Efficient Software Fault Prediction and Classification using Evolutionary Cost-Sensitive Deep Belief Network

HM Premalatha[1], CV Srikrishna[2]
[1]PES University, India
[2]PES Institute of Technology, India
(E-mail: premalathaphd2017@gmail.com)

***Abstract*—**In software engineering, Software Fault Prediction (SFP) is a vital approach, especially in software development for improving the software product quality. In software systems, faulty modules are in minority class while the majority class of the modules are non-faulty modules. This scenario affects the performance and efficiency of prediction models. Hence, faulty module's misclassification costs have more impacts on software quality. In this research paper, an efficient software fault detection method is proposed, namely Evolutionary Cost-Sensitive Deep Belief Network (ECS-DBN) algorithm. The proposed ECS-DBN method helps to avoid the misclassification of software faults and improves performance of classification. An experimental analysis was conducted on reputed database, namely NASA for fault prediction. The performance was measured using evaluation metrics such as precision, recall, f-measure, accuracy and error rate. Compared to the traditional CBA-SVM method, the ECS-DBN method achieved approximately 4.4% of improvement with respect to software defect prediction.

Keywords*—Defect Prediction; Deep Belief Network; Imbalance Data; Misclassification Rate; Software classification.*

I. INTRODUCTION

A software defect is a fault, error, or failure in a software. It produces either an incorrect, or unexpected result randomly, which considers as a deficiency in a software product [1]. The SFP is a significant process in system deployment because it helps to decrease the errors and improve the software quality. Earlier fault detection technique requires more time for fault identification and software product delivery. In software development life cycle, software metrics and efficient SFD methods help to construct the models to detect the faulty classes as well as non-faulty classes [2-5]. In software development phase, developers should collect the needful resources, then start the development process [6-8]. For, example if only 30% of testing resources are available, the knowledge of the weaker areas will help the testers in focusing the available resources on fixing the classes/modules that are more vulnerable to faults. Hence, a low cost, high quality and maintainable software can be produced in the given time and budget [9-11].

Numerous researchers introduced various SFP techniques with respect to imbalance data classification. For example, Nearest Neighbor (NN) algorithm identifies the defects in same class label, but it has the difficulty to handle the class imbalance learning issue [12]. In SFP, another challenging issue is the sample subset optimization problem [13]. The software faults are identified in the dataset, but class imbalance issue maximize the complexities of software defect predictions. Also, the misclassification prediction increases the costs of training instances [14 -15]. In this research work, an efficient SFP strategy ECS-DBN algorithm has been proposed. The proposed ECS-DBN algorithm identify the software defects and classify the imbalanced data. The significant contribution of the proposed SFP method is addressed below.

- The proposed ECS-DBN method significantly decreases the misclassification cost and error rates of software defect classification.
- In order to handle the imbalanced data problems, CSL method is applied to the DBN and improve the performance in training dataset.
- The Differential evolution operators are adopted in ECS-DBN method to solve the optimization problem by iteratively searching for a solution given in an evaluation metric.

This paper is organized as follows: Section 2 surveys several existing techniques of SFP. The proposed ECS-DBN algorithm is explained in section 3. Section 4 illustrates the SFP performance of the proposed method and existing methods. Finally, the conclusion of the research work is explained in section 5.

II. LITERATURE REVIEW

Numerous methods have been proposed by researchers on software testing. In this section, a brief review of some important contributions to the existing methodologies [16-21] of SFP is presented below.

R. Gao, W. E. Wong, Z. Chen, and Y. Wang, [16] presented Hamming Distance and K-Means Clustering algorithm (HM-KM) for predicting the success and failure of the execution result. The HM-KM algorithm avoid the output verification step and directly identifies the bugs in the software. Moreover, the HM-KM algorithm not only detects the single bugs, but also detects the multiple bugs by the coverage of each test sample. This framework was robust because it able to detect the noisy sample misidentification of success and failed test cases. The small number of executions shows the expected output but for the complex applications it not able to detect the faults.

Z. W. Zhang, X. Y. Jing, and T. J. Wang, [17] implemented a novel Non-Negative Sparse Graph based Label Propagation approach (NSGLP) for semi-supervised learning in software defect prediction, which uses not only few labeled data but also abundant unlabeled ones to improve the generalization capability. The paper constructed a class-balance labeled training dataset and learn a sparse graph for characteristics of defect data by Laplacian score sampling and sparse representation. The NSGLP method used the constructed graph to predict the labels of the unlabeled software modules through label propagation approach. As compared with several existing semi-supervised software defect prediction methods, the experiments on ten NASA datasets showed that the NSGLP approach provided better performance for the software defect prediction task, but this method leads poor performance in defect prediction due to insufficient labeled data.

K.N. Rao, and C.S. Reddy, [18] presented Improved Correlation over Sampling (ICOS) approach for SFP in imbalanced classes. The existing decision tree approaches failed to extract the relevant information because of issues in model construction. The proposed ICOS method was applied to the oversampling data to create a new object to extract the relevant information from the corpus. The experimental results confirmed that the proposed ICOS approach efficiently identified the modules which were error-prone using simple and less number of rules. The software engineering challenges like consistency, cost estimation, scalability were not considered in proposed ICOS method.

X. Rong, F. Li, and Z. Cui, [19] presented Support Vector Machine with Capacity of Bat Algorithm (SVM-CBA) for identifying the software faults. The major benefit of SVM-CBA algorithm is that it enclosed with the non-linear computing ability of SVM model and optimization capacity from CBA algorithm. The SVM-CBA strategy rectified the problem of redundant dataset and improved the prediction accuracy. Moreover, an experimental analysis of the SFP performance of the SVM-CBA method showed better results than the traditional methods. The SVM-CBA method not able to handle the all training data samples at a time, hence considers the limited number of training samples for fault prediction.

E. Erturk, and E. A. Sezer, [20] developed Mamdani kind of Fuzzy Inference System (MFIS) for SFP. At first, FIS model was used for predicting the software faults and not required the training process. The MFIS method generates the output in significant two steps, those are fuzzification, and rule evaluation. Aggregation of the rule output and de-fuzzification. The proposed models used for class-level metric for SFP to select the most valid and useful design alternatives. The MFIS model efficiently predicts the faults in the small scale dataset, but misclassification problem occurred in large scale dataset.

C. Manjula, and Lilly Florence, [21] introduced a hybrid approach (i.e. Improved Genetic algorithm (IGA) with Deep Neural Network (DNN)) by combining GA for feature optimization with DNN for classification. An IGA approach was the enhanced version of GA, it modified the chromosome designing and fitness function computation steps to find the optimal solution. The DNN classifier used the adaptive auto-encoder for better indication of selected software features. The method provides poor performance when the dataset is imbalance.

To overcome the above issues addressed by the existing methods, the proposed ECS-DBN method helps to estimate the misclassification costs automatically to improve the performance of CSDBN.

III. PROPOSED METHODOLOGY

The SFP has become a vital task in the software development process because the software products are growing rapidly. SFP is very important for minimizing fault effects in software modules and improving the effectiveness of the software development process. The existing SFP techniques suffers from class imbalance problem and it generates false predictions of software executions, hence the misclassification cost of each class increase. In this paper, the ECS-DBN algorithm is proposed to reduce the misclassification rates in SFP. The proposed SFP architecture includes several steps such as input data collection, preprocessing, SFP process, and prediction of outcome. The proposed block diagram is shown in the Fig. 1.

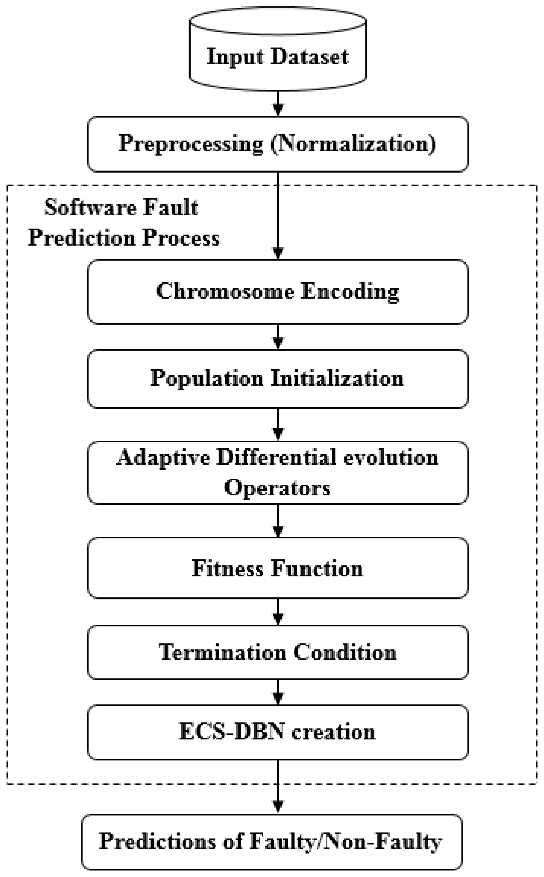
Input Dataset
Preprocessing (Normalization)
Software Fault Prediction Process
Chromosome Encoding
Population Initialization
Adaptive Differential evolution Operators
Fitness Function
Termination Condition
ECS-DBN creation
Predictions of Faulty/Non-Faulty

Fig. 1 Block diagram of proposed software fault prediction

A. *Preprocessing*

The data preprocessing step converts the raw data into a clean data set. An input data collected from the various sources and those data are not in a structured format, so it's difficult to use in further process. In this step, the normalization preprocessing method is applied for the data preprocessing. Normalization is a scaling technique; it rescales the size of the dataset. Normalization method helps to modify the unstructured dataset in structured one and provide the ranges between 0 to 1.

B. *Software Fault Prediction process*

In this research work, the ECS-DBN algorithm used for SFP. According to proposed algorithm, at first a population of misclassification cost is randomly initialized. Next, train the DBN algorithm with respect training dataset, then misclassification costs are applied to the DBN output. In order to change the misclassification costs, the mutation and crossover operators are used. Finally, during the runtime, evaluate the result of ECS-DBN algorithm with a training set to report the performance.

1) *Cost-sensitive Deep Belief Network*

The pre-processed software defect and non-defected data are forwarded to the SFP step to detect the defects by the ECS-DBN method. The ECS-DBN method use the Cost Sensitive Learning (CSL) model to reduce the overall cost in training sets. The graphical representation of different layers of CSDBN method is shown in the Fig. 2.

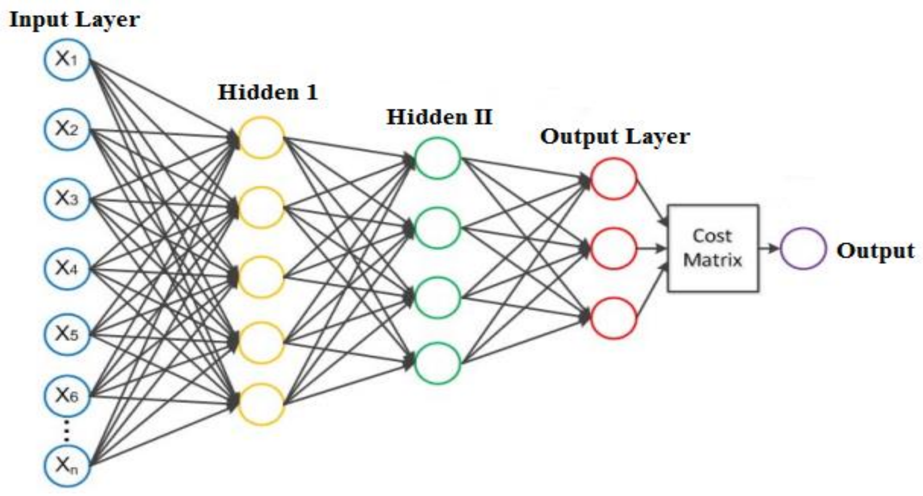

Fig. 2. Architecture of CSDBN

Consider, the overall classes are indicated as K , input data is represented as x , misclassification cost is represented as $C_{i,j} \in [0,1]$. Moreover, $C_{i,j} = 0$, when $i = j$, represents the correct classification cost is 0 . The $C_{i,j}$ is defined as misclassification cost of prediction class i and true class j . With the help of decision rule, it decrease the expectation cost $R(i \mid x)$ and mathematical expression is shown in Eq. (1).

$$R(i \mid x) = \sum_{j=1, j \neq 1}^{n} P(j \mid x) C_{i,j} \tag{1}$$

Where, $P(j \mid x)$ is the posterior probability estimation of classifying the data sample into class j . Based on the Bayes decision theory, classifier determines the classification risks and predict the risks of every class. However, in real-world applications the misclassification costs are essentially unknown and non-identical among various classes. The previous studies usually attempt to determine misclassification costs through trail-and-error, which does not lead to optimal misclassification costs. Some studies have designed some mechanisms to update misclassification costs based on the number of samples in different classes. To avoid hand tuning of misclassification costs and achieve optimal solution, Adaptive Differential Evolution (ADE) algorithm has been implemented in this paper. ADE algorithm is a simple yet effective evolutionary algorithm which could obtain the optimal solution by evolving and updating a population of individuals during several generations. It can adaptively self-updating control parameters without prior knowledge.

Consider, the probability of sample data is indicated as $x \in S$, particular class j and y is represented as stochastic variable and it's expressed in Eq. (2).

$$P(y = j \mid x) = soft\max_j (b + Wx) \tag{2}$$

The threshold value of misclassification helps to detect the posterior probabilities in each class and reduce the misclassification costs. The threshold value is indicated as C and attained posterior probability is signified as $P(y = j \mid x)$, the updated probabilities are indicated as P_ξ and calculated in Eq. (3).

$$P_{\xi}\left(y=j\mid x\right)=P\left(y=j\mid x\right).C \qquad (3)$$

Normally, compare to the majority classes the minority class misclassification threshold value is maximum. The hypothesized prediction $f(\psi)$ of the sample ψ is the member of the maximum probability among classes, can be obtained by using the following Eq. (4):

$$f(\psi)=_{j}^{\arg\max} P_{\xi}\left(y=j\mid x\right) \qquad (4)$$

The proposed CSL method only concerns the output layer of a DBN. For the pre-training and fine-tuning phase, the greedy layer based ECS-DBN method implemented.

2) Procedure for Training ECS-DBN

First, the cost sensitive learning method randomly initializes a population of misclassification costs. Second, use the training set to train a DBN. Third, according to the evaluation performance on the training set, select proper misclassification costs to generate the population of the next generation. Fourth, in the next generation, use mutation and crossover operator to evolve a new population of misclassification costs. ADE algorithm will proceed to the next generation and continue the mutation process to select the optimal solution from the search space. The training process of ECS-DBN is shown below.

Pre-training phase:

1. Let S_t be the training set.
2. Pre-train DBN using greedy layer-wise training algorithm with S_t.

Fine-tuning phase:

1. Misclassification costs are randomly initialized.
2. The DE operators of mutation and crossover helps to generate new population of misclassification costs.
3. Corresponding misclassification costs are multiplied to the training output and evaluate the errors in training data.
4. Based on the performance evaluation, select the misclassification costs and ignore the inappropriate attribute, move for the next generation.
5. Continuously repeat the mutation and selection process to reach the maximum number of generation.

Eventually, the method obtains the best misclassification costs and apply it on the output layer of DBN to form ECS-DBN. At run-time, test the resulting CSDBN with test dataset to report the performance. The misclassification costs help to encode the chromosome as numerical type with range [0, 1]. A maximum number of generation are set as the termination condition of the algorithm. The algorithm is terminated to converge upon the optimal solution. At the end of the optimization process, the best individual is used as misclassification costs to form an ECSDBN. Then test the performance of the generated ECS-DBN on test dataset.

IV. EXPERIMENTAL RESULT

The proposed ECS-DBN algorithm has been implemented in Java NetBeans 8.2 version and 32 bit operating system, and 8GB RAM.

A. Dataset Description

The CM1 dataset (NASA, http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff) is used for the performance analysis of the proposed work. CM1 is a NASA spacecraft instrument written in "C" language. It includes 497 modules with 9% fault-prone modules and overall 22 attributes were considered in this paper. Data were obtained from McCabe and Halstead feature code extractors. These features are already defined to objectively characterize the code features associated with software quality [22].

B. Performance Analysis

In this section, the performance of the proposed ECS-DBN technique is analyzed and compared with the existing techniques. Classification techniques play a significant role in identifying the fault-prone modules. The classification algorithm learns from the training set and creates a model to classify new objects. Here, approximately 497 samples are available in that 350 samples are training and 148 samples are testing. The ECS-DBN performs efficient and accurate prediction of the faulty module. According to the experimental studies, it is observed that the majority of software modules does not cause faults in software systems, only 20% of all software modules is found to be the faulty modules. The metrics used for the performance analysis are Accuracy, Overall error rate, Detection probability, FPR, False alarm probability, FNR, Precision, Recall and F1-measure.

1) Confusion Matrix

False Negative (FN) is the condition of predicting the faulty module as non-faulty module. False positive (FP) is labeling of the non-faulty module as faulty module. True positive (TP) is the correct classification of the faulty module, and true negative (TN) is the prediction of the faulty module as non-faulty module. Table 1 shows the confusion matrix values.

TABLE I. CONFUSION MATRIX FOR FAULT AND NON-FAULT MODULES

Actual Labels	Predicted Labels	
	Non-faulty	Faulty
Non-faulty	TN	FP
Faulty	FN	TP

2) Accuracy

The classification technique is used rapidly for the effective prediction of the software modules. Accuracy measures the probability of correctly classified fault-prone modules. The Eq. (5) explains the accuracy as:

$$accuracy=\frac{TN+TP}{TN+TP+FN+FP} \qquad (5)$$

3) Precision

This parameter calculates the accurate faulty modules from the total number of software modules. If the precision value is high, the time and effort required for testing and inspecting the software modules reduce, which is represented in Eq. (6).

$$Precision = \frac{TP}{FN + TP} \tag{6}$$

4) Recall

Recall is defined as the number of modules that are correctly predicted as faulty to the total number of software modules. It is also called as sensitivity expressed in Eq. (7).

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

5) F-Measure

F-measure is the harmonic mean of both precision and recall values. It is defined in Eq. (8)

$$F - measure = \frac{2 \times recall \times precision}{(recall + precision)} \tag{8}$$

6) False Positive Rate and False Negative Rate

The False Positive Rate (FPR) calculates the non-fault modules which are forecasted as faulty modules. The False Negative Rate (FNR) calculates the faulty modules which are predicted as non-faulty. The Eq. (9-10) explained the FPR and FNR.

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

$$FNR = \frac{FN}{TP + FN} \tag{10}$$

7) Overall Error Rate

In the software defect prediction process, the overall error rate is defined as the ratio of the fault prediction to the total number of predictions. Misclassification of the fault-prone modules generally incurs much higher rate than the misclassification rate of the non-fault-prone modules. The overall error rate is the ratio of the sum of FN and FP values to the sum of TP, TN, FP and FN values. The overall error rate is mathematically shown in Eq. (11).

$$overall\,error\,rate = \frac{FN + FP}{TP + FP + TN + FN} \tag{11}$$

8) Probability of False Alarm

Probability of false alarm (PF) is defined as the ratio of incorrect prediction of the faulty modules as non-faulty modules. It is defined as in Eq. (12).

$$PF = \frac{FN}{TN + FN} \tag{12}$$

C. Performance analysis of software fault prediction

In this section, SFP performance was measured with respect to different parameters such as accuracy, precision, recall and f-measure. The SFP performance was compared to the existing algorithms, namely Capacity of Bat Algorithm-SVM (CBA-SVM) [19], SVM, and Back Propagation Neural Network (BPNN). The existing CBA-SVM method handles large number of software modules, but data were redundant, hence the prediction performance was degraded. The traditional SVM algorithm helps to classify the software defects, but limited parameters are available. The proposed ECS-DBN algorithm shows better results than the existing methods.

TABLE II. PERFORMANCE OF SOFTWARE FAULT PREDICTION

Methodologies	**Parameters (%)**			
	Accuracy	**Precision**	**Recall**	**F-Measure**
BPNN [19]	90.5	94.2	88.2	85.4
SVM [19]	90.5	97.0	92.1	94.5
CBA-SVM [19]	94.0	98.0	94.0	96.0
ECS-DBN	98.4	98.0	98.4	98.0

Table 2 shows the performance analysis of ECS-DBN algorithm with respect to CM1 dataset. The CBA-SVM algorithm achieved approximately 94.0% of accuracy, 98.0%, 94.0% and 96.0% of precision, recall, and f-measure respectively. The proposed ECS-DBN algorithm achieved approximately 97.4%, 98.0%, 98.4%, and 98.0% in terms of accuracy, precision, recall, and f-measure respectively. The proposed ECS-DBN method able to handle the large scale dataset and avoids the redundant data samples. Moreover, mutation and crossover operators help to select the optimized solution and it's easy to classify the faulty and non-faulty samples. The graphical representation of performance analysis is shown in the Fig. 3.

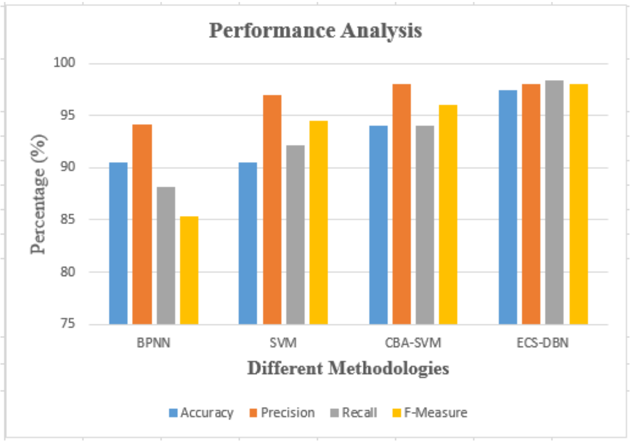

Fig. 3. Performance of software fault prediction

The Fig. 3 represents the performance analysis of the SFP. According to the graph, the BPNN and SVM method shows 90.0% of prediction accuracy. The traditional CBA-SVM method achieved 94.0% of accuracy. The proposed ECS-DBN method achieved approximately 98% of prediction accuracy. The table 3 depicts the performance of the ECS-DBN method.

TABLE III. PERFORMANCE OF ECS-DBN

Techniques	PF	FNR	FPR	Overall Error Rate
ANN [23]	0.097	1.0	0.00	0.098
HySOM [24]	0.071	0.62	0.05	0.08
Proposed ECS-DBN	0.013	0.49	0.004	0.02

The proposed technique shows less error rate compare to the existing technique and reduces the misclassification rate of software modules. So, overall performance of the proposed technique is better than the existing approaches. It is also observed that the classification and prediction performance of the proposed technique achieves high accuracy and minimum error rate.

D. Comparative Analysis

In this section, the existing SFP technique's fault detection performance was compared to the proposed ECS-DBN method and the result is tabulated in table 4. Rong, X., Li, F. and Cui, Z., [19] presented CBA-SVM method for fault prediction in software modules. This algorithm achieved approximately 94.0% of prediction accuracy. Finally, hybrid approach (i.e. Genetic algorithm with Deep Neural Network (GA+DNN)) in [23]. However, the performance of ANN-ABC was degraded when applying the same approach to other software defect prediction datasets.

TABLE IV. COMPARISON ANALYSIS OF ECS-DBN WITH EXISTING METHODS IN CM1 DATASET

Authors	Methodologies	Fault Prediction Accuracy (%)
Rong, X., Li, F. and Cui, Z., [19]	CBA-SVM	94.0
C. Manjula, and Lilly Florence, [23]	GA+ DNN	97.5
O. F. Arar, and K. Ayan, [25]	ANN+ABC	68
Proposed	ECS-DBN	98.0

The O.F. Arar, and K. Ayan, [25] presented ANN+ABC method for SFP. The parametric cost-sensitivity feature was added to ANN by introducing a new error function. The misclassification cost of positive and negative classes was set with related coefficients. The GA+DNN method provides poor performance when the CM1 dataset is imbalance. The ANN+ABC achieved approximately 97.5% of accuracy. Finally, the proposed ECS-DBN method achieved approximately 98.0% of accuracy.

V. CONCLUSION

Early detection and prediction of software defects plays an important role in the software industry in terms of quality measurement. To solve the issues in software bugs, various techniques have been developed by researchers in the past. In this research paper, to predict defect in the software module an efficient algorithm is proposed namely ECS-DBN. The ECS-DBN method used the CSL strategy to identify the unknown misclassification costs in the classes. In this experimental analysis, the input data were considered from the reputed dataset namely CM1 dataset. The performance of ECS-DBN model is measured using evaluation metrics such as Accuracy, Sensitivity, Specificity, Precision, false alarm, PD, FNR, FPR, the overall error rate and F-Measure. The proposed ECS-DBN method achieved 97.42% accuracy with 0.02 error rate, in addition, the precision is 98% and recall is 98.4%. Compared to the traditional CBA-SVM method, the proposed ECS-DBN method achieved approximately 4.4% of improvement with respect to software defect prediction. In future, the research can be extended for real time software development model using an effective flow model with a deep learning methodology which is capable of handling imbalance software module information.

ACKNOWLEDGMENTS

The authors would like to thank PES university for allowing to do research and would like to thank Dr. K N B Murthy, vice chancellor, PES university for his encouragement to do research.

REFERENCES

[1] R.S. Wahono, "A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks," Journal of Software Engineering, vol. 1, pp. 1-162015.

[2] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," Software Quality Journal, vol. 26, no. 1, pp. 97-125, 2018.

[3] W. Rhmann, "Cross project defect prediction using hybrid search based algorithms," International Journal of Information Technology, pp. 1-8., 2018.

[4] S.S. Rathore, and S. Kumar, "A decision tree logic based recommendation system to select software fault prediction techniques," Computing, vol. 99, pp. 255-285, 2017.

[5] H.B. Yadav, and D.K. Yadav, "A fuzzy logic based approach for phase-wise software defects prediction using software metrics," Information and Software Technology, vol. 63, pp. 44-57, 2015.

[6] D. Sharma, and P. Chandra, "A comparative analysis of soft computing techniques in software fault prediction model development," International Journal of Information Technology, pp. 1-10, 2018.

[7] S.S. Rathore, and S. Kumar, "Towards an ensemble based system for predicting the number of software faults," Expert Systems with Applications, vol. 82, pp. 357-382, 2017.

[8] M. Böhme, and S. Paul, "A probabilistic analysis of the efficiency of automated software testing," IEEE Transactions on Software Engineering, vol. 42, pp. 345-360, 2016.

[9] M.J. Siers, and M. Z. Islam, "Novel algorithms for cost-sensitive classification and knowledge discovery in class imbalanced datasets with an application to NASA software defects," Information Sciences, vol. 459, pp. 53-70, 2018.

[10] Y. Shao, B. Liu, S. Wang, G. Li, "A novel software defect prediction based on atomic class-association rule mining," Expert Systems with Applications, vol. 114, pp. 237-254.

[11] T.T. Khuat, and M.H. Le, "Binary teaching–learning-based optimization algorithm with a new update mechanism for sample subset optimization in software defect prediction," Soft Computing, pp. 1-17, 2018.

[12] D. Ryu, J. I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," Journal of Computer Science and Technology, vol. 30, pp. 969-980, 2015.

[13] T. T. Khuat, and M.H. Le, “Binary teaching–learning-based optimization algorithm with a new update mechanism for sample subset optimization in software defect prediction,” Soft Computing, pp. 1-17, 2018.

[14] D. Ryu, O. Choi, and J. Baik, “Value-cognitive boosting with a support vector machine for cross-project defect prediction”, Empirical Software Engineering, vol. 21, pp. 43-71, 2016.

[15] R. Moussa, and D. Azar, “A PSO-GA approach targeting fault-prone software modules”, Journal of Systems and Software, vol. 132, pp. 41-49, 2017.

[16] R. Gao, W.E. Wong, Z. Chen, and Y. Wang, Effective software fault localization using predicted execution results,” Software Quality Journal, vol. 25, pp. 131-169, 2017.

[17] Z. W. Zhang, X. Y. Jing, and T. J. Wang, “Label propagation based semi-supervised learning for software defect prediction,” Automated Software Engineering vol. 24, pp. 47-69, 2017.

[18] K. N. Rao, and C. S. Reddy, “An Efficient Software Defect Analysis Using Correlation-Based Oversampling,” Arabian Journal for Science and Engineering, pp. 1-21, 2018.

[19] X. Rong, F. Li, and Z. Cui, “A model for software defect prediction using support vector machine based on CBA,” International Journal of Intelligent Systems Technologies and Applications, vol. 15, pp. 19-34, 2016.

[20] E. Erturk, and E. A. Sezer, “Software fault prediction using Mamdani type fuzzy inference system,” International Journal of Data Analysis Techniques and Strategies, vol. 8, pp. 14-28, 2016.

[21] C. Manjula, and L. Florence, “Deep neural network based hybrid approach for software defect prediction using software metrics,” Cluster Computing pp. 1-17, 2018.

[22] H. Najadat, and I. Alsmadi, “Enhance rule based detection for software fault prone modules. International Journal of Software Engineering and Its Applications, vol. 6, pp. 75-86, 2012.

[23] G. Abaei, and A. Selamat, “Increasing the accuracy of software fault prediction using majority ranking fuzzy clustering,” Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Springer, Cham, pp. 179-193, 2015.

[24] G. Abaei, A. Selamat, and H. Fujita, “An empirical study based on semi-supervised hybrid self-organizing map for oftware fault prediction,” Knowledge-Based Systems, vol.74, pp. 28-39, 2015.

[25] Ö.F. Arar, and K. Ayan, “Software defect prediction using cost-sensitive neural network,” Applied Soft Computing, vol. 33, pp. 263-277, 2015.