Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Elma+ : An Ensemble Learning Based Model For Accurate Prediction Of Software Defects

MR..RAGHVENDRA OMPRAKASH SINGH[1] , DR. BLESSY THANKACHAN[2]

[1]Research Scholor Department of Computer and Systems Sciences Jaipur National University, Jaipur Inida.

[2]Assistant Director and Guide Department of Computer and Systems Sciences Jaipur National University, Jaipur Inida.

Abstract:

The recent developments made in the software engineering technologies has introduced the growth of the data. In order to handle the explosive growth of the data, many software quality assessments are introduced to validate designed software. Software defects assessment is one of the finest qualities of software engineering models. Software defects classification is the basis for effective management of software defects. This paper presents a novel ELMA+ technique to predict the classes of software defects. Here, an ensemble learning approach is taken to do the prediction systems. Initially, the defect data is collected from the public repository. A simpler exploratory data analysis is done to know the count of presence and absence of software defects. SMOTE technique is applied to preprocess the collected dataset. The presence of oversampling data has lowered the participation of minority classes during the training process. In order to leverage the minority classes and the presence of data ambiguity issues, the oversampling data are aligned with the synthetic data creation. The generated synthetic examples in alignment to the real-time data behave like feature space instead of data space. The minority of each class combines with the line segments of the nearest data points. Once the majority and minority classes are defined properly, then the oversampled data are sorted out. The scaled features are then fed into the ensemble of classifiers, namely, k-nearest neighbors (k-NN), Adaboost and Bagging. These three classifiers take the feature scaled data as input to classify the defects of the softwares. The proposed framework is simulated and has shown the efficacy of the proposed ensemble classifiers in terms of accuracy, sensitivity, specificity and the precision. The comparative analysis done from the perspective of before and after SMOTE application. It is clearly understood from the achieved

results that the feature scaled data into the ensemble classifiers has yielded better outcomes.

Keywords: Software engineering; Software defects; SMOTE technique; feature scaling process; Ensemble classifiers; Oversampling data.

1. Introduction:

Software Engineering (SE) is one of the eminent fields that deals with the organization of the developed and developing software. Quality Assurance (QA) of the software is a crucial role in the software development industry. QA is the kind of behavioral activity performed over the execution of the software projects (Wang et al,2016). Software Testing (ST) is an active research area that imbibes with the quality of the software products. Generally, the testing consumes a higher time and effort to test the test cases. There are different kinds of software testing available, even though the bugs (or) errors remain to be unresolved due to the lack of time and effort (Yang et al, 2015). Therefore, a need of improvising the software testing field that preserves the time and cost of the organizations. The developer (or) the tester should be proficient in analyzing and predicting the software bugs at an earlier phase. Software bugs are inevitable in nature. The recent developments made in the software industry have been emerging with complex software applications. It is mandated that the developed software should be free from bugs (Huo X et al, 2016). The errors caused by programming functions are known as software bugs. Significantly, it reduces the performance of the entire software applications. Each software has maintained a Bug Tracking System (BTS) which acquires, organizes, and monitors the bugs and its reports. With the BTS, the software entities like users, developers and the testers make use of it, to generate the test reports. Once a bug is found, it undergoes several phases to resolve it. Analysis and Prediction of the bugs using ensemble classifiers is being considered widely in this research study. The design of predictive classifiers is possible with the detailed analysis of the causes of software bugs (CJ Clemente, et al, 2018). Different prediction models related to the bug indicators are developed to predict the count of bugs and its resolving procedures. Prediction models assist the developers to provide software testers and the developers. Several steps are taken to support the bug prediction models, so as to improve the quality of the software products. It prevents the bug severity (Li et al, 2017) in future bugs. Software Defect Prediction (SDP) is an important field of study that determines the software quality and its administration with reduced costs. The continuous failures of the software over time are labeled as software defects. The software bugs are obtained from the software developer and stakeholders. The role of a software defect prediction system is to predict the defects by improving the software quality with reduced costs. Several researchers are working on improving the static attributes which define the software quality metrics using effective learning approaches. It relies on software attributes such as Line of Code (LoC) and also predicts the software quality systems. In the angle of software quality systems, a different version of prediction techniques are available for different testing and software modules environments.

1.1 Contributions of the study

The highlights of this paper are:

a) The historical and synthetic data are employed to resolve the minority classes and data ambiguity issues.
b) The deployment of SMOTE technique has leveraged the minority classes by creating synthetic data in order to find out the nearest data points.
c) The data space is treated as feature space to handle oversampling data.
d) Ensemble of classifiers, like K- Nearest Neighbors, Adaboost and Bagging techniques has improved the predictive quality by using feature scaled data.
e) The prediction accuracy is higher than the conventional classifiers.

1.2 Organization of the paper

The paper is organized as follows:

Section II presents the literature survey that reviews the existing techniques.

Section III presents the Proposed framework that explains the proposed phases.

Section IV presents the Experimental Results and Discussion which describes the achieved performance using the proposed techniques.

Section V presents the Conclusion that portrays the findings of the study.

2. Related work:

The review of existing studies from the aspects of objectives, techniques designed, merits and the demerits in this field is discussed. (Rudolf Ferenc et al, 2020) explored the scope of deep learning in bug prediction systems. It was explored on the bugs of Java classes and obtained 55.27% F-measure than the conventional prediction systems. The dimensions of the bugs are not considered for analytic purposes which brings replication issues. (Cheng Zhou et al, 2020) presented the prediction model for named entity recognition which intensively worked on the information extraction process. Based on the obtained fine-grained factual information, the accessibility of bug prediction under neural networks was improved. The concept of the conditional random fields has helped to resolve the prediction error 0f 91.1%. (Sushant Kumar Pandey et al, 2019) has presented deep features and ensemble learning techniques. It was designed to enhance the testing efforts on the Promise repository NASA dataset. The bug features are deeply represented and analyzed under auto-encoders that enhance the performance of the bug predictions via the highest correlation coefficients. Bug prediction on the successive count of the software systems was explored by (Sushant Kumar Pandey et al, 2020). The metadata of each bug was given into the training layers of deep features which has avoided the overfitting and class imbalance issues. It is explored on seven datasets that have reduced the MSE value from 0.71 to 4.715 and MAE from 0.22 to 1.679.

(Yan Xiao et al, 2018) has improved the localization of the bugs using enhanced convolutional neural networks. Here, the frequency of the bugs are estimated and then fixed using word embedding models. The system has reduced the bug fixing computational time and the test cases generation time. The fundamental thought is to sort out the bug by developer with reference to comparative skill with the set of predefined bug classes. (Guo et al, 2020) has presented a Word2vec to the CNN implementation for the bug summary process. Some researchers have studied the components, products, priority and severity to enhance the prediction accuracy of the bug assignment process. (Zhao et al, 2019) has recommended a topic model based Latent Dirichlet Allocation (LDA) method to compute the similarity of bug reports. With the help of the multiple attribute data, some inconsistencies prevailing in bug reports are done. The supervision of the data is done by LDA that derived the main subjects (Xia et al, 2017). Regardless of it, labeling of bug reports with different label information can lead to the context information loss. Due to the information retrieval methods, topic modeling systems take the largest data size. A semi-supervised text categorization model (Xuan et al, 2017) was studied to accurately generate the bug reports data. Along with the concept of bayesian approach, the report data generation was quite better. Social networking techniques (Xuan et al, 2012) has presented the bug classes by prioritizing the developers. Several influential factors such as characteristics of products, time variation and the noise tolerance have prioritized the developers.

Approaches like tossing paths were to fix the bug reports. (Jeong et al, 2009) has presented the transfer graph model that classifies the bugs. It also reported the accuracy of the bug report assignment. However, it does not ensure the feasibility rate of the classified bugs. (Sajedi Badashian et al, 2020) has introduced the transfer graph model with different attributes. The bug tossing issues in bug triaging has reported that 93% of bug reports are tossed. It has stated that the bugs were not properly ensured for different environments. (Shokripour et al, 2015) has yielded the class score for the bug parameters. With the help of fuzzy logic systems, the bug reports are classified into their respective classes. (Couto et al, 2012) has described a robust model that portrayed the causality between software metrics and the bugs occurrence. With the help of corpus, the prediction of bugs were determined. However, the false positive rate of bug classification has increased.

HyGRAR, a hybrid classification model that merges the hidden layer functionalities of ANN with the association rule mining (Miholca et al, 2018a) were introduced. The relationship between the software metrics was classified into defective and non-defective software entities. It was explored on the NASA PC software databases. Hidden units were highly increased that developed imbalance class issues. Unsupervised learning (Miholca et al, 2017) under coupling metrics was explored to maintain the evolution of the bugs. An object oriented system was introduced to quantify the coupling of application scenarios. The representation of high dimensional datasets has increased the textual data performance. Doc2Vec conversion has increased the structural class rates. The role of semantic and structural features (Miholca et al, 2019) under the learning process increased the contextual information rate. The files were coded

under Abstract Syntax Trees (AST) and converted into the token vectors. CNN classifiers was used to work out with the token vectors. . Based on the semantic features, the classes were done with 99.5% accuracy rate. In continuation with, a hybrid algorithm was studied using association rules and artificial neural networks (Miholca et al, 2018b). Here, 10 open source data was implemented to enhance the classifier's efficiency, however, the heavy computational steps were observed.

In (Radjenovic D. Heri, 2013), a systematic review was done on quality metrics of software defect prediction. The review has stated that the prediction of fault location is really helpful in influencing the quality metrics and thus, contextual properties explored the accessibility of selecting the quality metrics. An intelligent software fault prediction was studied by Deep Belief Networks (Wang et al, 2016). . With the help of syntax trees, the semantic features were gathered and analyzed on the within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). Both the prediction model has decreased the false positive rate by 10% than TCA+ algorithm. In (Jing et al, 2017), the author was specific to resolve the class imbalance problems by an improved Subclass Discriminant Analysis (SDA). SDA has deliberately resolved the feature learning modules by evenly distributing the source and target data consistently.From the perspective of cross-project domain, the prediction model has lowered the class separation based on logical constraints. Two stage cost sensitive learning module (Liu et al, 2014) was designed to leverage the cost sensitivity analysis on the deep learning algorithms. In alignment to that, a cost-sensitive approach was also designed for feature selection algorithms such as Cost-Sensitive Variance Score (CSVS), Cost-Sensitive Laplacian Score (CSLS), and Cost-Sensitive Constraint Score (CSCS). Though it decreased the computational time and also leveraged the cost metrics based on the software quality, the efficiency of determining contextual information was not studied. Similar study was studied using a ranking approach (Yang et al, 2015) that stated the efficient use of testing resources. Depending on the count of defects, the module was designed to cope up with the noisy data as well as the feature learning modules. The characterization of the relationship between the square errors exhibited in the classifier were rectified under optimization schemes.

In (Czibula et al, 2014), authors have explored latent semantic indexing in the computational linguistics fields. The processing capabilities of the genetic ontology was studied latent semantic analytics. The similarity among the topics were extracted to process the experimental data. Some of the structured data was not extracted under heterogeneous classes. Relational Association Rules (RAR) (Haghighi et al, 2012) was designed to resolve the defect prediction under different association rules. It was explored on NASA datasets that required heavy time on the data cleaning process. It was processed on different datasets, however, the role of the constructed class is not focussed on a semi-supervised algorithm. One of the software qualities is the cost reduction which was studied by data mining approach (Kirbas et al, 2017a). Naive bayes classifier was employed , along with the performance of other 37 different classifiers, so as to detect the faults. The authors have stated that the decreased costs enhanced the classifiers'

performances. The system has increased the classification accuracy, yet, it is not feasible to determine the boundaries of classes in bagging algorithms. The software faults in industrial sectors (Kirbas et al, 2017b)] have distorted the relationship between the evolutionary coupling and defects. It experimented on the legacy financial system and a modern telecommunications system. Regression analysis was employed to find out the relationship between the software metrics. The different effects of evolutionary coupling were employed to detect the faults based on their types, size and process metrics. Likewise, the infrastructure issues are not studied for different testing cycles. In (Cataldo et al, 2009), the author has introduced a conceptual coupling based metrics for software defect prediction systems. Coupling metrics is one of the software quality metrics that defines the performance of the prediction modules. It was established by estimating the similarities among the bugs. However, it can't handle the class imbalance issues during data crowdedness. Similar study was extended using empirical study (Chen et al, 2013). The coupling class includes the structural, semantic and dynamic measures of the software systems. Here, Java classes such as Argo UML, J Hot Draw and jEdit were studied. The semantic coupling was studied more than other coupling measures. The encapsulation between the classes were revised using call methods (Sushan et al, 2020). The logical constraints on class call methods are not studied.

3. Proposed Methodology:

This section presents the working of the ELMA+ that classifies software defects by means of ensemble classifiers.

3.1 Data collection & preprocessing:

The database is collected from the public repository, named, Promise datasets. It contains several classes of the software bug classes. The dataset contains many noises ie. missing and unbounded data. These imbalances are rectified using random sampling techniques. This step is known as data preprocessing. The collected datasets are raw which have to be preprocessed using preprocessing techniques to remove the irrelevant and uncertainties from the data. Generally, sampling methods are employed to preprocess the data. The data sampling reduces the computational steps of the training data. Here, the sampling methods are classified into two approaches, namely, under sampling methods and over sampling methods. In this study, automatic proper sampling methods are explored to reduce the computational space of irrelevant data. The larger dataset is converted into smaller random samples. Each sample consists of T tuples data. While drawing a sample from the tuples T, it is likely to be the probability of tuples under the dataset D ie. 1/T, then all tuples are equally sampled. Some tuples are recorded before being replaced with the other tuples. By doing so, all tuples are sampled under the given functions which could refine the noisy and redundant data.

3.2 Feature extraction:

In the proposed work, a novel Synthetic Minority Over-Sampling Technique (SMOTE) technique

is employed for an oversampling process so as to alleviate the class imbalance problem and make the training dataset contain adequate samples of both majority and minority classes. Many oversampling data is replaced so as to enhance the minority class recognition. The effects of oversampling under different regions of feature space is considered. Regardless of it, the minority class is reduced by lowering the feature space. By creating 'synthetic' examples, the oversampling data is replaced for the improvements of minority classes. Here, additional data operators are performed on the real data. It makes use of rotation and skew operators on the training data. The generated synthetic examples in alignment to the real-time data behave like feature space instead of data space. The minority of each class combines with the line segments of the nearest data points. Once the majority and minority classes are defined properly, then the oversampled data are sorted out. The scaled features are then fed into the ensemble of classifiers. The combination of k-Nearest Neighbors and Gaussian distributions are merged to scale the features.

a) K- Nearest Neighbors (k-NN):

In this step, a k-Nearest Neighbor Classifier is built. While performing classification, k-NN considers the unbiased weighting of instances in the decision rule regardless of the distance. The steps of the conventional k-NN classifier are:

- Finding the k-training instances
- Selecting the most common data on these k-instances
- Estimating the distance values between those k-instances.

The decision rule of the KNN method is based on item strength rather than majority vote. Any test point to be evaluated is assigned the label of the class with the highest summation of item strength. Here, the training data is splitted into smaller subsets. A classifier model is designed for each subset and then, a distance approach is applied to classify the testing data. Based on the estimated distance on the testing data will determine the performance of the classifiers. Since it deals with the continuous attributes, the distance estimation between the data is easily calculated using the euclidean distance formula. Let the data in the first subset is represented as, $(x_1, x_2 \ldots . x_p)$and the data in the second subset is represented as, $(y_1, y_2 \ldots . y_q)$. Then, the euclidean distance between two subsets is given as,

$$\text{Distance} = \sqrt{\quad}$$

For each data in the subsets are analyzed using the above equation. It is found that the largest values create an issue with the smallest values.

b) Gaussian Distribution:

The extracted features have helped to determine the variance of the cases related to software defects. Then, multivariate Gaussian distribution was formulated in testing instances for differentiating the bug classes. By applying multivariate Gaussian models, the interaction between the data has strongly evoluted with reduced complexity. Let us assume x, as a number of bugs

classes, optimal fit parameter €. The maximum likelihood value is determined for the training set. The features used for training purposes are determined with mean and standard deviation value using Gaussian Distribution p(x). It is expressed as

$$P(x) = \prod_{k=1}^{n} P(x_k : \mu_k \sigma_$$

k= Count of features taken from the sample data

μ = mean value of the gaussian distribution function

σ = standard deviation

The assumptions formulated to define the anomaly classes are

If P(x) < optimized fit value €, then the value is said to be bug class

If P(x)>= optimized fit value €, then the value belongs to normal class

3.3 Ensemble Classifier:

The design of ensemble classifiers is done to find out the defect classification. Here, three well-known classifiers such as random forest, adaboost and bagging classifiers are formulated.

a) Random forest classifiers:

The proposed steps of the random forest classifiers are:

- The diversity of each record under a tree ensures correlated features between a node and the leaves.
- Random features are selected to build the trees.
- Root node is not altered at any causes i.e stopping criterion, parameter fitness and so on.
- Root nodes are altered only when the information gain changes.
- Ensured efficient training features to the classifier models.

Consider a possible set of N records of input data. Each record is denoted as a d-dimensional vector, along with single aspects C, which is the label of the random trees under single-objective. It is expressed as:

$$S_0 = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$$

Where,

X and Y are the labels of records.

A splitting criterion in two dimensions for a random tree are given as,

$$h(\mathbf{x};\theta_1,\theta_2) = \begin{cases} 1 & \text{if } x_{\theta_1} > x_{\theta_2} \\ 0 & \text{otherwise.} \end{cases}$$

The random tree is formed by the splitting criterion with the mutual information. In specific to, the splitting functions assist to construct the left and right child of a tree. The below equation presents the constraints of the left and right child of a tree.

$$\begin{aligned} \mathcal{S}_i^L(\boldsymbol{\theta}) &= \{\mathbf{x} \in \mathcal{S}_i \mid h(\mathbf{x};\theta_1,\theta_2) = 1\} \\ \mathcal{S}_i^R(\boldsymbol{\theta}) &= \{\mathbf{x} \in \mathcal{S}_i \mid h(\mathbf{x};\theta_1,\theta_2) = 0\} \end{aligned}$$

The construction of a child in a tree represents the quality of the data sets Si. The information gain of each record is estimated from eqn. (20):

$$I_i(\boldsymbol{\theta}) = H(\mathcal{S}_i) - \sum_{j\in\{L,R\}} \frac{|\mathcal{S}_i^j(\boldsymbol{\theta})|}{|\mathcal{S}_i(\boldsymbol{\theta})|} H(\mathcal{S}_i^j(\boldsymbol{\theta})),$$

Where,

H(s) is the entropy value of set S.

Each tree denotes the class with the information gain of datasets Si. Henceforth, the incessant records sampling Thus, the trees are formed by continuous sampling of records by the above two equations. Specifically, internal nodes of a tree are assigned by the maximum information gain. The overall split of a tree for depth of a tree as, T= d(d-1), is given in below equations.

$$\boldsymbol{\theta}_i^* = \arg\max_t I_i(\boldsymbol{\theta}_t).$$

During the construction of the trees, i.e placing the node as, left (or) right branch of a tree, an overfitting of the data may occur. In the case of constructing the random forest, the diversity issue brings significant change over the classifiers. The collection of trees under forest is represented as F= (T1… Tf). The left and right branches of a tree are regularized before forming the forests. Each tree is independently trained.

b) Adaboost classifier:

Boosting is an approach that creates prediction rules by merging weak and relatively inaccurate rules. It discovers the class by moving forward. The pseudocode of adaboost algorithm is,

Pseudocode:
Given data: $(x_1, y_1 \ldots\ldots (x_m, y_m)$ where, x and y are the set of samples
Initialize: $D_1(a) = 1/n \text{ for } a = 1, \ldots, n$
For q=1, ….. P
Train the weak learner using distribution D_q.
Get the weak hypothesis, h_t: $\aleph \rightarrow \{-1, +1\}$
Select the h_t with the low weighted error,

$$\epsilon_t = P_{ra\ D_q}[h_t(x_a \neq y_m$$

Select $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Update for, a= 1,...., n:

$$D_{t+1}(a) = \frac{D_t(a) \exp(-\alpha_t y_m h_t(x_m))}{Z_t}$$

Where, Z_t is the normalization factor.
The final hypothesis output : $H(a) = \text{sign} h_t$ (a))

c) Bagging classifier:

Bagging classifier takes the decision with the different learners into one prediction model which is implemented by voting approach. Regardless of it, the bagging process is done in three easy ways, namely, the alternative which leads to the experts. For a case that one gets more votes than other classes, it is considered as correct. The classes are categorized by the voting modules. Similar to that, the weight is also randomly estimated in the training sets.

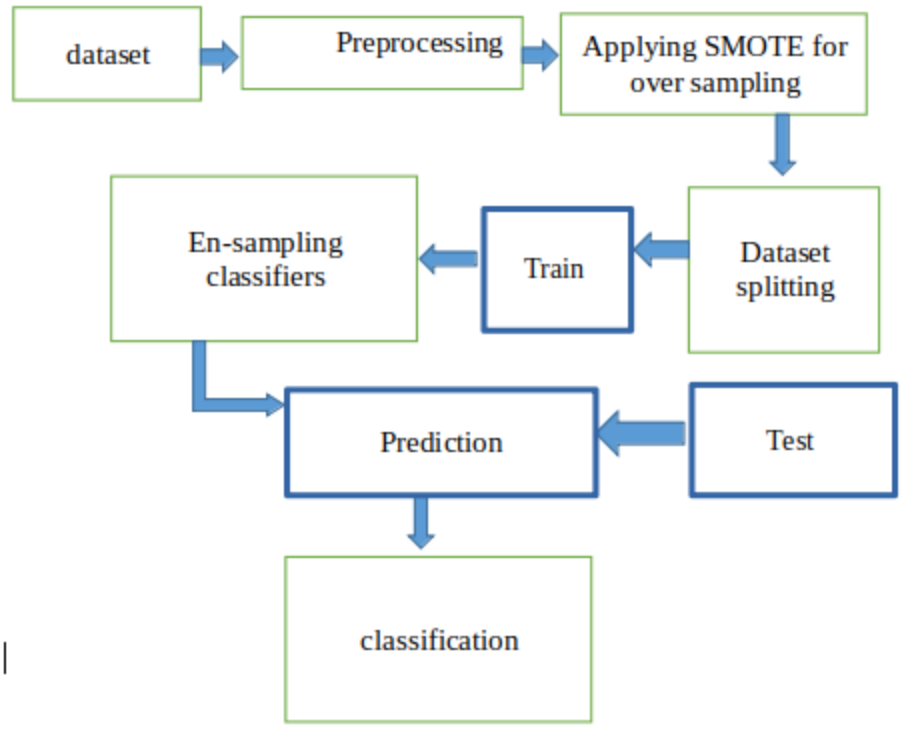

Fig.1. Proposed workflow

4. Results and Discussion:

The proposed framework is experimented under certain performance metrics. The dataset is collected from the public repository, named, promise software defect prediction is implemented in this study. The data analysis is done using Python, a high level programming language. Python is the recent programming language employed in the data analytics study. It has a great choice of Machine Learning (ML) and Artificial Intelligence (AI) libraries that explores the efficiency of the real-time analytic system. In our proposed study, the confusion matrix is given as,

True positive (TP)- No. Of samples correctly identified as true

False positive (FP)- No. Of samples incorrectly identified as true

True Negative (TN)- No. Of samples correctly identified as false

False Negative (FN)- No. Of samples incorrectly identified as false

Class	Label values
True	0
False	1

Table 1. Class Distribution

Premises:

Labels: True (presence of software defect) and false (absence of software defect)

a) Recall: The proportion of true cases to the predicted true cases which is expressed as

Recall= (TP)/(TP+FN)

b) Precision: It is defined as the proportion of predicted true that are correctly real true value. It is given as:

Precision= (TN)/(TN+FP)

c) Prediction Accuracy: It is the most intuitional performance that measures the correctly predicted observation to the total observations. It defines the ability of distinguishing true and false abnormal cases. It is given as:

Accuracy= (TP+ TN)/ (TP+TN+FP+FN)

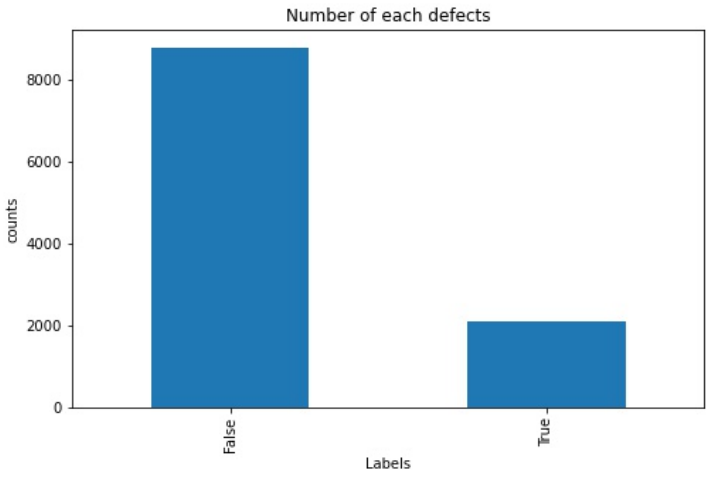

Fig.2. Data analysis before the SMOTE application

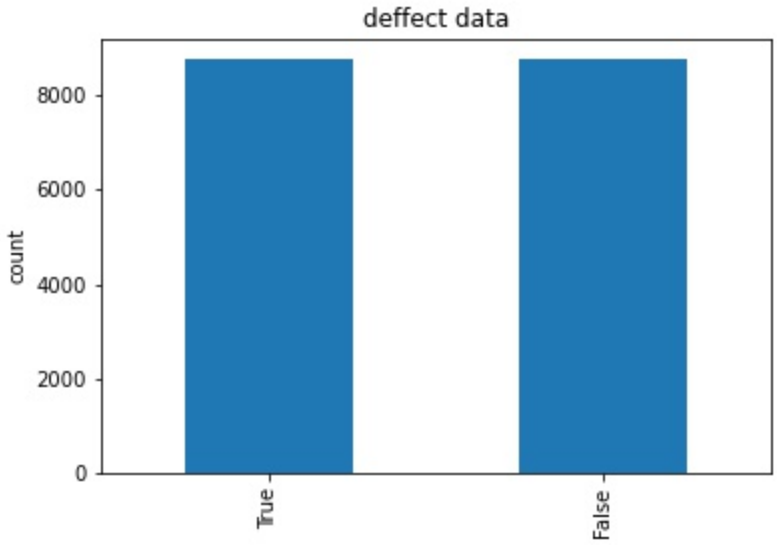

Fig.3. Data analysis after the SMOTE application

The fig.2 and 3 presents the data analysis of the SMOTE application. Here, the considered dataset is explored in terms of true and false i.e presence of software defects (True) and the absence of software defects (false). It helps to explore the distribution of majority and minority classes. The need of oversampling processing by means of SMOTE technique is to leverage minority data classes.

Parameters	Existing	Proposed
Sensitivity	96	96
Specificity	17	82
Precision	96	96
Accuracy	82	89

Table 2: Comparison between existing and proposed classifiers

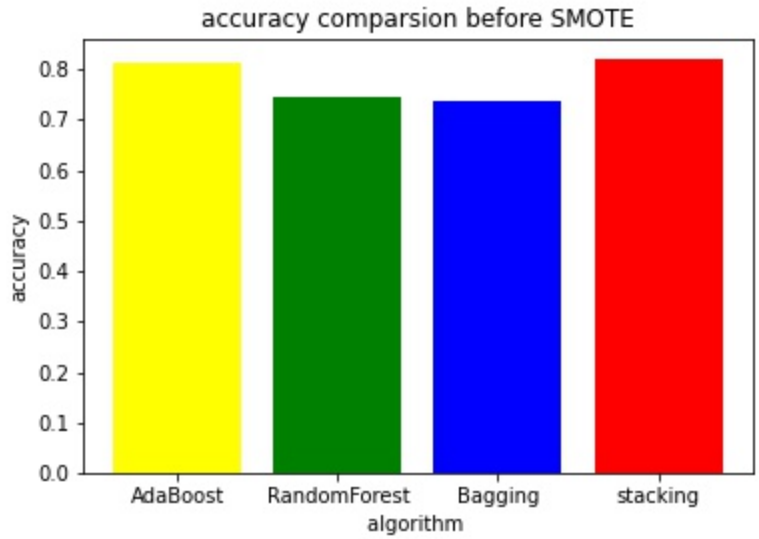

Fig. 3. Existing - Accuracy performance of ensemble classifiers

The fig.3 presents the accuracy performance of the ensemble of classifiers before the SMOTE technique usage. It is observed from the results that the classifiers have yielded better accuracy, however, the sensitivity and specificity of the software defect classification are taken the higher values. It proves the inability of the classifiers.

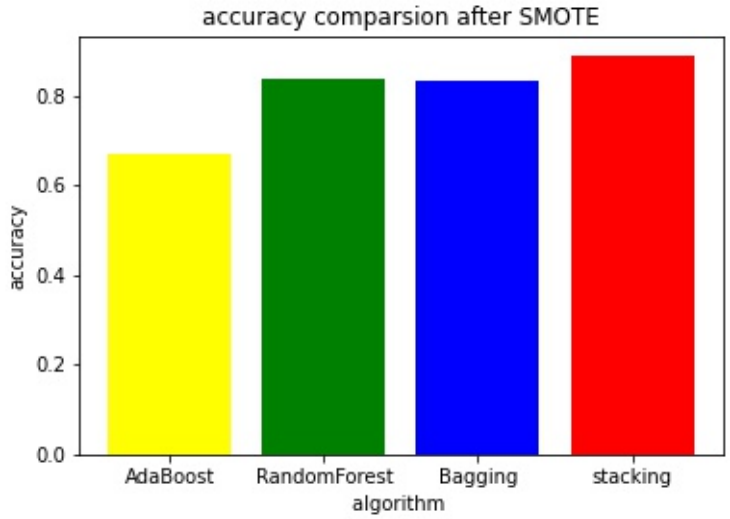

Fig.4. Proposed - Accuracy performance of ensemble of classifiers

The fig.4 presents the accuracy performance of the ensemble of classifiers after the SMOTE technique usage. Here, X-axis represents the set of ensemble classifiers and Y-axis represents the set of classifiers. It is clearly understood that the proposed ensemble classifiers have sorted out the class imbalance issues with the SMOTE technique.

5. Conclusion:

This paper designs a novel ELMA+ technique that intends to accurately predicts the software defect classes. An ensemble of well-known classifiers such as k-nearest neighbors (k-NN), Adaboost and Bagging are analyzed in this study. To begin the study, the defect data is collected from the public repository. A simpler exploratory data analysis is done to know the count of presence and absence of software defects. SMOTE technique is applied to preprocess the collected dataset. The presence of oversampling data has lowered the participation of minority classes during the training process. In order to leverage the minority classes and the presence of data ambiguity issues, the oversampling data are aligned with the synthetic data creation. The generated synthetic examples in alignment to the real-time data behave like feature space instead of data space. The minority of each class combines with the line segments of the nearest data points. The scaled features are then fed into the ensemble of classifiers, namely, k-nearest neighbors (k-NN), Adaboost and Bagging. These three classifiers take the feature scaled data as input to classify the software defects. In specific, minority classes are resolved and then analyzed to resolve the data ambiguity issues. The experimental results have shown the efficiency of the proposed ensemble classifiers by achieving the better value of accuracy, sensitivity, specificity and precision. The comparative analysis done from the perspective of before and after SMOTE application. It is clearly understood from the achieved results that the feature scaled data into the ensemble classifiers has yielded better prediction outcomes.

REFERENCES

1. Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: Software engineering (ICSE), 2016 IEEE/ACM 38th international conference on. IEEE; 2016. p. 297–308.
2. Yang X, Lo D, Xia X, Zhang Y, Sun J. Deep learning for just-in-time defect prediction. QRS; 2015. p. 17–26.
3. Huo X, Li M, Zhou Z-H. Learning unified features from natural and programming languages for locating buggy source code. IJCAI; 2016. p. 1606–12.
4. Clemente CJ, Jaafar F, Malik Y. Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?. In: 2018 IEEE international conference on software quality, reliability and security (QRS). IEEE; 2018. p. 95–102.
5. Li J, He P, Zhu J, Lyu MR. Software defect prediction via convolutional neural network. In: 2017 IEEE international conference on software quality, reliability and security (QRS). IEEE; 2017. p. 318–28.
6. Rudolf Ferenc et al.,. Deep learning in static, metric-based bug prediction. Array (Elsevier), 6, 2020.
7. Cheng Zhou et al.,. Improving software bug-specific named entity recognition with deep neural networks. The Journal of Systems and Software, 165, 2020.
8. Sushan Kumar Pandey et al.,. BPDET: An Effective Software Bug Prediction Model using

Deep Representation and Ensemble Learning Techniques. Expert Systems With Applications. 2019.

9. Sushan Kumar Pandey et al.,. BCV-Predictor: A bug count vector predictor of a successive version of the software system . Knowledge based systems. 2020.
10. Yan Xiao et al.,. Improving Bug Localization with Word Embedding and Enhanced Convolutional Neural Networks. Information and Software Technology.2018.
11. Guo, Shikai, Zhang, Xinyi, Yang, Xi, Chen, Rong, Guo, Chen, Li, Hui, Li, Tingting, 2020. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. Neural Processing Letters 51 (3), 2589–2606.
12. Zhao, Huimin, Zheng, Jianjie, Xu, Junjie, Deng, Wu, 2019. Fault Diagnosis Method Based on Principal Component Analysis and Broad Learning System. IEEE Access 7, 99263–99272.
13. X. Xie, W. Zhang, Y. Yang, and Q. Wang, ‘‘DRETOM: Developer recommendation based on topic models for bug resolution,” 2012, doi: 10.1145/ 2365324.2365329.
14. Xuan, J., Jiang, H., Ren, Z., Yan, J., Luo, Z., 2017. Automatic bug triage using semi-supervised text classification. arXiv.
15. Xuan, Jifeng, Jiang, He, Hu, Yan, Ren, Zhilei, Zou, Weiqin, Luo, Zhongxuan, Wu, Xindong, 2015. Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. 27 (1), 264–280.
16. G. Jeong, S. Kim, and T. Zimmermann, ‘‘Improving bug triage with bug tossing graphs,” 2009, doi: 10.1145/1595696.1595715.
17. Sajedi-Badashian, Ali, Stroulia, Eleni, 2020. Guidelines for evaluating bug-assignment research. Journal of Software: Evolution and Process. 32 (9).
18. Shokripour, Ramin, Anvik, John, Kasirun, Zarinah M., Zamani, Sima, 2015. A time- based approach to automatic bug report assignment. Journal of Systems and Software 102, 109–122.
19. Miholca, D., 2018. An improved approach to software defect prediction using a hybrid machine learning model, in: 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 443–448. doi:10.1109/SYNASC.2018. 00074.
20. Miholca, D., Czibula, G., Zsuzsanna, M., Czibula, I.G., 2017. An unsupervised learning based conceptual coupling measure, in: 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017, pp. 247–254.
21. Miholca, D.L., Czibula, G., 2019. Software defect prediction using a hybrid model based on semantic features learned from the source code, in: Douligeris, C., Karagiannis, D., Apostolou, D. (Eds.), Knowledge Science, Engineering and Management -LNCS, volume 11775, Springer International Publishing, Cham. pp. 262–274.
22. Miholca, D.L., Czibula, G., Czibula, I.G., 2018. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural

networks. Information Sciences 441, 152 – 170.
23. Radjenovic, D., Heri ´ cko, M., Torkar, R., ˇ Zivkovi ˇ c, A., 2013. Software fault prediction metrics: A systematic literature review. In- ˇ formation and Software Technology 55, 1397 – 1418.
24. Wang, S., Liu, T., Tan, L., 2016. Automatically learning semantic features for defect prediction, in: Proc. of the 38th Int. Conf. on Softw. Engineering, ACM, New York, NY, USA. pp. 297–308
25. Jing XY, Wu F, Dong XW, Xu BW. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans Softw Eng 2017;43:321–39
26. Liu M, Miao L, Zhang D. Two-stage cost-sensitive learning for software defect prediction. IEEE Trans Reliab 2014;63:676–86.
27. Yang XX, Tang K, Yao X. A learning-to-rank approach to software defect prediction. IEEE Trans Reliab 2015;64:234–46.
28. Czibula, G., Marian, Z., Czibula, I.G., 2014. Software defect prediction using relational association rule mining. Inf. Sci. 264, 260–278
29. Haghighi, A.S., Dezfuli, M.A., Fakhrahmad, S., 2012. Applying mining schemes to software fault prediction: A proposed approach aimed at test cost reduction, in: Proc/ of the World Congress on Engineering, IEEE Computer Society, Washington, DC, USA. pp. 1–5.
30. Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., Bener, A., 2017. The relationship between evolutionary coupling and defects in large industrial software. J. Softw. Evol. Process 29, 1–19.
31. Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., Bener, A., 2017. The relationship between evolutionary coupling and defects in large industrial software. J. Softw. Evol. Process 29, 1–19
32. Cataldo, M., Mockus, A., Roberts, J.A., Herbsleb, J.D., 2009. Software dependencies, work dependencies, and their impact on failures. IEEE Transactions on Software Engineering 35, 864–878.
33. Chen, H., Martin, B., Daimon, C., Maudsley, S., 2013. Effective use of latent semantic indexing and computational linguistics in biological and biomedical applications. Frontiers in Physiology 4, 8
34. Sushan Kumar Pandey et al.,. BCV-Predictor: A bug count vector predictor of a successive version of the software system . Knowledge based systems. 2020.