Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 865–869
DOI: 10.15439/2023F363
ISSN 2300-5963 ACSIS, Vol. 35

Experiments on Software Error Prediction Using Decision Tree and Random Forest Algorithms

Ilona Bluemke
0000-0002-2894-5976
Warsaw University of Technology,
Institute of Computer Science,
Nowowiejska 15/19 00-665
Warsaw, Poland
Email: Ilona.Bluemke@pw.edu.pl

Paweł Borsukiewicz
00000-0002-2934-6115
Warsaw University of Technology,
Institute of Computer Science,
Nowowiejska 15/19 00-665
Warsaw, Poland
Email:
pborsukiewicz99@gmail.com

***Abstract*—Machine learning algorithms are widely used in the assessment of error-proneness in software. We conducted several experiments with error prediction on public PROMISE repository. We used Decision Tree and Random Forest algorithms. We also examined techniques aiming at the improvement of performance and accuracy of the model – such as oversampling, hyperparameter optimization or threshold adjustment. The outcome of our experiments suggests that Random Forest algorithm, with 100 – 1000 trees, can be used to obtain high values of evaluation parameters such as accuracy and balanced accuracy. However, it has to be implemented with a set of techniques countering imbalance of the datasets used to assure high values of precision and recall that correspond with correct detection of erroneous software. Additionally, it was shown that the oversampling and hyperparameter optimization could be reliably applied to the algorithm, while threshold adjustment technique was not found to be consistent.**

***Index Terms*—error prediction, error proneness, decision tree, random forest, PROMISE repository, machine learning.**

I. Introduction

HIGH quality of software is essential in software production. Software testing is a key part of the software development process, especially for quality assurance, but it requires a lot of time and resources. It is estimated that testing activities consume more than half of the cost of the whole software development process [1], [2]. Application of software error prediction is almost 30% cheaper than testing, according to recent studies [3]. Ability to predict faulty components in the early phase may highly increase cost-effectiveness. There is abundant literature on software defect prediction, some works are mentioned in section II. There are also several systematic literature reviews on this subject eq. [4], [5], [6].

Contemporary solutions put strong emphasis on the usage of machine learning algorithms in software defect prediction. However there are many papers on software error prediction we decided to conduct some experiments on the usage of machine learning algorithms – Decision Tree and Random Forest on public PROMISE repository. We wanted to examine the relationship between computational effort and accuracy of obtained results. Techniques aiming at the improvement of performance and accuracy of the model – such as oversampling, hyperparameter optimization or threshold adjustment were also analyzed and addressed.

Paper is structured as follows. Section II introduces related work. Methodology is stated in section III. The experiment and its analysis are presented in section IV. Section V concludes the paper, highlighting some issues.

II. Related Work

Lately various studies have been conducted in order to compare algorithms used to determine error-proneness. Many of them were based on wide variety of machine learning algorithms [7] and used formerly mentioned PROMISE [8] repository as its dataset.

Random Forest, Naive Bayes, J48, Immunos 1 & 2, CLONALG, AIRS (Artificial Immune Recognition System algorithm) 1 & 2 and AIRS 2 Parallel were algorithms studied by Catal et al. [9],[10]. When AIRS algorithm was taken into consideration the researchers concluded that the best results were obtained when CK metrics and LOC metric were combined.

AIRS is a system inspired by human's immunological system with B-cells and T-cells as our guardians. In the past the main application was supervised learning, however, in 2001 it was demonstrated that algorithms based on this system can be also used for classification domain [9]. Other slightly different algorithms using AIS include CLONALG [11] and Immunos [12].

As previously mentioned, Catal et al. [9] have also directed their research towards Random Forests (RF). This algorithm is based on existence of high number of so called "trees". Each tree is independent, but at the end the majority voting result of all the trees in the forest is taken as a final result. Model can be trained with various performance enhancing

techniques. One of them is bootstrap sampling or bagging (bootstrap aggregating), which means randomly taking only small part of the dataset for each training process and repeating it multiple times. In case of singular trees one may also consider pruning – technique based on removal of some of the nodes that are not essential and could result in overfitting. Nonetheless, as it was stated in study by L. Breiman [13] that procedure is not needed in case of RF algorithm when bagging is applied. According to the study by Mundada et al. [14] RF is the best algorithm for NASA datasets, which are a part of the PROMISE repository.

J48 is an algorithm that implements C4.5 decision tree learning [9], [15].

Mundada et al. [14] directed their study towards Artificial Neural Network (ANN) and Resilient Back Propagation (RBP) using JM1 dataset. As a result of this experiment, researchers concluded that the better accuracy of ANN algorithm was reached, when compared with already existing analytical models.

Bishnu et al. [16] studied performance of QUAD Tree-Based K-Means Clustering Algorithm using AR3, AR4 and AR5 datasets. It was concluded that error rates of this algorithm are comparable to the ones obtained with other algorithms. In order to obtain the best values, data sets partitioning has to ensure that the sum of distances within the clusters is properly reduced [17].

Okutan and Taner [18] used 9 datasets from PROMISE dataset to research Bayesian Networks. The results of the study stated that the LOC, RFC and LOCQ metrics are the best choice due to their effectiveness when this algorithm is considered. An important advantage of this network is the fact that it can be used even when the metrics are incomplete for some sets.

Kumudha et al. [19] have introduced a significant development in the field. Their research focused on conventional Radial Basis Function Neural Network (RBFNN) and the novel Adaptive Dimensional Biogeography Based Optimization Model (ADBBO). Having based the research in CM1, JM1, KC1, KC2, and PC1 datasets, results obtained during this study showed that newly proposed method is more effective when compared with already existing algorithms.

Gupta and Gupta. [20] have used derived metrics from PROMISE repository datasets to determine fault classification. In this study, the emphasis was put on the data distribution and skewness rather than the algorithms itself.

Erturk and Akcapinar [21] have used projects from PROMISE repository to conduct research on Fuzzy Inference Systems (FIS) [22] and Adaptive Neuro Fuzzy Inference System (ANFIS). Those new methods deploy iterative software error-proneness prediction to automatically detect fault prone sections.

Alighardashi et al. [23] have used ten PROMISE and NASA datasets to test feature selection method. Five filter methods were used during this study. Weighted filter (WF) method was determined to be able to detect best features that would allow the fault prediction accuracy to be the increased in the fastest way possible.

After preparation of the above related work recent publications in this domain appeared e.g. [24], [25], [26], [27]. These works are not included in the above text.

III. Methodology

For the purpose of the experiment Decision Tree and Random Forest algorithms [9] were selected. Random Forest, being composed of Decision Trees, is a flexible algorithm that can be applied both in classification and regression problems. As the purpose of the experiment is to assess error-proneness of the samples within the datasets, one can consider the problem primarily as classification problem. One can also assume that there are two classes of results - code either is correct or incorrect. However, as it is possible to apply regression version of Random Forest algorithm in this particular scenario and to some extend treat its values as a probability of existence of an error, it was used and compared against its classification counterpart.

Python [28] was used for implementation and functions from NumPy [29], Pandas [30], scikit-learn [31] and imbalanced-learn [32] libraries were a basis for the implementation of the algorithm and the evaluation of performance such as accuracy, recall, precision, etc.

Hyperparameter optimization was not initially performed as some hyperparameters, such as forest size, were the focus of the study and in order to better understand what are the disadvantages of the basic model. Optimization of multiple hyperparameters would result in largely extended training times, especially if larger forests were to be considered. In further parts optimization techniques were used in order to increase the performance and assess the full potential of Random Forest in error-prediction field.

Similarly, SMOTE [33] oversampling was another technique that was not used initially, but was introduced later in order to improve the performance of the model. By default bootstrapping was enabled throughout the whole experiment and pruning was not performed as recommended by Breiman [13]. The order of processes is presented in Fig1.

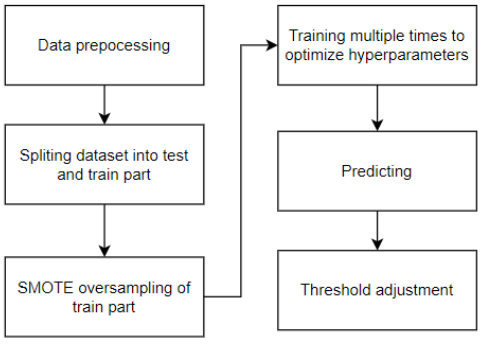

Fig 1. Phases of experiment

IV. EXPERIMENTS

Public NASA datasets were used, including those available within the PROMISE repository. The scope of the tests includes CM1, KC1, KC2, PC1, PC2 and PC3 sets.

Experiments were conducted on two devices: personal laptop and virtual machine provided by the Warsaw University of Technology.

A. Experiment results

Experiment was divided into a series of incremental steps. Each step introduced new technique or method, or combined those previously assessed. The final outcome was the process presented in Fig 1.

Initially the datasets were analyzed and some results of analysis are presented in Table I. It can be seen that datasets are highly imbalanced.

TABLE I.
DATASET PROPERTIES

Dataset	Dataset size	Error-free software in dataset [%]	Number of metrics
CM1	498	90.2	22
KC2	522	79.5	22
PC1	1109	93.1	22
PC3	1563	89.8	38
KC1	2109	84.6	22
PC2	5589	99.6	37

Before introduction of any enhancement mechanism, it was assumed that the optimal number of trees for the experiment should be in the range from 100 to 1000. This observation was confirmed throughout the experiment. Above 1000 trees any substantial improvement to the evaluation metrics was not observed as shown in Table II. It is worth noting that the training time grows almost linearly with the forest size, therefore lower forest sizes are generally preferred when training time is limited. Even though, all experiments were performed for forest sizes of 1 (decision tree), 10, 100, 1000, 10000, 25000 and 50000, results provided in this paper were obtained for forests with 1000 trees unless stated otherwise.

TABLE II.
BASIC CLASSIFICATION ACCURACY

Number of trees	1	10	100	1000	10000
Dataset	Accuracy				
CM1	0.81	0.87	0.86	0.86	0.86
KC1	0.81	0.84	0.85	0.85	0.85
KC2	0.74	0.82	0.82	0.80	0.82
PC1	0.90	0.91	0.92	0.92	0.92
PC2	0.99	0.99	0.99	0.99	0.99
PC3	0.85	0.90	0.90	0.90	0.90

Even though, accuracy score is rather high, it is usually very close to the percentage of error-free samples in the dataset. Analyzing metrics such as recall, precision, and balanced accuracy it was clearly visible that model tends to classify vast majority of samples as error-free, thus making from few to even no useful detections (true positive values) as in case of dataset CM1, as shown in Table III.

TABLE III.
BASIC CLASSIFICATION FOR CM1 DATASET

Accuracy	0.86
Recall	0.00
Precision	0.00
Balanced accuracy	0.49
F1 score	0.00

First technique aiming to improve error-prone software detection that was assessed was threshold adjustment. As this problem deals with two classes – erroneous software and error-free software it is primarily a classification problem. However, one may take a regression approach with the 0.5 threshold as a default one. Adjustment of that selection threshold, either its lowering or increasing, could potentially lead to better classification. Exemplary outcome of the experiment for CM1 dataset is presented in Fig 2.

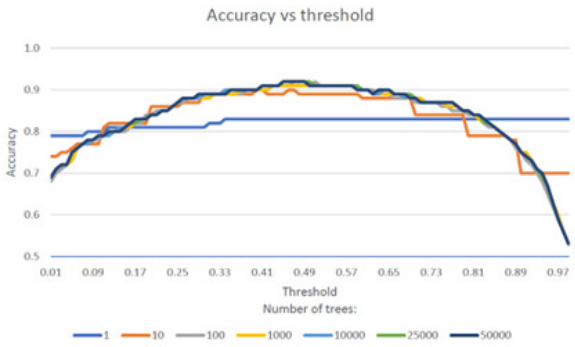

Fig 2. Threshold adjustment for CM1 dataset

Slight change of the threshold around the 0.5 mark in some cases, provided some minor improvements. Nonetheless, no direct pattern could be established observing various datasets and forest sizes. Moreover, it did not help in any way to tackle the problem of datasets imbalance, still being strongly biased towards error-free classes. Given method was also combined with subsequently described methods, however, at each step it was too unpredictable and, as a result, it was discarded.

Further studies were done on SMOTE oversampling technique. As presented in Table IV, it aimed to reduce the learning bias resulting from dataset imbalance by equalizing proportions via creating artificial samples.

TABLE IV.
DATASET SIZES BEFORE AND AFTER OVERSAMPLING

	Before SMOTE		After SMOTE	
	Faulty	Not faulty	Faulty	Not faulty
CM1	45	428	428	428
KC1	307	1696	1696	1696
KC2	96	399	399	399
PC1	73	980	980	980
PC2	21	5288	5288	5288
PC3	151	1333	1333	1333

As presented in Table V and Table VI, significant improvements could be noticed. Not only, was the accuracy improved, but more importantly precision and recall values also, which indicate that the true error-free software was now properly detected and classified.

TABLE V.
DATASET SIZES BEFORE AND AFTER OVERSAMPLING

	Accuracy	
	Before	After
CM1	0.86	0.94
KC1	0.85	0.91
KC2	0.80	0.85
PC1	0.92	0.96
PC2	0.99	1.00
PC3	0.90	0.93

TABLE VI.
OVERSAMPLED CLASSIFICATION FOR CM1 DATASET

Accuracy	0.94
Recall	0.98
Precision	0.88
Balanced accuracy	0.91
F1 score	0.93

In the final part of the experiment hyperparameter optimization for the previously analyzed oversampling technique was added. Similarly to the forest size selection, this step becomes more and more computationally intensive with the increase in number of the combinations that have to be considered. Performing grid search and cross-validation proved to be successful in improving results, as can be seen in Table VII. Comparison of balanced accuracies obtained throughout the experiment was presented in Fig 3.

TABLE VII.
RESULTS AFTER HYPERPARAMETER OPTIMIZATION

	Dataset					
	CM1	KC1	KC2	PC1	PC2	PC3
Accuracy	0.95	0.91	0.87	0.98	1.00	0.94
Recall	0.98	0.91	0.89	0.98	1.00	0.95
Precision	0.91	0.91	0.86	0.97	0.99	0.93
F1 score	0.95	0.91	0.87	0.98	1.00	0.94
Balanced accuracy	0.95	0.91	0.86	0.98	1.00	0.94

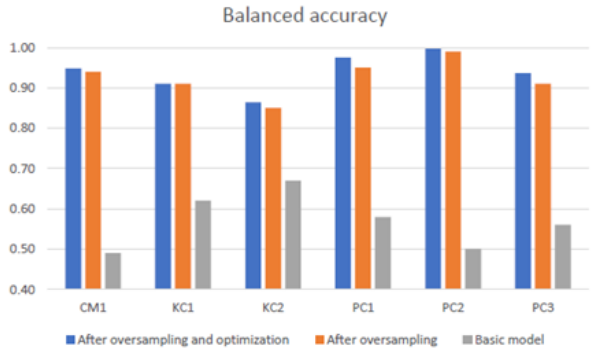

Fig 3. Balanced accuracy comparison

B. *Comparison with other studies*

Comparing obtained results with other studies within the domain, one can reference AUC obtained in a study by Catal et al. [10]. As presented in Fig. 4, results for all of the datasets that were covered by both experiments have significantly improved.

Nonetheless, when comparing the results with studies based on neural networks, such as the ones obtained via implementation of ADBBO by P. Kumudha et al. [19], presented in Fig. 5, it can be observed that the Random Forest provided better results for PC1 and CM1 datasets, while it was outperformed by Neural Network in case of KC1 and KC2 datasets.

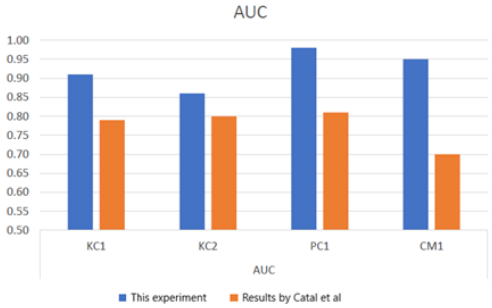

Fig 4. AUC comparison with prior study

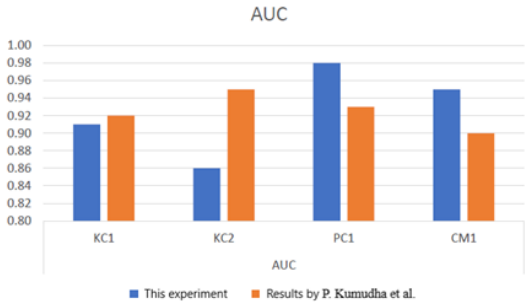

Fig 5. AUC comparison with prior study

A recent study by T.F. Husin et al. [26] was analyzing Least Square Support Vector Machine (LSSVM) combined with the use of SMOTE technique. Even though, it was concluded that SMOTE significantly improved obtained results, as presented in Fig. 6, those results were not close to the results obtain in this or any of two previously mentioned studies.

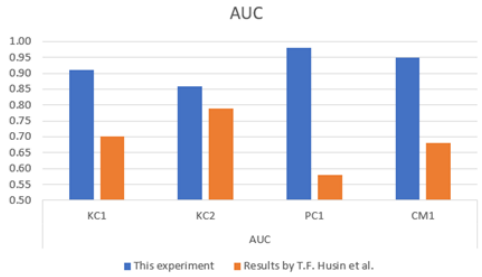

Fig 6. AUC comparison with new study

V. CONCLUSIONS

The aim of our study was to assess the viability of application of Decision Tree and Random Forest algorithms within the scope of error-proneness detection field. The series of experiments was conducted for six different datasets and total algorithm training time was approximately 150 hours with the majority of this time spent on the final version. Therefore, due to vastness of collected data detailed results presented in section IV focused only one of them – CM1. Study was performed on the data acquired from PROMISE repositories started with the analysis of the most basic models, which turned out to be insufficient due to the bias towards error-free classification resulting from dataset imbalance. Subsequently, a set of techniques was deployed in order to improve its performance. They included hyperparameter optimization, basic

feature selection, threshold adjustment and SMOTE oversampling technique.

As a result it was possible to observe that implementation of mechanisms aiming at improvement of performance of algorithms resulted in models being able to quite accurately classify samples present within PROMISE repository. High values of precision and recall, in most cases above 90%, may assure one that software errors can be well detected using Random Forest algorithm. It was also shown that usually random forests of sizes between 100 and 1000 are the most appropriate as above that values accuracy does not seem to improve, while computation time does. Nonetheless, it is also worth mentioning that single decision trees also provided useful results, however, they cannot quite compete with the anti-overfitting properties of the forest. Further, if predictions trained on PROMISE datasets are to be reasonable, one shall counter negative effects of imbalanced dataset – oversampling was proved to be a viable solution that significantly increased values of evaluation parameters such as balanced accuracy. Additionally, if training time is not limited, hyperparameter optimization may further improve obtained results. Finally, there has not been found any reason to use regression instead of classification it this particular classification problem. Throughout the study, it was found that threshold adjustment technique could result in slight improvements, however, it could not be reliably used.

In order to further improve results obtained by Random Forest, one may consider application of more advanced feature selection methods. Similarly, it would be reasonable to use MOOD and QMOOD object metrics. Further, one could consider creation of their own datasets, based on publicly available repositories. Performing a training on data gathered from projects in the same language, technology or domain as the target test set could also make prediction algorithm more sensitive to crucial aspects of assessing error-proneness for a given case.

REFERENCES

[1] F. Elberzhager, A. R. Rosbach, Eschbach, J. Münch, "Reducing Test Effort: A Systematic Mapping Study on Existing Approaches", Information and Software Technology, vol. 54, no. 10, 1092-1106, 2012.

[2] K. Bareja, A. Singhal, "A Review of Estimation Techniques to Reduce Testing Efforts in Software Development", http://dx.doi.org/ 10.1109/ACCT.2015.110, 2015.

[3] J. Hryszko, L. Madeyski, "Cost Effectiveness of Software Defect Prediction in an Industrial Project", http://dx.doi.org/ 10.1515/fcds-2018-0002, 2018.

[4] Y.Z. Bala, P.A. Samat, K.Y. Sharif, N. Manshor, "Current Software Defect Prediction: A Systematic Review", http://dx.doi.org/ 10.1109/AiIC54368.2022.99114586, 2022

[5] F. Matloob et al., "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review", http://dx.doi.org/ 0.1109/ACCESS.2021.3095559, 2021.

[6] Y. Zhao, K. Damevski, H,Chen, "A Systematic Survey of Just-in-Time Software Defect Prediction", http://dx.doi.org/ 10.1145/3567550, 2023.

[7] T. Menzies , J. DiStefano, A. Orrego , R. Chapman, " Assessing predictors of software defects", in *Proc Predictive software models workshop*, pp. 1-5, 2004.

[8] G. Boetticher, T. Menzies, T. Ostrand, PROMISE Repository of Empirical Software Engineering Data, West Virginia University, Department of Computer Science 2007.

[9] C. Catal, B. Diri, B. Ozumut, "An artificial immune system approach for fault prediction in object oriented software", pp. 238-245, http://dx.doi.org/ 10.1109/DEPCOS-RELCOMEX, 2007.

[10] C. Catal, B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem", http://dx.doi.org/ 10.1016/j.ins.2008.12.001, 2009.

[11] J. Brownlee, "Clonal selection theory & CLONALG. The clonal selection classification algorithm", in *Technical Report 2-02*, Swinburne University of Technology, 2005.

[12] J. H. Carter, "The immune system as a model for pattern recognition and classification", http://dx.doi.org/10.1136/jamia.2000.0070028, 2001.

[13] L. Breiman, "Bagging predictors.", *Mach Learn* 24, pp.123–140, https://doi.org/10.1007/BF00058655Y, 1996.

[14] D. Mundada, A. Murade, O. Vaidya, and J. N. Swathi, "Software Fault Prediction Using Artificial Neural Network And Resilient Back Propagation", *Int. J. Comput. Sci. Eng*., vol. 5, no. 03, pp. 173–179, 2016.

[15] Z. Xiang, L. Zhang, "Research on an Optimized C4.5 Algorithm Based on Rough Set Theory", http://dx.doi.org/ 10.1109/ICMeCG.2012.74, 2012.

[16] P. Bishnu and V. Bhattacherjee, "Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm", pp. 1146–1150, http://dx.doi.org/ 10.1109/TKDE.2011.163, 2012.

[17] P. Bishnu and V. Bhattacherjee, "Outlier Detection Technique Using Quad Tree" in *Proc Int'l Conf. Computer Comm. Control and Information Technology*, pp. 143-148, 2009.

[18] A. Okutan and O. Taner, "Software defect prediction using Bayesian networks", http://dx.doi.org/ 10.1007/s10664-012-9218-8, 2014.

[19] P. Kumudha, R. Venkatesan, "Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction", http://dx.doi.org/ 10.1155/2016/2401496, 2016.

[20] S. Gupta, D. Gupta, "Fault Prediction using Metric Threshold Value of Object Oriented Systems", *International Journal of Engineering Science and Computing*, vol. 7, no. 6, pp. 13629–13643, 2017

[21] E. Erturk, E. Akcapinar, "Iterative software fault prediction with a hybrid approach", http://dx.doi.org/ 10.1016/j.asoc.2016.08.025, 2016.

[22] J. S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system", http://dx.doi.org/ 10.1109/21.256541, 1993.

[23] F. Alighardashi, M. Ali, Z. Chahooki, "The Effectiveness of the Fused Weighted Filter Feature Selection Method to Improve Software Fault Prediction", pp. 5, http://dx.doi.org/10.22385/jctecs.v8i0.96, 2016.

[24] C. Lakshmi Prabha, Dr.N. shivakumar "Software Defect Prediction Using Machine Learning Techniques" , *Proc. of the Fourth International Conference on Trends in Electronics and Informatics*, IEEE Xplore Part Number: CFP20J32-ART; ISBN: 978-1-7281-5518-0, 2020.

[25] Y. Shen, S. Hu, S, Cai, M. Chen, "Software Defect Prediction based on Bayesian Optimization Random Forest", http://dx.doi.org/ 10.1109/DSA56465.2022.00149, 2022.

[26] T.F. Husin, M.R. Pribadi, Yohannes, "Implementation of LSSVM in Classification of Software Defect Prediction Data with Feature Selection", *9th Int. Conf. on Electrical Engineering, Computer Science and Informatics (EECSI2022)*, pp.126-131, 2022.

[27] MD.A. Jahangir, MD. A.Tajwar, W. Marma, "Intelligent Software Bug Prediction: An Empirical Approach", http://dx.doi.org , 101109/ICREST57604.2023.10070026, 2023.

[28] Python Core Team, "Python: A dynamic, open source programming language", Python Software Foundation, accessed 28.04.2022, https://www.python.org/

[29] C.R. Harris, K.J. Millman, S.J. van der Walt et al. "Array programming with NumPy", Nature 585, pp. 357–362, http://dx.doi.org/ 10.1038/s41586-020-2649-2, 2020.

[30] W. McKinney, "Data structures for statistical computing in python", *Proc. of the 9th Python in Science Conference*, vol 445, pp. 56-61, http://dx.doi.org/ 10.25080/Majora-92bf1922-00a, 2010.

[31] Pedregosa et al., "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research 12*, pp. 2825-2830, 2011.

[32] G. Lematre, F. Nogueira, C. K. Áridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning", *Journal of Machine Learning Research* 17, pp. 1-5, http://dx.doi.org/ 10.48550/arXiv.1609.06570, 2017.

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique", *Journal of artificial intelligence research*, pp. 321-357, 2002.