ISSN 2278-3091
Volume 9 No.2, March - April 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse173922020.pdf
https://doi.org/10.30534/ijatcse/2020/173922020

An Ensemble DeepBoost Classifier for Software Defect Prediction

K. Sri Kavya[1], Dr. Y. Prasanth[2]
[1] M. Tech Student, srikavyaketagani987@gmail.com
[2] Professor, prasanthyalla@kluniversity.in
Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh - 522502, India

ABSTRACT

The main objective of a software development team is to have maximum customer defects in the software will reduce its quality. Thereby increasing its development cost. Several algorithms have been proposed for predicting software defects. But most of these algorithms are not appropriate when the dataset is imbalanced. In this paper an Ensemble DeepBoost Classifier (EDC) is built to predict the software defects effectively by addressing two major issues – curse of dimensionality and class distribution imbalance problem. Firstly, EDC uses Genetic Algorithm (GA) to find out the features that are relevant for software defect prediction. Thus, achieving dimensionality reduction. Later it uses Safe Line SMOTE (SLS) algorithm to achieve equal class distribution. Finally, it uses DeepBoost algorithm to predict whether the samples are defective or not based on the historical software defect data. The experiment was carried out on 7 PROMISE repository datasets and the results of EDC were compared with similar algorithms. The experimental results indicate that EDC has outperformed various existing algorithms in most evaluation metrics.

Key words: Class Imbalance, DeepBoost, Dimensionality Reduction, Genetic Algorithm, Safe Line SMOTE.

1. INTRODUCTION

The main objective of a software development team is to have maximum customer satisfaction by developing a defect free software product. The success of a software project depends on its quality in terms of cost, time, effort etc. Code review, inspection and testing are the traditional practices for improving software quality prior to the official release. However, testing is the most expensive phases of the software development life cycle [1].

Researchers have proposed a lot of methods to enhance the prediction results. Most of them found that the quality of datasets, such as class distribution imbalance and curse of dimensionality, has great impact on the overall performance of the model. If the number of features in the dataset is very large, it is called the "Curse of Dimensionality." There are two ways to reduce the dimensions in the dataset: by selecting the most appropriate features from the actual dataset called feature selection or by finding a smaller subset of new features called feature extraction. Dimensionality reduction enables the model to train faster. It improves the accuracy of a model and reduces over fitting. At the same time, most of the software defect datasets are not balanced, i.e., there is a huge variation of data distribution. However a small difference does not matter. In this case, the percentage of defective module is very less than the percentage of normal module. The normal and defective modules are considered as the major and minor classes respectively. When the dataset is imbalanced, standard algorithms have a bias towards the instances of major class. The classifier treats the minor class features as noise and ignores those features. There are 2 methods that deal with class imbalanced data. At data level, sampling techniques are applied to the dataset to either increase the sample count of minor class (called oversampling) or decrease the sample count of major class (called under-sampling) [2]. This is done to obtain the same rate of class distribution. At algorithmic level, the existing classification algorithm is changed to reduce the impact of imbalanced class distribution on the prediction model.

In this paper, considering these challenges, an Ensemble DeepBoost Classifier is constructed to predict the software defects efficiently. Initially the data is pre-processed using Genetic Algorithm to retrieve the features that are relevant for classification and the dataset is balanced using Safe Line SMOTE algorithm. Finally Ensemble DeepBoost algorithm is used for classification purpose.

The rest of this paper is organized as follows: Section 2 presents the summary of related work. Section 3 contains the description of proposed model EDC. Section 4 presents an overview of the datasets and evaluation metrics used. Section 5 demonstrates the performance of EDC. Section 6 concludes the paper along with future work.

2. BACKGROUND AND RELATED WORK

Various algorithms based on deep learning, machine learning and data mining such as fuzzy decision trees [3], Association rules [4], Random Forest [5], Artificial Neural Networks [6], Convolution Neural Networks [7], Bayesian Network [8], Support Vector Machine [9] have been used to predict software defects. However, these algorithms don't perform well when the dataset has large number of features and class imbalance problem.

For curse of dimensionality, Chao Ni *et al*. [10] proposed a novel method FeSCH (Feature Selection using Clusters of Hybrid data) for feature selection in cross project software defect prediction. This approach contains two stages. During the initial stage, using Density Peaks Clustering technique the original set of features is divided into multiple clusters. During the final stage, from each cluster, the proper set of features is selected based on three different ranking strategies - SFD (Similarity of Feature Distributions), LDF (Local Density of Features) and FCR (Feature-Class Relevance). Haijin Ji *et al*. [11] proposed a new feature selection method (NASM) based on Maximal Information Coefficient and Automatic Clustering. Firstly, a coefficient matrix between the features is computed that contains the maximal information, and then based on this matrix, features are clustered by spectral clustering. Calinski-Harabasz measure is adopted to determine the optimum number of clusters in the procedure of automatic clustering. Finally, the set of relevant features is selected. Qiao YU *et al*. [12] proposed a feature selection method for software defect prediction based on Similarity Measure (SM). They have designed a feature ranking algorithm. Using this algorithm, the weights of features are updated based on the resemblance of instances that belong to different classes. Then the weights of features are sorted in decreasing order. After sorting, a feature ranking list is obtained. Finally, from the list obtained after sorting, all the feature subsets are selected and evaluated sequentially on a KNN model. The Area Under curve metric is used to assess the performance of the classification model.

For class imbalance, at data view Shamshul Huda *et al*. [13] proposed An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction. This methodology uses a combination of Majority Weighted Minority Oversampling Technique, random oversampling and Fuzzy-Based Feature Instance Recovery to construct an ensemble classification model. This oversampling strategy involves in generating pseudo positive instances from the minor class. The proposed model has reduced false negative rate. Lina Gong *et al*. [14] proposed an approach Cluster-based Over-sampling with noise filtering (KMFOS) to handle imbalance class distribution in SDP. Initially, KMFOS splits the samples of minor class into K clusters, and pseudo minor class samples are generated by interpolation between samples of each two clusters. Then, these pseudo minor class samples would distinctly spread in the space of software defect dataset. Then, this cluster-based over-sampling is extended through the Closest List Noise Identification (CLNI) to clean the noise samples. At algorithmic view, cost sensitive learning and ensemble learning procedures are used to increase the performance of the classifier. Ensemble learning helps improve the performance of the algorithm by combining multiple models. Naeem Seliya *et al*. [15] proposed an Ensemble learning approach "Roughly Balanced Bagging" (RBBag) algorithm for prediction of software defects in imbalanced datasets. This algorithm is evaluated against two classification models, C4.5 decision tree and naive bayes. The results indicate that naive bayes classifier outperforms C4.5 decision tree. Zhiqiang Li *et al*. [16] proposed a novel approach "Ensemble Multiple Kernel Correlation Alignment" (EMKCA) for Heterogeneous Software Defect Prediction. First, based on multiple kernel leaning, the source and target project data are mapped into high dimensional kernel space, such that the defective and normal modules can be well split. Later, a kernel correlation arrangement procedure is used to distribute the source and target project data alike in the kernel space. Lastly, multiple kernel classifiers are integrated to minimize the effect triggered by imbalanced class distribution.

3. METHODOLOGY

EDC is a software defect prediction model for prediction of software defects in class imbalanced data. The outline of proposed methodology is shown in Figure 1. The first phase is data preprocessing. Statistical analysis is carried out in order to find out whether there are any missing values and outliers in the software defect dataset. Then Genetic Algorithm is used for feature extraction and data sampling is done using Safe Line SMOTE algorithm. The second phase is classification where Ensemble DeepBoost algorithm is used to construct the software prediction model.

3.1 Data Preprocessing

Data preprocessing involves tasks like data cleaning, data integration, data reduction and transformation. The missing values in the dataset are filled during the data cleaning stage. The missing values are imputed with the corresponding mean value. Then, the features that are relevant for classification are selected in order to reduce dimensions in the dataset. At the same time, most of the software defect datasets are not balanced. In this paper, Genetic Algorithm (GA) is used for feature extraction and the dataset is balanced using Safe Line SMOTE algorithm.

A. Feature Extraction

Most of the software defect datasets contain large number of features. Some of these features may be relevant for classification while others may redundant or irrelevant or correlated which leads to curse of dimensionality. Selecting

the best set of relevant features will increase the performance of the classifier.

EDC uses Genetic Algorithm for feature extraction. Genetic Algorithm (GA) mimics Darwin's idea of natural selection. The first step in GA involves in creating an initial population i.e., set of possible features and calculating their respective fitness scores. Here, the fitness function used is maximization of Area Under ROC Curve. Each feature is referred to as an individual. The features are encoded using binary scheme; a feature is either included (represented by 1) or not included (represented by 0) in the subset. To produce offsprings of the next generation, the features with the best fitness score are selected and combined randomly. The fitness score of an individual is determined using the formula (1).

$$\text{Fitness (i)} = \text{ROC/Number of features} \quad (1)$$

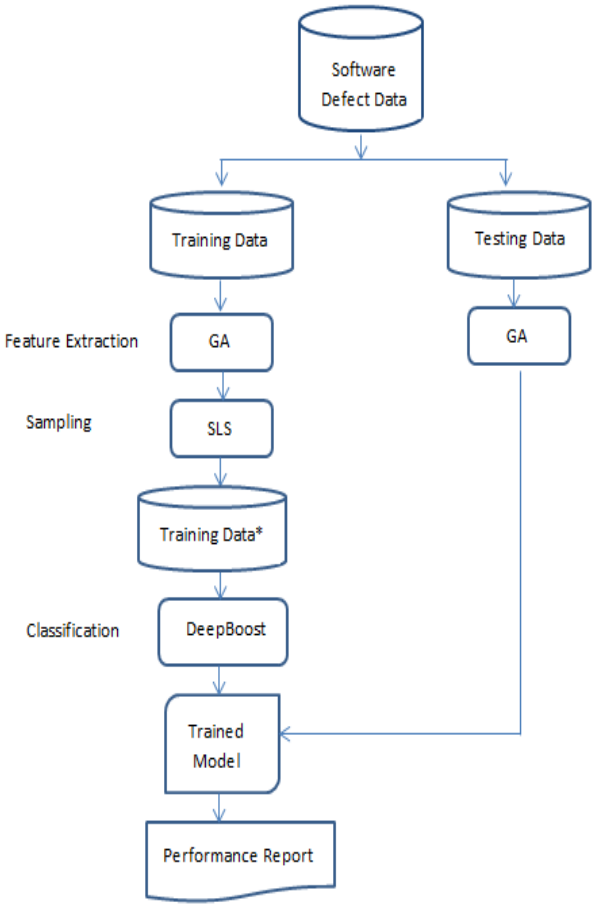

Figure 1: Software Defect Prediction Model of EDC

The objective of a Genetic Algorithm is to integrate different solutions to extract the ideal set of features from each generation. The advantage of GA over other techniques is, one can obtain the optimal solution from the best of prior solutions. Figure 2 depicts the flow chart of Genetic Algorithm.

B. Sampling

Most of the software defect datasets are not balanced, i.e., they do not have equal number of instances in the respective classes. When the dataset is imbalanced, standard algorithms have a bias towards the instances of major class. The classifier treats the minor class features as noise and ignores those features.

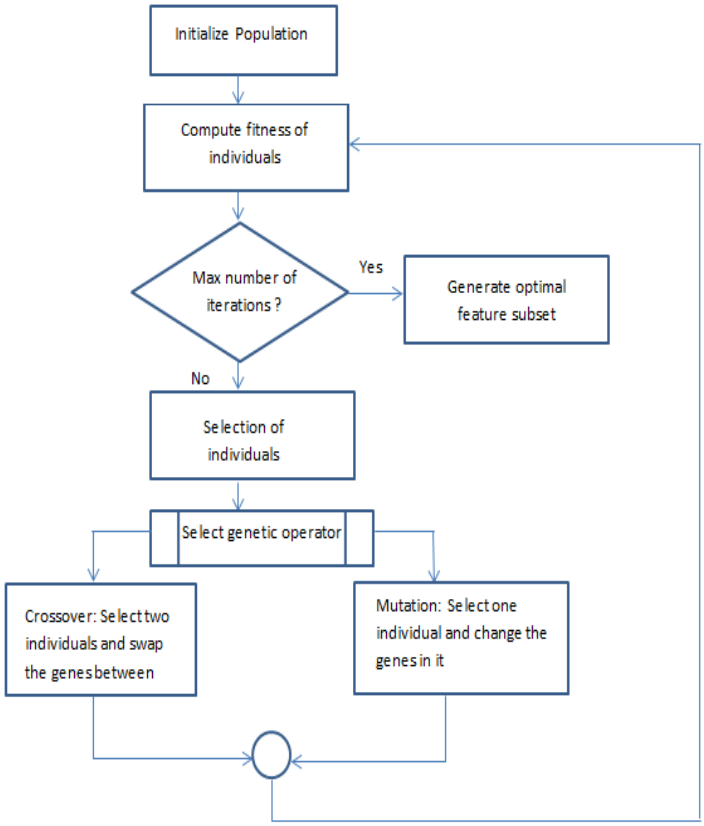

Figure 2: Flow Chart of Genetic Algorithm

Safe Level SMOTE (Safe Level Synthetic Minority Oversampling TEchnique) is a sampling technique, similar to SMOTE with a difference of generating synthetic minor class instances (also called data points) along the same line with different weight degree, called safe level. Before sampling, SLS algorithm assigns each positive data point a safe level. The safe level (sl) of a data point is calculated using formula (2). When sl of a data point is near to 0, the data point is nearly noise. The data point is considered safe if its sl is close to k. The safe level ratio is used to choose the safe positions to generate synthetic data points. The safe level ratio is determined using formula (3). The Safe Line SMOTE algorithm is described in Algorithm 1.

$$\text{safe level (sl)} = \text{number of positive data points in k nearest neighbors} \quad (2)$$

$$\text{safe level ratio} = \text{sl of a positive data point} / \text{sl of a nearest neighbor} \quad (3)$$

Algorithm 1 Safe Level SMOTE

Input: Dataset of original positive instances S
Output: Dataset of synthetic positive instances S'

1. S' = ∅ // Initially set of synthetic instances is empty
2. for i in S, Calculate k nearest neighbors and randomly select one from the k nearest neighbors. Let it be n.
3. for i in S sl_i = number of positive data points in k nearest neighbors
4. for n in S sl_n = number of positive data points in k nearest neighbors
5. if ($sl_n \neq 0$) then compute SLRatio = sl_i / sl_n
7. else Set SLRatio = ∞
8. if (SLRatio = ∞ AND $sl_i = 0$) then positive synthetic instance is not generated //i.e., i and n are noises
9. else for (j = 1 to feature_count)
10. if (SLRatio = ∞ AND $sl_i \neq 0$) then Set dist = 0
11. else if (SLRatio = 1) then dist = a random number∈[0,1
12. else if (SLRatio > 1) then dist = a random number ∈ [0,1/SLRatio]
13. else set dist = a random number ∈ [1 – SLRatio, 1]
14. difference = n[j] - i[j]
15. s[j] = i[j] + dist·difference
16. S' = S' ∪ {s} //Add synthetic instance to the set.
17. return S'.

3.2 Classification

While constructing the classifier, the smaller the error is, the more stable the classifier is. Noise, bias and variance are the factors that are responsible for generating error in the model. Ensemble procedures support to minimize these factors as Ensemble learning techniques conclude by considering the decisions from multiple models. Therefore, EDC uses Ensemble DeepBoost algorithm to reduce these factors.

DeepBoost is an ensemble learning algorithm that uses a hypothesis set *H* consisting of deep decision trees, or models of other families as base classifiers. The basic assumption is that the hypothesis set *H* is partitioned as the combination of *p* disjoint families $H_1, \ldots, H_p$ taking values in [−1, +1] ordered by increasing Rademacher complexity, $R_m(H_k)$, where H_k, is the set of decision trees of depth k, or a set of functions of degree k for $k \in [1, p]$. It is also assumed that the hypothesis sets H_k are symmetric i.e., there exists $(-h) \in H_k$, for any $h \in H_k$. For each hypothesis h, we keep either h or −h in $\{h_1, \ldots, h_N\}$ using the notation defined in formula (4). Training and testing samples are drawn from certain distribution D_t over the input space I. The weighted error $\varepsilon_{s,j}$ of hypothesis h_j for the distribution Ds, for $s \in [1, T]$ is calculated using the formula (5).

$$\wedge_j = \lambda r_j + \beta \quad (4)$$

$$\varepsilon_{s,j} = [1 - E(y_i\, h_j(x_i)] \quad (5)$$

The DeepBoost algorithm is described in Algorithm 2. At each iteration, w.r.t specified criterion, it looks for the base hypothesis that is ideal. First, via an exhaustive search, the optimal tree $h_1^* \in H_1$ trees is found. Next, a local optimal tree $h_k^* \in H_k$ trees is found $\forall 1 < k \leq K$. Finally, from the set of hypotheses selected in previous iterations, the best hypothesis is selected.

Algorithm 2 DeepBoost

Input: S' $((x_1, y_1), \ldots, (x_m, y_m))$ – Dataset of m training samples
Output: Composite Model - f

1. for i from 1 to m compute $D_1(i) = 1/m$
2. for t from 1 to T do // T = maximum number of iterations
3. for j from 1 to N do // Search for the optimal base hypothesis where N = Number of distinct base functions
4. if $\alpha_{t-1,\,j} \neq 0$ //When the mixture coefficient is not equal to zero, compute the hypothesis set's index as $d_j = (\varepsilon_{t,\,j} - 0.5) + \text{sgn}(\varepsilon_{t-1,\,j})\,(\wedge_{jm}/2S_t)$
5. else if $|\varepsilon_{t,\,j} - 0.5| \leq (\wedge_{jm}/2S_t)$ then Set hypothesis set's index set to 0
6. else $d_j = (\varepsilon_{t,j} - 0.5) - \text{sgn}((\varepsilon_{t,\,j} - 0.5)\,(\wedge_{jm}/2S_t)$
7. k = argmax $|d_j|$; where $j \in [1,N]$
8. $\varepsilon_t = \varepsilon_{t,\,k}$
9. if $|(1-\varepsilon_t)e^{\alpha}_{t-1,\,k} - \varepsilon_t e^{-\alpha}_{t-1,\,k}| \leq (\wedge_{jm}/2S_t)$ then compute step size as $\Pi_t = -\alpha_{t-1,\,k}$
10. else if $|(1-\varepsilon_t)\, e^{\alpha}_{t-1,\,k} - \varepsilon_t e^{-\alpha}_{t-1,\,k}| > (\wedge_{jm}/2S_t)$ then compute step size as $\Pi_t = \log[-(\wedge_{km}/2S_t) + \sqrt{(\wedge_{km}/2S_t)^2 + ((1-\varepsilon_t)/\,\varepsilon_t)}]$
11. else $\Pi_t = \log[-(\wedge_{km}/2S_t) + \sqrt{(\wedge_{km}/2S_t)^2 + ((1-\varepsilon_t)/\,\varepsilon_t)}]$
12. $\alpha_t = \alpha_{t-1} + \Pi_t e_k$ // e_k= kth unit vector in R^N.
13. $S_{t+1} = \Sigma_{i=1\text{to}\,m}\ \phi'(1 - y_i \Sigma_{j=1\text{ to }N}\, \alpha_{t,\,j}\, h_j(x_i))$
14. for 1: 1 to m calculate the distibution $D_{t+1}(i) = \phi'(1 - y_i \Sigma_{j\,=\,1\text{ to }N}\, \alpha_{t,\,j}\, h_j(x_i)) / (S_{t+1})$ //S_{t+1} is the normalization factor
15. $f = \Sigma_j\, \alpha_{T,j}\, h_j$; j: 1 to N
16. return family of functions f.

4. EXPERIMENTAL SETUP

4.1 Dataset Description

The datasets used in this research were taken from the PROMISE repository. The experiment was carried out on 7 NASA MDP datasets namely, CM1, PC1, PC2, PC3, KC1, KC2 and JM1. Table 1 provides the basic information of the datasets used in this experiment. The defect rate of the datasets is shown in Figure3. From Figure 3, it was clear that the percentage of defective modules in all the datasets is almost less than 20%.

Table 1: NASA MDP Datasets

Dataset	Defective	Non-Defective	Features	Proportion of Defects
CM1	49	449	22	9.83%
PC1	77	1032	22	6.94%
PC2	16	1569	37	1.00%
PC3	160	1403	38	10.23%
KC1	326	1783	22	15.45%
KC2	107	415	22	20.49%
JM1	2106	8779	22	19.34%

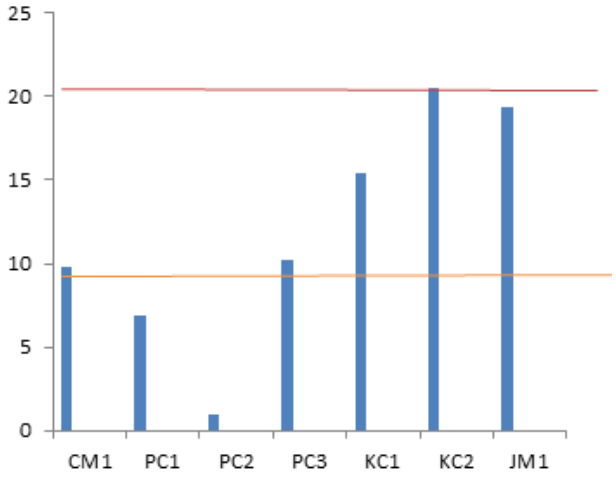

Figure 3: Defect Rate of NASA MDP Datasets

4.2 Evaluation Measures

The predictive performance of the model is assessed by adopting various metrics such as AUC, specificity, sensitivity, precision, accuracy and balance. These metrics are computed from the confusion matrix. Table 2 provides the definition of confusion matrix.

Accuracy is referred to as the proportion of correctly predicted observations to the total number of observations. Sensitivity (also known as Recall) is defined as the proportion of the number of correctly classified positive observations to the total number of positive observations. Precision (also known as Positive Predictive value) is defined as the proportion of total number of correctly classified positive observations to the total number of predicted positive observations. F-measure is the combination of Recall and Precision. AUC (Area Under Curve) is the probability that the classifier is capable of distinguishing between the classes. The greater the value of AUC, the better the performance of the classification model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (6)$$

Table 2: Confusion Matrix

Actual	Predicted	
	Positive	Negative
Positive	TP (True Positive)	FN (False Negative)
Negative	FP (False Positive)	TN (True Negative)

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (7)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (8)$$

$$\text{F –measure} = \frac{2*\text{Sensitivity}*\text{Precision}}{\text{Sensitivity} + \text{Precision}} \quad (9)$$

4.3 Experimental Design

The experiment was carried out using RStudio software. The software defect dataset is partitioned into training and testing sets in the proportion of 70% and 30% respectively. After partitioning the dataset, Genetic algorithm is used to find out the optimal set of relevant features. The maximum number of iterations is set to 20. Binary encoding scheme is used for encoding the genes (1 for inclusion of the feature and 0 for exclusion of the feature). The probabilities of crossover and mutation are set to 0.8 and 0.3 respectively. In the later stage, Safe Line SMOTE is used to balance the dataset. The number of nearest neighbors during the process of sampling and calculation of safe level are set to 5 and 4 respectively. Finally, Ensemble DeepBoost algorithm is used to construct the classifier. The number of iterations and the maximum depth of a single decision tree in the model are set to 10 and 6 respectively.

Table 3: Experimental Results

Dataset	Accuracy	Senstivity	Precision	AUC	Balance
CM1	89.73%	0.88	0.94	0.97	0.89
PC1	92.38%	0.93	0.90	0.98	0.92
PC2	94.06%	0.92	0.98	0.97	0.93
PC3	95.78%	0.94	0.96	0.95	0.95
KC1	80.03%	0.84	0.81	0.88	0.81
KC2	93.21%	0.93	0.87	0.95	0.93
JM1	91.24%	0.90	0.88	0.97	0.90

5. ANALYSIS OF EXPERIMENTAL RESULTS

The procedural results are shown in Table 3. The corresponding values of the evaluation metrics- Accuracy, Sensitivity, Precision, AUC, and Balance are summarized in Table 3.

Table 4: Comparison of different algorithms

Measure	Algorithm	CM1	PC1	PC2	PC3	KC1	KC2	JM1	Average
Accuracy (%)	DNC	75.04	78.30	84.22	78.35	**84.61**	76.14	82.02	79.81
	CEL	79.58	83.48	87.65	81.26	73.83	77.05	80.02	80.44
	MEKL	68.03	69.00	74.76	69.13	65.48	66.53	77.31	70.03
	DPRAR	87.16	**95.60**	85.42	**96.70**	82.30	83.00	89.10	88.46
	EDC	**89.73**	92.38	**94.06**	95.78	80.03	**93.21**	**91.24**	**90.92**
F – measure	DNC	0.32	0.38	0.39	0.40	0.47	0.33	0.31	0.37
	CEL	0.27	0.32	0.37	0.36	0.36	0.33	0.30	0.33
	MEKL	0.40	0.50	0.65	0.46	0.50	0.44	0.50	0.49
	DPRAR	0.65	0.78	0.84	0.86	0.71	0.66	0.76	0.62
	EDC	**0.90**	**0.91**	**0.94**	**0.94**	**0.82**	**0.89**	**0.88**	**0.89**
Balance	DNC	0.65	0.68	0.75	0.74	**0.83**	0.66	0.62	0.70
	CEL	0.54	0.61	0.65	0.57	0.55	0.49	0.46	0.55
	MEKL	0.71	0.70	0.72	0.63	0.68	0.66	0.75	0.69
	DPRAR	**0.90**	**0.92**	0.83	0.89	0.82	0.58	**0.91**	0.87
	EDC	0.89	**0.92**	**0.93**	**0.95**	0.81	**0.93**	0.90	**0.90**
AUC	DNC	0.79	0.87	0.88	0.82	**0.89**	0.80	0.71	0.82
	CEL	0.70	0.83	0.90	0.80	0.81	0.82	0.71	0.80
	MEKL	0.71	0.71	0.75	0.64	0.70	0.66	0.72	0.70
	DPRAR	0.90	0.92	0.94	0.92	0.82	0.85	0.92	0.89
	EDC	**0.96**	**0.97**	**0.97**	**0.95**	0.88	**0.95**	**0.97**	**0.94**

Since accuracy is the main indicator of the overall performance of the classifier, the results in Table 3 indicate that EDC has achieved approximately 90% of accuracy in most of the datasets except in KC1 dataset where the accuracy is only 80.03%. As most of the software defect datasets are imbalanced, the false alarm of minor class is overlooked. Therefore, it is not comprehensive to consider only the Accuracy as standard for evaluation. Another metric such as AUC value is considered to make further evaluation. Table 3 shows that AUC values of all the datasets is above 0.9 except for KC1 dataset. It indicates that the proposed method has effectively handled the two problems mentioned earlier.

At the end, this paper compared other ensemble learning algorithms like Relational association rule mining (DPRAR) [8], Dynamic version of Adaboost Negative Correlation Learning (DNC) [17], Multi-core kernel ensemble learning (MEKL) [18] and Coding based Ensemble Learning (CEL) [19], with EDC to validate the predictive performance of the proposed model. The results of comparative study are summarized in Table 4.

From Table 4, it was evident that the mean accuracy of EDC is 90.92% which is higher than the other algorithms except in certain cases where DPRAR has 95.60% and 96.70% for PC1 and PC3 datasets. However, DPRAR has achieved an average accuracy of 88.46% which is slightly lower than EDC. Other indicators including F-measure and balance also showed better results when compared with other algorithms. The mean AUC of EDC is 0.94 which indicates that EDC has better prediction performance. The results of comparison of various algorithms are shown graphically in Figure 4. From Figure 4, it was clear that EDC has outperformed various similar algorithms in Accuracy and AUC.

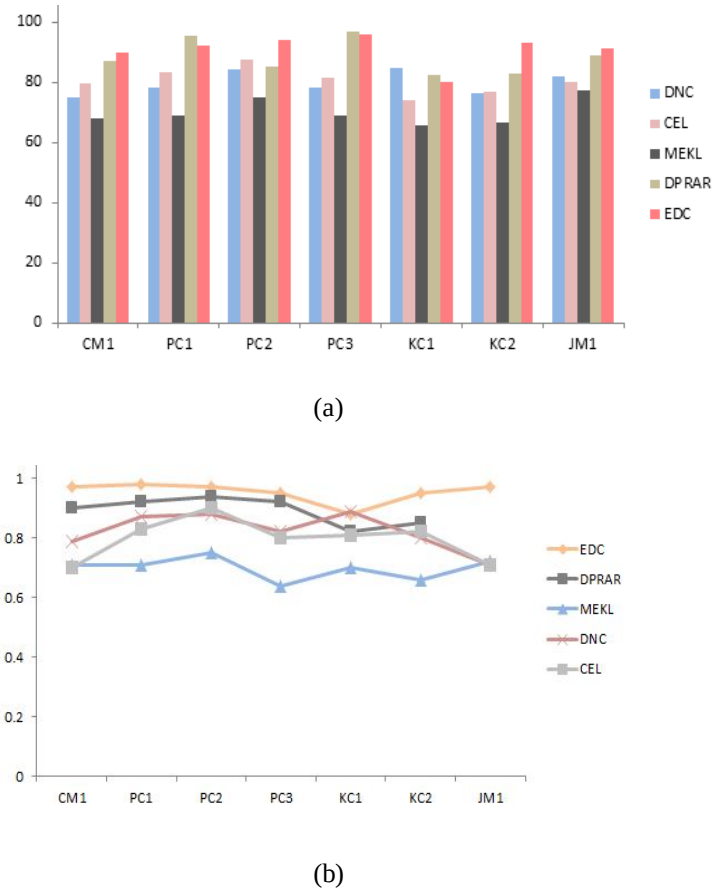

Figure 4: Comparison of different algorithms on (a) Accuracy and (b) AUC

6. CONCLUSION

In this paper, an Ensemble DeepBoost algorithm, EDC, is proposed for prediction of software defects. The two main threats in constructing software prediction models are curse of dimensionality and imbalanced class distribution. Therefore, EDC uses Genetic Algorithm to find out the relevant features of the dataset and Safe Line SMOTE to achieve approximately equal number of samples in major and minor classes. Moreover, EDC uses ensemble DeepBoost algorithm for classification. Then EDC is compared with similar algorithms and the experimental results indicate that EDC outperforms other algorithms including DNC, CEL, MEKL and DPRAR in various evaluation metrics. However, this paper adopts a common approach for data preprocessing for all datasets. It doesn't take into account, the unique features of each dataset. In future, dataset specific preprocessing method will be used for different datasets.

REFERENCES

1. Sandeep Dalal, Kamna Solanki, Sudhir, Diksha. **Exploring the essentials and principles of software development**, *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8, No. 6, pp 3504 – 3510, 2019. https://doi.org/10.30534/ijatcse/2019/129862019
2. B. Jabber, P. Sai Venkat, K. Sri Sai Nikhil, B. Lakshmi Avinash. **A novel sampling approach for balancing the data and providing health care management system by government**, *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8, No. 6, pp 2753 - 2761, 2019. https://doi.org/10.30534/ijatcse/2019/12862019
3. Vashisht, Vipul, Manohar Lal, and G. S. Sureshchandar. **A framework for software defect prediction using neural networks**, *Journal of Software Engineering and Applications*, Vol. 8, No. 08, 384, 2015. https://doi.org/10.4236/jsea.2015.88038
4. Li, Jian, Pinjia He, Jieming Zhu, and Michael R. Lyu. **Software defect prediction via convolutional neural network**, In 2017 *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 318-328. IEEE, 2017.
5. Okutan, Ahmet, and Olcay Taner Yıldız. **Software defect prediction using Bayesian networks**, *Empirical Software Engineering*, Vol. 19, No. 1, 154-181, 2014. https://doi.org/10.1007/s10664-012-9218-8
6. Soe, Yan Naung, Paulus Insap Santosa, and Rudy Hartanto. **Software Defect Prediction Using Random Forest Algorithm**, In 2018 12th *South East Asian Technical University Consortium (SEATUC)*, Vol. 1, pp. 1-5. IEEE, 2018.
7. Ricky, Michael Yoseph, Fredy Purnomo, and Budi Yulianto. **Mobile application software defect prediction**, In 2016 *IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 307-313. IEEE, 2016. https://doi.org/10.1109/SOSE.2016.25
8. G. Czibula, Z. Marian, and I. G. Czibula. **Software defect prediction using relational association rule mining**, *Information Sciences*, Vol. 264, pp. 260 - 278, Apr. 2014.
9. Marian, Zsuzsanna, Ioan-Gabriel Mircea, Istvan-Gergely Czibula, and Gabriela Czibula. **A novel approach for software defect prediction using fuzzy decision trees**, In 2016 18th *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 240-247. IEEE, 2016. https://doi.org/10.1109/SYNASC.2016.046
10. Ni, Chao, Wangshu Liu, Qing Gu, Xiang Chen, and Daoxu Chen. **FeSCH: a feature selection method using clusters of hybrid-data for cross-project defect prediction**, In 2017 IEEE 41st Annual *Computer Software and Applications Conference (COMPSAC)*, Vol. 1, pp. 51-56. IEEE, 2017.
11. Ji, Haijin, Song Huang, Yaning Wu, Zhanwei Hui, and Xuewei Lv. **A New Attribute Selection Method Based on Maximal Information Coefficient and Automatic Clustering**, In 2017 *International Conference on Dependable Systems and Their Applications (DSA)*, pp. 22-28. IEEE, 2017. https://doi.org/10.1109/DSA.2017.13
12. Yu, Qiao, Shu-juan Jiang, Rong-cun Wang, and Hong-yang Wang. **A feature selection approach based on a similarity measure for software defect prediction**, *Frontiers of Information Technology & Electronic Engineering*, Vol. 18, No. 11, pp 1744-1753, 2017.
13. Huda, Shamsul, Kevin Liu, Mohamed Abdelrazek, Amani Ibrahim, Sultan Alyahya, Hmood Al-Dossari, and Shafiq Ahmad. **An ensemble oversampling model for class imbalance problem in software defect prediction**, *IEEE access*, Vol. 6, pp 24184-24195, 2018
14. Gong, Lina, Shujuan Jiang, and Li Jiang. **Tackling Class Imbalance Problem in Software Defect Prediction through Cluster-Based Over-Sampling With Filtering**, *IEEE Access*, Vol. 7, 145725-145737, 2019. https://doi.org/10.1109/ACCESS.2019.2945858
15. Seliya, Naeem, Taghi M. Khoshgoftaar, and Jason Van Hulse. **Predicting faults in high assurance software**, In 2010 *IEEE 12th International Symposium on High Assurance Systems Engineering*, pp. 26-34. IEEE, 2010.
16. Li, Zhiqiang, Xiao-Yuan Jing, Xiaoke Zhu, and Hongyu Zhang. **Heterogeneous defect prediction through multiple kernel learning and ensemble learning**, In 2017 *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 91-102. IEEE, 2017. https://doi.org/10.1109/ICSME.2017.19
17. Wang, Shuo, and Xin Yao. **Using class imbalance learning for software defect prediction**, *IEEE Transactions on Reliability*, Vol. 62, No. 2, 434-443, 2013.

https://doi.org/10.1109/TR.2013.2259203

18. Sun, Zhongbin, Qinbao Song, and Xiaoyan Zhu. **Using coding-based ensemble learning to improve software defect prediction**, *IEEE Transactions on Systems, Man, and Cybernetics*, Part C (Applications and Reviews), Vol. 42, No. 6, 1806-1817, 2012. https://doi.org/10.1109/TSMCC.2012.2226152
19. Wang, Tiejian, Zhiwu Zhang, Xiaoyuan Jing, and Liqiang Zhang. **Multiple kernel ensemble learning for software defect prediction**, *Automated Software Engineering*, Vol. 23, No. 4, 569-590, 2016. https://doi.org/10.1007/s10515-015-0179-1