Effective Prediction of Software Defects using Random-tree Entropy based Feature Selection Framework

Abdulaziz Alhumam
Department of Computer Science
College of Computer Sciences and Information Technology
King Faisal University, Al-Ahsa, Saudi Arabia

***Abstract*—Software systems have grown in size and complexity. These characteristics increase the difficulty of preventing software errors. As a result, forecasting the frequency of software module failures is critical to a developer's efficiency. Many methods for defect detection and correcting problems exist. Hence, Machine Learning (ML) classification performance has to be greatly improved. Thus, in this study, a novel approach is proposed for predicting the number of software defects based on relevant variables using ML. First, feature entropy on each raw features is performed and then identifying the un-pruned random feature. Then is selected the relevant feature through the identical existence among the entropy and un-pruned feature. And finally, the software defect dataset of National Aeronautics and Space Administration (NASA) PC-1 is sent to an ML-based model to estimate the number of faults. Initial PC-1 dataset comprises 37 raw features from this only 8 critical characteristics are utilized to enhance the ML model. A random tree feature selection strategy is shown to be accurate and potentially outperform existing methods in the experimental results. The proposed method considerably outperformed the performance of current ML models by obtaining the accuracy of 97.76% in Random Forest (RF) model.**

***Keywords*—Software defect prediction; machine learning; classification; feature entropy**

I. INTRODUCTION

In the recent years, the researcher tried to find different techniques and tools in taming the quality, dependability, and reliability of the software systems [1]. A software defect can cause minor inconvenience or catastrophic failure. Pre-deployment fault prediction for testing is supported by recent research in software fault prediction (SFP). Object-oriented programming is harder than procedural programming due of inheritance. By identifying faulty software modules before to the start of the testing process, software defect prediction can help enhance software quality and testing efficiency. These findings aid software engineers in allocating scarce resources to more prone-to-failure modules. Complex software application can deliver high efficient, accurate and powerful work to modern organizations [2]. Software defect prediction (SDP) has grown in popularity during the previous two decades. The results of the SDP assist in allocating resources for software testing. However, defect prediction is often employed for activities with a high degree of precision. It is difficult to ensure resource allocation prior to software testing or without prior execution data. Machine learning is used to identify problematic modules, as it reveals hidden patterns in software properties [3]. The feature selection activity removes non-classification features with low performance [4]. The variant selection activity selects the best versions of classification methods for their ensemble [5].

A data collection method based on regular expressions and bug-code linking [6] is proposed. In terms of accuracy and consistency, our strategy outperforms other commonly used data collection methods and their publicly available datasets [7]. Around 65 publicly available base datasets containing Chidamber and Kemerer (CK) and other inheritance indicators were used to determine the effect of inheritance on SFP [8]. They investigate the degree to which an inheritance metric accurately predicts software fault proneness. Additionally, they choose CK measures and inheritance metrics for predicting software problems. In SFP experiments, metrics such as exclusive usage and inheritance viability are analyzed [9]. They combed publicly available inheritance metrics data sets and discovered approximately 40 that contained inheritance metrics. Their initial cleanup included nine metrics relating to inheritance.

They preprocessed selected data sets and then merged them using all possible inheritance metrics combinations. The study [10] examined defect prediction datasets. There is no memory data management strategy proposed, nor is a mechanism for defect detection proposed. The proposed technique for defect prediction keeps track of the error rate performance. The defect prediction detector initiates the generation of defects, warning, and control flags. The proposed technique outperforms the conventional technique (p-value 0.05) and within-group comparisons yield statistically significant effect sizes. We observe that increasing the error rate results in DP, which results in suboptimal prediction performance. To overcome the difficulties associated with zero value thresholds, a spectral classifier based on the median absolute deviation threshold was developed [11]. Rather than using a measure of central tendency, this method makes use of the dispersion of eigenvector values. The report's baseline technique is a zero-value threshold spectral classifier, and the entity class is predicted using a heuristic technique.

The highest co-entropy criteria [12] successfully handle the non-Gaussian noise for SDP. A new classifier is created after instance filtering, feature selection, and reduction. It also finds a non-normal distribution for the 21 most significant software indicators. The hybrid feature selection (HFS) [13] is divided into two stages and it clusters features first using hierarchical agglomerative clustering and then eliminates un-normalized and duplicate features using two wrapper methods. Three distinct classifiers with four performance metrics were evaluated empirically on 11 well-studied NASA programs such as accuracy, precision, recall, and F-measure.

II. Related Work

To predict defects on NASA datasets, decision tree (DT), random forest (RF), Naive Bayes (NB), multi-layer perceptron (MLP), radial basis function (RBF), support vector machine (SVM), and k-nearest neighbour classifiers are used [14]. Precision, Recall, F-Measure, Accuracy, Matthew Correlation co-efficient, and ROC Area are used to evaluate classification performance. A two-stage data pre-processing method for software failure prediction models and semi-supervised deep fuzzy C-mean clustering feature extraction is presented [15]. The main goal is to optimise intra-cluster class and feature using deep multi-clusters of unlabelled and labelled data sets. A new strategy called conditional domain adversarial adaptation (CDAA) [16] can help with a variety of SDP problems. The CDAA has a generator, discriminator, and classifier. This is how the generator learns to move between spaces. The discriminator learns to spot the generator's bogus instances. The classifier learns to classify occurrences appropriately. In our CDAA, both classifier and discriminator loss functions propagate to generator. The enhanced wrapper feature selection (EWFS) [17] method selects features in stages while keeping previous choices in mind. This feature selection improves subset assessment while maintaining model performance. On software defect datasets of various granularities, the DT and NB classifiers were used to evaluate EWFS. This feature selection outperformed existing metaheuristics and sequential search-based WFS techniques in the experiments.

For feature exploration and categorization, neural forest (NF) [18] combines deep neural network with decision forest. After the neural network, a decision forest is connected to perform classification and guide feature representation learning. For efficient defect prediction, NF combines NN and decision forests, and the performance of this hybrid method is examined [19]. The hybrid approach [20] improved classification accuracy compared to existing methods. This method investigates the relationship between defect density, velocity, and introduction time. An integrated machine learning approach is used in ten PROMISE data sets with 22838 instances.

To see how FRFS (filter-based ranking feature selection) [21] methods affect software defect using feature selection methods that are too computationally costly. Empirically, they look at three large-scale web applications. Then they build SVP models using a random forest classifier and seven FRFS methods. To address the prediction model's low classification rates, a hybrid strategy called DELT (diverse ensemble learning technique) [22] is presented. Unlabelled test modules are predicted by majority voting. The DPAHM (Defect Prediction based association hierarchy method) [23] is used to allocate resources for coarse-level activities. FAHP (Fuzzy Analytical Hierarchy Process) is a prevalent multi-criteria decision-making method [24]. Conversely, this evaluation methodology employs a wide range of performance indicators. They may now trust study findings more, avoid misleading conclusions and set realistic restrictions. They employed 11 defect classifiers and 22 prominent performance measurements. The study used KNIME data mining and 12 NASA MDP software defect data sets.

With KMFOS, the class imbalance problem is solved [25]. KMFOS creates additional faulty instances by interpolating between two clusters. They would then spread out in the flawed dataset space. To reduce the noise, CLNI uses cluster-based oversampling. To develop an HDP model, a structured unsupervised deep domain adaptation is applied [26]. They start by combining data from both source and target projects into one statistic. The authors then develop an SNN (simple neural network) model to manage the various and class-imbalanced difficulties in SDP. The hybrid defect prediction model [27] uses the cross-entropy loss function as the classification loss function to reduce distribution mismatch. A heterogeneous defect prediction approach [28], [29] addresses the issue of extreme class imbalance in real-world software datasets. Minority samples in defect data are balanced using the Majority technique based on Mahalanobis distance in the first step. Ensemble learning and joint similarity measurement are used in the second stage to identify the most relevant and representative features across the source and target projects. At last, knowledge transmission from source to target project inside Grassmann manifold space.

The PROMISE Source Code (PSC) dataset was created to expand the CNN research's initial PSC dataset [30]. Our study used 30-repetition holdout and 10-fold cross-validation. An improved CNN model was then proposed and compared to previous CNN findings and an empirical study. It is used to identify contributing elements and independent variables [31]. Defect-free modules have their bugs replaced by a negative number, while faulty modules have their bugs left alone. Negate the false values of defect-free modules while increasing the false values of defective modules. In the next step, algorithms from NASA, SoftLab, and Promise are used. RKEE [32] is preceded by feature selection and rough set-based KNN noise filtering. Remove redundant features first using the feature ranking algorithm. A rough-KNN noise filter removes noisy samples from both minority and majority classes in the second stage. Both the minority and majority classes deal with ambiguity and overlap. NASA and Eclipse data sets have been used to test our technique.

There are considerable discrepancies in data sharing between the source and destination projects, which leads to inconsistencies in metrics. First, we present a clustering-based metric matching approach. An extract multi-granularity metric feature vector unifies the metric dimension while keeping maximum information. A strategy for predicting cross-project defects [33]. That is, it converts the project's original feature space into a manifold space, then uses that manifold space to train a superior naive Bayes prediction model. FSLBDA (few-shot learning based balanced distribution adaptation) technique [34] for unique defect prediction. Under-sampling can correct class imbalance in defect datasets, but reduces the size of training datasets. They remove redundant measurements from severe gradient boosting datasets. Dual innovative approaches [35] for learning from imbalanced data sets to improve minority class forecasting accuracy. These strategies try to distinguish between oversampling and misclassification costs. Experiment findings showed that identifying problematic modules accurately reduced detection system costs by G-mean and AUC. Instance weight is determined by information gravity among source and destination domains, whereas feature load is determined by high correlation with the learning goal, low correlation with other features, and low domain difference. Using 25 real-world datasets, the suggested methodology outperforms existing CPDP (cross project defect prediction) approaches [36]. The suggested approach builds a better CPDP model by allocating weights based on the varying contribution of characteristics and cases to the predictor.

III. Proposed Method

This section summarizes the software defect classification framework, as well as the significance of each feature. There are a total of 37 software defect attributes in total, with 8 significant features chosen for model performance evaluation. The proposed framework's system block diagram is shown in Fig. 1. NASA software defect datasets must be analyzed using machine learning models. The model is trained using six classification methods in this experiment: DT, EB, RF, SVM, LM, and NN. The experiments are carried out with the help of the R programming language, which trains models to classify software defects. The RF random tree and DT entropy have used feature values for each measurement and measurement class as inputs.

A. Dataset Description

The publicly available NASA Defect Dataset of PC1 was used in this study which is presented in Table I. In the dataset, there are 759 samples and 37 features, respectively. Lines of code, normalised cyclomatic complexity, cyclomatic density, essential complexity, maintenance severity, halstead content, halstead difficulty, parameter count, and other metrics are included in the data as presented in the Table II.

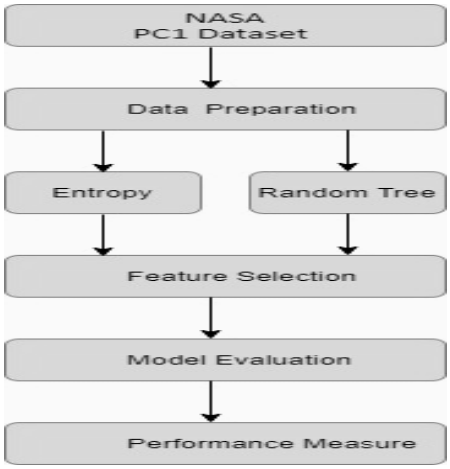

Fig. 1. Experiment Workflow with NASA– PC1.

TABLE I. PC1 Features of NASA Data Set

F1_Loc_blank	F19_Halstead_difficulty
F2_Branch_count	F20_Halstead_effort
F3_Call_pairs	F21_Halstead_error_est
F4_Loc_code_and_comment	F22_Halstead_length
F5_Loc_comments	F23_Halstead_level
F6_Condition_count	F24_Halstead_prog_time
F7_Cyclomatic_complexity	F25_Halstead_volume
F8_Cyclomatic_density	F26_Maintenance_severity
F9_Decision_count	F27_Modified_condition_count
F10_Decision_density	F28_Multiple_condition_count
F11_Design_complexity	F29_Node_count
F12_Design_density	F30_Normalized_cylomatic_complexity
F13_Edge_count	F31_Num_operands
F14_Essential_complexity	F32_Num_operators
F15_Essential_density	F33_Num_unique_operands
F16_Loc_executable	F34_Num_unique_operators
F17_Parameter_count	F35_Number_of_lines
F18_Halstead_content	F36_Percent_comments
	F37_Loc_Total

TABLE II. PC1 Features Selected from NASA Data Set

Cyclomatic_density	Halstead_difficulty
Essential_complexity	Maintenance_severity
Parameter_count	Normalized_cylomatic_complexity
Halstead_content	Number_of_lines

B. Algorithm

The algorithm was input with different dataset of different raw features along with m sample, the different models were trained with, and the performance model was observed for True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) for the model performance evaluation. The steps were followed for each feature with various entropy features using recursive partitioning and with different decision criteria.

Input dataset with n raw features $F_r = \{f_1, f2, f3, \ldots, f_n\}$ and m samples.

1. Train various models $TM_s = \{tm_1, tm_2, tm_3, tm_s\}$ to observe the TP, TN, FP, FN for model performance evaluation.
2. **for** each feature **do**
   if ∃ a entropy ∈ raw feature F_r **then**
   Execute entropy features $F_e = \{f_1, f2, f3, \ldots, f_e\}$ using recursive partitioning and decision criteria.
   end if
   end for
3. **for** each feature **do**
   if ∃ a significance ∈ raw feature F_r **then**
   Execute RF variable significance features $RF_v = \{rf_1, rf2, rf3, \ldots, rf_v\}$ using un-pruned random tree with less error.
   end if
   end for
4. Obtain the significant features $SF_k = \{sf_1, sf_2, sf_3, \ldots, sf_k\}$ using the Step 2 and 3 as follows:
   Significant features $SF_k = F_e \cap SF_k$
5. Evaluate the model TM_s performance using significant features SF_k of *Step 4*.
6. **for** each model TM_s **do**
   if ∃ a model with high accuracy then
   Select the model for classification
   end if
   end for

IV. Result and Discussion

In this section, the summary of the experimental results obtained by various machine-learning models are presented. These experiments are conducted on the dataset NASA PC1 Dataset. The results obtained from various ML models are shown in Table III. In the next stage, the ML method on the dataset with all the features of confusion matrix calculated and shown in Fig. 2. The accuracy and precision are also calculated and are shown in Fig. 2 and Fig. 3. The Sensitivity and Specificity are also calculated the results are shown in Fig. 4.

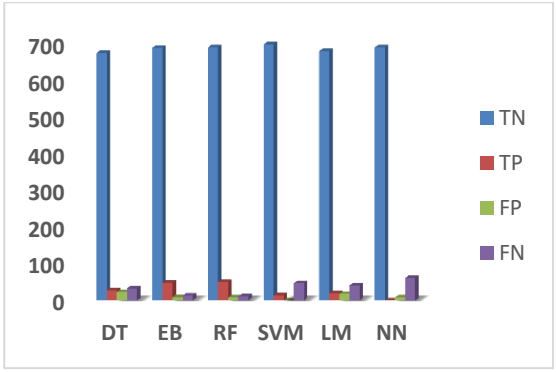

Fig. 2. Confusion Matrix of ML Models.

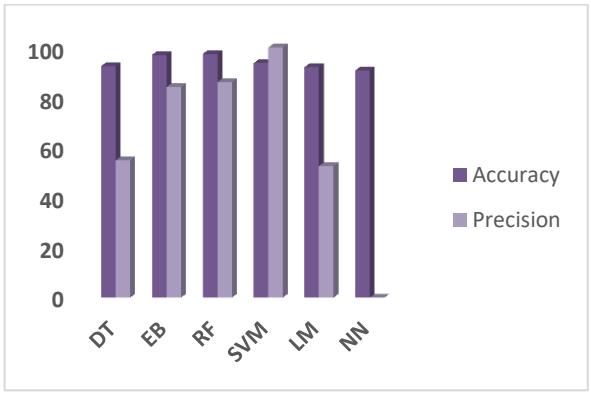

Fig. 3. Accuracy and Precision of ML Models.

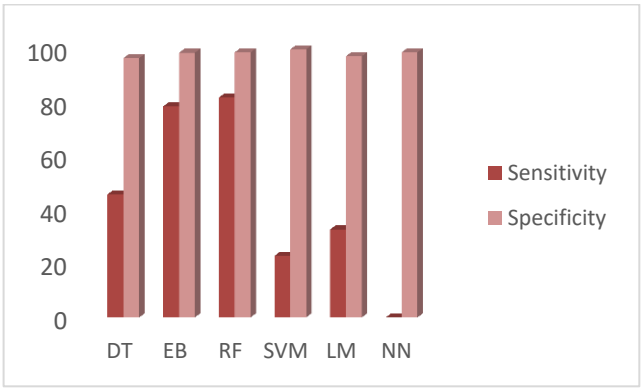

Fig. 4. Sensitivity and Specificity of ML Models.

The results obtained from ML models are shown in Fig. 5.

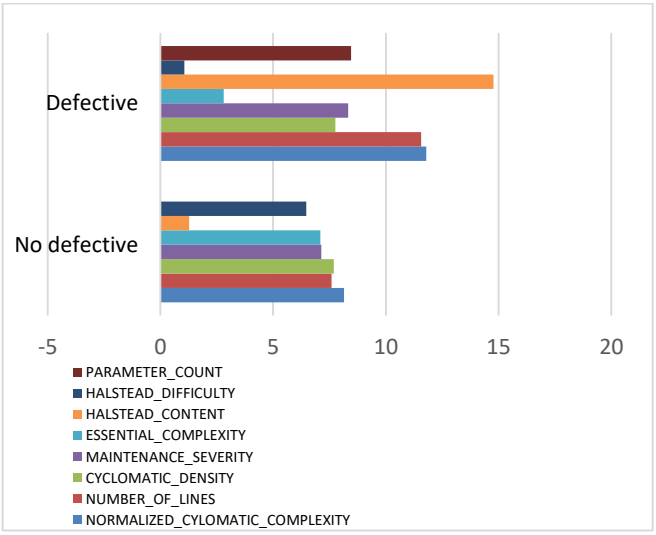

Fig. 5. Significant Features of Defective and Non-defective Class.

In the next stage, the ML method on the dataset with significant features are applied and confusion matrix calculated and shown in Fig. 6. The accuracy and precision with significant features are calculated and the presented in Fig. 7. The Sensitivity and Specificity are also calculated the results are shown in Fig. 8.

TABLE III. RESULTS OF ML MODEL WITHOUT SIGNIFICANT FEATURES

ML	n	TN	TP	FP	FN	Accuracy	Error Rate	Sensitivity	Specificity	Precision
DT	759	675	28	23	33	92.62	7.38	45.90	96.70	54.90
EB	759	689	48	9	13	97.10	2.90	78.69	98.71	84.21
RF	759	690	50	8	11	97.50	2.50	81.97	98.85	86.21
SVM	759	698	14	0	47	93.81	6.19	22.95	100.00	100.00
LM	759	680	20	18	41	92.23	7.77	32.79	97.42	52.63
NN	759	690	0	8	61	90.91	9.09	-	98.85	-

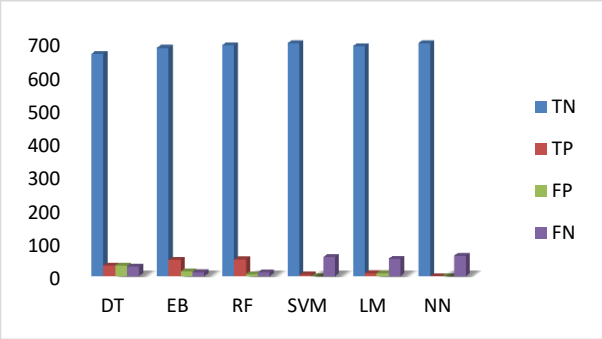

Fig. 6. Confusion Matrix of ML Models with Significant Features.

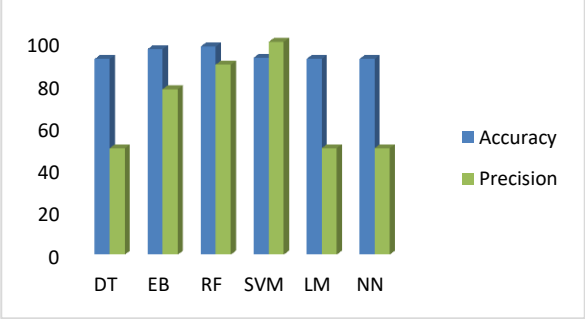

Fig. 7. Accuracy and Precision of ML Models with Significant Features.

Although this result is obtained by using with only 8 features out of 37 features, the proposed approach time consuming due to the large number of parameters in features selection. Since the proposed model utilizes only important features and avoid features which are not have high impact. Machine Learning models are used in to find out optimal feature selection and significant result improvement achieved by using Random Forest method in selection process.

The results revealed that our proposed method performed better than existing methods without significant features. The machine learning models with all features accuracy results obtained 97.50 % by using Random Forest method. The same dataset with significant features results in accuracy improvement 97.76 is achieved. Six distinct models are investigated for software defect data classification with selected features. As a result, the results of all six classification methods are compared using the outputs of the suggested feature ranking algorithms as input. The experimental results in Table IV shows that the suggested feature with an RF model have the greatest accuracy scores of all six features.

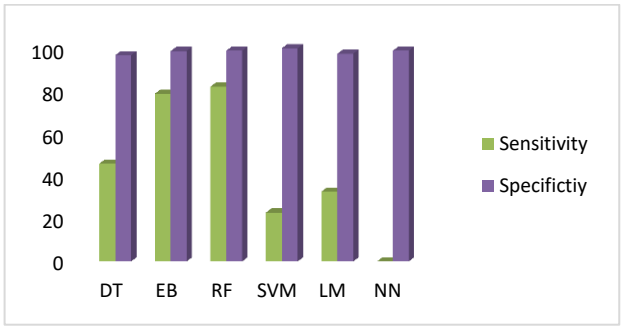

Fig. 8. Sensitivity and Specificity of ML Models with Significant Features.

TABLE IV. RESULTS OF ML MODEL WITH SIGNIFICANT FEATURES

ML	TN	TP	FP	FN	Accuracy	Precision	Sensitivity	Specificity
DT	666	32	32	29	91.96	50.00	52.46	95.42
EB	684	49	14	12	96.57	77.78	80.33	97.99
RF	692	50	6	11	97.76	89.29	81.97	99.14
SVM	698	4	0	57	92.49	100.00	6.56	100.00
LM	689	9	9	52	91.96	50.00	14.75	98.71
NN	698	0.1	0.1	61	91.95	50.00	0.16	99.99

V. CONCLUSION

Software defect prediction method plays important role and important to prevent and predict the bugs in the software in early stages are very difficult and challenging. However, this work using machine learning models perform evaluation of defect prediction with all features used in NASA dataset. The Machine learning models like DT, EB, RF, SVM, LM, and NN are used. The evaluation process carried out using with significant features and all features. The experimental results analyzed and summarized based on confusion matrix, accuracy, precision, sensitivity and specificity. The accuracy is plays major role and error rate also evaluated by using random forest the results are improved. The comparison results with all features and significant features used with ML models shows improvements. As future work, many more ML models and performs comparison among them to make more optimal results.

ACKNOWLEDGMENT

The author wishes to thank the College of Computer Sciences and Information Technology, King Faisal University, Saudi Arabia, for providing the infrastructure for this study.

REFERENCES

[1] S. K. Alferidah and S. Ahmed, "Automated Software Testing Tools," Proceedings-2020 IEEE, International Conference on Computing and Information Technology, ICCIT-1441, 2020, pp. 1-4.

[2] A. A. Alsayyah and S. Ahmed, "Energy Efficient Software Development Techniques for Cloud based Applications," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, no. 5, pp. 8043–8054, 2020.

[3] D. Chen, X. Chen, H. Li, J. Xie and Y. Mu, "DeepCPDP: Deep learning based cross-project defect prediction," IEEE Access, vol. 7, pp. 184832-184848, 2019.

[4] U. Ali, S. Aftab, A. Iqbal, Z. Nawaz, M. S. Bashir et al., "Software Defect Prediction Using Variant based Ensemble Learning and Feature Selection Techniques," International Journal of Modern Education and Computer Science, vol. 12, no. 5, pp. 29-40, 2020.

[5] H. Alsawalqah, N. Hijazi, M. Eshtay, H. Faris, A. A. Radaideh et al., "Software Defect Prediction Using Heterogeneous Ensemble Classification Based on Segmented Patterns," Applied Science, vol. 10, no. 1745, 2020.

[6] G. Mauša, T.G. Grbac and B.D. Bašić, "A systematic data collection procedure for software defect prediction," Computer Science and Information Systems, vol. 13, no. 1, pp. 173-197, 2016.

[7] M. A. Alshammari and M. Alshayeb, "The effect of the dataset size on the accuracy of software defect prediction models: An empirical study," Inteligencia Artificial, vol. 24, no. 68, pp. 72-88, 2021.

[8] S. R. Aziz, T. Khan and A. Nadeem, "Experimental validation of inheritance metrics' impact on software fault prediction," IEEE Access, vol. 7, no. 8742643, pp. 85262-85275, 2019.

[9] S.R. Aziz, T. A. Khan and A. Nadeem, "Exclusive use and Evaluation of Inheritance Metrics Viability in Software Fault Prediction—an Experimental Study," Peer. J Computer Science, 7, pp. 1-47, 2021.

[10] M. A. Kabir, J. W. Keung, K. E. Bennin and M. Zhang, "A Drift Propensity Detection Technique to Improve the Performance for Cross-Version Software Defect Prediction," Proceedings - 2020 IEEE 44th Annual Computers, Software, and Applications Conference, COMPSAC 2020, art. no. 9202527, pp. 882-891.

[11] A. Marjuni, T. B. Adji and R. Ferdiana, "Unsupervised software defect prediction using median absolute deviation threshold based spectral classifier on signed Laplacian matrix," Journal of Big Data, vol. 6, no. 1, 2019.

[12] H. Ji and S. Huang, "A New Framework Consisted of Data Preprocessing and Classifier Modelling for Software Defect Prediction," Mathematical Problems in Engineering, no. 9616938, 2018.

[13] Y. Jian, X. Yu, Z. Xu and Z. Ma, "A hybrid feature selection method for software fault prediction," IEICE Transactions on Information and Systems, vol. 10, pp. 1966-1975, 2019.

[14] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana et al., "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, pp. 300-308, 2019.

[15] A. Arshad, S. Riaz, L. Jiao and A. Murthy, "The empirical study of semi-supervised deep fuzzy c-mean clustering for software fault prediction," IEEE Access, vol. 6, no. 8439927, pp. 47047-47061, 2018.

[16] L. Gong, S. Jiang and L. Jiang, "Conditional Domain Adversarial Adaptation for Heterogeneous Defect Prediction," IEEE Access, vol. 8, no. 9169630, pp. 150738-150749, 2020.

[17] A.O. Balogun, S. Basri, L.F. Capretz, S. Mahamad, A. A. Imam et al., "Software defect prediction using wrapper feature selection based on dynamic re-reranking strategy," Symmetry, vol. 13, no. 11, 2021.

[18] Y. Qiu, Y. Liu, A. Liu, J. Zhu and J. Xu, "Automatic Feature Exploration and an Application in Defect Prediction," IEEE Access, vol. 7, no. 8794540, pp. 112097-112112, 2019.

[19] E. A. Felix and S.P. Lee, "Integrated Approach to Software Defect Prediction," IEEE Access, vol. 5, no. 8058420, pp. 21524-21547, 2017.

[20] M. Banga, A. Bansal and A. Singh, "Proposed hybrid approach to predict software fault detection," International Journal of Performability Engineering, vol. 15, no. 8, pp. 2049-2061, 2019.

[21] X. Chen, Z. Yuan, Z. Cui, D. Zhang and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," IET Software, vol. 15, no. 1, pp. 75-89, 2021.

[22] U. S. Bhutamapuram and R. Sadam, "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique," Journal of King Saud University - Computer and Information Sciences, (In Press), 2021.

[23] C. Cui, B. Liu, P. Xiao and S. Wang, "Can Defect Prediction Be Useful for Coarse-Level Tasks of Software Testing?," Applied Sciences, vol. 10, no. 15, 2020.

[24] H. Ghunaim and J. Dichter, "Applying the FAHP to Improve the Performance Evaluation Reliability of Software Defect Classifiers," IEEE Access, vol. 7, no. 8710236, pp. 62794-62804, 2019.

[25] L. Gong, S. Jiang and L. Jiang, "Tackling Class Imbalance Problem in Software Defect Prediction through Cluster-Based Over-Sampling with Filtering," IEEE Access, vol. 7, no. 8861051, pp. 145725-145737, 2019.

[26] L. Gong, S. Jiang, Q. Yu and L. Jiang, "Unsupervised deep domain adaptation for heterogeneous defect prediction," IEICE Transactions on Information and Systems, E102D, pp. 537-549, 2019.

[27] Y. Shao, J. Zhao, X. Wang, W. Wu and J. Fang, "Research on Cross-Company Defect Prediction Method to Improve Software Security," Security and Communication Networks, no. 5558561, 2021.

[28] K. Jiang, Y. Zhang, H. Wu, A. Wang and Y. Iwahori, "Heterogeneous defect prediction based on transfer learning to handle extreme imbalance," Applied Sciences, vol. 10, no. 1, 2020.

[29] A. Wang, Y. Zhang and Y. Yan, "Heterogeneous Defect Prediction Based on Federated Transfer Learning via Knowledge Distillation," IEEE Access, vol. 9, no. 9352701, pp. 29530-29540, 2021.

[30] C. Pan, M. Lu, B. Xu and H. Gao, "An improved CNN model for within-project software defect prediction," Applied Sciences, vol. 9, no. 10, 2019.

[31] J.-H. Ren and F. Liu, "Predicting software defects using self-organizing data mining," IEEE Access, vol. 7, no. 8758097, pp. 122796-122810, 2019.

[32] S. Riaz, A. Arshad and L. Jiao, "Rough Noise-Filtered Easy Ensemble for Software Fault Prediction," IEEE Access, vol. 6, no. 8435900, pp. 46886-46899, 2018.

[33] Y. Sun, X-Y. Jing, F. Wu, and Y.Sun, "Manifold embedded distribution adaptation for cross-project defect prediction," IET Software, vol. 14, no. 7, pp. 825-838, 2020.

[34] A. Wang, Y. Zhang, H. Wu, K. Jiang and M. Wang, "Few-Shot Learning Based Balanced Distribution Adaptation for Heterogeneous Defect Prediction," IEEE Access, vol. 8, no. 8999527, pp. 32989-33001, 2020.

[35] J. Zheng, X. Wang, D. Wei, B. Chen and Y. Shao, "A Novel Imbalanced Ensemble Learning in Software Defect Predication," IEEE Access, vol. 9, no. 9404009, pp. 86855-86868, 2021.

[36] Q. Zou, L. Lu, S. Qiu, X. Gu and Z. Cai, "Correlation feature and instance weights transfer learning for cross project software defect prediction," IET Software, vol.15, no. 1, pp. 55-74, 2021.