel function and indicated that SVM performed better than ANN.
In a study by [12], researchers presented an application back-propagation neural network (BPNN) based on three cost-sensitive boosting algorithms and tested it over four NASA datasets. The designs of two of the BPNNs were based on weight updating, while the other was based on threshold. The empirical results indicated that the threshold-based feed forward neural network performed better than other methods particularly for object-oriented software modules. In a study by [13], researchers presented an experiment for comparing several statistical and machine learning techniques using AR1 and AR6 public domain datasets to find the relationship between the static code metrics and the fault proneness of a module. Performance was evaluated using area under the curve (AUC) values. Results revealed that decision tree achieved better results than other applied techniques.

Researchers in a study by [14], detected faulty components by applying the radial basis function neural network with novel adaptive dimensional biogeography-based optimization model to investigate five NASA datasets from the PROMISE repository. Results were satisfactory compared to conventional models. Researchers in a study done in [15], constructed a graphical user interface (GUI) tool with the help of MATLAB for software defect prediction based on the Bayesian regularization neural network (BRNN) technique, which led to a reduction in the software cost by limiting the squared errors and weights. The performance of the technique was compared with Levenberg Marquardt and back propagation neural network algorithms, and according to the results, the BRNN performed better.

Feature selection has a major role in detecting the most significant attributes and consequently improving software defect prediction to alleviate the high dimensionality concerns. Several studies have explored the effectiveness of feature selection methods on the performance of defect prediction models [16–18]. Researchers [16], detected faulty components by applying four feature ranking and two wrapper methods on the code change-based bug prediction over eleven software projects. They found that feature selection step led to an improvement in classification speed and scalability, and optimally reasonable results were obtained with only 3% of the total feature set. Researchers [17] built a prediction model based on several feature selection models. They applied seven feature ranking and two wrapper methods and one embedded method. They applied several changes and source code metrics, and they tested this system over the noisy NASA dataset. Results indicate no significant differences on defect prediction over the datasets were found. In [18], researchers combined the output of six feature-ranking based methods and thoroughly investigated the performance of two ensemble methods over three datasets. The results on NASA dataset showed that differences between ranking method, classifier and software dataset significantly impacted the classification outcomes, and the ensemble method improved the fault prediction results.

In a study by [18], researchers combined the output of six feature-ranking based methods and thoroughly investigated the performance of two ensemble methods over three datasets. The results on the NASA dataset showed that differences between ranking method, classifier, and software dataset had a significant impact on classification outcomes, and the ensemble method led to an improvement in fault prediction results.

3. MATERIALS AND METHODS

The aim of this study was to suggest a feature selection-based clustering model for predicting software defects. This section describes the methodology used in this study. The proposed framework consists of three stages: (1) feature selection, (2) clustering, and (3) decision making. An experimental framework was implemented in two dimensions, and initially, datasets were

directly fed into classifiers before the feature selection stage. Yet, in the second dimension, the datasets started with the feature selection stage. We employed several classifiers to discover the best algorithms to resume feature selection step.

3.1. Datasets

The NASA benchmark datasets have been extensively used in software defect prediction. Each dataset implements a NASA-based software system, which includes different metrics that are closely related to software quality. The datasets contain records classified by a target class and consists of one value from two of the following: 'Y' or 'N'. 'Y' indicates that the particular record (software module) is defective, and 'N' indicates that it is non-defective. Two cleaned version of NASA datasets are provided by [19]. The first version is called D' and includes duplicate and inconsistent instances, whereas the second version D", does not include duplicate and inconsistent instances. We used the D'' version, available for download at [20]. This cleaned version has already been adapted in the literature [10–12,14,17,18]. Many researchers agree on considering D'' superior to its predecessor dataset as it does not contain redundant records and has a lower complexity level of data. Seven cleaned NASA datasets were used in this research for experiment. The chosen datasets include KC1, KC3, MC2, MW1, PC1, PC3, and PC4 (Tables 1 and 2).

Table 1. National Aeronautics and Space Administration (NASA) cleaned dataset D" details

Dataset	**No. Features**	**No. of records**	**Defective**	**Non-Defective**	**Defective (%)**
KC1	22	1162	294	868	25.3
KC3	40	194	36	158	18.5
MC2	40	124	44	80	35.4
MW1	38	250	25	225	10
PC1	41	1107	76	1031	6
PC3	41	1563	160	1403	10.2
PC4	38	1270	176	1094	13.8

Table 2. Features of the D'' dataset

No.	**Feature**	**KC1**	**KC3**	**MC2**	**MW1**	**PC1**	**PC3**	**PC4**
1.	LOC_BLANK	✓	✓	✓	✓	✓	✓	✓
2.	BRANCH_COUNT	✓	✓	✓	✓	✓	✓	✓
3.	CALL_PAIRS		✓	✓	✓	✓	✓	✓
4.	LOC_CODE_AND_COMMENT	✓	✓	✓	✓	✓	✓	✓
5.	LOC_COMMENTS	✓	✓	✓	✓	✓	✓	✓
6.	CONDITION_COUNT		✓	✓	✓	✓	✓	✓
7.	CYCLOMATIC_COMPLEXITY	✓	✓	✓	✓	✓	✓	✓
8.	CYCLOMATIC_DENSITY		✓	✓	✓	✓	✓	✓
9.	DECISION_COUNT		✓	✓	✓	✓	✓	✓
10.	DECISION_DENSITY		✓	✓	✓	✓	✓	
11.	DESIGN_COMPLEXITY	✓	✓	✓	✓	✓	✓	✓
12.	DESIGN_DENSITY		✓	✓	✓	✓	✓	✓
13.	EDGE_COUNT	✓	✓	✓	✓	✓	✓	✓
14.	ESSENTIAL_COMPLEXITY	✓	✓	✓	✓	✓	✓	✓
15.	ESSENTIAL_DENSITY		✓	✓	✓	✓	✓	✓

No.	Feature	KC1	KC3	MC2	MW1	PC1	PC3	PC4
16.	LOC_EXECUTABLE	✓	✓	✓	✓	✓	✓	✓
17.	PARAMETER_COUNT		✓	✓	✓	✓	✓	✓
18.	GLOBAL_DATA_COMPLEXITY		✓	✓				✓
19.	GLOBAL_DATA_DENSITY		✓	✓				✓
20.	HALSTEAD_CONTENT	✓	✓	✓	✓	✓	✓	✓
21.	HALSTEAD_DIFFICULTY	✓	✓	✓	✓	✓	✓	✓
22.	HALSTEAD_EFFORT	✓	✓	✓	✓	✓	✓	✓
23.	HALSTEAD_ERROR_EST	✓	✓	✓	✓	✓	✓	✓
24.	HALSTEAD_LENGTH	✓	✓	✓	✓	✓	✓	✓
25.	HALSTEAD_LEVEL	✓	✓	✓	✓	✓	✓	✓
26.	HALSTEAD_PROG_TIME	✓	✓	✓	✓	✓	✓	✓
27.	HALSTEAD_VOLUME	✓	✓	✓	✓	✓	✓	✓
28.	MAINTENANCE_SEVERITY		✓	✓	✓	✓	✓	✓
29.	MODIFIED_CONDITION_COUNT		✓	✓	✓	✓	✓	✓
30.	MULTIPLE_CONDITION_COUNT		✓	✓	✓	✓	✓	✓
31.	NODE_COUNT		✓	✓	✓	✓	✓	✓
32.	NORMALIZED_CYLOMATIC_COMPLEXITY		✓	✓	✓	✓	✓	✓
33.	NUM_OPERANDS	✓	✓	✓	✓	✓	✓	✓
34.	NUM_OPERATORS	✓	✓	✓	✓	✓	✓	✓
35.	NUM_UNIQUE_OPERANDS	✓	✓	✓	✓	✓	✓	✓
36.	NUM_UNIQUE_OPERATORS	✓	✓	✓	✓	✓	✓	✓
37.	NUMBER_OF_LINES	✓	✓	✓	✓	✓	✓	✓
38.	PERCENT_COMMENTS		✓	✓	✓	✓	✓	✓
39.	LOC_TOTAL	✓	✓	✓	✓	✓	✓	✓

3.2. Classification via Clustering Techniques

Unsupervised clustering algorithms sorts observations into similar sets or groups according to the values of their features even when class labels are absent. We will investigate three types of clustering techniques that work under different assumptions: (1) X-means, (2) farthest first, and (3) self-organizing map (SOM). The details of these techniques are described in the following sections.

3.2.1. X-Means Clustering

The popular K-means clustering algorism has many limitations as it tends to scale poorly and it is computationally heavy. Moreover, the solution depends on the initial positions of the cluster centers and the number of clusters K has to be delivered by the user as it can only find linearly separable clusters. In an attempt to solve these issues, Pelleg and Moore introduced a new and efficient algorithm [21]. Their innovations included two ways of exploiting cached sufficient statistics and an efficient test that selects the most promising subset of classes for refinement. The X-means algorithm searches the space of cluster locations and number of clusters based on the outcome of the Bayesian Information or the Akaike Information Criterion. Experimental results showed that the X-means algorithm provides a fast and effective way to cluster unstructured data. A generalized X-means algorithm procedure is described in the next section.

Step 1: Initialize K= K_{min}

Step 2: Rub K-Means algorithm

Step 3: For k=1,..., k: Replace each centroid by two centroids $\mu_1$and μ_2

Step 4: Run K-means algorithm with K=2 over the cluster K. Replace or retain each centroid based on the model selection

Step 5: If convergence condition is not satisfied, go to Step 2, Otherwise Stop

Algorithm 1: Procedure for X-Means Algorithm [21]

3.2.2. Farthest First Clustering Technique

Farthest First is a unique clustering algorithm that combines both hierarchical and distance-based clustering. This algorithm builds a hierarchy of clusters using an agglomerative hierarchical clustering method with a distance measurement criterion that is similar to the one used by K-Means algorithm. Farthest First assigns a centre to a random point, and then computes the k most distant points [22].

This algorithm starts with randomly choosing an instance as a cluster centroid, assigning the objects in the cluster, and then computing the distance between each remaining instance and its nearest centroid. The algorithm takes an arbitrary cluster centroid and calculates the distance of one centroid from other as the maximum cluster assignment using Farthest First. When outlier detection is performed on the dataset, objects that are outliers can be detected. This places the cluster centre at the point further from the present cluster. The process is repeated until the number of clusters is greater than a predetermined threshold value [23]. This clustering algorithm can ultimately speed up the clustering process as this algorithm requires fewer data relocations and adjustments.

3.2.3. SOM-Based Cluster Analysis Technique

The SOM has received increasing attention since it was proposed by Kohonen in 1990 [24]. SOM is a competitive unsupervised neural network that consists of both input and output layers with numerous neurons. The SOM methodology has been used in data analysis as tool for resolving and visualizing nonlinear relationships in complex data, topology-based cluster analysis, vector quantization, and projection of multidimensional data. It achieves clustering through dimensionality reduction using topographically ordered nodes that represent the distribution characteristics of the input samples.

If one assumes that ‘m’ cluster units exist, which are arranged in a one- or two-dimensional (1D/2D) array, and the input signals are n-tuples [25]. The cluster unit whose weight vector matches the input pattern is approximately selected as winner. The input vector is compared with the target vector and if they differ, the weights of the network are altered slightly to reduce the error in the output. The Euclidean distance (D) is computed between the input vector and weight vector$_{wij}$ and is represented in Equation 1.

$$D(i) = \sum \left(w_{ij} - x_i\right)^2 \quad i = 1 \quad to \quad n \quad and \quad j = 1 \quad to \quad m \qquad 1$$

The smallest distance was computed, and the weights were updated using Equation 2.

$$w_{ij(new)} = w_{ij(old)} + \alpha \left[x_i - w_{ij(old)} \right] \quad 2$$

in which x denotes the input vector, i and j indicates index values. This process is repeated many times and with many sets of vector pairs until the network gives the desired output.

4. RESULTS AND DISCUSSION

In our work for the comparison of various clustering algorithms we used Weka platform [26]. Weka is one of data-mining tool which contains a collection of machine learning algorithms. In the proposed approach we have used WEKA with 10-foldcross validation method to evaluate data and compare results. The performance of the proposed model is measured using sensitivity, accuracy, precision, and f-measures [27].

The next step includes using several swarm-based algorithms and performs further tests on the NASA dataset by combining it with several clustering algorithms. Our main objectives were to improve the new method, test it on a relevant dataset, and compare this new method with another feature selection. Dataset selection was the first stage of proposed framework. The performance was compared using seven widely used clustering algorithms, namely K-MEANS, expectation maximization (EM), density-based (DB), Farthest First, SOM, learning vector quantization (LVQ), neural network, and X-Means clustering algorithms at the first-stage. The results of the proposed framework were evaluated through sensitivity, accuracy, precision, and f-measure measures. The proposed framework was implemented on seven cleaned NASA Datasets (D''). The results are described in Table 3 to 14. Highest scores are highlighted in bold for easy identification.

Tables 3–6 present the prediction performance results for each of the classification via clustering algorithms before the feature selection step. The first noteworthy observation in Table 3 was that Farthest First algorithm outperformed other algorithms in terms of sensitivity, accuracy, and f-measures readings. But regarding precision, SOM scored a better result in with KC1, while X-Means outperformed the others in KC3 dataset. Results of MC2 and MW1 datasets are reflected in Table 4. Regarding sensitivity and f-measures, Farthest First outperformed the other classifiers, while we noticed that the performance accuracy of all the algorithms and precision in most of the classifiers was fairly consistent. The results of PC1, PC3, and PC4 datasets are given in Tables 5 and 6. It is clear that Farthest First algorithm outperformed other algorithms in sensitivity, accuracy, and f-measures readings. However, regarding precision, EM scored a better result in all the datasets. Eventually, we noticed that the best results were related with X-MEANS, Farthest First, and SOM.

Table 3. Clustering results for KC1 and KC3datasets

	KC1				**KC3**			
Algorithm	sensitivity	accuracy	precision	f-measures	sensitivity	accuracy	precision	f-measures
EM	0.632	0.616	0.838	0.721	0.576	0.550	0.826	0.679
DB	0.831	0.721	0.803	0.817	0.797	0.727	0.857	0.826
K-MEANS	0.880	0.740	0.794	0.835	0.861	0.758	0.845	0.853
X-MEANS	0.629	0.615	0.848	0.722	0.802	0.733	**0.861**	0.830
FarthestFirst	**0.998**	**0.748**	0.749	**0.856**	**0.962**	**0.814**	0.835	**0.894**
SOM	0.579	0.583	**0.851**	0.689	0.700	0.653	0.848	0.767
LVQ	0.925	0.741	0.773	0.842	0.924	0.799	0.844	0.882

Table 4. Clustering results for MC2 and MW1 datasets

	MC2				**MW1**			
Algorithm	sensitivity	accuracy	precision	f-measures	sensitivity	accuracy	precision	f-measures
EM	0.440	0.477	**0.786**	0.564	0.744	0.738	0.944	0.832
DB	0.925	**0.694**	0.698	0.796	0.777	0.755	0.941	0.851
K-MEANS	0.950	0.685	0.685	0.796	0.830	0.802	0.944	0.883
X-MEANS	0.825	0.667	0.725	0.772	0.718	0.712	0.955	0.820
Farthest First	**1.000**	0.685	0.672	**0.804**	**0.969**	**0.884**	0.908	**0.938**
SOM	0.733	0.663	0.759	0.746	0.772	0.771	**0.960**	0.856
LVQ	0.950	0.685	0.685	0.796	0.858	0.816	0.932	0.894

Table 5. Clustering results for PC1 and PC3 datasets

	PC1				PC3			
Algorithm	sensitivity	accuracy	precision	f-measures	sensitivity	accuracy	precision	f-measures
EM	0.700	0.708	**0.981**	0.817	0.651	0.673	**0.969**	0.779
DB	0.686	0.669	0.937	0.792	0.637	0.609	0.884	0.741
K-MEANS	0.684	0.654	0.918	0.784	0.693	0.644	0.874	0.773
X-MEANS	0.683	0.690	0.968	0.801	0.649	0.666	0.956	0.773
Farthest First	**0.998**	**0.920**	0.922	**0.958**	**1.000**	**0.876**	0.876	**0.934**
SOM	0.804	0.770	0.931	0.863	0.620	0.640	0.941	0.748
LVQ	0.998	0.920	0.922	0.958	1.000	0.876	0.876	0.934

Table 6.Clustering results for PC4 dataset

Algorithm	sensitivity	accuracy	precision	f-measures
EM	0.719	0.728	**0.949**	0.818
DB	0.671	0.650	0.896	0.767
K-MEANS	0.671	0.633	0.874	0.759
X-MEANS	0.703	0.640	0.833	0.763
Farthest First	**0.999**	**0.862**	0.863	**0.926**
SOM	0.829	0.731	0.837	0.833
LVQ	0.989	0.856	0.863	0.922

In the second step, we chose to modify the best-performing clustering algorithms to reach a higher prediction reading using four popular swarm intelligence algorithms described in the literature, specifically, PSO, Cuckoo, Bat, and GWO algorithms. The primary goal was to understand the trends and the relationship in their performance. The results of the selected optimization algorithms are reported in Table 7. For each of the selected datasets, we obtained four distinctive subsets. In general, we observed that the bio-based feature selection techniques helped remarkably in reducing the features numbers. For example, we noticed that feature selection techniques led to a reduction in the features in the KC1 dataset from 22 to 7–14 and in the PC1 dataset from 41 to 12–14 features. After constructing feature sets using the bio-based feature selection techniques, the techniques were individually applied to compare their performances. These features were then fed into X-MEANS, Farthest First, and SOM classifiers.

Table 7.Selected features of bio-based feature algorithms

Algorithm	**KC1**	**KC3**	**MC2**	**MW1**	**PC1**	**PC3**	**PC4**
Bat	LOC_BLANK BRANCH_COUNT LOC_COMMENTSHALSTEAD_CONTENT HALSTEAD_DIFFICULTY HALSTEAD_LENGTH HALSTEAD_LEVEL NUM_OPERANDS NUM_UNIQUE_OPERATORS Total = 9	LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT NORMALIZED_CYLOMATIC_COMPLEXITY NUMBER_OF_LINES PERCENT_COMMENTS Total =6	LOC_BLANK CALL_PAIRS LOC_COMMENTS CYCLOMATIC_COMPLEXITY DESIGN_COMPLEXITY EDGE_COUNT ESSENTIAL_COMPLEXITY ESSENTIAL_DENSITY GLOBAL_DATA_COMPLEXITY GLOBAL_DATA_DENSITY HALSTEAD_DIFFICULTY HALSTEAD_EFFORT NODE_COUNT Total = 13	LOC_BLANK CALL_PAIRS LOC_COMMENTS DESIGN_COMPLEXITY EDGE_COUNT ESSENTIAL_COMPLEXITY NODE_COUNT NUM_UNIQUE_OPERANDS NUMBER_OF_LINES Total =9	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_COMPLEXITY CYCLOMATIC_DENSITY PARAMETER_COUNT HALSTEAD_CONTENT HALSTEAD_DIFFICULTY HALSTEAD_LENGTH MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERATORS NUMBER_OF_LINES PERCENT_COMMENTS LOC_TOTAL Total =15	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_DENSITY HALSTEAD_CONTENT HALSTEAD_LENGTH NORMALIZED_CYLOMATIC_COMPLEXITY NUM_OPERANDS NUM_UNIQUE_OPERANDS NUMBER_OF_LINES PERCENT_COMMENTS Total =11	LOC_BLANK LOC_CODE_AND_COMMENT CONDITION_COUNT ESSENTIAL_COMPLEXITY HALSTEAD_CONTENT MULTIPLE_CONDITION_COUNT NORMALIZED_CYLOMATIC_COMPLEXITY PERCENT_COMMENTS Total =8

Algorithm	**KC1**	**KC3**	**MC2**	**MW1**	**PC1**	**PC3**	**PC4**
Cuckoo	LOC_COMMENTS HALSTEAD_CONTENT HALSTEAD_DIFFICULTY HALSTEAD_LEVEL HALSTEAD_VOLUME NUM_UNIQUE_OPERANDS NUM_UNIQUE_OPERATORS Total =7	LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT NORMALIZED_CYLOMATIC_COMPLEXITY PERCENT_COMMENTS Total =5	LOC_BLANK CALL_PAIRS LOC_COMMENTS CYCLOMATIC_COMPLEXITY DESIGN_COMPLEXITY EDGE_COUNT GLOBAL_DATA_COMPLEXITY GLOBAL_DATA_DENSITY HALSTEAD_DIFFICULTY HALSTEAD_EFFORT MAINTENANCE_SEVERITY Total = 11	LOC_BLANK CALL_PAIRS CYCLOMATIC_COMPLEXITY EDGE_COUNT ESSENTIAL_COMPLEXITY HALSTEAD_CONTENT MODIFIED_CONDITION_COUNT NODE_COUNT NUM_UNIQUE_OPERANDS NUMBER_OF_LINES Total =10	LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_DENSITY PARAMETER_COUNT HALSTEAD_CONTENT MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS NUM_UNIQUE_OPERATORS NUMBER_OF_LINES PERCENT_COMMENTS LOC_TOTAL Total = 14	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS HALSTEAD_CONTENT HALSTEAD_LENGTH NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS NUM_UNIQUE_OPERATORS NUMBER_OF_LINES PERCENT_COMMENTS Total =10	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_COMPLEXITY CYCLOMATIC_DENSITY PARAMETER_COUNT MULTIPLE_CONDITION_COUNT NUMBER_OF_LINES PERCENT_COMMENTS Total = 9

Algorithm	**KC1**	**KC3**	**MC2**	**MW1**	**PC1**	**PC3**	**PC4**
PSO	LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_DENSITY PARAMETER_COUNT HALSTEAD_CONTENT HALSTEAD_EFFORT MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS NUMBER_OF_LINES PERCENT_COMMENTS LOC_TOTAL Total =14	BRANCH_COUNT LOC_CODE_AND_COMMENT NORMALIZED_CYLOMATIC_COMPLEXITY Total =3	LOC_BLANK LOC_COMMENTS DESIGN_COMPLEXITY EDGE_COUNT GLOBAL_DATA_COMPLEXITY GLOBAL_DATA_DENSITY HALSTEAD_DIFFICULTY HALSTEAD_EFFORT HALSTEAD_PROG_TIME MAINTENANCE_SEVERITY Total =10	LOC_BLANK LOC_COMMENTS CONDITION_COUNT HALSTEAD_CONTENT NODE_COUNT NUM_UNIQUE_OPERANDS Total =6	LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_DENSITY PARAMETER_COUNT HALSTEAD_CONTENT HALSTEAD_EFFORT MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS NUMBER_OF_LINES PERCENT_COMMENTS LOC_TOTAL Total =14	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS DESIGN_COMPLEXITY HALSTEAD_CONTENT HALSTEAD_LENGTH MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_OPERANDS NUM_UNIQUE_OPERANDS PERCENT_COMMENTS Total =11	LOC_BLANK LOC_CODE_AND_COMMENT CONDITION_COUNT CYCLOMATIC_DENSITY DESIGN_DENSITY ESSENTIAL_COMPLEXITY HALSTEAD_CONTENT NORMALIZED_CYLOMATIC_COMPLEXITY PERCENT_COMMENTS Total =9

Algorithm	**KC1**	**KC3**	**MC2**	**MW1**	**PC1**	**PC3**	**PC4**
GWO	LOC_BLANK LOC_COMMENTS HALSTEAD_CONTENT HALSTEAD_DIFFICULTY HALSTEAD_LEVEL HALSTEAD_VOLUME NUM_UNIQUE_OPERANDS NUM_UNIQUE_OPERATORS Total =8	BRANCH_COUNT LOC_CODE_AND_COMMENT NORMALIZED_CYLOMATIC_COMPLEXITY Total =3	LOC_BLANK CALL_PAIRS LOC_COMMENTS CYCLOMATIC_COMPLEXITY ESSENTIAL_DENSITY GLOBAL_DATA_COMPLEXITY GLOBAL_DATA_DENSITY HALSTEAD_DIFFICULTY HALSTEAD_EFFORT NODE_COUNT Total =10	LOC_BLANK LOC_COMMENTS EDGE_COUNT HALSTEAD_CONTENT MODIFIED_CONDITION_COUNT NODE_COUNT NUM_UNIQUE_OPERANDS Total =7	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS CYCLOMATIC_DENSITY PARAMETER_COUNT HALSTEAD_CONTENT NODE_COUNT NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS NUMBER_OF_LINES PERCENT_COMMENTS LOC_TOTAL Total =12	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS HALSTEAD_CONTENT HALSTEAD_LENGTH MAINTENANCE_SEVERITY NORMALIZED_CYLOMATIC_COMPLEXITY NUM_UNIQUE_OPERANDS PERCENT_COMMENTS Total =9	LOC_CODE_AND_COMMENT PARAMETER_COUNT MULTIPLE_CONDITION_COUNT PERCENT_COMMENTS Total =4

Tables 8–11 present the performance of the clustering scheme based on the selected datasets. According to Table 8, highest sensitivity scores were obtained using KC1 feature set construction methods for Bat and PSO, with using 41% and 64% of the original feature size. KC3 and PC4 recorded the highest sensitivity scores using Cuckoo feature set method, with nearly 13% and 24% of the feature set respectively. The overall highest sensitivity scores were obtained when MW1, PC1 and PC3 applied the features extracted from Bat, Cuckoo and GWO. Worth noting that the highest scores were obtained with the Farthest First scheme.

Table 8.Sensitivity results of each clustering algorithm

Algorithm		KC1	KC3	MC2	MW1	PC1	PC3	PC4
Bat	X-MEANS	0.625	0.797	0.763	0.584	0.584	0.649	0.696
	Farthest First	**0.993**	0.949	0.975	**1.000**	**1.000**	**1.000**	0.988
	SOM	0.647	0.708	0.793	0.692	0.692	0.620	0.676
Cuckoo	X-MEANS	0.685	0.738	0.714	0.818	0.610	0.661	0.658
	Farthest First	0.985	**0.975**	0.975	0.960	1.000	1.000	0.992
	SOM	0.640	0.650	0.723	0.884	0.700	0.667	0.659
PSO	X-MEANS	0.714	0.821	0.714	0.775	0.626	0.691	0.691
	Farthest First	**0.993**	0.962	**1.000**	0.978	1.000	0.960	0.960
	SOM	0.704	0.568	0.810	0.901	0.712	0.712	0.712
GWO	X-MEANS	0.693	0.821	0.746	0.806	0.714	0.654	0.657
	Farthest First	0.984	0.962	0.975	0.969	0.993	1.000	**0.991**
	SOM	0.652	0.568	0.793	0.914	0.704	0.707	0.681

Tables 9 present the performance of the clustering scheme based on the selected datasets according to accuracy scores. The highest accuracy scores were obtained using MW1 and PC1 feature set construction methods for Bat, Cuckoo, and PSO with 24%, 34%, and 34% of the original feature size, respectively. PC3 yielded good accuracy scores using the Bat, Cuckoo, and GWO feature set methods with nearly 28%, 24%, and 22% of the feature set respectively. PC4 and KC3 scored 86% and 81% based on GWO and Cuckoo features, respectively, while KC1 obtained the highest scores with features extracted from Bat and PSO. Most of the best results were obtained using Farthest First algorithm, except the case in which MC2 scored the highest accuracy using the SOM and GWO feature sets.

Table 9. Accuracy results of each clustering algorithm

Algorithm		KC1	KC3	MC2	MW1	PC1	PC3	PC4
Bat	X-MEANS	0.582	0.737	0.651	0.615	0.615	0.666	0.692
	Farthest First	**0.748**	0.804	0.677	**0.922**	**0.922**	**0.876**	0.853
	SOM	0.612	0.693	0.718	0.697	0.697	0.640	0.691
Cuckoo	X-MEANS	0.636	0.671	0.607	0.809	0.633	0.666	0.656
	Farthest First	0.742	**0.808**	0.677	0.900	**0.922**	**0.876**	0.856
	SOM	0.608	0.617	0.654	0.868	0.706	0.662	0.651
PSO	X-MEANS	0.643	0.742	0.607	0.772	0.634	0.709	0.709
	Farthest First	**0.748**	0.794	0.694	0.916	**0.922**	0.838	0.838
	SOM	0.655	0.580	0.651	0.882	0.717	0.719	0.719
GWO	X-MEANS	0.635	0.742	0.667	0.798	0.643	0.663	0.665
	Farthest First	0.745	0.794	0.677	0.908	0.748	**0.876**	**0.865**
	SOM	0.620	0.580	**0.721**	0.894	0.655	0.700	0.697

According to Table 10, the highest precision scores were obtained using MW1 and PC1feature set construction methods for Bat method with 24% and 37% of the original feature size, respectively. PC3 and PC4 give the second highest precision scores using Bat and PSO feature set methods with nearly 28% and 27% of the feature sets, respectively. The highest precision scores of KC3, MC2, and KC1 datasets were obtained when the features extracted from Bat, Cuckoo and PSO were applied. It should be noted that the highest scores were obtained with the SOM and X-MEANS schemes.

Table 10. Precision results of each clustering algorithm

Algorithm		KC1	KC3	MC2	MW1	PC1	PC3	PC4
Bat	X-MEANS	0.775	0.869	0.744	**0.978**	**0.978**	0.956	0.926
	Farthest First	0.751	0.833	0.672	0.922	0.922	0.876	0.861
	SOM	0.788	**0.911**	0.793	0.963	0.963	0.941	**0.968**
Cuckoo	X-MEANS	0.791	0.849	0.714	0.959	0.966	0.935	0.904
	Farthest First	0.749	0.823	0.672	0.931	0.922	0.876	0.862
	SOM	0.793	0.838	**0.825**	0.965	0.967	0.930	0.887
PSO	X-MEANS	0.792	0.855	0.714	0.959	0.936	**0.966**	0.966
	Farthest First	0.750	0.817	0.678	0.932	0.922	0.867	0.867
	SOM	**0.814**	0.888	0.723	0.965	0.967	0.962	0.962
GWO	X-MEANS	0.785	0.855	0.758	0.959	0.792	0.936	0.923
	Farthest First	0.752	0.817	0.672	0.932	0.750	0.876	0.870
	SOM	0.799	0.888	0.793	0.965	0.814	0.935	0.954

Tables 11 presents the performance of the clustering scheme based on the selected datasets according to f-measures scores. The best scores were obtained using MW1 and PC1 feature set

construction methods for Bat, Cuckoo, and PSO with 24% to 37% of the original feature size. PC3 yielded f-measures scores of 93.4% using Bat, Cuckoo, and PSO, while PC4 yielded 92.6% based on GWO features, which represents 11% of the original feature size. Overall, the highest f-measures scores were obtained when KC3, KC1, and MC2 was presented using the features extracted from Bat, Cuckoo, and GWO. It is worth noting that the highest scores were obtained mostly with the Farthest First scheme.

Table 11. f-measures results of each clustering algorithm

Algorithm		KC1	KC3	MC2	MW1	PC1	PC3	PC4
Bat	X-MEANS	0.692	0.832	0.753	0.732	0.732	0.773	0.795
	Farthest First	**0.855**	0.888	0.796	**0.959**	**0.959**	**0.934**	0.920
	SOM	0.710	0.797	0.793	0.806	0.806	0.748	0.796
Cuckoo	X-MEANS	0.734	0.789	0.714	0.883	0.748	0.775	0.762
	Farthest First	0.851	**0.892**	0.796	0.945	**0.959**	**0.934**	0.922
	SOM	0.708	0.732	0.770	0.923	0.812	0.777	0.756
PSO	X-MEANS	0.751	0.838	0.714	0.857	0.750	0.805	0.805
	Farthest First	**0.855**	0.884	**0.808**	0.954	**0.959**	0.911	0.911
	SOM	0.755	0.693	0.764	0.932	0.820	0.818	0.818
GWO	X-MEANS	0.736	0.838	0.752	0.875	0.751	0.770	0.767
	Farthest First	0.852	0.884	0.796	0.950	0.855	**0.934**	**0.926**
	SOM	0.718	0.693	0.793	0.939	0.755	0.806	0.795

In Figures 1 and 2, consistency has also been validated by drawing boxplots diagrams based on accuracy rates of the clustering results versus the accuracy rates of the original dataset when addressing all datasets. Figure 1 shows the standardized boxplots of accuracy values for each dataset on all the clustering methods, whereas Figure 2 shows the boxplots of accuracy values in the second phase after applying bio-based feature selection techniques. It can be observed that the superiority of the proposed technique is evident on the majority of data sets as compared to original dataset. So, the suggested clustering technique is suitable for analysing our experimental results on the clean NASA dataset. In almost all cases, the proposed technique generates more consistent and convincing results as compared to other algorithms.

Having insight into the clustering results before and after feature selection, it is clear that most of the performance metrics performed better as the dimensionality of features decreased considerably. Performance results reveals that the Bat and Cuckoo algorithms yielded the best performance on the datasets. It is evident that the Farthest First clustering algorithm is good for predicting software faultiness, while Bat and Cuckoo were useful in comparison to all other metaheuristic algorithms. Although the class imbalance issue, which is the main reason of biased performance in any classification problem, was not considered in this study we suggest including resampling technique in any future framework system.

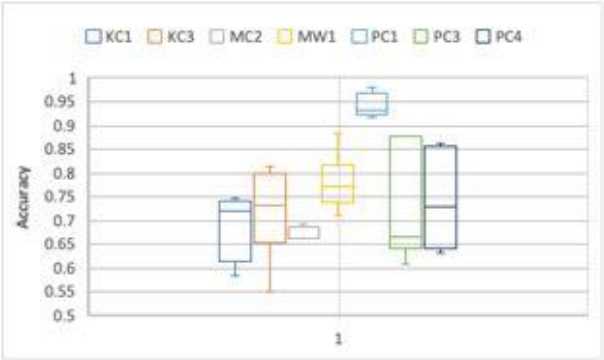

Figure 1. Boxplots of accuracy rates of clustering on original datasets

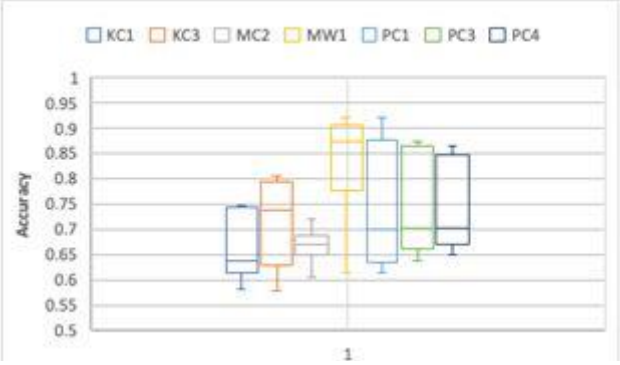

Figure 2. Boxplots of accuracy rates of clustering using the proposed technique

5. CONCLUSION

The main contribution of this article is the implementation of a bio-inspired feature selection-based clustering framework for software fault prediction. The proposed framework consisted of three stages, including feature selection, clustering, and evaluation. Two different dimensions were used in the framework, one with and one without feature selection. This research compared the ability of PSO, Bat, Cuckoo, and GWO bio-inspired algorithms. Several clustering algorithms were applied: (1) X-means, (2) Farthest First, and (3) SOM. For experiments, seven cleaned publicly available NASA datasets were used. Results showed the effectiveness of Farthest First clustering algorithm in predicting software faultiness, and Bat and Cuckoo were useful in comparison to all other metaheuristic algorithms.

REFERENCES

[1] E. Erturk and E. Akcapinar, A comparison of some soft computing methods for software fault prediction, Expert Syst. Appl., vol. 42, no. 4, pp. 1872–1879, 2015.
[2] Y. Ma, G. Luo, X. Zeng, and A. Chen, Transfer learning for cross company software defect prediction, Inf. Softw. Technol., vol. 54, no. 3, Mar. 2012.
[3] R. Malhotra, Empirical research in software engineering: concepts, analysis, and applications. Chapman and Hall/CRC, 2016.
[4] P. Goodman, 1993, Practical Implementation of Software Metrics, McGraw Hill, London.
[5] G. Czibula, Z. Marian, and I. G. Czibula, "Software defect prediction using relational association rule mining," Information Sciences, vol. 264, pp. 260-278, 2014.
[6] I., Lee, Y.J. Shin, Machine learning for enterprises: Applications, algorithm selection, and challenges. Business Horizons, 63(2), 157e170, 2020.

[7] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim. Reducing features to improve code change-based bug prediction. IEEE Transactions on Software Engineering, 39(4): 552-569, 2013.
[8] A. Balogun, M. Mabayoje, S. Salihu, and S. Arinze, "Enhanced Classification Via Clustering Using Decision Tree for Feature Selection," International Journal of Applied Information Systems (IJAIS), vol. 9, no. 6, pp. 11-16, 2015.
[9] F. Lanubile, A. Lonigro, and G. Vissagio, Comparing models for identifying fault-prone software components. Seke, no. July, pp. 312– 319, 1995.
[10] K. O. Elish and M. O. Elish, Predicting defect-prone software modules using support vector machines, J. Syst. Softw., vol. 81, no. 5, pp. 649– 660, 2008.
[11] I. Gondra, Applying machine learning to software fault-proneness prediction, J. Syst. Softw., vol. 81, no. 2, pp. 186–195, 2008.
[12] J. Zheng, Cost-sensitive boosting neural networks for software defect prediction, Expert Syst. Appl., vol. 37, no. 6, pp. 4537–4543, 2010.
[13] R. Malhotra, Comparative analysis of statistical and machine learning methods for predicting faulty modules, Appl. Soft Comput. J., vol. 21, pp. 286–297, 2014.
[14] P. Kumudha and R. Venkatesan, Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction, Sci. World J., vol. 2016, 2016.
[15] R. Mahajan, S. K. Gupta, and R. K. Bedi, Design of software fault prediction model using BR technique, in Procedia Computer Science, vol. 46, no. Icict 2014, pp. 849–858, 2015.
[16] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim. Reducing features to improve code change-based bug prediction. IEEE Transactions on Software Engineering, 39(4): 552-569, 2013.
[17] K. Muthukumaran, A. Rallapalli, and N. L. Murthy. Impact of feature selection techniques on bug prediction models. In Proceedings of the 8th India Software Engineering Conference(ISEC). ACM, 120-129, 2015.
[18] H. Wang, T. M. Khoshgoftaar, J. V. Hulse, and K. Gao. Metric selection for software defect prediction. International Journal of Software Engineering and Knowledge Engineering, 21(02): 237-257, 2011.
[19] M. Shepperd, Q. Song, Z. Sun and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," IEEE Trans. Softw. Eng., vol. 39, pp. 1208–1215, 2013.
[20] NASA Defect Dataset, [Online]. Available: https://github.com/klainfo/NASADefectDataset. [Accessed: 28-September-2020].
[21] D.PELLEG, A.MOORE, X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proceedings 17th ICML, Stanford University, 2000.
[22] S. D.Hochbaum and B. D. Shmoys, A Best Possible Heuristic for the k-Center Problem, Mathematics of Operational Research, 10(2): pp. 180-184,1985.
[23] S. Dasgupta and P. M. Long. Performance guarantees for hierarchical clustering, Journal of Computer and System Sciences, 70(4):555-569, 2005.
[24] T. Kohonen, The self-organizing map, Neurocomputing, vol. 21, no. 1–3, pp. 1–6, 1998.
[25] K. Melody, Extending the Kohonen self-organizing map networks for clustering analysis. Comput Stat Data Anal 2001; 38: 161-180, 2001.
[26] M., Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: An update, ACM SIGKDD Explor. News, 11, 10–18, 2009.
[27] X.Y. Liu, Q.Q. Li, Z.H. Zhou, Learning imbalanced multiclass data with optimal dichotomy weights, Proceeding of IEEE 13th International Conference on Data Mining, Dallas, TX, USA, pp. 478–487, 2013.