A real time analysis on different machine learning models for software defect testing

Dr K Butchi Raju, Department of CSE, GRIET, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, INDIA, butchiraju.katari@gmail.com

Abstract

As the size of the defects increases, it becomes difficult to predict the different types of software defects with high true positive rate. The main objective of the machine learning models for software defect-based testing application is to improve the defect prediction rate with less error rate. Evaluating the software metrics and defect prediction are the two key quality features that determine the success of a software product. Most of the conventional meta-heuristic-based software defect testing models are independent of dynamic parameters estimation. Also, these conventional models are used to predict the defect in the homogeneous software testing systems with limited number of feature space. In this paper, different types of software defect prediction systems and its models are discussed along with the limitations on various metrics.

Keyword: Software defect prediction, software testing, machine learning models, classification models.

I. Introduction

Detection and analysis of software defects at a very early stage is very much essential in the domain of software engineering. It also influences the decision-making process related to allocation of resources for evaluation or verification. Software quality assurance can be defined as a significant phenomenon for the implementation of various machine learning techniques in defect detection. These techniques basically emphasize on single product-based software defects rather than the multi-product-based defects. Detection and analysis of software defects at a very early stage is very much essential in the domain of software engineering. It also influences the decision-making process related to allocation of resources for evaluation or verification. Software quality assurance can be defined as a significant phenomenon for the implementation of various machine learning techniques in defect detection. These techniques basically emphasize on single product-based software defects rather than the multi-product-based defects. In recent decades, the size of the object-oriented defects increases, the prediction of multi-level defects also increasing exponentially. The main objective of the software defect prediction models is to improve the true positive rate of the defects with minimum time and cost. Traditional software prediction classifiers are developed to assess the metrics in the application level. Bayesian network (BN), Naïve Bayes, SVM, linear regression approaches as well as bagging approaches are used to assess the software defects with limited feature space. Most of the traditional software defect prediction models are focused on limited defect features in a single application. One of the major limitations is that, lack of training information in the early phases of software testing process. As the size of the metrics increases, it is difficult to process high dimensional features due to impact on memory space and time. Find Bugs framework use BCEL Java binary parser for binary code pattern matching by class structure analysis, linear code analysis, control flow analysis, and data flow analyses[1]. In [2], the authors introduce Find Bugs approach for finding concurrency bugs in Java programs. Contest infrastructure has a feature which utilizes bug patterns to trigger concurrency errors during test runs. It would be appropriate to mention that software defects reduce the quality of software, increase costs and delay the schedule of development. A software development team can forecast the possible bug and its severity in the initial stage of software development through software defects prediction techniques. The process of locating defective components in the software is known as Software Default Prediction prior to the start of the testing stage. One of the most active areas of software engineering research is prediction of software defects that lead to increased customer satisfaction, more reliable software, reduced development time, and reduced rework effort and cost-effectiveness[3]. Prediction is known as the job of predicting continuous or ordered values for a given input. Thus, the practice of predicting defects is considered to be extremely important in order to achieve software quality and to learn from earlier errors. Software metrics[4][5] is considered one of the components required to identify and predict the software defect. However, identifying the correct software metrics is a major challenge for the developer.

Defect forecast offers an optimized manner to identify the vulnerabilities that occur owing to manual or automatic mistakes in the SDLC stages. As software program addiction increases, software quality in the present era is becoming increasingly crucial. Software defects such as errors and faults may influence the software quality resulting in client discontent [6]. It is too hard to create a quality end item due to the growing software limitations and modular nature.

A software test is a study to inform stakeholders about the quality of the tested product or service. A series of software error detection activities. Testing is a process that is used to detect computer software correctness, integrity and quality. A Software Defect / Bug is a condition of a software product which does not meet the expectations or requirements of the user (not specified but reasonable). In other words, a malfunctioning program or incorrect coding or logic error produces wrong, unintentional findings. The current forecasting work concentrates on estimating the number of faults in software systems; (ii) the discovery of fault associations and (iii) classification of fault-pronounced software components, which are typically faulted rather than fault-pronounced, in two classes. The second type of work is carried out by the community association of data mining to disclose software defects that can be used for three purposes[7]. This technique first finds a candidate concurrency bugs through code patterns. And then, it inserts noise injections at the candidate bug site in order to detect concurrency bugs with high probability in testing[8]. Upon ConTest, this technique contributes to active testing of concurrent Java program. They adopted bug patterns for assisting code review process[9]. The authors extend the regular expression in Perl language for bug specifications and bug detections. As a preprocessing to code review by experts, this technique automatically attach the comments on a code which is corresponding to a given bug specification[10].As the software industry evolves, the monitoring and enhancement of software quality is increasingly engaged in software businesses. In 1992 IBM conceived the Orthogonal Default Classification (ODC) in quantitative and qualitative assessment to satisfy these criteria. Software defect prediction is a significant guidance for studies into software reliability[11]. The technology for defect prediction can be used to discover high-risk software module. Software designers can focus on risky modules with more defects to save costly testing and time[12], and then use a restricted test funds for risky modules.The significant thing is to discover high-risk modules in the software goods for anticipating software errors. In the classification method the characteristics in the samples play various roles for issues of classification. At the same moment, the interaction between the different characteristics impacts classification performance[13]. Very little study focuses on relationships between attributes. In most times the characteristics of traditional algorithms are always presumed to be distinct during the classification phase. In practical issues however, the interplay of characteristics occurs. Therefore, when predicting software defects, the interaction between the characteristics must be taken into account[14].Fuzzy integral is a non-linear component, based on fuzzy measures. The non-additiveness of fluid measurements makes the interaction between classification features complementary to the fluid. The fluorescent measures corresponding to these are essential to achieving high-grade efficiency in the essential classification of the fluorescent. In general, the measurement of fluctuations is very complex[15]. The reciprocal data between characteristics is an significant tool for efficient assessment of the related degree between attributes[6] in order to evaluate the correlations between the characteristics of data theory.

Open-default information metrics are evaluated and a sub-set of smaller-scale software metrics are developed from current NASE project information from PROMISE[16]. Research has shown that, compared to two other kinds of subset function algorithms, the suggested Algorithm enhances the efficiency of the three famously classified kinds.Most techniques of this approach extend the type systems of existing programming languages. The extended type systems check a program correctly follows a given programming rules to ensure that the desired properties holds for the program. The type systems are normally implemented as a part of compilers. [17]Introduce an extended Java type system to avoid concurrency errors including deadlock and data race. This type system requires every shared member has a specification of its synchronization object. However, there are two shortcomings of this approach to be applied to general programs. First, these systems restrict programmers to write codes in simple manner strictly. Second, these methods require programmers to specify the additional information related to synchronization used in a code[18]. These two shortcomings are unfeasible for targeting system programs. In system programs, fine tuning of synchronization operations are common in order to improve performance. Moreover, the size of program is normally too large for programmers to give the additional information manually.

II.Related Works

Rossa[19] used Complexity Metrics to predict defects. Complexity metrics are considered to be superior predictors of potential fault compared to other reputed past fault predictors, i.e. previous alterations and previous errors. By knowing which program is prone to defects, the development process or the program can be correlated with defect density. The database of bugs is the reliable basis for information about malfunctions. The code that changes a lot is more likely to fail than the unchanged code. The techniques of machine learning have a higher accuracy rate and are therefore much more stable. Some researchers think that an optimum place can be used by a single method of choosing characteristics. Therefore, techniques such as a ensemble technology can be promoted, which incorporates distinct selection techniques, not a single method,[20] and an iteration technique that repeatedly re-examples the features. Software metrics are also used using other techniques such as correlation assessment, logistic regression[21], and mutual information analysis[8]. There are research.Dynamic analysis techniques aim to verify a certain property of a program by evaluating its actual executions. By observing internal states during target program executions, the dynamic analysis techniques can use accurate information of program behaviors. In dynamic analysis, it is possible to achieve value-sensitive and alias-sensitive analysis with much less computation cost than in static analysis.

Dynamic analysis extends traditional testing to check meaningful properties using intermediate state information in program executions. Dynamic analytical methods share inherently the test constraints. Full evaluation for target programs can not be supported by dynamic analysis because the controlled partial conduct of the goal programs is used. The other restriction is that it is hard to apply dynamic analysis methods unless target programs are full. Executable settings and sample instances are required for the dynamic analysis technique. These can only be provided at the subsequent stage of software development, in particular for embedded software[22].

The defect is a software program flaw that may lead the program to fail to fulfill its tasks. Defect forecast offers an optimized manner to identify the vulnerabilities that occur owing to manual or automatic mistakes in the SDLC stages. As software program addiction increases, software quality in the present era is becoming increasingly crucial. Software defects such as mistakes and faults can influence the software quality that contributes to discontent with the client. Software metrics and computations are instruments or procedures that include software project or system evaluation or evaluation in order to provide constant or nominal characteristics The results were compared to the performance of the classifier proposed with 17 other data extraction technologies. In general, the predictive precision metric classification is helpful. The results also demonstrated the importance not as generally assumed of specific classification algorithms. Only eleven program and classification-engine data software metrics were calculated. The error rate was 10% and the classification was high. The Bayesian Naïve algorithm was tested for by [23]. In data mining and machine learning, Naïve Bayes is one of the most common learning algorithms. It is popular thanks to its effective inductive learning algorithms. Due to its conditional independence, the Naïve Bayes classification provides extremely competitive performance. [24] tried and identified and implemented metrics in a common dataset, to provide Software Engineered Management software reliability. This research aims to improve all actions by combining additional metrics. The testing of the Artificial Immunology System Classifier has produced good results. A new method for analyzing software systems ' defective distributions was proposed by [25]. While the prediction systems have local meaning only, it is not necessary to use OO technology suites; thus measuring technology becomes easier to access. A machine study classification was used to assess the similarity between previous changes or clear changes in buggies. The change classification has been predicted for bugs. There are several software metrics available in the model of defect prediction, such as product and process metrics. Successful quality control of software involves prevention of defects, removal of defects, and measuring defects. Prevention of defects includes all activities that in the first place minimize the likelihood of creating an error or defect. Default gaging consists of different matrices of detected malfunctions during the development phase and includes the deficiencies that the customer is pointing out after release. Default removal is responsible for all activities that detect and eliminate deficiencies and errors in any type of deliverable product[26]. For example, software metrics, product metrics and process metrics are at the heart of models for bug prediction.

The classification system is trained with functions from the history of software revision that classifies software changes as buggy or clean when applied. There was a 78 percent accuracy in the results. The changes were higher due to a small granular prediction and the seminal information on the source code was not required for classification. A wide range of programming languages are used for the changes of classification. The best way to model the software components at various failure levels is to have the strong back propagation algorithm based on the neural network. [27] introduced a software reliability evaluation methodology for Fuzzy-Neural. This paper identified an adaptive network-based fluid inference system (ANFIS) reliability prevision model to enhance the evaluation accuracy. The model uses the software's reliable information as input data (default lines every thousand lines), using reliability prediction as output data, the neural network trainings Adaptive–Fuzzy, membership of defect counters every thousand lines. The new software defect benchmark frame was presented by [28]. This includes both evaluation and forecasting. During the evaluation phase, the selected scheme evaluates various systems of learning. In the prediction phase, a predictor with all historical data is then used with the best learning scheme. Finally, the predictor is employed for the prediction of the new data defect. Bayesian networks used in [29] to determine the likelihood of influence between software metrics and defect proneness. [30] compares the k-NN Network which has been implemented as either fault or non-default susceptible in classifying software components. The hierarchical clustering technique The performance is better compared with the cluster-based approach when it comes to neural network approach. A general software defect prediction framework was proposed and evaluated by [31] to support the biased and thorough comparison between competing systems. The results show that different learning arrangements (i.e. no dominant scheme) for different data sets should be selected. Multiple data sets from software projects have been used to model software quality in order to resolve this problem. Because there are numerous defects in just a few of the modules, it is necessary to investigate the modules which are severely affected in comparison to other modules. The use of the Neural Feed Network was examined by [32].

Multiple Linear Regression (MLR), logistic regression for data modelling[33] used the earliest attempts to model maintainability and defects with static source code metrics as predictors. The problem with MLR is that it is not easy to interpret relationships between predictor software metrics and response variables[34]. The choice of modelling technique greatly influences the accuracy of maintenance and prediction of defects models, but different prediction model comparison studies have reached different, inconsistent and divergent conclusions about the superiority of one modelling technique over the other[35]. [36] conducted a systematic review of the prediction of defects and found that two-thirds of the prediction of defects studies were based on private datasets and their results could not be verified. With a small number of private data sets, different experiment design, different measurements of accuracy and lack of application of statistical significance tests, it is not possible to understand the strengths and weaknesses of different machine learningtechniques[37]. The work embodied in this work concerns the comparison with public data sets of a wide range of machine learning techniques for early maintenance and prediction defects. Public data sets allow other researchers to examine the validity of proposed models by replicating experiments and constructing reproducible or refutable models[38]. The work in this work also evaluates machine learning techniques as predictor variables for defects with different metric categories such as source code metrics[39], micro-interaction metrics[40] and software entropy metrics.[41] presented an effective multi-objective naïve Bayes learning for cross-project defect prediction [42]. They introduced multi objective learning mechanisms and implemented those in cross project environments. This approach has three prime objectives and those objectives completely depend upon the process of class imbalance. In this piece of research work, a new algorithm known as harmony search algorithm is implemented. The above proposed algorithm has the responsibility of resolving multi objective Bayes issues. Numbers of solutions along with various PD, PF balance values are generated by analysing the source data. After that NB or NBNN is constructed along with an individual optimal solution. Additionally, it can determine the fault proneness of the targeted data.It is responsible for producing frequent item sets for every individual partition. The item set has numbers of abnormalities and known as focused item set. Depending upon real item set, they introduced a new pre-processing technique which is responsible for setting real items those are missing in partition only. These changed data have significant role during the development process of Naïve Bayes classifier. It is also responsible for detection of defective software modules. In the evaluation phase, the performance of NB model with ten bins is considered. It can be noticed that, this performance is not much satisfactory. It may either increase or decrease with respective to the inclusion of missing item sets.

Maintenance Index (MI)[43] is a traditional model used to predict software application maintenance. It includes various metrics of Halstead, cyclomatic complexity of McCabe, lines of code (LOC) and number of comments. Researchers criticized MI model as they found problems applying this model to large and diverse collections of mission-critical projects. Recently, [44] found that MI's predicted for five software system releases were the same where the actual maintenance effort observed to maintain these systems varied considerably. [45] proposed a linear software maintenance prediction model based on a minimum set of software design level metrics.[46] studied for a maintenance period of three years two commercial object-oriented systems and developed a predictability model using Multiple Linear Regression (MLR). [47] studied C++ systems software maintenance using MLR as predictors with object-oriented metrics. [48] studied the relationship with the MLR modeling technique between design metrics and sustainability. Machine learning techniques[49] were not considered in these studies. When there is no acceptable theory that can relate maintenance to its software predictor metrics, parametric techniques such as MLR are not useful. Machine learning techniques can therefore be used to predict the maintenance of software because they are non-parametric in nature. [50] used Bayes Network, Regression Trees and MLR to predict the maintenance of software. They concluded that only one system studied was superior to MLR by the Bayes Network. These studies[51-53] did not compare their results with other techniques of machine learning or MLR or used different measurements of accuracy, so the results were not comparable. [54] used fuzzy logic techniques for software maintenance measurement. [55]conducted a systematic review of the prediction and metrics of software maintenance. Various predictors of maintenance collected at source code level, maintenance prediction techniques, accuracy measurements and maintenance metrics have been summarized. It was concluded that there were no obvious choices to build predictive models for maintainability. A number of techniques for machine learning were investigated in prediction of defects. The nature of data sets for defect prediction is skewed[56]. Non-defective modules are negative examples (or negative class, majority class) in terms of machine learning literature, and defective modules in training data are positive examples (or positive class, minority class). This is referred to as the problem of class imbalance. Class imbalance greatly degrades the performance of machine learning techniques. [57] investigated the methods of ensembles bagging and boosting over NASA MDP datasets and found that ensembles bagging and boosting were more accurate than single base classifiers (learners). In Bagging and Boosting Ensembles, they employed seven base learners. However, through statistical significance tests, they did not evaluate their models. It involves an unbiased and comprehensive comparison technique. A minor modification during the evaluation phase may influence the resulted outcomes significantly. This suggested technique is very much efficient for real world implementation irrespective of the nature of data. There are certain cases, where data is skewed. Model cannot predict sufficient number of defective instances for the process of learning. Suppose a method is performing very well in case of balanced dataset, it will result poorest performance in case of imbalanced dataset.

Selecting the most suitable set of attributes that represent a problem, from large set of attributes is also a challenging task. Some attributes might be irrelevant, redundant, or containing useful information only when combined together. We must select the best possible features before feeding them into the algorithm since this influence the quality of the prediction model as well as the computer resources (such as calculation time, memory usage *etc.*). In feature selection, the wrapper is the model evaluation based on different feature combinations. The evaluation result (e.g. the accuracy from a 10-fold cross validation) allows the identification of the best-performing model and thus, the best-performing feature combination. So, the best performing feature combination is the feature combination to select from all features. There are three decisions to make to perform this kind of feature selection. First, what is the selection criterion to apply. Typically, the outcome of a classifier evaluation is the accuracy or the area under the ROC curve AUC[58]. These measures are the mostly used selection criteria following the rule: the higher, the better. Second, which algorithm to use Although, the wrapper approach is concerned to be a black box approach to score the feature sub-sets, the algorithm choice has some influence on the results of the final model. Third, we have to determine the appropriate search strategy. Ideally, wrapper methods would make use of all possible feature combinations to determine the feature contributions (exhaustive, complete search).

In the feature selection, there are two fundamental search procedures, the forward and backward selection. Forward selection starts from scratch and adds new variables one-by-one while evaluating the optimal search path. The backward selection does the opposite: the search starts from a model based on all variables and eliminates one-by-one. The results of both approaches can differ due to non-independent variables and different stopping points when a certain quality threshold value is reached. In other wrapper application fields also other search techniques such as evolutionary search and simulated annealing are used.In a larger dataset, not all variables are so important to consider, the greater the number of variables, the greater the complexity.

III.Conclusion

In this paper, various machine learning models are discussed on software testing defect databases along with variation in the software metrics. Most of the conventional software defect prediction modelsare difficult to handle large heterogeneous data types for feature extraction and classification processfor software testing systems. In this paper, different feature selection measures and meta-heuristic classification models are studied for software defect prediction process. In the future work, a hybrid meta-heuristic based software defect classification framework is designed to improve the decision making process of software testing systems.

References

[1]Z. Xu et al., “A comprehensive comparative study of clustering-based unsupervised defect prediction models,” Journal of Systems and Software, vol. 172, p. 110862, Feb. 2021, doi: 10.1016/j.jss.2020.110862.

[2]F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, “A large empirical assessment of the role of data balancing in machine-learning-based code smell detection,” Journal of Systems and Software, vol. 169, p. 110693, Nov. 2020, doi: 10.1016/j.jss.2020.110693.

[3]D.-L. Miholca, G. Czibula, and I. G. Czibula, “A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks,” Information Sciences, vol. 441, pp. 152–170, May 2018, doi: 10.1016/j.ins.2018.02.027.

[4]Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," Expert Systems with Applications, vol. 114, pp. 237–254, Dec. 2018, doi: 10.1016/j.eswa.2018.07.042.

[5]H. Alsolai and M. Roper, "A systematic literature review of machine learning techniques for software maintainability prediction," Information and Software Technology, vol. 119, p. 106214, Mar. 2020, doi: 10.1016/j.infsof.2019.106214.

[6]R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504–518, Feb. 2015, doi: 10.1016/j.asoc.2014.11.023.

[7]N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," Information and Software Technology, vol. 122, p. 106287, Jun. 2020, doi: 10.1016/j.infsof.2020.106287.

[8]C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," Information and Software Technology, vol. 107, pp. 125–136, Mar. 2019, doi: 10.1016/j.infsof.2018.11.005.

[9]P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," Journal of Systems and Software, vol. 137, pp. 184–196, Mar. 2018, doi: 10.1016/j.jss.2017.11.066.

[10]R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software," Applied Soft Computing, vol. 49, pp. 1034–1050, Dec. 2016, doi: 10.1016/j.asoc.2016.04.032.

[11]R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," Neurocomputing, vol. 343, pp. 120–140, May 2019, doi: 10.1016/j.neucom.2018.04.090.

[12]Y. Zhang, D. Jin, Y. Xing, and Y. Gong, "Automated defect identification via path analysis-based features with transfer learning," Journal of Systems and Software, vol. 166, p. 110585, Aug. 2020, doi: 10.1016/j.jss.2020.110585.

[13]F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro, and J. Bernardino, “Automating orthogonal defect classification using machine learning algorithms,” Future Generation Computer Systems, vol. 102, pp. 932–947, Jan. 2020, doi: 10.1016/j.future.2019.09.009.

[14]S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques,” Expert Systems with Applications, vol. 144, p. 113085, Apr. 2020, doi: 10.1016/j.eswa.2019.113085.

[15]D. P. P. Mesquita, L. S. Rocha, J. P. P. Gomes, and A. R. Rocha Neto, “Classification with reject option for software defect prediction,” Applied Soft Computing, vol. 49, pp. 1085–1093, Dec. 2016, doi: 10.1016/j.asoc.2016.06.023.

[16]Z. Sun, J. Zhang, H. Sun, and X. Zhu, “Collaborative filtering based recommendation of sampling methods for software defect prediction,” Applied Soft Computing, vol. 90, p. 106163, May 2020, doi: 10.1016/j.asoc.2020.106163.

[17]J. Chen, K. Hu, Y. Yang, Y. Liu, and Q. Xuan, “Collective transfer learning for defect prediction,” Neurocomputing, vol. 416, pp. 103–116, Nov. 2020, doi: 10.1016/j.neucom.2018.12.091.

[18]D.-L. Miholca, G. Czibula, and V. Tomescu, “COMET: A conceptual coupling based metrics suite for software defect prediction,” Procedia Computer Science, vol. 176, pp. 31–40, Jan. 2020, doi: 10.1016/j.procs.2020.08.004.

[19]R. Rossa, A. Borella, and N. Giani, “Comparison of machine learning models for the detection of partial defects in spent nuclear fuel,” Annals of Nuclear Energy, vol. 147, p. 107680, Nov. 2020, doi: 10.1016/j.anucene.2020.107680.

[20]S. Feng et al., “COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction,” Information and Software Technology, vol. 129, p. 106432, Jan. 2021, doi: 10.1016/j.infsof.2020.106432.

[21]C. Jin, “Cross-project software defect prediction based on domain adaptation learning and optimization,” Expert Systems with Applications, vol. 171, p. 114637, Jun. 2021, doi: 10.1016/j.eswa.2021.114637.

[22]R. Özakıncı and A. Tarhan, “Early software defect prediction: A systematic map and review,” Journal of Systems and Software, vol. 144, pp. 216–239, Oct. 2018, doi: 10.1016/j.jss.2018.06.025.

[23]A. T. Haouari, L. Souici-Meslati, F. Atil, and D. Meslati, “Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction,” Applied Soft Computing, vol. 96, p. 106686, Nov. 2020, doi: 10.1016/j.asoc.2020.106686.

[24]H. Wei, C. Hu, S. Chen, Y. Xue, and Q. Zhang, “Establishing a software defect prediction model via effective dimension reduction,” Information Sciences, vol. 477, pp. 399–409, Mar. 2019, doi: 10.1016/j.ins.2018.10.056.

[25]Z. Ding and L. Xing, “Improved software defect prediction using Pruned Histogram-based isolation forest,” Reliability Engineering & System Safety, vol. 204, p. 107170, Dec. 2020, doi: 10.1016/j.ress.2020.107170.

[26]T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, “Improving defect prediction with deep forest,” Information and Software Technology, vol. 114, pp. 204–216, Oct. 2019, doi: 10.1016/j.infsof.2019.07.003.

[27]X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, and D. Mu, “Improving high-impact bug report prediction with combination of interactive machine learning and active learning,” Information and Software Technology, vol. 133, p. 106530, May 2021, doi: 10.1016/j.infsof.2021.106530.

[28]Z. Xu et al., “LDFR: Learning deep feature representation for software defect prediction,” Journal of Systems and Software, vol. 158, p. 110402, Dec. 2019, doi: 10.1016/j.jss.2019.110402.

[29]O. Meqdadi, N. Alhindawi, J. Alsakran, A. Saifan, and H. Migdadi, “Mining software repositories for adaptive change commits using machine learning techniques,” Information and Software Technology, vol. 109, pp. 80–91, May 2019, doi: 10.1016/j.infsof.2019.01.008.

[30]X. Huo and M. Li, “On cost-effective software defect prediction: Classification or ranking?,” Neurocomputing, vol. 363, pp. 339–350, Oct. 2019, doi: 10.1016/j.neucom.2019.05.100.

[31]K. Shi, Y. Lu, J. Chang, and Z. Wei, "PathPair2Vec: An AST path pair-based code representation method for defect prediction," Journal of Computer Languages, vol. 59, p. 100979, Aug. 2020, doi: 10.1016/j.cola.2020.100979.

[32]P. K. Chaubey and T. K. Arora, "Software bug prediction and classification by global pooling of different activation of convolution layers," Materials Today: Proceedings, Dec. 2020, doi: 10.1016/j.matpr.2020.10.598.

[33]Y. Shao, B. Liu, S. Wang, and G. Li, "Software defect prediction based on correlation weighted class association rule mining," Knowledge-Based Systems, vol. 196, p. 105742, May 2020, doi: 10.1016/j.knosys.2020.105742.

[34]Z. Xu et al., "Software defect prediction based on kernel PCA and weighted extreme learning machine," Information and Software Technology, vol. 106, pp. 182–200, Feb. 2019, doi: 10.1016/j.infsof.2018.10.004.

[35]I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," Information and Software Technology, vol. 58, pp. 388–402, Feb. 2015, doi: 10.1016/j.infsof.2014.07.005.

[36]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, Apr. 2018, doi: 10.1016/j.infsof.2017.11.008.

[37]L. Zhao, Z. Shang, L. Zhao, T. Zhang, and Y. Y. Tang, "Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks," Neurocomputing, vol. 352, pp. 64–74, Aug. 2019, doi: 10.1016/j.neucom.2019.03.076.

[38]W. Rhmann, B. Pandey, G. Ansari, and D. K. Pandey, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study," Journal of King Saud University - Computer and Information Sciences, vol. 32, no. 4, pp. 419–424, May 2020, doi: 10.1016/j.jksuci.2019.03.006.

[39]X. Yang, D. Lo, X. Xia, and J. Sun, "TLEL: A two-layer ensemble learning approach for just-in-time defect prediction," Information and Software Technology, vol. 87, pp. 206–220, Jul. 2017, doi: 10.1016/j.infsof.2017.03.007.

[40]J. Cui, L. Wang, X. Zhao, and H. Zhang, “Towards predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications,” Computer Communications, vol. 155, pp. 125–131, Apr. 2020, doi: 10.1016/j.comcom.2020.02.078.

[41]Y. Ma, G. Luo, X. Zeng, and A. Chen, “Transfer learning for cross-company software defect prediction,” Information and Software Technology, vol. 54, no. 3, pp. 248–256, Mar. 2012, doi: 10.1016/j.infsof.2011.09.007.

[42]R. Malhotra and L. Bahl, “A defect tracking tool for open source software,” in 2017 2nd International Conference for Convergence in Technology (I2CT), Apr. 2017, pp. 901–905, doi: 10.1109/I2CT.2017.8226259.

[43]H. Chai, N. Zhang, B. Liu, and L. Tang, “A Software Defect Management System Based on Knowledge Base,” in 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Jul. 2018, pp. 652–653, doi: 10.1109/QRS-C.2018.00118.

[44]K. Okumoto, A. Asthana, and R. Mijumbi, “BRACE: Cloud-Based Software Reliability Assurance,” in 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Oct. 2017, pp. 57–60, doi: 10.1109/ISSREW.2017.48.

[45]T. Kim, J. Park, I. Kulida, and Y. Jang, “Concolic Testing Framework for Industrial Embedded Software,” in 2014 21st Asia-Pacific Software Engineering Conference, Dec. 2014, vol. 2, pp. 7–10, doi: 10.1109/APSEC.2014.82.

[46]M. Nafreen, M. Luperon, L. Fiondella, V. Nagaraju, Y. Shi, and T. Wandji, “Connecting Software Reliability Growth Models to Software Defect Tracking,” in 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Oct. 2020, pp. 138–147, doi: 10.1109/ISSRE5003.2020.00022.

[47]B. Doherty, A. Jelfs, A. Dasgupta, and P. Holden, “Defect Analysis in Large Scale Agile Development: Quality in the Agile Factory Model,” in 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), Oct. 2016, pp. 180–180, doi: 10.1109/IWSM-Mensura.2016.034.

[48]A. Perera, A. Aleti, M. Böhme, and B. Turhan, “Defect Prediction Guided Search-Based Software Testing,” in 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), Sep. 2020, pp. 448–460.

[49]L. C. Júnior, “Operational Profile and Software Testing: Aligning User Interest and Test Strategy,” in 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), Apr. 2019, pp. 492–494, doi: 10.1109/ICST.2019.00062.

[50]G. Ranieri, “Planning of Prioritized Test Procedures in Large Integrated Systems: Best Strategy of Defect Discovery and Early Stop of Testing Session, The Selex-ES Experience,” in 2014 IEEE International Symposium on Software Reliability Engineering Workshops, Nov. 2014, pp. 112–113, doi: 10.1109/ISSREW.2014.106.

[51]Y. Guo, M. Shepperd, and N. Li, “Poster: Bridging Effort-Aware Prediction and Strong Classification - A Just-in-Time Software Defect Prediction Study,” in 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), May 2018, pp. 325–326.

[52]A. Tosun, O. Turkgulu, D. Razon, H. Y. Aydemir, and A. Gureller, “Predicting Defects Using Test Execution Logs in an Industrial Setting,” in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), May 2017, pp. 294–296, doi: 10.1109/ICSE-C.2017.148.

[53]J. Gao, L. Zhang, F. Zhao, and Y. Zhai, “Research on Software Defect Classification,” in 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Mar. 2019, pp. 748–754, doi: 10.1109/ITNEC.2019.8729440.

[54]K. Sneha and G. M. Malle, “Research on software testing techniques and software automation testing tools,” in 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Aug. 2017, pp. 77–81, doi: 10.1109/ICECDS.2017.8389562.

[55]H. Chen, X. Wang, and L. Pan, “Research On Teaching Methods And Tools Of Software Testing,” in 2020 15th International Conference on Computer Science Education (ICCSE), Aug. 2020, pp. 760–763, doi: 10.1109/ICCSE49874.2020.9201788.

[56]Chun Shan, Boyang Chen, Changzhen Hu, Jingfeng Xue, and Ning Li, “Software defect prediction model based on LLE and SVM,” in 2014 Communications Security Conference (CSC 2014), May 2014, pp. 1–5, doi: 10.1049/cp.2014.0749.

[57]F. M. Tua and W. D. Sunindyo, “Software Defect Prediction Using Software Metrics with Naïve Bayes and Rule Mining Association Methods,” in 2019 5th International Conference on Science and Technology (ICST), Jul. 2019, vol. 1, pp. 1–5, doi: 10.1109/ICST47872.2019.9166448.

[58]D. Garg and A. Singhal, “A critical review of Artificial Bee Colony optimizing technique in software testing,” in 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), Feb. 2016, pp. 240–244, doi: 10.1109/ICICCS.2016.7542311.