DATASETS INCONSISTENCY IN DEFECT PREDICTION

[1]EBUBEOGU AMARACHUKWU FELIX, [2]ONWUEMENE ODIAKA INNOCENT, [3]ADIEZE ENYINNAYA CHUKS, [4]AMAECHI FESTUS CHINEDU, [5]AKUESHI CHIOMA SONIA

[1]University of Malaya, Kuala Lumpur, Malaysia 50603
[2]MAHSA University 42610 Jenjarom, Selangor Malaysia,
[3,4]Kuala Lumpur Metropolitan University College 50350 Kuala Lumpur Malaysia,
[5]University Putra Malaysia 43400 Seri Kembangan, Selangor Malaysia
E-mail: [1]felixbosken@siswa.um.edu.my, [2]fada4all@yahoo.com, [3]enyidaddy08@yahoo.com,
[4]amaechichinedu51@gmail.com, [5]chiomasonia85@gmail.com

Abstract - Data pre-processing is important in software defect prediction. Despite achievements in defect prediction, data reliability remains an issues because of class imbalance; the success of a prediction study relies on the quality of data utilized. In this paper, we present a data pre-processing technique that can accurately identify defective and defect-free modules in a dataset and renders the dataset suitable for defect classification. We applied a top-down technique that considers datasets as a unit to identify both the defective and defect-free classes on 10 projects from the PROMISE repository. The support vector machine classifier achieved an average classification accuracy and specificity of 89.78% and 98.90%; the neural network classifier achieved an area under the receiver operating characteristic curve, Brier score, MCC, precision, and g-mean of 83.53%, 15.12%, 34.37%, 63.04%, and 41.87%; respectively; the naïve Bayes classifier achieved a recall, and a J-coefficient of 78.53% and 31.89%, respectively, and the K-nearest-neighbors classifier achieved an average information score of 36.14%. This manuscript calls for the need to properly pre-process datasets before they are applied in machine learning studies to avoid misleading results.

Keywords - Machine Learning, Data Pre-Processing, Classification Algorithms, Defect Prediction, Imbalanced Data.

I. INTRODUCTION

The need for an accurate data pre-processing approach in software defect prediction remains an achievable goal in software engineering. However, software companies are still faced with the challenges of software defects despite the remarkable achievements in software defects prediction studies. Notably, the success of any prediction study relies on the quality of data applied. And such, data pre-processing remains vital in defect prediction [4], [42].

Notably, data preparation remains a challenging task in defect prediction because of the impurities associated with existing datasets which results to data imbalance likely to affect the prediction accuracy in a research study. However, if a defect prediction study is carried out with a reliable dataset, the finding of such a study may not be biased; hence, the use of poor-quality data may produce a misleading results. In the absence of a substantial effort in data preparation, the outcome on the resources invested in a project could be discouraging [27]. Therefore, this manuscript calls for proper data pre-processing to avoid misleading results in defect prediction studies.In this paper, we present a top-down data pre-processing technique that considers the entire datasets as a unit and carefully identifies both the majority and minority classes to make the dataset suitable for defect prediction. Thereafter, the pre-processed dataset is used in the training of the classifications models; an experiment was conducted using 10 different projects collected from the PROMISE repository [31]. Results show that the support vector machine classifier achieved an average classification accuracy and specificity of 89.78% and 98.90%; the neural network classifier achieved an area under the receiver operating characteristic curve, Brier score, MCC, precision and g-mean of 83.53%, 15.12%, 34.37%, 63.04% and 41.87%; respectively; the naïve Bayes classifier achieved a recall, and a J-coefficient of 78.53% and 31.89%, respectively, and the K-nearest-neighbors classifier achieved an average information score of 36.14%. The remainder of this paper is organized as follows. Section II discusses related work. Section III presents the evaluation metrics. In Section IV, we present our experiment. We then present our results and discussion in Section V. Section VI discusses the potential threats to the validity of this study. Finally, Section VII presents our conclusion and directions for future work.

II. RELATED WORK

A number of studies have proposed approaches to leverage data pre-processing and have achieved noteworthy performance in an attempt to provide a reliable means of pre-processing datasets applied in machine learning and data mining activities. For instance, the authors of [32] outlined important rules necessary to remove erroneous data from National Aeronautics and Space Administration (NASA) datasets. Their effort is remarkable and also encourages re- searchers to make available the source of data applied in a research study, spend additional time to pre-process datasets and report in details the steps followed in pre-preprocessing their datasets.

In addition to these outlined rules for data pre-processing, the authors of [26] identified additional rules that can be applicable to remove defective data. Although these rules have been applied in data pre-processing, datasets are associated with defects following data cleaning. Obviously, datasets obtained from repositories are prone to defect and noise because of impurities in the datasets. These impurities can lead to inaccurate prediction outcome and, consequently, produce misleading results [29].

The erroneous nature of datasets applied in defect prediction studies has become a cause of great concern. The authors of [14] pro- posed a distance-based approach in selecting the data used in defect prediction. They found that a distance-based technique enhances the success rate of cross-project defect prediction. The performance of a classification model can be affected by erroneous datasets. To overcome erroneous datasets, the authors of [2] and [15] proposed a generic algorithm applied in oversampling of data, through the generic algorithm, an evolutionary approach was used to create a synthetic minority instance-oriented classes which indicates an adjustment in the data. This approach showed that data adjustment can improve classification accuracy.

According to [38], dataset attribute influences data pre-processing. That study explicitly stated how data metrics can affect prediction outcome and further reported that poor selection of data metrics reduces the accuracy of prediction models; the impact of poor metric selection can override the impact of the researcher conducting the study, and consequently, lead to misleading results. The authors of [30] compared different data mining approaches including sampling and cost-sensitive, ensemble and hybrid approaches. Though that study relied on the datasets cleaned and pre-processed by Shepperd et al., and concluded that the approaches compared can enhance the classification accuracy of the minority class.

Early detection of defects in datasets can improve data quality. As such, the authors of [9] and [10] proposed some steps through which data pre-processing can be improved by identifying defects early in datasets. As mentioned in [25], using the attributes of datasets in building a prediction model is essential. Notably, these attributes can contain defects that are responsible for data imbalance and also can impact the performance of prediction models. One approach to overcome the imbalanced nature of datasets was reported in [41]. The proposed approach can identify high-impact defects. Their approach is in contrast with the traditional approach in which only the binary classification is considered and the effect of defect is disregarded.

The authors of [11] identified class overlap as a significant factor affecting datasets. Class overlaps render dataset samples invalid and noisy. That study concluded that training a classification model with class overlap-free data improves the performance of such a model. Consequently, training such a model with data containing class overlaps can decrease model performance. To address the class overlap issue, the authors of [1], [3], and [11] proposed a synthetic minority oversampling technique (SMOTE) that can identify the overlapped instances in datasets and improve the prediction out- come. The authors of [3] applied a combination of over-sampling technique for the minority class and under-sampling techniques for the majority class to improve classification accuracy in terms of area under the ROC curve. This approach performs better than single under-sampling as confirmed by the authors of [12] who identified issues with the existing under-sampling techniques in addressing the data imbalance problem. Their study reported that the existing under-sampling techniques cannot improve the understandability of classification models in terms of the AUC. In an attempt to solve the problem with the existing under-sampling techniques that study proposed genetic algorithm-based under-sampling (GAUS) to improve the classification accuracy.

Class imbalance affects the performance of classification models as reported in [1], [8], [13], [16], [24], [28], [30], [34], and [37]. To address this issue, the authors of [43] conducted a laplacian score sampling for data training. Their results confirmed that laplacian score sampling can improve data quality by addressing the imbalance issue in datasets. Another approach to improve the quality of data in prediction is through a relational association rule that can provide a detailed information regarding the attributes of datasets as reported in [5]. The association rule can be applied to identify defects in datasets. Furthermore, few studies such as [21] and [33] have reported that feature selection can as well be used to improve the quality of dataset; applying dimension reduction on datasets can improve the performance of prediction models.

As the issue of class imbalance in a dataset continues to receive attention, the authors of [34] through a reliability-based balancing algorithm proposed an approach that can effectively balance the defective and defect-free classes in datasets. The authors of [22] and [36] categorized some of the proposed approaches into data sampling, cost-sensitive learning, algorithm modification and bagging, and boosting-based. These approaches cause the loss of some essential parts of data, which may lead to data overfitting as reported in [36].

III. EVALUATION METRICS

In this section, we present a list of the evaluation metrics applied in determining the performances of the classification models applied in this study.

Classification accuracy (CA): The ratio of the number of correct predictions to the total number of predictions. The CA is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Precision: The proportion of all predicted positive instances that are correctly predicted.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Recall (sensitivity): The proportion of all actually positive in- stances that are correctly predicted.

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Specificity: The proportion of all actually negative instances that are correctly predicted.

$$Specificity = \frac{TN}{TN + FP} \quad (4)$$

Matthews correlation coefficient (MCC): Applied in binary
classification as a measure of the achieved performance, the MCC is an important correlation coefficient that represents an overall assessment of the observed and predicted classes in binary classification. It considers all four aspects of the confusion matrix, including the true negatives [23]. It is calculated directly from the confusion matrix as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (5)$$

where TP = the number of instances that are actually positive and are also predicted to be positive, also called true positives; FP = the number of instances that are negative but are predicted to be positive, also called false positives; FN= the number of positive instances that are incorrectly predicted to be negative, also called false negatives and TN = the number of negative instances that are correctly predicted to be negative, also called true negatives.

J-coefficient: The sum of the recall and specificity minus one. It is equivalent to the recall minus the false positive rate (FPR).

$$\textbf{J-coef} = Recall - FPR = Recall + Specificity - 1 \quad (6)$$

F-measure (F-score or F1): The harmonic average of the precision and recall. It is calculated using the following formula:

$$\textbf{F-score} = 2 \times (Precision \times Recall) / (Precision + Recall) \quad (7)$$

Area under the ROC curve (AUC): A metric based on a graphical representation of the performance of a binary classifier. It can also be used to compare different classifiers. The sensitivity, which represents the proportion of actually positive instances that are correctly classified, is plotted on the y axis against 1 minus the specificity, which represents the proportion of actually negative instances that are correctly classified, on the x axis. A larger AUC value indicates better classifier performance. As reported in [39], the possible values of the AUC lie between 0 and 1. An AUC value of 0 indicates the worst possible performance, whereas a value of 1 indicates the best possible performance.

Geometric mean (G-mean): The square root of the product of the recall and precision, as proposed in [20] and [19]. Thus, the geometric mean is calculated as follows:

$$\textbf{G-mean} = \sqrt{Recall \times Precision} \quad (8)$$

Equation (8) represents the geometric mean for the minority target class (True class).

Brier score (BS): A score that measures the accuracy of probabilistic predictions. The authors of [39] applied the Brier score in their study to measure the gap between the estimated probability and the result achieved. A Brier score of 0 is the best achievable score, and 1 is the worst achievable score. The Brier score is calculated using the following formula:

$$BS = \frac{1}{N} \sum_{t=1}^{N} (f_t - o_t)^2 \quad (9)$$

where
f_t = the forecasted likelihood of occurrence,
o_t = the true event result at time t, and
N = the number of predicted instances.

Information score (IS): The average amount of information per classified instance, as defined in [18]. The average classifier performance $(\overline{P}_{av})$, which represents the mean performance of a given classifier for the target class of True is calculated as follows:

$$\overline{P}_{av} = \frac{1}{n} \sum_{i=1}^{n} p_n = \frac{1}{n} (p_1 + p_2 + \ldots p_n) \quad (10)$$

where
n = the number of experiments performed,
P_{av} = the average classifier performance, and
p_i = the individual performance in the i-th experiment (here, each experiment corresponds to a different project).

IV. EXPERIMENTS

Our experiment was conducted with a computer having a 1.70 GHz Intel (R) Core (TM) i5-3317U CPU and 4 GB of RAM running 64-bit Windows 10. We evaluated the performance of the classification models using Orange 2.7.

A. Data collection and pre-processing technique

In the current this study, 10 different datasets were used from the PROMISE repository [31]. These

NASA Metric Data Program datasets are widely used in software defect prediction studies. Furthermore, these datasets contain a significant amount of imbalanced data that require pre-processing and cleaning to avoid misleading results.

Table 1 presents the statistics of 10 datasets we collected from the PROMISE repository. Each field in the table, from left to right, represents the dataset name, the number of modules, the number of defective modules, the number of attributes, and the percentage of defective modules in each dataset. Figure 1 represents a flow diagram of data pre-processing. The first phase of our experiment is data cleaning. At this phase, we analyzed the datasets to understand in detail the 10 datasets applied in this study.

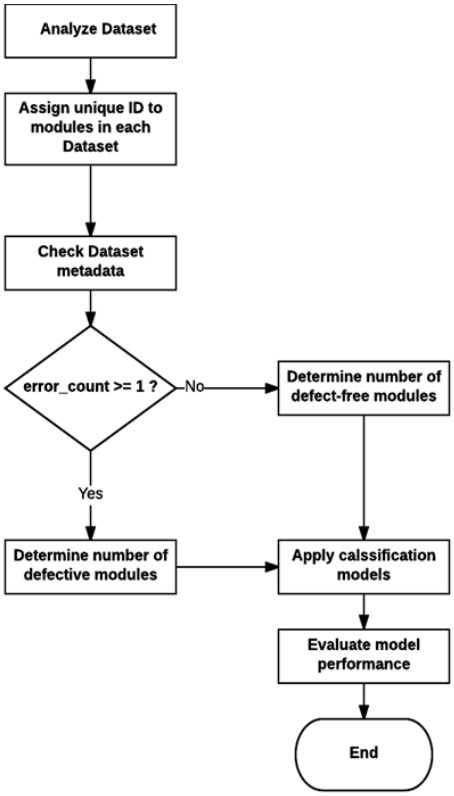


Figure 1: Flow diagram of data pre-processing

Dataset	No. of modules	No. of Defective modules	No. of Attributes	% Defective modules
KC1	2109	326	21	15.46
KC2	522	107	21	20.50
KC3	458	43	39	9.39
MC1	9466	68	38	0.72
MC2	161	48	39	29.81
MW1	403	31	37	7.69
PC1	1109	77	21	6.94
PC2	5589	21	36	0.38
PC3	1563	159	37	10.17
PC4	1458	174	37	11.93
Total	22,838	1054	326	4.62

Table 1: Dataset statistics

We determined the source, size, and format of the datasets. Second, for each dataset, we assigned a unique identifier to accurately determine the minority class. The number of defective modules, number of attributes, and percentage of defects in each dataset were determined and recorded. We analyzed the individual module of each dataset. The unique identifiers assigned to all the modules in each dataset. The unique identifiers assigned to all the modules in each dataset DS_1−n simplified the analysis of these modules.

In identifying both the defective and defect-free modules (MD1−n), a binary decision is made on each module as a function of (0, 1),

f : $MD_1 \rightarrow$ n (0, 1) to ensure that the accurate records of the mi- nority class is obtained. Thus, an error count greater than or equal to 1indicates a defective module; meanwhile, an error count of 0 indicates a defect-free module. The process is repeated for all the modules in each dataset as presented in algorithm 1. Up to 80% of the pre-processed data was used as the training data, and our validation was performed using the remaining 20% of the data. The six selected classifiers were implemented on each dataset to deter- mine their individual classification performances; thereafter, the average classification performance of each classifier on all datasets was determined accordingly.

B. Algorithm 1: Data pre-processing technique

Algorithm 1 Top-down data pre-processing technique for defect prediction

procedure STEPS FROM DATA PRE-PROCESSING TO DEFECT PREDICTION
Input: Datasets DS
Output: Pre-processed data for average classifier performance $\overline{P}_{av}$ based on $M \times N$ cross-validation
  M = 10 /* number of folds for cross-validation
  N = 10 /* number of repetitions of training/testing
  Analyze datasets (DS_{1-n})
  Analyze modules MD_{1-n} of each (DS_{1-n})
  for i = 1 to n **do**
    Assign unique ID to MD_{1-n} of each (DS_{1-n})
    Check attributes in each module of each dataset
    Check incomplete and missing values
    Check outliers and inliers
    Identify defective and defect-free modules MD_{1-n}
    for i = 1 to n **do**
      $f : MD_{1-n} \rightarrow (0, 1)$
      if error_count $\geq$ 1 **then**
        $f : (MD) = 1$
        MD = defective
        Record number of defects
        Calculate percentage of defects
      else
        $f : (MD) = 0$
        MD = defect-free
      end if
      Divide DS_{1-n} into two parts: 80% and 20%, for **training** and **testing**, respectively
      Evaluate individual classifier performance p
      Calculate average classifier performance $\overline{P}_{av} = \frac{1}{n}\sum_{i=1}^{n} p_n = \frac{1}{n}(p_1 + p_2 + \ldots p_n)$
      Repeat the entire algorithm for each dataset DS
    end for
  end for
end procedure

V. RESULTS AND DISCUSSION

In this section, we present the results obtained through the proposed dataset pre-processing technique applied in this study as presented in algorithm 1. Although the

pre-processing of the datasets was time consuming, we still ensured that enough time was invested in cleaning the datasets before applying them in model training. When addressing imbalanced data, both the minority and majority classes should be correctly identified to avoid misleading results during classification. In this study, we carefully identified both the minority and majority classes in each dataset.
We used numerous evaluation metrics to assess the consistency of the classifiers' performance, including the CA, AUC, Brier score, MCC, recall, specificity, J-coefficient, f-score, precision, g-mean, and information score. We applied a large number of metrics because the use of more evaluation metrics results in a significant likelihood of obtaining useful information on the accuracy of the models and in a reliable prediction [17].

Table 2 presents the average performance of each of the six selected classification models. In terms of the CA and specificity, the SVM model achieved the best performance of 89.78% and 89.90%, respectively. The neural network model achieved the best AUC, Brier score, precision, G-mean, and MCC performance, with 83.53%, 15.12%, 63.04%, 41.87%, and 34.37%, respectively. The Brier score is an important performance metric for probabilistic prediction; a score of 100% indicates the worst achievable score, whereas a score close to 0% indicates good performance. Therefore, the neural network model, with a Brier score of 15.12%, achieved the best performance in terms of this metric, whereas the naïve Bayes model showed the worst performance of 78.53%. In terms of the J-coefficient, the naïve Bayes model achieved the best performance of 31.89% in the classification of the minority classes. The KNN model achieved the best performance of 3.61% overall in terms of the average amount of information per classified instance, as the information score indicated, and an F-score of 37.92%.

In terms of the recall, which indicates the proportion of actually positive instances that are correctly classified, the naïve Bayes model achieved a far superior result of 71.20% and J-coeff of 31.89. Furthermore, the results of this study are significant compared with the results reported in [40] in terms of the CA, recall, specificity, precision, MCC, and AUC.

We compared the AUC performances of the six selected classifiers with respect to the minority classes in Figures 2 to 11 because the AUC depicts all performance-related information and allows the essential links related to classifier performance to be clearly observed [6].The present study shows that incorrect pre-processing of a dataset can generate misleading results in defect prediction study. Also, the possibility of the classifiers losing their performance capabilities when impurities are present in datasets.

VI. THREATS TO VALIDITY

This section presents the potential threats to the validity of this study, of which we differentiated three types, namely, construct, internal, and external threats.

A. Construct threats

We constructed an algorithm that shows steps in pre-processing the datasets applied in this study and equations for calculating the average performances of the selected classification models to determine unbiased results. These algorithm and equations may pose some threats to the validity of this study.

B. Internal Threats

Previous studies have achieved noteworthy results in terms of data pre-processing for binary defect classification. Our proposed data pre-processing technique also aimed at leveraging data pre-processing in binary classification. The consideration of the average performance result obtained in this study may have affected the validity of this study. Thus, a more accurate approach for data pre-processing in binary defect classification is needed.

C. External Threats

In this study, our experiments were conducted using 10 datasets that may pose some threats to the validity of this study. These datasets are widely used NASA projects collected from the PROMISE repository and have been used in several previous defect prediction studies. For example, the authors of [35], [28], [7], and [43] applied these NASA datasets in their early studies. The correctness of our results depends on our ability to correctly identify the minority modules in these datasets during data pre-processing. We encourage other researchers to replicate this study to improve the overall evaluation of binary defect classification.

VII. CONCLUSION AND FUTURE WORK

The issue of data reliability in defect prediction studies has yet to be resolved. Therefore, in the present study, we have presented a top-down data pre-processing approach to accurately pre-process datasets before applying them to classification models. This method can provide practical outputs to software defect prediction studies because it exposes the necessary information about the dataset. In our experiments, we used 10 PROMISE datasets containing 22,838 instances. The support vector machine classifier achieved an average classification accuracy and specificity of 89.78% and 98.90%, respectively. The neural network classifier achieved an area un- der the ROC curve, Brier score, MCC, precision, and g-mean of 83.53%, 15.12%, 34.37%, 63.04%, and 41.87%; respectively. The naïve Bayes classifier achieved a recall, and a J-coefficient of 78.53% and 31.89%, and the K-nearest-neighbors classifier achieved an average information score of 36.14%.

Classifier	CA	Sen	Spec	AUC	IS	F1	Prec	Recall	Brier	MCC	J-coeff	G-mean
RF	0.8948	0.1819	0.9772	0.8058	-0.0319	0.2667	0.5527	0.1819	0.1525	0.2649	0.1591	0.3171
SVM	**0.8978**	0.1460	**0.9890**	0.7000	0.0229	0.2143	0.5354	0.1460	0.1605	0.2301	0.1350	0.2796
KNN	0.8813	0.3464	0.9390	0.7418	**0.0361**	**0.3792**	0.4220	0.3464	0.1888	0.3112	0.2854	0.3823
Neural Network	0.8969	0.2781	0.9638	**0.8353**	0.0057	0.3631	**0.6304**	0.2781	**0.1512**	**0.3437**	0.2419	**0.4187**
LR	0.8960	0.2387	0.9661	0.8226	-0.0344	0.3202	0.5302	0.2387	0.1532	0.2966	0.2048	0.3558
Naïve Bayes	0.5962	0.7120	0.6069	0.7196	-1.7393	0.3121	0.2330	**0.7120**	0.7853	0.2322	**0.3189**	0.4073

Table 2: Average classifier performances for target class = True

The results of the present study must be confirmed to verify the acceptability of our approach for data pre-processing. In our future work, we will investigate the performance of the selected classifiers for both true and false target classes, compare the results of the average classifiers performance, and apply this data pre-processing approach on datasets from other sources. This paper also encourages other researchers to replicate the steps applied herein and calls for the need to properly pre-process datasets before they are applied in machine learning studies to avoid misleading results.

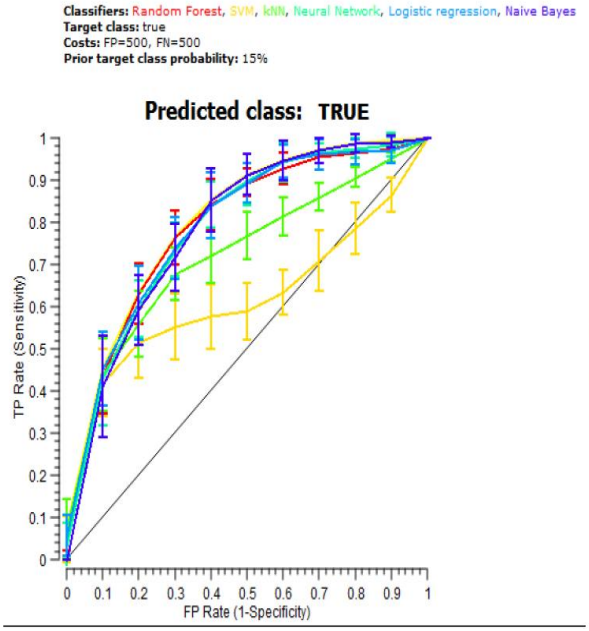


Fig. 2: ROC curve for KC1, target = True

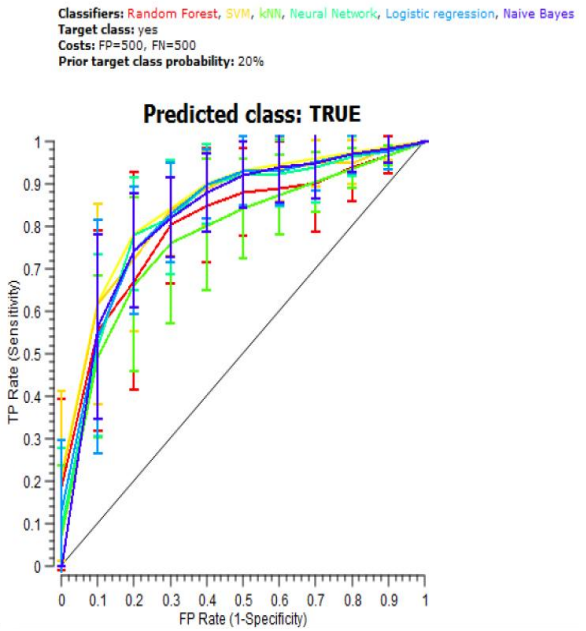


Fig 3: ROC Curve for KC2 target = True

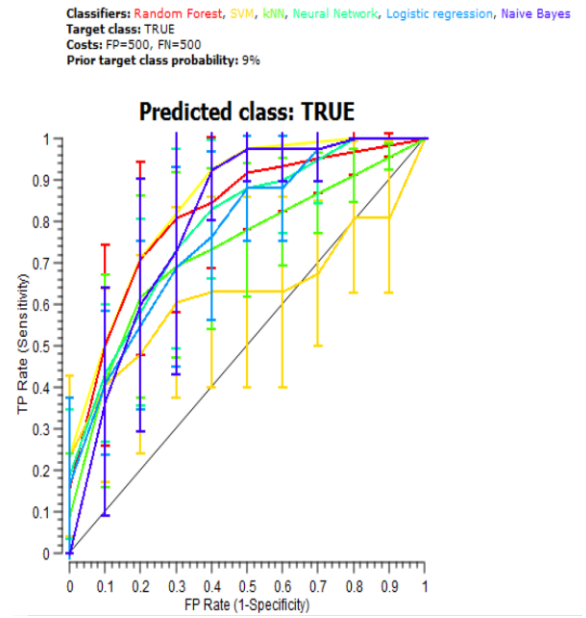


Fig. 4: ROC curve for KC3, target = True

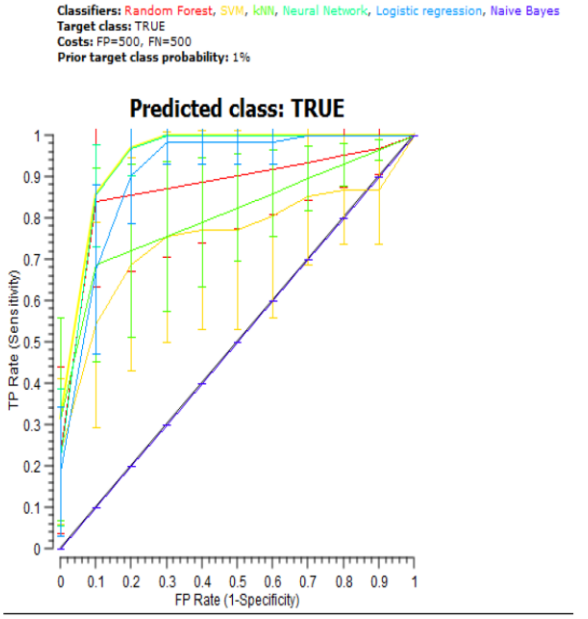


Fig. 5: ROC curve for MC1, target = True

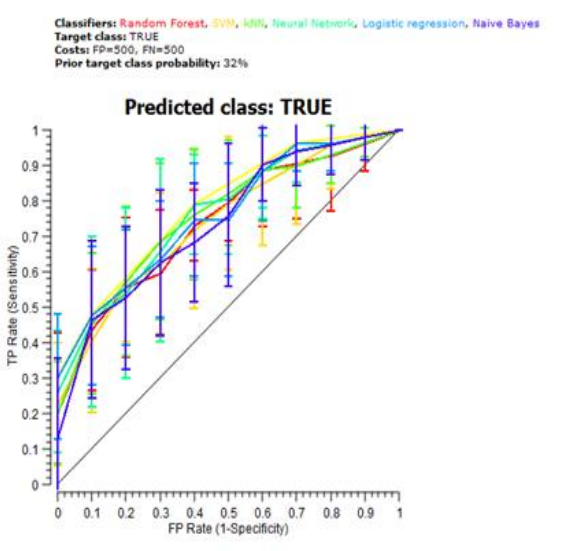


Fig. 6: ROC curve for MC2, target = True

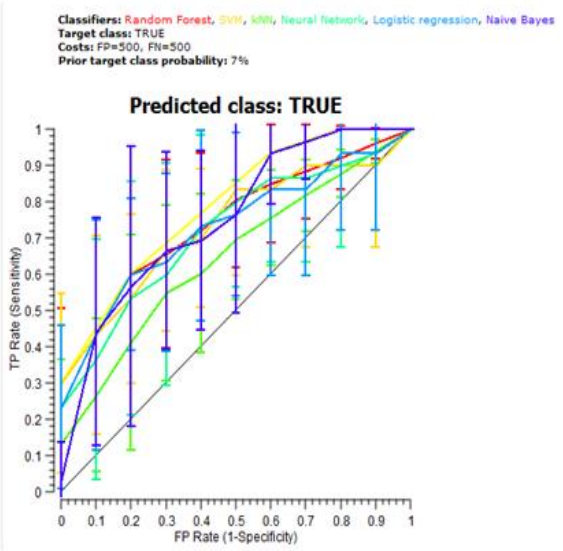


Fig. 7: ROC curve for MW1, target = True

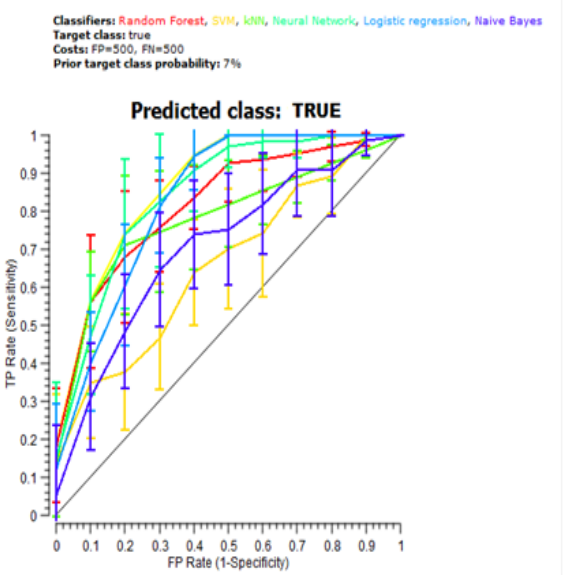


Fig. 8: ROC curve for PC1, target = True

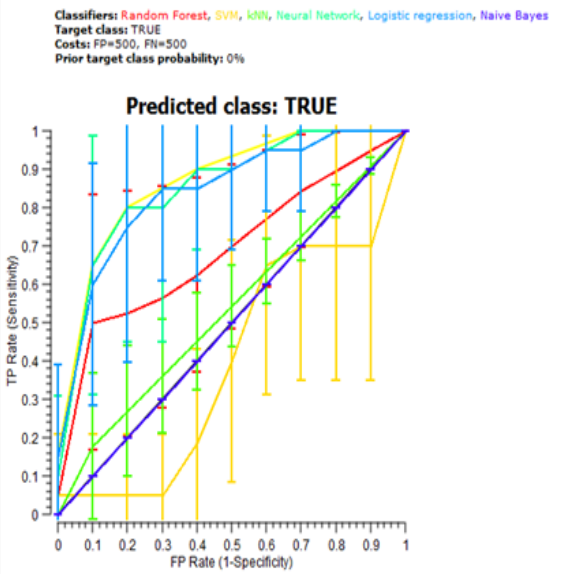


Fig. 9: ROC curve for PC3, target = True

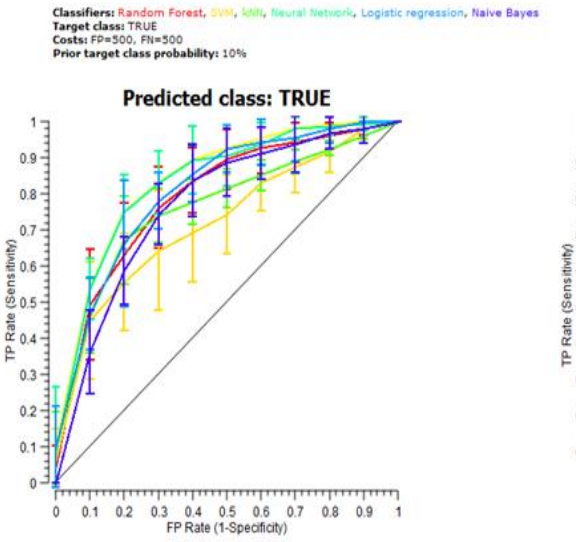


Fig. 10: ROC curve for PC3, target = True

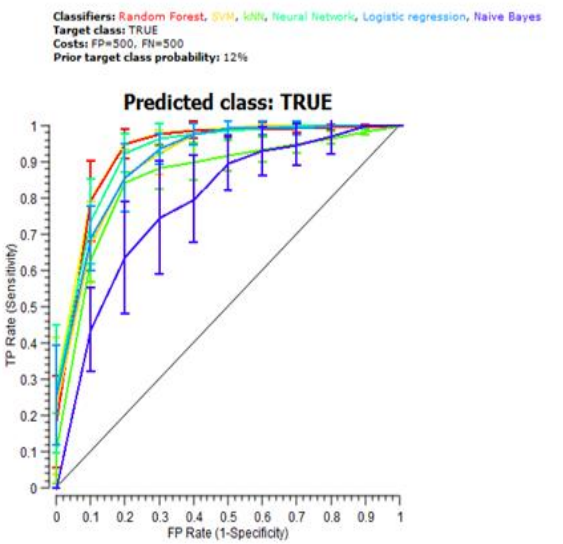


Fig. 11: ROC curve for PC4, target = True

ACKNOWLEDGMENTS

The researchers would like to express our profound gratitude to the unanimous reviewers for their time and thoughtful comments while reviewing this work.

REFERENCES

[1] Gustavo Batista, Diego Silva, and Ronaldo Prati. 2012. An experimental design to evaluate class imbalance treatment methods. In Machine Learning and Applications (ICMLA), 2012 11th International Conference on, Vol. 2. IEEE, 95–101.

[2] Marcelo Beckmann, Beatriz Souza LP de Lima, and Nelson FF Ebecken. 2011. Genetic algorithms as a pre-processing strategy for imbalanced datasets. In Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. ACM, 131–132 .

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16 (2002), 321–357.

[4] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Editorial: special issue on learning from imbalanced data sets. ACM Sigkdd Explorations Newsletter 6, 1 (2004), 1–6.

[5] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. 2014. Software defect prediction using relational association rule mining. Information Sciences 264 (2014), 260–278.

[6] Chris Drummond and Robert C Holte. 2006. Cost curves: An improved method for visualizing classifier performance. Machine learning 65, 1 (2006), 95–130.

[7] Karim O Elish and Mahmoud O Elish. 2008. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software 81, 5 (2008), 649–660.

[8] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. 2004. A multiple resampling method for learning from imbalanced data sets. Computational intelligence 20, 1 (2004), 18–36.

[9] David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. 2011. The misuse of the NASA metrics data program data sets for automated software defect prediction. In Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on. IET, 96–103.

[10] David Gray, David Bowes, Neil Davey, Y Sun, and Bruce Christianson. 2012. Reflections on the NASA MDP data sets. IET software 6, 6 (2012), 549–558.

[11] Shivani Gupta and Atul Gupta. 2017. A set of measures designed to identify overlapped instances in software defect prediction. Computing (2017), 1–26.

[12] Jihyun Ha and Jong-Seok Lee. 2016. A New Under-Sampling Method Using Genetic Algorithm for Imbalanced Data Classification. In Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication. ACM, 95.

[13] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology 59 (2015), 170–190.

[14] Steffen Herbold. 2013. Training data selection for cross-project defect prediction. In Proceedings of the 9th International Conference on Predictive Models in Software Engineering. ACM, 6.

[15] Theodoros Iliou, Christos-Nikolaos Anagnostopoulos, Marina Nerantzaki, and George Anastassopoulos. 2015. A novel machine learning data preprocessing method for enhancing classification algorithms performance. In Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS). ACM, 11.

[16] Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. Intelligent data analysis 6, 5 (2002), 429–449.

[17] Xiaoyuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 496–507.

[18] Igor Kononenko and Ivan Bratko. 1991. Information-based evaluation criterion for classifier's performance. Machine Learning 6, 1 (1991), 67–80.

[19] Miroslav Kubat, Robert C Holte, and Stan Matwin. 1998. Machine learning for the detection of oil spills in satellite radar images. Machine learning 30, 2-3 (1998), 195–215.

[20] Miroslav Kubat, Stan Matwin, and others. 1997. Addressing the curse of imbalanced training sets: one-sided selection. In ICML, Vol. 97. Nashville, USA, 179–186.

[21] Issam H Laradji, Mohammad Alshayeb, and Lahouari Ghouti. 2015. Software defect prediction using ensemble learning on selected features. Information and Software Technology 58 (2015), 388–402.

[22] López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. 2013. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Information Sciences 250 (2013), 113–141.

[23] Zaheed Mahmood, David Bowes, Peter CR Lane, and Tracy Hall. 2015. What is the Impact of Imbalance on Software Defect Prediction Performance? In Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering. ACM, 4.

[24] Wentao Mao, Jinwan Wang, Ling He, and Yangyang Tian. 2017. Online sequential prediction of imbalance data with two-stage hybrid strategy by extreme learning machine. Neurocomputing (2017).

[25] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. IEEE transactions on software engineering 33, 1 (2007).

[26] Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. 2016. The jinx on the NASA software defect data sets. In Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. ACM, 13.

[27] Dorian Pyle. 1999. Data preparation for data mining. Vol. 1. morgan kaufmann.

[28] YU Qiao, Shujuan Jiang, and Yanmei Zhang. 2017. The Performance Stability of Defect Prediction Models with Class Imbalance: An Empirical Study. (2017).

[29] RS Ramya, S Priyadarshini, and S Karthik. 2012. An enhanced secure preserving for pre-processed data using DMI and PCRBAC algorithm. In Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology. ACM, 283–287.

[30] Daniel Rodriguez, Israel Herraiz, Rachel Harrison, Javier Dolado, and José C Riquelme. 2014. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, 43.

[31] J. Sayyad Shirabad and T.J. Menzies. 2005. The PROMISE Repository of Software Engineering Databases.". School of Information Technology and Engineering, University of Ottawa, Canada". (2005). "http://promise.site.uottawa.ca/SERepository"

[32] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data quality: Some comments on the nasa software defect datasets. IEEE Transactions on Software Engineering 39, 9 (2013), 1208–1215.

[33] Kandarp P Shroff and Hardik H Maheta. 2015. A comparative study of various feature selection techniques in high-dimensional data set to improve classification accuracy. In Computer Communication and Informatics (ICCCI), 2015 International Conference on. IEEE, 1–6.

[34] Paolo Soda. 2011. A multi-objective optimisation approach for class imbalance learning. Pattern Recognition 44, 8 (2011), 1801–1810.

[35] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. 2011. A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering 37, 3 (2011), 356–370.

[36] Zhongbin Sun, Qinbao Song, Xiaoyan Zhu, Heli Sun, Baowen Xu, and Yuming Zhou. 2015. A novel ensemble method for classifying imbalanced data. Pattern Recognition 48, 5 (2015), 1623–1637.

[37] Yuchun Tang, Yan-Qing Zhang, Nitesh V Chawla, and Sven Krasser. 2009. SVMs modeling for highly imbalanced classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39, 1 (2009), 281–288.

[38] Chakkrit Tantithamthavorn. 2016. Towards a better understanding of the impact of experimental components on defect prediction modelling. In Proceedings of the 38th International Conference on Software Engineering Companion. ACM, 867–870.

[39] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering 43, 1 (2017), 1–18.

[40] Xiao Xuan, David Lo, Xin Xia, and Yuan Tian. 2015. Evaluating defect prediction approaches using a massive set of metrics: An empirical study. In Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, 1644–1647.

[41] Xin-Li Yang, David Lo, Xin Xia, Qiao Huang, and Jian-Ling Sun. 2017. HighImpact Bug Report Identification with Imbalanced Learning Strategies. J. Comput. Sci. & Technol 32, 1 (2017).

[42] Mohammed J Zaki, Wagner Meira Jr, and Wagner Meira. 2014. Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press.

[43] Zhi-Wu Zhang, Xiao-Yuan Jing, and Tie-Jian Wang. 2017. Label propagation based semi-supervised learning for software defect prediction. Automated Software Engineering 24, 1 (2017), 47–69

★★★