International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Enhanced Software Defect Prediction Through Homogeneous Ensemble Models

[1]R. Mamatha, [2]Dr. P. Lalitha Surya Kumari, [3]Dr. A. Sharada

Submitted: 11/09/2023 **Revised**: 21/10/2023 **Accepted**: 08/11/2023

Abstract: The authors of this paper recommend employing state-of-the-art Machine Learning methods for fault prediction in computer programmes. The Promise software engineering repository, where NASA stores its data, serves as an example. The basic objective of software defect prediction is the early discovery of software flaws. Machine learning algorithms can help with this by making predictions based on historical data. The initial experiments' results suffered from low precision and recall since they relied on outdated machine learning methods. Modern machine learning methods such as Naive Bayes, Boosting, and Grid Search were incorporated to increase the model's accuracy. The performance of the software defect prediction model has been greatly enhanced through the use of state-of-the-art machine learning techniques. The precision and recall rates, two measures of how well a system can forecast errors, have also grown. Naive Bayes, Boosting, and Grid Search are just a few of the modern machine learning methods that helped improve the software defect prediction model. The algorithms' increased accuracy and recall rates show how effective they are at finding and predicting software defects. The importance of using state-of-the-art machine learning methods to the task of defect prediction is emphasised. Using techniques such as Naive Bayes, Boosting, and Grid Search can significantly increase the model's effectiveness. These methods have improved software development processes by accelerating and better isolating bugs.

Keywords: *effective, methods, Boosting, accelerating, isolating*

1 Introduction

Identification during development: During the software production process, software defects are identified by developers and testers. This can happen during various stages such as coding, integration, or system testing. Defects are typically logged and tracked in a bug-tracking system. Software defects can significantly impact the productivity of the development team. Fixing defects requires additional time and effort, diverting resources from other tasks. It can lead to delays in the software production timeline and increased costs. Quality assurance: Software defects affect the quality of the software. They can cause malfunctions, crashes, or incorrect behavior, resulting in poor user experience. Quality of the application of modern machine learning methods such as Nave Bayes, Boosting, and Grid Search aided in the improvement of the software fault prediction model. y assurance activities, such as testing and debugging, are performed to identify and rectify these defects. Maintenance and support: After the software is released, defects reported by users need to be addressed through maintenance and support activities. This involves investigating the reported issues, diagnosing the underlying cause, and providing bug fixes or patches. Software defects can consume a significant portion of the maintenance efforts.

Customer satisfaction and reputation: Software defects impact customer satisfaction and the reputation of the software product and its development company. Frequent defects can lead to frustration among users and tarnish the image of the software. Conversely, a software product with fewer defects is more likely to gain positive feedback and maintain a good reputation. Identifying and fixing software defects is part of an iterative improvement process. Feedback from defect reports helps developers enhance their understanding of the software, leading to improved designs, better code quality, and more robust development practices.

To mitigate the impact of software defects, software development teams employ various strategies such as code reviews, automated testing, continuous integration, and quality assurance processes. The proposed study suggests a machine learning paradigm that aims to identify and rectify defects early in the software production cycle, reducing their impact on productivity, quality, and customer satisfaction.

In order to build a successful software system, it is necessary to draw on a wide range of resources, each with their own unique set of skills, knowledge, and expertise. The software development tasks are intricate. Assigning software development team members to certain projects and tasks is a critical element of every project manager's

1Research Scholar, Department of CSE, Koneru Lakshmaiah Education Foundation, Hyderabad-500075, Telangana, India. mamatha.racharla@gmail.com
2Professor, Department of CSE, Koneru Lakshmaiah Education Foundation, Hyderabad-500075, Telangana, India. vlalithanagesh@gmail.com
3Professor, Department of CSE, G. Narayanamma Institute of Technology & Science, Hyderabad, Telangana, India. sharada@gnits.ac.in

job. The best time for delivering a project depends on the manager taking into account the available skill sets, the interdependencies across activities, and the constraints on available resources [1-4]. It has been managed in a number of formats using a variety of optimisation strategies with the goal of reducing total project duration [3,5,7]. Allocation decisions made using these techniques frequently factor in time and resource constraints. Instead, the NP-Completeness of the allocation problem is typically dealt with by employing the strategy of traversing the optimisation space [3]. Managers of projects have the option of using the automated SSSP (Single Source Shortest Path) technique to streamline the allocation process, which may be tailored to account for a wide variety of project and resource variables.

Machine Learning (ML) techniques are employed in Software Defect Prediction (SDP) to identify software modules or components that are susceptible to failure. The objective of SDP is to assist software developers in efficiently allocating their resources for software testing and maintenance, by prioritising the focus on potentially flawed modules or components prior to the software's release [21–24]. SDP models are constructed using software characteristics such as software development history, source code complexity, software cohesiveness, and coupling. The software qualities mentioned are quantified statistically to assess the quality and reliability of software systems.

Machine learning techniques that take advantage of software's features are used to build SDP models. Models of SDPs have been built using supervised and unsupervised machine learning techniques. It's the triple-digits: 31-32-33. The major objective is to create precise and accurate SDP models for foreseeing software issues. The quality of the software metric datasets used to create SDP models is crucial to the success of the models. The predictive ability of SDP models is highly sensitive to the software characteristics used in their construction [25, 26, 27, 28]. There may be a class imbalance problem because SDP models' software properties are often complex and involved. If there is a disproportionate number of faulty cases compared to non-defective ones, then the SDP has a class imbalance. The SDP model loses some of its predictive power when there is a significant income gap between participants in the programme. [34, 35].

A small group of developers completing the greatest number of tasks (a bug, feature request, or task) in a quantifiable manner may have impacts depending on both how successfully the bug-fixing technique is completed and how many bugs can be solved in a specific length of time. When there are hundreds or even thousands of defects in a repository, assigning team members to problems becomes a major burden. It is much more important to allocate the most capable developers to the appropriate tasks through smart resource allocation. The majority of bug triage techniques [17] rely on text categorization. These methods, however, are plagued by poor bug reporting, which might cause the triage process to assign bugs to the incorrect developers [19], [20]. Low recall levels are another issue with these methods [17], [18]. In this study, the primary factor used to determine how jobs are assigned is the project bug.

This study introduces a new framework for Software Defect Prediction (SDP) that uses ensemble methods (specifically Boosting) to enhance the prediction performance of SDP models. The framework utilizes the cat boost (CB) and XGboost (XB) algorithms as classifiers, applied to pre-processed datasets consisting of 616 instances. The evaluation of the proposed techniques is conducted using metrics such as accuracy, Area Under Curve (AUC)) Region of Coverage (ROC), and Precision Recall analysis (PR curve). This study's key contribution is empirical validation of the effect of the homogeneous ensemble on the prediction performance of SDP models.

The paper is divided into sections, as described below:

Section 2 provides a comprehensive analysis of pertinent literature, with a specific focus on the issue of high dimensionality in SDP. Section 3 elucidates the research methodology employed in the study. Section 4 presents and illustrates experimental findings and analyses. Section 5 serves as the final conclusion of the research..

2 Related Work

The predictive power of Software Defect Prediction (SDP) models has been shown to suffer when a class imbalance problem is present. Overfitting and a lack of confidence in SDP models are common results of class imbalance. Scholars have proposed a number of approaches, including ensemble methods, data sampling, and cost-sensitive analysis, to address this issue.

Singh, Misra, and Sharma [9] examined the effectiveness of ensemble methods, including voting and Bagging, for predicting the severity of issues in order to rectify the class imbalance present in the bug dataset. Ensemble approaches performed better than individual classifiers, indicating that this group's methodology may effectively resolve class imbalance.

El-Shorbagy, El-Gammal, and Abdelmoez [10] used a heterogeneous ensemble technique called stacking with the SMOTE (Synthetic Minority Over-sampling Technique) algorithm. The goal was to combine the efficiency of different base classifiers to take use of the benefits of dealing with minority class labels. In terms of accurately classifying the minority group, their new method surpassed prior approaches. The stacking

ensemble method requires a large number of permutations of simple classifiers, which can be laborious and time-consuming.

Balogun, Basri, Abdulkadir, Adeyemo, Imam, and Bajeh [5] conducted an empirical study to assess the predictability of SDP models using data sampling techniques. The researchers examined the use of under-sampling (Random Under-Sampling: RUS) and over-sampling (SMOTE) methodologies across different imbalance ratios. Experimental results indicate that the presence of a class imbalance in SDP datasets has a substantial influence on the prediction performance of SDP models. In addition, they suggested employing the SMOTE technique as a solution to the problem of class imbalance in SDP. These findings align with the results obtained by Yu, Jiang, and Zhang. [3].

Laradji, Alshayeb, and Ghouti [13] conducted a study to examine the application of feature selection in combination with ensemble approaches for SDP. Their objective was to utilise ensemble approaches to tackle the issue of class imbalance and eradicate feature redundancy through the use of feature selection algorithms. Laradji, Alshayeb, and Ghouti [13] found that incorporating feature selection with ensemble approaches improved the accuracy of SDP model prediction. The models improved their performance by meticulously choosing pertinent traits and minimising feature duplication.

The results show that class imbalance has a significant impact on SDP models' ability to predict outcomes. Experts have stressed the significance of addressing this issue by employing an appropriate mixture of data collection methods, classifiers, and ensemble approaches.

Based on their empirical research, Song, Guo, and Shepperd [15] concluded that class imbalance reduces the predictive accuracy of SDP models. The right combination of data sampling methods and classifiers was also stressed as crucial for maximising predicted accuracy.

To emphasise the critical relevance of adopting data sampling methods, Goel, Sharma, Khatri, and Damodaran [3] all agreed that the class imbalance problem in SDP can be effectively solved by implementing an appropriate data sample methodology. After analysing the prediction performance of seven different boosting ensemble methods, Malhotra and Jain [14] concluded that data sampling methods should be employed prior to using the boosting ensemble methodology.

Several methods for resolving class differences in SDP were compared by Wang and Yao [35]. They discovered that ensemble strategies outperformed data sampling and cost-sensitive approaches. The findings of Rodriguez, Herraiz, Harrison, Dolado, and Riquelme [7] reinforce this finding, showing that the combination of different approaches can yield better results than the use of any of them alone.

To identify critical source code metrics for defect detection, Kumar, Misra, and Rath [36] used correlation analysis and multivariate linear regression feature selection. After that, neural networks and ensemble methods were used to train the datasets. The results of their study demonstrated the usefulness of ensemble methods for Semantic Dependency Parsing (SDP), particularly when combined with feature selection procedures.

This research proposes a strategy for improving the accuracy of SDP model prediction by integrating data sampling with homogeneous ensemble methods (Bagging and Boosting), specifically to deal with the problem of class imbalance.

3 Methodology

3.1Classification Algorithm

The primary prediction models utilised in this study consist of the Support Vector Machine (SVM) and Random Forest (RF) algorithms. These algorithms have been commonly employed in SDP experiments and have consistently shown excellent predictive capabilities. Furthermore, they have been demonstrated to exhibit stability while handling datasets that are skewed [3, 5].

3.2Homogeneous Ensemble Methods

The boosting ensemble approach is a methodology that trains a sequence of weak classifiers progressively on re-weighted training data. Each weak classifier is trained to concentrate on examples misclassified by the prior classifiers. The boosting ensemble makes its final judgment by combining the predictions of all weak hypotheses using a majority voting procedure [17].

Boosting utilises weighted averages to enhance the effectiveness of weak classifiers. During each iteration, the weights assigned to the training samples are adjusted, prioritising the misclassified data by increasing their significance. Boosting aims to enhance the overall performance of the ensemble by giving more emphasis on incorrectly identified samples.

Furthermore, boosting also incorporates a feature selection mechanism. Each weak classifier in the boosting process decides which features or attributes are most informative for the next iteration. This iterative feature selection process helps the boosting ensemble to focus on the most relevant features and improve the prediction performance.

Overall, boosting ensemble combines the predictions of multiple weak classifiers by assigning weights to their

outputs and using a majority voting mechanism. Through the iterative training process and feature selection, boosting goals to augment the performance of the ensemble and create a stronger overall classifier.

3.3 Dataset of Software Defects

In this section, the software defect dataset used in the study is discussed.

CM Dataset:

CM is a software defect prediction dataset developed by NASA, utilizing the NASA Metrics Data Program (MDP). It specifically targets a spacecraft instrument developed by NASA, which is executed in the C programming language. The dataset includes features extracted from Halstead metrics, which provide insights into the characteristics of the source code at the segment, function, or method level. These metrics serve as valuable indicators for predicting software defects in the CM1 dataset.

KC1 Dataset:

The KC dataset is a software defect prediction dataset created by NASA, utilizing the NASA Metrics Data Program (MDP). It specifically targets a storage management project developed by NASA, which involves receiving and processing ground data. The project is implemented in the C++ programming language. The dataset includes features derived from Halstead metrics, which provide valuable information about the complexity and characteristics of the code. These metrics are utilized to build models for predicting software defects in the KC1 dataset.

KC2 Dataset:

The KC dataset is a software defect prediction dataset created by NASA, utilizing the NASA Metrics Data Program (MDP). It specifically focuses on the science data processing portion of the KC1 project, which is another project developed by NASA. The code in KC2 is written in the C++ programming language and incorporates third-party software libraries in addition to the code from KC1. The dataset consists of 522 instances, and the data is derived from Halstead metrics extracted from the source code. These metrics provide valuable insights into the complexity and characteristics of the code, enabling the development of software defect prediction models using the KC2 dataset.

PC Dataset:

NASA generated the PC dataset to anticipate software defects using the NASA Metrics Data Program (MDP). It focuses on flight software made for an earth-orbiting satellite; a vital component developed by NASA. The software is implemented in the C programming language. The PC1 dataset consists of data derived from Halstead metrics, which provide valuable insights into the complexity and characteristics of the source code. These parameters are used to develop models for predicting software defects in the PC1 dataset, with the aim of enhancing the reliability and performance of the flight software.

3.4 Algorithms used:

Catboost: In classification tasks, CatBoost employs a default encoding technique for categorical variables that have a set of distinct values exceeding the minimum size required for one-hot encoding. This encoding technique is utilized to transform categorical features into numerical features, facilitating their effective use in the classification models built by CatBoost.

XGBoost: XGBoost uses gradient descent on decision trees to build a series of models that are iteratively integrated, with each successive model fixing the errors of the preceding ones. This iterative approach allows XGBoost to generate a final optimal model for the given task. Notably, XGBoost exhibits remarkable efficiency in terms of computational resource utilization and processing speed. It efficiently utilizes available resources while providing fast and effective model training and prediction capabilities. This efficiency makes XGBoost a popular choice for various applications where processing large datasets or real-time predictions are critical considerations.

To expedite the model training process, we employ GPUs for both XGBoost and CatBoost algorithms. Leveraging GPUs significantly accelerates the training time, enabling us to complete our experiments within a practical timeframe. Specifically, for XGBoost, we found that explicitly setting the learning rate to 0.1 and constraining the maximum depth of its constituent decision trees to 6 were necessary to achieve reasonable training times. These parameter settings strike a balance between model complexity and training efficiency, ensuring that the training process is efficient without compromising the model's performance. The use of GPUs and the carefully selected parameter settings allow us to conduct our experiments effectively and obtain meaningful results in a feasible amount of time.

3.5 Performance Assessment Metrics in the Context of software defect prediction:

performance assessment metrics include:

True Positive (TP): The model correctly predicts a software problem.

True Negative (TN): The model accurately predicts a non-defective occurrence.

False Positive (FP): A false alarm occurs when the model predicts a software defect that does not exist.

False Negative (FN): When there is a software defect, the model fails to predict it (missed detection).

These assessment measures have been widely used and proved to be reliable in SDP studies [3,13,19].

Accuracy: Calculates the proportion of correctly classified cases (TP and TN) to the total number of examples to determine the overall correctness of the model's predictions.

Precision is the ratio of correctly predicted faults (TP) to the total number of instances predicted as defects (TP + FP). It represents the model's ability to predict defects.

The proportion of accurately anticipated defects (TP) to the total number of actual defects (TP + FN) is referred to as recall. It represents the model's ability to discover problems.

F1 rating: The harmonic mean of precision (the model's accuracy in predicting flaws) and recall (the model's ability to recognize problems) is used to calculate the F1 score. This balanced measure takes precision and recall into account, making it particularly useful when there is a class imbalance or when both false positives and false negatives occur.

AUC-ROC (Area Under the Receiver Operating Characteristic Curve): This metric measures the model's ability to distinguish between defective and non-defective cases at different probability levels. A higher AUC-ROC value indicates better prediction performance.

4 Experimental Framework

In order to evaluate the performance of the suggested approach to SDP, the following methods were incorporated into the experimental framework of this research:

Dataset Selection: The dataset was chosen from the NASA MDP Promise repository. The specific dataset was selected based on its size and relevance to the study.

Classifier Selection: SVM, Random Forest, Logical regression, and Neural network classifiers were selected based on their widespread use in SDP studies and their potential to handle the dataset effectively and compared with ensembled machine learning algorithms like EDA, Boosting, Cat boost, XGBoost.

Evaluation Metrics: The performance of each classifier was assessed based on parameters such as precision, recall, and accuracy. These metrics offer insights into the classifier's accuracy in correctly identifying instances and its overall performance on the dataset.

Result Analysis: After evaluating the classifiers, the results were analysed to regulate the strengths and weaknesses of each algorithm. The researchers examined the precision, recall, and accuracy values of each classifier to understand their performance on the specific dataset.

Algorithm Suitability: Based on the results, conclusions regarding the suitability of each algorithm for different types of datasets is given. Factors such as the classifier's accuracy, precision, recall, and overall performance are considered to identify which algorithm performed best for the given dataset.

Figure 1 illustrates the framework for predicting software defects.

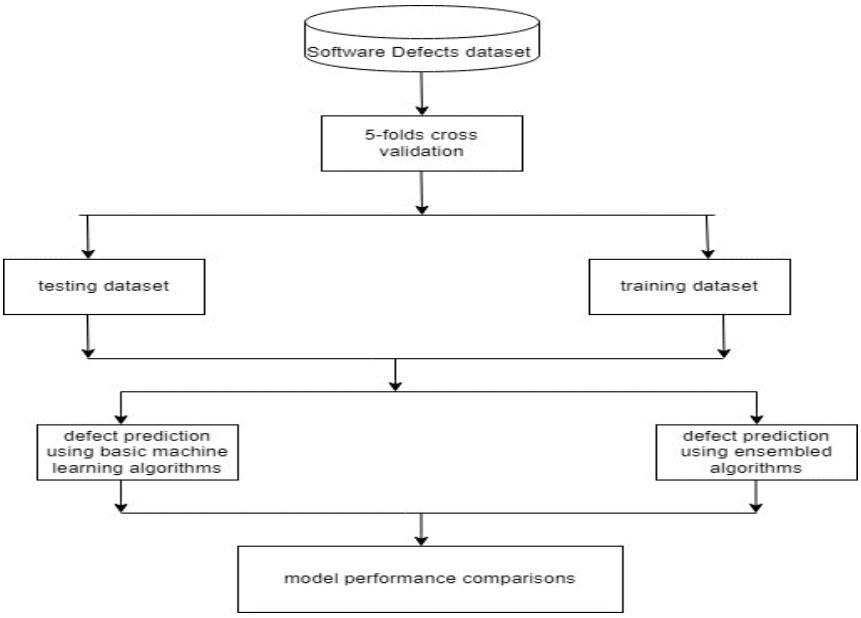

Fig.1: Outline for software defect prediction

5 Results

Popular methods such as SVM, Random Forest, Logistic Regression, and Neural Networks were chosen to assess the effectiveness of different classifiers in SDP studies. These classifiers were selected due to their prevalent use in the field and their potential to effectively handle the dataset.

Ensemble machine learning algorithms were adopted to improve the model's performance even further. Ensemble methods like such as EDA, Boosting, CatBoost, and XGBoost combine multiple individual models to create a more powerful predictive model. Upon applying the ensemble algorithm to the dataset, significant improvements in the performance of the models are observed. The ensemble techniques allowed us to pull the strengths of each individual classifier, leading to enhanced accuracy and predictive power. By combining the predictions from multiple models, bias, variance, and overfitting were able to reduced resulting in more robust and reliable predictions.

Overall, incorporating ensemble algorithms alongside the chosen classifiers proved to be a beneficial technique for the SDP study. This approach not only expanded the range of techniques used but also significantly improved the performance of the models, enabling to achieve more accurate and reliable results for the given dataset.

The graph below depicts A classification model's performance across all classification thresholds. This graph illustrates two parameters: the True Positive Rate and the False Positive Rate. The percentage of false positives. Figures 5.1 and 5.2 show that the model has strong discriminatory power and distinguishes between positive and negative events across various classification levels. The lift curve, depicted in Fig 5.3, is a graphical depiction that aids in determining the effectiveness of a binary classification model for targeting positive cases. Finally, Figure 5.4 depicts the overall performance of the binary classification model in terms of precision and recall.

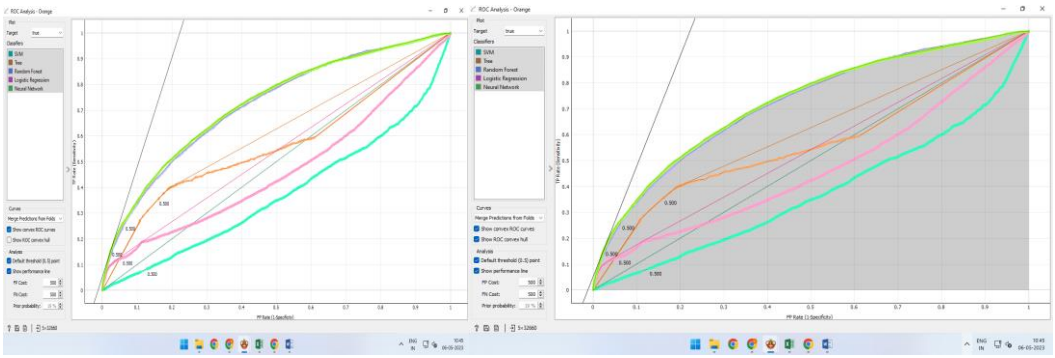

Fig 5.1: Convex ROC Curves for the models

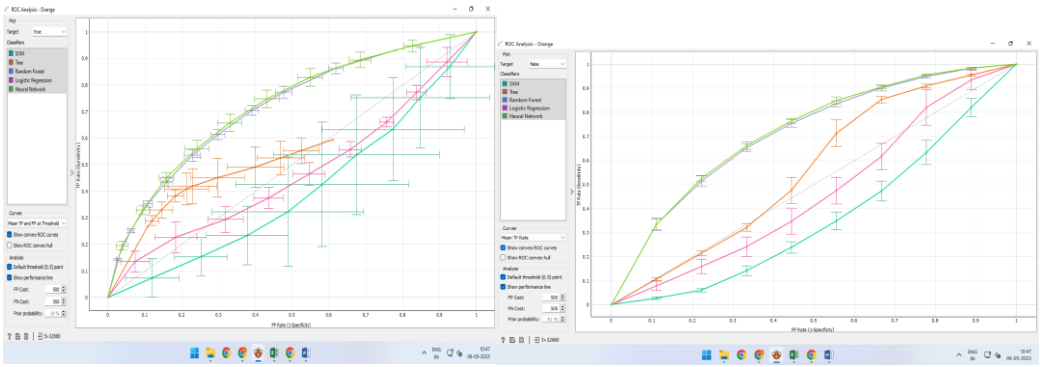

Fig 5.2: ROC Curve Analysis for the models

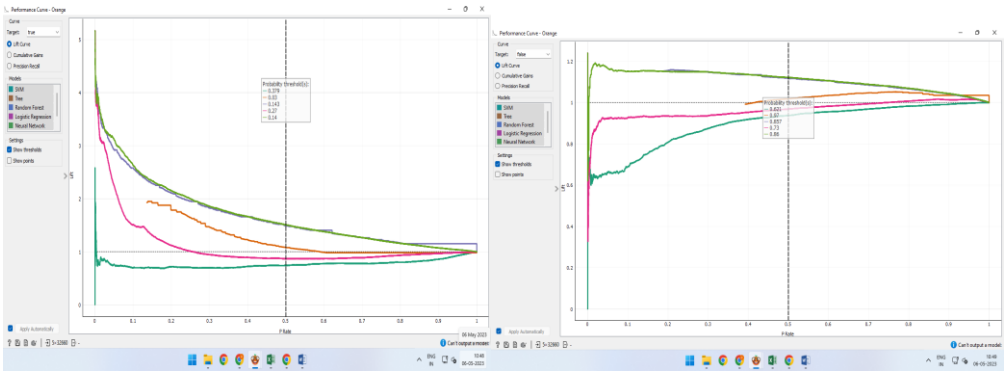

Fig 5.3: Lift Curves

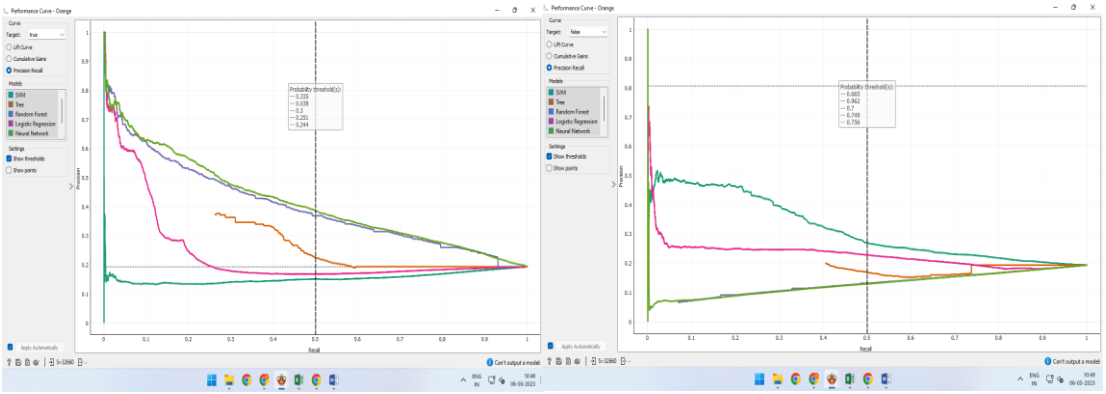

Fig 5.4: Precession recall

6 Conclusion

This article shows an SDP method that has been tested in the real world using the uniform ensemble (Bagging and Boosting). It was shown in the research that the method can improve prediction performance while also making the base classifiers better at prediction. This study will look into SDP methods further to improve ensemble parameters and the level of data sampling..

References:

[1] Tsai, H.-T., H. Moskowitz, and L.-H. Lee, Human resource selection for software development projects using Taguchi's parameter design. European Journal of Operational Research, 2003. 151(1): p. 167-180.

[2] Di Penta, M., M. Harman, and G. Antoniol, The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. Software: Practice and Experience, 2011. 41(5): p. 495-519.

[3] Goel, L., Sharma, M., Khatri, S.K., Damodaran, D.: Implementation of data sampling in class imbalance learning for cross project defect prediction: an empirical study. In: 2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT), pp. 1–6. IEEE (2018)

[4] Peixoto, D.C., G.R. Mateus, and R.F. Resende. The Issues of Solving Staffing and Scheduling Problems in Software Development Projects. in Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual. 2014. IEEE.

[5] Balogun, A.O., Basri, S., Abdulkadir, S.J., Adeyemo, V.E., Imam, A.A., Bajeh, A.O.: Software defect prediction: analysis of class imbalance and performance stability. J. Eng. Sci. Technol. 14, 3294–3308 (2019)

[6] Alba, E. and J.F. Chicano, Software project management with GAs. Information Sciences, 2007. 177(11): p. 2380- 2401.

[7] Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., Riquelme, J.C.: 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, pp. 1–10 (2014)

[8] Uddin, J., Ghazali, R., Deris, M.M., Naseem, R., Shah, H.: A survey on bug prioritization. Artif Intell Rev 47(2), 145–180 (2017). DOI 10.1007/s10462-016-9478-6. URL http://link.springer.com/10.1007/s10462-016-9478-6

[9] Singh, V., Misra, S., Sharma, M.: Bug severity assessment in cross-project context and identifying training candidates. J. Inf. Knowl. Manag. 16, 1750005 (2017)

[10] El-Shorbagy, S.A., El-Gammal, W.M., Abdelmoez, W.M.: Using SMOTE and heterogeneous stacking in

ensemble learning for software defect prediction. In: Proceedings of the 7th International Conference on Software and Information Engineering, pp. 44–47 (2018)

[11] Ferrucci, F., M. Harman, and F. Sarro, Search-Based Software Project Management, in Software Project Management in a Changing World, G. Ruhe and C. Wohlin, Editors. 2014, Springer Berlin Heidelberg. p. 373-399.

[12] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in Proc. Intl. Conf. Software Engineering & Knowledge Engineering (SEKE 10), 2010, pp. 209–214.

[13] Laradji, I.H., Alshayeb, M., Ghouti, L.: Software defect prediction using ensemble learning on selected features. Inf. Softw. Technol. 58, 388–402 (2015)

[14] Singh, V., Misra, S., Sharma, M.: Bug severity assessment in cross-project context and identifying training candidates. J. Inf. Knowl. Manag. 16, 1750005 (2017)

[15] Song, Q., Guo, Y., Shepperd, M.: A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans. Softw. Eng. 45, 1253–1269 (2018).

[16] A. Tamrawi, T. Nguyen, J. Al-Kofahi, and T. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011, pp. 365–375.

[17] Sun, B., Chen, S., Wang, J., Chen, H.: A robust multi-class AdaBoost algorithm for mislabeled noisy data. Knowl.-Based Syst. 102, 87–102 (2016) [18] J. Anvik, "Automating bug report assignment," in Proceedings of the 28th international conference on Software engineering. ACM, 2006, pp. 937–940.

[18] Yang, X., Lo, D., Xia, X., Sun, J.: TLEL: a two-layer ensemble learning approach for justin-time defect prediction. Inf. Softw. Technol. 87, 206–220 (2017)

[19] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zim- ¨ mermann, "What makes a good bug report?" in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, 2008, pp. 308–318.

[20] Goel, L., Sharma, M., Khatri, S.K., Damodaran, D.: Implementation of data sampling in class imbalance learning for cross project defect prediction: an empirical study. In: 2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT), pp. 1–6. IEEE (2018)

[21] Hamdy, A., El-Laithy, A.: SMOTE and Feature Selection for More Effective Bug Severity Prediction. Int. J. Softw. Eng. Knowl. Eng. 29, 897–919 (2019)

[22] Iqbal, A., Aftab, S.: A classification framework for software defect prediction using multifilter feature selection technique and MLP. Int. J. Mod. Educ. Comput. Sci. 12(1), 18–25 (2020). https://doi.org/10.5815/ijmecs.2020.01.03

[23] Chang, C.K., M.J. Christensen, and T. Zhang, Genetic algorithms for project management. Annals of Software Engineering, 2001. 11(1): p. 107-139.

[24] Izadi, M., Ganji, S., Heydarnoori, A., Gousios, G.: Topic recommendation for software repositories using multi-label classification algorithms (2020)

[25] Ghotra, B., McIntosh, S., Hassan, A.E.: A large-scale study of the impact of feature selection techniques on defect classification models. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 146–157. IEEE (2017)

[26] Xu, Z., Liu, J., Yang, Z., An, G., Jia, X.: The impact of feature selection on defect prediction performance: an empirical comparison. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 309–320. IEEE (2016)

[27] Gupta, A., Suri, B., Misra, S.: A systematic literature review: code bad smells in java source code. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 665–682. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_49

[28] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in Proceedings of the 28th international conference on Software engineering. ACM, 2006, pp. 361–370.

[29] D. Cubrani ˇ c and G. C. Murphy, "Automatic bug triage using text categorization," in In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering. Citeseer, 2004, pp. 92–97.

[30] Balogun, A., Oladele, R., Mojeed, H., Amin-Balogun, B., Adeyemo, V.E., Aro, T.O.: Performance analysis of selected clustering techniques for software defects prediction. Afr. J. Comput. ICT 12, 30–42 (2019)

[31] Ghotra, B., McIntosh, S., Hassan, A.E.: A large-scale study of the impact of feature selection techniques on defect classification models. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 146–157. IEEE (2017)

[32] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zim- ¨ mermann, "What makes a

good bug report?" in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, 2008, pp. 308–318.

[33] J.-W. Park, M.-W. Lee, J. Kim, S. won Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems." in AAAI, W. Burgard and D. Roth, Eds. AAAI Press, 2011.

[34] El-Shorbagy, S.A., El-Gammal, W.M., Abdelmoez, W.M.: Using SMOTE and heterogeneous stacking in ensemble learning for software defect prediction. In: Proceedings of the 7th International Conference on Software and Information Engineering, pp. 44–47 (2018)

[35] Kumar, L., Misra, S., Rath, S.K.: An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Comput. Stand. Interfaces 53, 1–32 (2017)

[36] Ms. Nora Zilam Runera. (2014). Performance Analysis On Knowledge Management System on Project Management. International Journal of New Practices in Management and Engineering, 3(02), 08 - 13. Retrieved from http://ijnpme.org/index.php/IJNPME/article/view/28

[37] Ghazaly, N. M. . (2020). Secure Internet of Things Environment Based Blockchain Analysis. Research Journal of Computer Systems and Engineering, 1(2), 26:30. Retrieved from https://technicaljournals.org/RJCSE/index.php/journal/article/view/8