Using Machine Learning Techniques to Predict Bugs in Classes: An Empirical Study

Musaad Alzahrani
Department of Computer Science
Albaha University
Al-Baha 65799, Saudi Arabia

***Abstract*—Software bug prediction is an important step in the software development life cycle that aims to identify bug-prone software modules. Identification of such modules can reduce the overall cost and effort of the software testing phase. Many approaches have been introduced in the literature that have investigated the performance of machine learning techniques when used in software bug prediction activities. However, in most of these approaches, the empirical investigations were conducted using bug datasets that are small or have erroneous data leading to results with limited generality. Therefore, this study empirically investigates the performance of 8 commonly used machine learning techniques based on the Unified Bug Dataset which is a large and clean bug dataset that was published recently. A set of experiments are conducted to construct bug prediction models using the considered machine learning techniques. Each constructed model is evaluated using three performance metrics: accuracy, area under the curve, and F-measure. The results of the experiments show that logistic regression has better performance for bug prediction compared to other considered techniques.**

***Keywords*—Software bugs; bug prediction; machine learning techniques; software metrics; unified bug dataset**

I. Introduction

Software development is an error-prone process. Mistakes and errors that occur during the development process result in bugs that can ultimately cause software failure [1]. Software testing is one of the most important phases in the software development process which aims to identify bugs and to ensure the overall quality of systems before they are released. However, the testing cost and effort can grow dramatically when the size and complexity of a system increase. It is estimated that the cost of the testing activities constitutes around 25% of the total cost of the software development budget and it can reach to 50% when the size and complexity of the system increase [2], [3]. Therefore, the testing resources should be allocated efficiently in order to minimize the total cost of the overall development process.

Software Bug Prediction (SBP) is one of the most useful techniques that can be used to decrease the testing cost and effort [4]. The main goal of SBP is to identify the modules that are likely to have bugs. Software professionals use SBP models at the beginning of the testing phase to classify the modules of a system into bug-prone modules and non-bug-prone modules based on a set features extracted from the modules. The most commonly used features are software metrics that measures different characteristics of the module such complexity, size, coupling, and cohesion. Fig. 1 shows the basic architecture of a SBP model. Most of the testing cost and effort should be allocated on bug-prone modules in order to use the limited testing resources efficiently.

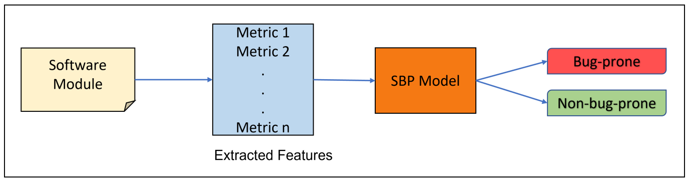

Fig. 1. Basic Architecture of a SBP Model.

Many approaches have been introduced into the literature that have shown the success of using machine learning algorithms to construct SBP models (e.g., [5], [6], [7], [8]) based on public available bug datasets such as NASA and Colombus [9]. However, the size and quality of the bug datasets used to train SBP models can have a great impact on the performance of the models and can limit their generality. Ferenc, Rudolf et al. [10] explained several issues in most commonly used bug datasets including the missing of the source code elements associated with data; dissimilarities in terms of granularity, features, and format between the datasets; the missing values in the datasets; and the existence of contrasting bug information. To mitigate these issues, they produced a unified bug dataset at class and file level by analyzing the source code of the reported systems in 5 public bug datasets. The Unified Bug Dataset [10] contains the values of 60 metrics and bug information for 47,618 classes and for 43,744 files and their corresponding source code. Although the Unified Bug Dataset is considered to be a large and clean dataset and has better quality compared to most commonly used bug datasets, it has been used in only a few studies in literature (e.g., [10], [11]) to build machine learning based SBP models.

Therefore, this paper empirically investigates the performance of 8 well-known machine learning techniques, namely, Logistic Regression (LR), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naive Bayes(NB), Decision Tree (DT), Bagging, Random Forest (RF), and AdaBoost based on the Unified Bug Dataset. The contribution of the paper is twofold:

- Constructing 8 SBP models based on the Unified Bug Dataset.
- Evaluating and comparing the performance of the constructed SBP models.

The rest of the paper is organized as follows. Related studies are summarized and discussed in Section II. Section III describes the used methodology. The results and discussion are given in Section IV. The threats to validity are presented in Section V. Section VI concludes the study and provides future work.

II. Related Work

SBP refers to the process of predicting buggy modules in a software system. In the last two decades, many approaches in the literature have introduced SBP models that tried to identify a causal relationship between a set of characteristics or features of a given software module and the existence of bugs in that module based on historical bug datasets. The most commonly used features to predict bugs are software metrics such as McCabe's Cyclomatic Complexity [12], Halstead Metrics [13], and Chidamber and Kemerer object-oriented metrics suite [14].

The majority of SBP models are constructed using machine learning algorithms such as LR (e.g., [5], [6]), SVM (e.g., [7], [8]), and KNN (e.g., [8]). The study by Basili et al. [5] was one of the early studies that used LR to construct a SBP model for the purpose of empirically validating Chidamber and Kemerer object-oriented metrics as quality indicators based data collected from 8 small object-oriented systems. The results of the study showed that the object-oriented metrics are beneficial to some degree to predict bugs in software systems in early phases of their development life cycle. Osman et al. [8] studied the impact of adjusting the parameters of two machine learning algorithms: SVM and KNN to build better SBP models. They conducted a set of experiments on five systems and the results showed that tuning the parameters of the two algorithms can improve their accuracy when compared to the default parameters setting. Researchers in [15] experimentally compared the performance of NB to DT for defect prediction based NASA Datasets [9]. Their results suggested that NB is more accurate and useful in predicting software defects when compared to DT. Singh et al. [16] analyzed the performance of five machine learning algorithms namely Artificial Neural Network, Particle Swarm Optimization, DT, NB, and SVM. They carried out a set of experiments on NASA datesets to compare the accuracy of the considered algorithms when used for software defect prediction. The output of the experiments showed that SVM outperformed the other 4 algorithms. Matloob et al. [17] conducted a systematic literature review on software defect prediction using ensemble learning methods. They considered only studies that were published in the period from 2012 to 2021. The results of their systematic literature review showed that RF, boosting, and bagging are the most frequently proposed methods in the literature during the considered period. In addition, their results showed that most of the proposed ensemble models were built based on PROMISE datasets [9] (which consists of a set of public datasets mostly NASA datasets). In a recently published study [11], the performance of the two ensemble learning methods: AdaBoost and Bagging was investigated. A set of experiments were conducted on the Unified Bug Dataset [10]. The results indicated that AdaBoost with a DT as a base learner outperformed Bagging technique.

A considerable amount of previously published studies have indicated the effectiveness of using machine learning algorithms to build SBP models. However, most of these studies used bug datasets that are small or have erroneous data such as NASA datasets [18], [11]. Therefore, this study tries to bridge this gap by using a set of well-known machine learning algorithms to construct SBP models based on the Unified Bug Dataset [10], which is a large and clean dataset and which has been used in only a few studies in the literature [11].

III. Methodology

A. Motivation

Many machine learning algorithms have been used in previous studies to construct SBP models. However, the results of these studies are not always agreeing on the superiority of a machine learning algorithm or technique over others. In addition, most of the previous studies built SBP models based on bug datasets (such as NASA datasets) that have been shown to be noisy and containing erroneous data [19], [18], [11], which can have a significant impact on the performance of these models.

Motivated by the previously mentioned remarks, this study aims to answer the following research question:

- RQ: What is the most effective machine learning technique (in terms of performance metrics) for bug prediction in classes based on the Unified Bug Dataset?

B. Research Framework

The overall research framework of this study is depicted in Fig. 2. The input dataset is first preprocessed. The preprocessing step includes data normalization and feature selection. After the preprocessing of the input dataset, the ten-fold cross-validation is used to train and evaluate the considered machine learning models. The ten-fold cross-validation randomly divides the dataset into 10 equal size subdatasets. For each subdataset, the remaining 9 subdatasets are used train a model and the subdataset is used to test the performance of the model. Finally, the results of the ten-fold cross-validation for each model are averaged and reported.

C. Dataset

The Unified Bug Dataset [10] is used to construct the SBP models. The dataset contains information for 47,618 classes and for 43,744 files and their corresponding source code. The information includes the values of 60 software metrics (including McCabe's Cyclomatic Complexity [12], Halstead Metrics [13], and Chidamber and Kemerer object-oriented metrics suite [14]) and the number of bugs in each class and file. The values of the software metrics for the classes and files were calculated from their source code using the open-source OpenStaticAnalyzer tool [20]. The bug information of the classes and file were collected from 5 public bug datasets namely: PROMISE [21], Eclipse Bug Dataset [22], Bug Prediction Dataset [23], Bugcatchers Bug Dataset [24], and GitHub Bug Dataset [25].

D. Dependent and Independent Variables

The dependent variable Y in this study is binary (i.e., $Y \in \{0, 1\}$) where 0 means that the class does not have a bug (referred as non-buggy class) and 1 means the class has

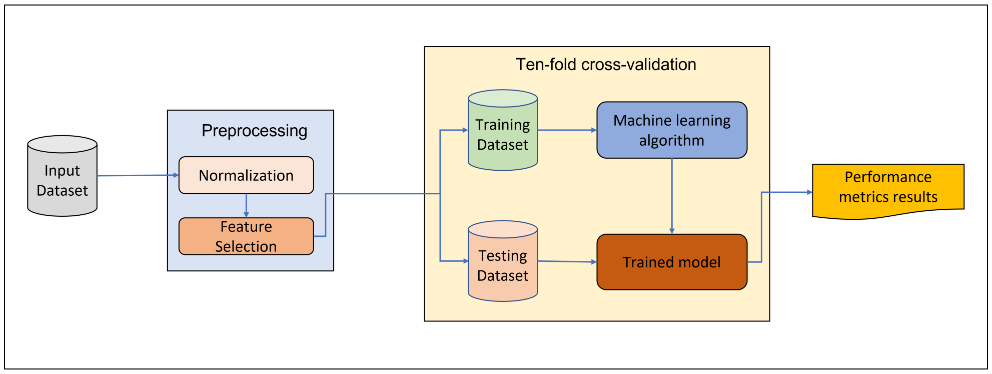

Fig. 2. The Research Framework of this Study.

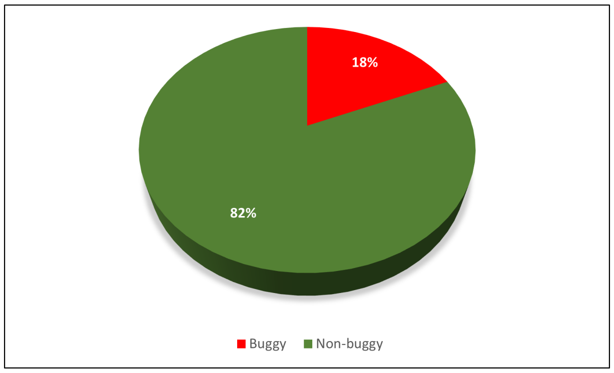

Fig. 3. The Distribution of Classes in the used Dataset.

at least one bug (referred as buggy class). Figure 3 shows the distribution of buggy and non-buggy classes in the Unified Bug Dataset. It can be seen from Fig. 3 that the Unified Bug Dataset has imbalanced distribution in terms of buggy and non-buggy classes. The non-buggy classes make up to 82% of the total classes in dataset whereas the buggy classes form only 18%. The ultimate goal of a SBP model is to label a class in question with 0 or 1 based on the values of a set of independent variables. The software metrics of the classes reported in the Unified Bug Dataset are used as independent variables in this study. Software metrics of a class are quantitative measurements that indicate the degree to which a class possesses a property or attributes such as class complexity, size, coupling and cohesion.

E. Data Preprocessing

Two common techniques are applied sequentially to preprocess the considered dataset namely: MinMaxScaler [26], and correlation-based filter-subset feature selection with BestFirst search [27].

One issue in the Unified Bug Dataset is that the dataset includes software metrics that are not normalized (i,e., they do not have an upper bound) and they differ in the order of magnitude [4]. This issue can have a negative impact on the accuracy of a prediction model [4]. Normalization is a commonly used technique that is used to address this issue by rescaling of the original values of variables to a specific range. In this paper, the MinMaxScaler [26] technique to is applied in the data preprocessing to transform the original values of all the metrics in the Unified Bug Dataset between the closed interval 0 and 1.

High dimensionality is another issue in the Unified Bug Dataset. The dataset includes 60 software metrics. Fig. 4 shows the Spearman correlation coefficients between each pair of software metrics in the dataset. As it can be seen from Fig. 4, some of these metrics have strong correlations with each other. Building SBP models based on high dimensional redundant dataset takes more time and computational resources and can negatively affect the performance models [4], [28]. Therefore, researchers often apply feature selection techniques to address this problem before constructing prediction models [1], [29], [30], [31]. In this study, the correlation-based filter-subset feature selection with best first search is used. This technique was found to be the best when used in the field of SBP among 30 feature selection techniques that were analyzed in a large-scale study [27].

F. Learners

The learners that are used to build SBP models in this study include: Logistic Regression (LR), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naive Bayes (NB), Decision Tree (DT), Bagging, Random Forest (RF), and AdaBoost. The former 5 learners referred in the literature as traditional learners whereas the latter 3 learners referred as ensemble learners. In traditional learning techniques, a single learner (e.g., LR) is used to build a prediction model. On the other hand, a combination of learners are used to build a prediction model in ensemble learning techniques [17]. A brief description of each learner is given in the following.

Logistic Regression (LR): LR is a statistical model used to predict a binary dependent variable based on a set of independent variables using the following equation:

$$\pi(X_1, X_2, ..., X_n) = \frac{1}{1 + e^{-(C_0+C_1X_1+C_2X_2+...+C_nX_n)}} \quad (1)$$

where $X_1, X_2, ..., X_n$ are the independent variables and $C_1, C_2, ..., C_n$ are estimated regression coefficients. The larger the absolute value of the coefficient, the stronger the impact of the independent variable is on the dependent variable. π is the probability that a the dependent variable is 0 or 1.

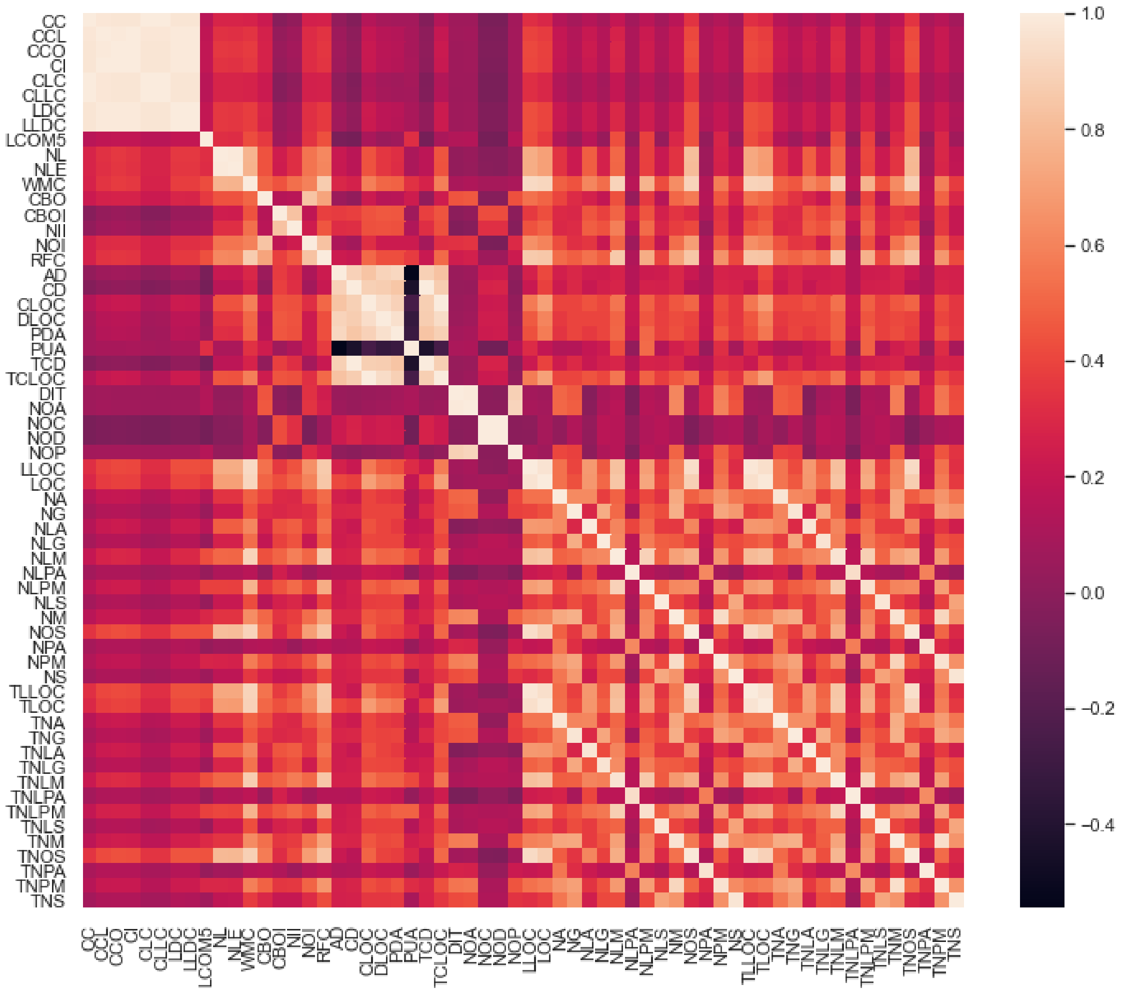

Fig. 4. A Heat Map Showing the Spearman Correlation Coefficients between the Software Metrics Existing in the used Dataset.

Support Vector Machine (SVM): SVM is a discriminative classifier algorithm that separates data samples into (generally) two different classes. The data samples are represented on 2-dimensional space and SVM tries to find an optimal hyperplane that divides the 2-dimensional space into two parts such that the data samples of each class reside on one part.

K-Nearest Neighbor (KNN): KNN is simple algorithm that solves the machine leaning classification problems by finding the k nearest neighbours (which has been previously classified) to the data point to be classified. Then KNN classifies the data point to the most frequent class in k nearest neighbours.

Naive Bayes (NB): NB a simple technique that is used to build a probabilistic classifier based on Bayes' theorem. NB classifiers assume that the input features are statistically independent. Thus, each feature contributes independently to the probability that an instance data belongs to a certain class regardless the correlation between the considered feature and other features.

Decision Tree (DT): DT is a learning technique that constructs a decision tree model for classification problems based on the given dataset. Each internal node in the constructed tree symbolizes a test condition on a feature, each branch denotes an output of a test condition, and each leaf node denotes a label (or a decision).

Random Forest (RF): RF is an ensemble learning method that uses a set of unpruned decision trees for classification. The decision trees are constructed based samples of the dataset. During the classification of a data sample, the data sample is given to each decision tree and the class label of the instance is determined by taking the mode of the outputs of the decision trees.

Bagging: Bagging (aka Bootstrap aggregating) is an ensemble learning method that aims to improve the accuracy of machine learning algorithms most commonly decision trees. It reduces the variance of a model and helps to avoid the overfitting of data. It creates n subdatasets each of which contains a subset of features and data samples that are randomly selected with replacement from the original training dataset. The n subdatasets are used in parallel to construct n base (or weak) prediction models. The label class predicted by the majority of the base models is chosen to be the output of the bagging classifier.

AdaBoost: AdaBoost (stands for Adaptive Boosting) is an ensemble learning method that is used to combine multiple weak classifiers into a single strong classifier for the purpose of improving the accuracy and performance of the weak classifiers. The weak classifiers in Adaboost are usually decision stumps which are decision tree with just one node (the root) and two leaves. The weak classifiers are trained sequentially. Samples misclassified by a weak classifier are given more weight in subsequent classifiers. Also each weak classifier is given a weight according to its accuracy. The final output of Adaboost classifier is the weighted sum of the outputs of the weak classifiers.

G. Performance Evaluation

Three performance metrics are used to evaluate the performance of the constructed models including accuracy (ACC), F-measure (F1), and Area under the ROC Curve (AUC). These metrics are widely used in the literature to evaluate SBP models.

Accuracy measures the fraction of the predictions that are classified correctly by a model. The value of accuracy ranges from 0 to 1 where a higher value means better accuracy. It is calculated for a binary classification according to the following equation:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

The F-measure is a commonly used performance metric (especially in case of the existence of an imbalanced classification problem) that considers both the precision and recall of a model. It is the harmonic mean of the precision and recall and its value ranges from 0 to 1 where a large value means a better performance. The following equation is used to compute the F-measure:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

where the Precision and Recall are calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

AUC is a performance metric for classification problems across all the various classification thresholds. The value of AUC lies between the closed interval [0, 1] where a larger value of AUC means better performance for a classifier. AUC is calculated based on the following equation:

$$AUC = \frac{\sum rank(All\, Positive\, Samples) - \frac{X(X+1)}{2}}{P * N} \tag{6}$$

Where P and N represent the number of positive and negative samples, respectively.

H. Tools

All the experiments in this study are implemented using the open source scikit-learn tools [26]. They are simple and efficient tools used to implement machine learning techniques in Python.

IV. Results and Discussion

Following the research framework shown in Fig. 2, eight SBP models were constructed using the considered machine learning techniques and the used bug dataset. Out of the 60 features (Software metrics), 25 features were selected to construct the models after the application of the correlation-based filter-subset feature selection with best first search. The results of the considered performance metrics for each constructed model are given in Fig. 5.

The highest accuracy value is 0.82 which is achieved by LR, SVM, and RF classifiers. The accuracy values of NB, Adaboost, and KNN are 0.81, 0.8, and 0.79, respectively. The Bagging classifier has an accuracy of 0.76. DT has the lowest accuracy value which is 0.7.

LR, SVM, and RF classifiers attained the highest AUC value (0.77). NB achieved the second largest value of AUC (0.76). Adaboost has a value of 0.73 for AUC. The values of AUC for KNN, Bagging, and DT are 0.67, 0.62, and 0.53, respectively, which are relatively low compared to the values of AUC for other classifiers.

For the F1 values, LR and NB have the highest value (0.77). SVM, RF, and Adaboost attained the second largest value of F1 (0.76). KNN, Bagging, and DT have F1 values of 0.75, 0.73, and 0.7, respectively.

Answering RQ: From the results depicted in Fig. 5, it can be said that LR is the most effective machine learning technique (in terms of performance metrics) for bug prediction in classes based on the Unified Bug Dataset as it achieved the highest values of accuracy, AUC, and F1 measure. However, the performance of LR is not significantly better than the performance of the other considered techniques. In fact, some of the other classifiers have exactly the same performance of LR (for some of the used performance metrics) such as SVM and RF for the accuracy and AUC metrics and NB for the F1 metric.

V. Threats to Validity

There are several issues that may impact the results of this study and limit their generality.

The quality of the bug dataset used to build the SBP models was not evaluated in this paper. The values of the software metrics and the bug information were used in all the experiments conducted in this study without verification or validation. However, the dataset was extensively evaluated and validated in [10] and it has been used in other studies in the literature (e.g. [11])

Software metrics included in the used dataset are not the only factors that can have impact on software bug proneness. Other factors such as the experience of software engineers involved in the development process of a software unit (e.g., class) and development environment can also make a software

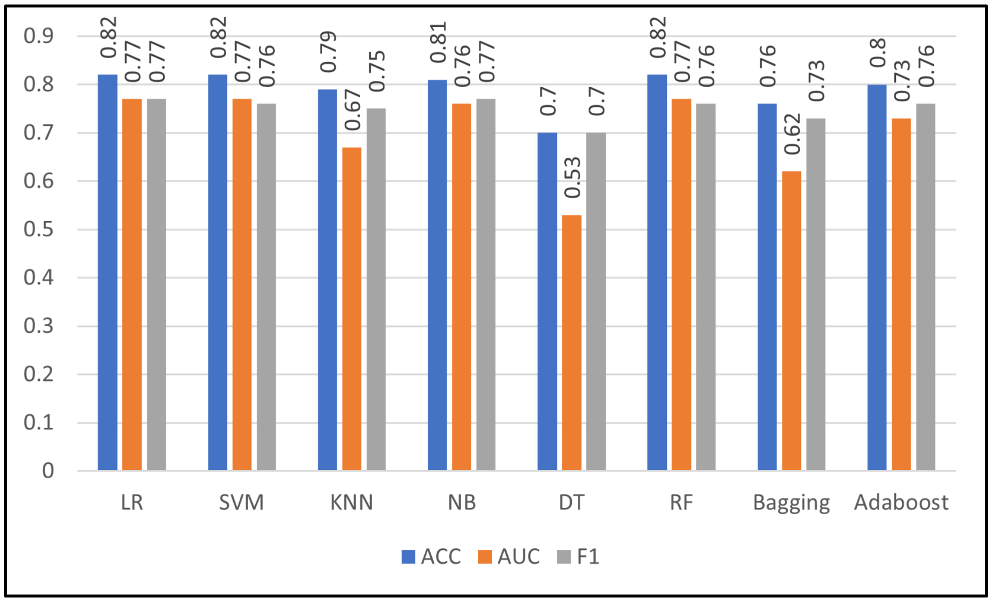

Fig. 5. The Average Results of the used Performance Metrics for the Constructed Models.

unit to be bug prone. However, these factors are out of the scope of this study and the aim of the study is to compare the performance of well-known machine learning techniques based on the used bug dataset which only includes software metrics and bug information.

The performance of the SBP models constructed in this study were evaluated based on only accuracy, F-1, and AUC. There are several other well-known performance metrics which were not used in this study such as Precision, Recall, Balance, and G-mean. However, there is no previous study on SBP that has used all the existing performance metrics to evaluate SBP models. In this study, the accuracy was used because it is one of the most commonly used metric to evaluate the performance of prediction models. In addition, the F-1 and AUC metrics were used in this study because the considered dataset is greatly imbalanced and these two metrics are good performance metrics for evaluating models constructed based on a biased dataset.

VI. Conclusion and Future Work

In this study, the performance of 8 widely used machine learning techniques were investigated. A set of experiments were conducted using the Unified Bug Dataset which includes bug information for 47,618 classes. LR was found to be the most effective technique for predicting buggy classes. It attained 0.82, 0.77, and 0.77 for the accuracy, AUC, and F1, respectively.

The correlation-based filter-subset with best first search was the only feature selection technique applied in this study. There are many other feature selection and transformation techniques that have been introduced in the literature. A future work can extend this study by comparing the performance the considered 8 machine learning techniques when applying different feature selection and transformation techniques.

References

[1] A. O. Balogun, S. Basri, L. F. Capretz, S. Mahamad, A. A. Imam, M. A. Almomani, V. E. Adeyemo, A. K. Alazzawi, A. O. Bajeh, and G. Kumar, "Software defect prediction using wrapper feature selection based on dynamic re-reranking strategy," *Symmetry*, vol. 13, no. 11, p. 2166, 2021.

[2] V. Nguyen, V. Pham, and V. Lam, "qestimation: a process for estimating size and effort of software testing," in *Proceedings of the 2013 International Conference on Software and System Process*, 2013, pp. 20–28.

[3] H. Ohtera and S. Yamada, "Optimal allocation and control problems for software-testing resources," *IEEE Transactions on Reliability*, vol. 39, no. 2, pp. 171–176, 1990.

[4] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[5] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[6] X. Yang, H. Yu, G. Fan, and K. Yang, "A differential evolution-based approach for effort-aware just-in-time software defect prediction," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Representation Learning for Software Engineering and Program Languages*, 2020, pp. 13–16.

[7] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2009, pp. 223–234.

[8] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter optimization to improve bug prediction accuracy," in *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2017, pp. 33–38.

[9] [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[10] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java and its assessment regarding metrics and bug prediction," *Software Quality Journal*, vol. 28, no. 4, pp. 1447–1506, 2020.

[11] Z. J. Szamosvölgyi, E. T. Váradi, Z. Tóth, J. Jász, and R. Ferenc, "Assessing ensemble learning techniques in bug prediction," in *International Conference on Computational Science and Its Applications*. Springer, 2021, pp. 368–381.

[12] T. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[13] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[14] S. Chidamber and C. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[15] T. Wang and W.-h. Li, “Naive bayes software defect prediction model,” in *2010 International Conference on Computational Intelligence and Software Engineering*. Ieee, 2010, pp. 1–4.

[16] P. D. Singh and A. Chug, “Software defect prediction analysis using machine learning algorithms,” in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. IEEE, 2017, pp. 775–781.

[17] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, and T. R. Soomro, “Software defect prediction using ensemble learning: A systematic literature review,” *IEEE Access*, 2021.

[18] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, “The jinx on the nasa software defect data sets,” in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–5.

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

[20] Sed-Inf-U-Szeged, “Sed-inf-u-szeged/openstaticanalyzer: Openstaticanalyzer is a source code analyzer tool, which can perform deep static analysis of the source code of complex systems.” [Online]. Available: https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer

[21] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in *Proceedings of the 6th international conference on predictive models in software engineering*, 2010, pp. 1–10.

[22] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Third International Workshop on Predictor Models in Software Engineering (PROMISE’07: ICSE Workshops 2007)*. IEEE, 2007, pp. 9–9.

[23] M. D’Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” in *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. IEEE, 2010, pp. 31–41.

[24] T. Hall, M. Zhang, D. Bowes, and Y. Sun, “Some code smells have a significant but small effect on faults,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 1–39, 2014.

[25] Z. Tóth, P. Gyimesi, and R. Ferenc, “A public bug database of github projects and its application in bug prediction,” in *International Conference on Computational Science and Its Applications*. Springer, 2016, pp. 625–638.

[26] “Sklearn.preprocessing.minmaxscaler.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[27] B. Ghotra, S. McIntosh, and A. E. Hassan, “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)*. IEEE, 2017, pp. 146–157.

[28] Z. Mahmood, D. Bowes, P. C. Lane, and T. Hall, “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*, 2015, pp. 1–4.

[29] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, “A two-stage data preprocessing approach for software fault prediction,” in *2014 Eighth International Conference on Software Security and Reliability (SERE)*. IEEE, 2014, pp. 20–29.

[30] M. Liu, L. Miao, and D. Zhang, “Two-stage cost-sensitive learning for software defect prediction,” *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 676–686, 2014.

[31] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,” *Information and Software Technology*, vol. 59, pp. 170–190, 2015.