Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 2, April 2021, pp. 1063~1070
ISSN: 2302-9285, DOI: 10.11591/eei.v10i2.2752

Threshold benchmarking for feature ranking techniques

Ruchika Malhotra[1], Anjali Sharma[2]
[1]Department of Software Engineering, Delhi Technological University, Delhi, India
[2]Planning Monitoring and Evaluation, CSIR-National Physical Laboratory, New Delhi, India

Article Info

Article history:

Received Sep 9, 2020
Revised Nov 10, 2020
Accepted Dec 5, 2020

Keywords:

Feature ranking techniques
Machine learning
Threshold

ABSTRACT

In prediction modeling, the choice of features chosen from the original feature set is crucial for accuracy and model interpretability. Feature ranking techniques rank the features by its importance but there is no consensus on the number of features to be cut-off. Thus, it becomes important to identify a threshold value or range, so as to remove the redundant features. In this work, an empirical study is conducted for identification of the threshold benchmark for feature ranking algorithms. Experiments are conducted on Apache Click dataset with six popularly used ranker techniques and six machine learning techniques, to deduce a relationship between the total number of input features (N) to the threshold range. The area under the curve analysis shows that ≃ 33-50% of the features are necessary and sufficient to yield a reasonable performance measure, with a variance of 2%, in defect prediction models. Further, we also find that the $\log_2(N)$ as the ranker threshold value represents the lower limit of the range.

Corresponding Author:

Anjali Sharma
Planning Monitoring and Evaluation
CSIR-National Physical Laboratory
New Delhi, India 110012
Email: anjali@nplindia.org

1. INTRODUCTION

The success of the software defect prediction models lies in the choice of the optimal input features subset as the redundant and irrelevant input features can over or under estimate the performance due to their correlation and/or noisy characteristics [1-4]. The feature selections techniques optimize the size of the input features by removing the redundant metrics which also helps in lessen the computational bulkiness [5, 6].

In practice, there are different approaches for feature selection, which are based on filters [7, 8], wrappers [9], embedded and hybrid [2, 4, 10-12] techniques. Both embedded and wrapper based methods for feature selection depends upon the classification algorithm, unlike filter based approaches which are classifier independent. Hence, the filter techniques are being widely adopted for feature selection as they are usually faster, have low computational complexity and more scalable. Filter methods are mainly classified into two; (i) space search methods and (ii) ranking based methods. They employ statistical correlation between a set of features and the target feature, which thereby determine the importance of target variable [13, 14]. In search space methods, a subset is chosen and assigned with specific weights for a set of possible features. On the other hand, in ranker methods every feature is independently ranked by the use of its descriptive score functions. The features are then sorted out in decreasing order on the basis of its significance score. Ranker techniques are computationally efficient but do not provide the list of the redundant features. Since, the features are only ranked and not eliminated as like in correlation-based filtering methods, what remains as a challenge is the threshold value of features ranked, the latter which determines the cut-off dimensionality [14-19].

On empirical grounds, previous works [6, 7], have proposed $\log_2(N)$ to be the number of selected features for defect prediction models, where N is the total number of features in the metric space. This method of metric space optimization, otherwise known as threshold based feature selection, has been validated by other authors [3, 5]. However, the inferences are not very consistent with respect to the prevalence of one feature set choice over others [4, 20] and few questions remain unchecked. For instance, the direct dependency of the threshold on N poses serious limitation [10]. Since ranker techniques do not check for correlated features, there can be a chance of such features set ranked highest, which in combination can result in low performance values, either due to high unwarranted bias or variance [1, 21-23]. As highlighted in reference [2], the threshold on the number of the ranked features depends on the nature of the dataset. Furthermore, for a given dataset, it also does not pose an upper or lower limit to the value of N to be used, a priori subjected to ranker algorithms. In fact several studies have demonstrated that feature selection can greatly improve the performance of subsequently applied classification methods [7, 11, 24, 25]. An interesting question which then follows is that given there exists different ranker techniques and machine learning algorithms which are based on different theoretical mainframes, whether a particular threshold value, say $\log_2(N)$, can be universally be adopted to measure the prediction model performance, both precisely and accurately [2].

The goal of the present research work is to conduct an empirical study determine an universally acceptable threshold to the number of features derived from ranker algorithms by means of simple, fast and computationally tractable method. For the same, we employ combinations of various feature selection methods with different machine learning algorithms. This is anticipated to provide not only a set of excellent features that are distinct, but also to reduce computational burden. The selected classifiers are very different in their theoretical frameworks, which includes parametric, non-parametric and ensemble machine learning techniques. For the evaluation measure of the classifier performance we choose area under the curve (AUC). The empirical investigation suggests that the threshold benchmark for the feature ranking algorithms is more or less a range rather than a number. This is in contrast with the current practice that $\log_2(N)$, N=total number of features would determine the threshold. In fact, we find that $\log_2(N)$ is the lower limit of the range we predict from our experiments.

The remainder of this paper is organized as follows. Section 2 highlights the method followed in this study and also introduces the preliminaries, *i.e.*, the feature selection methods and machine learning algorithms studied in this work. In section 3 we elaborate and discuss the empirical results of the study, and also highlight the threats to validity of the results. In section 4, we conclude the result of our study.

2. RESEARCH METHOD

The experiment design followed in this empirical study is shown schematically in Figure 1. The change logs maintained by different software repositories corresponding to different software are analyzed to extract the OO metrics and fault data from the reports using defect collection and reporting system (DCRS) module [26]. The faults are associated with the corresponding classes of the software module. The source code for the Apache Click application, developed in Java language, was identified from the GitHub repository https://github.com/apache/click. Using the DCRS, the faults were collected from the defect logs from two consecutive releases of Apache Click, which were then mapped to the classes in the source code. To meet the objective of the study, the data was binary categorized into 'defective' which for finite defects were considered as 1 and with no-defects were taken to be 0.

2.1. Variables, data collection and application selection

The software metric suites of C&K [27], Lorenz & Kidd and QMOOD [28, 29], Tang, Kao & Chen [30] and Martin's [31] are used to serve as input in this empirical study. The WMC, CBO, LCOM, RFC, DIT, LCOM3, NOC, IC, CBM, AMC metrics are chosen as the complex classes are more prone to defects. The value of the NPM, DAM, CAM, MOA, MFA metrics proportionate to the quality associated with structural, behavioral and class design of the software, while the higher metric value of Ca and Ce depict complex relationship between components, leading to defects. The LOC size metric is also considered as the maintenance and development of large software modules need to be optimized. These eighteen metrics, extracted from DCRS, serve as input to the seven ranker algorithms. Both, the reduced subset and also the whole feature set are served as input to defect prediction models by applying nine ML techniques. Thereafter, the models are subjected to 10-fold cross validation and performance is evaluated using AUC. Our choice of Apache dataset was based on its application size and programming language used. Also, we find these open source Java-based applications fit for the study as these had more than 300 classes and multiple release versions. The Apache Click provides a web framework which is relatively easy to learn and helps understand the client style programming model.

2.2. Feature ranking techniques

Filter-based feature ranking methods evaluate each feature separately by assigning individual feature a score according to an indicator. In this work, we adopt to the seven feature ranking methods which are based on statistics and entropy paradigms. The non-parametric chi-square (CS) [32] statistical method compares a distribution of observed and theoretical frequencies, to identify which feature variables are better related to the outcome variable.

The information gain (IG) is an entropy-based strategy, which estimates the information contained in each attribute to select the features with more distinctive qualities. However, IG has the bias for dealing with multi-valued features. The gain ratio (GR) [33] method utilizes the gain in the information by taking into account the number of scores yielded by the attribute test conditions and compensates for the bias of IG. Even though GR technique surpasses the limitation of the information gain technique; it shows a bias toward attributes with more distinct values. The symmetrical uncertainty (SU) overcomes this effect by normalizing values to the range 0-1. Also, SU compensates the inherent built bias in IG by considering the sum of the entropies of the two variables.

ReliefF [34] is a feature selection algorithm inspired by the instance-based learning algorithm [35, 36] and a modification of the original works of Kira and Rendell [33]. It is found to be robust with ability to deal with incomplete and noisy data. One rule (OneR) is the classifier based feature selection technique that a single rule for each chosen features in the training data and, then selects the rule which has the least error. For all these feature selection methods, the larger indicator value signifies stronger relevance between the feature and class label.

2.3. Classifiers

The six machine learning (ML) techniques used in this study are (i) naive bayes (NB), (ii) logistic regression (LR), (iii) multi-layer perceptron (MLP), (iv) random forest (RF), (v) bagging (BAG), (vi) logit boost (LB). The LR and NB techniques are statistical in nature, and MLP represents a neural network solution. The RF, BAG and LB are ensemble-based methods. The Weka suite [37] with default parameters was used to build the predictive models.

2.4. Evaluation measure

The software defect prediction data are the class imbalanced datasets for which the receiver operating characteristic (ROC) is the most widely used performance indicator [38]. Hence, we have used AUC as the performance measure to determine the predictive capability of the model. We note that the AUC measure is insensitive to the noise and imbalance in the dataset. Furthermore, we apply the 10-fold cross-validation of the models [39] to yield the best possible result.

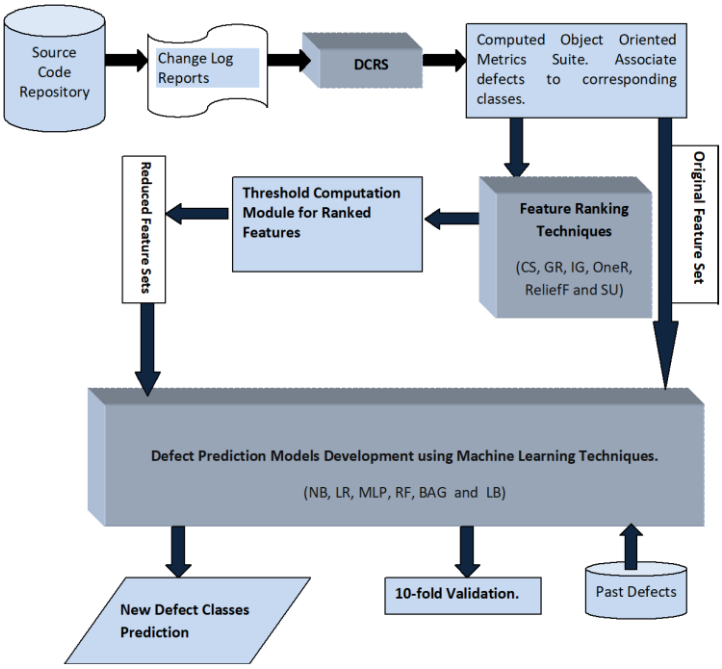

Figure 1. Research method adopted in the present work

3. RESULTS AND DISCUSSION

For the Apache Click data sets considered, we first compute the performance measure of the six classifiers with all metrics as input by means of AUC. The classifier AUC values were determined as NB(0.74), LR(0.77), MLP(0.76), BAG(0.78), LB(0.76), RF(0.78). In terms of these values, which vary well within ±2%, the results show little significance on the choice of the classifiers.

We now focus towards the problem, on how the performance measure varies with input features. As a systematic approach, we show in Table 1, the metrics obtained from six ranker algorithms according to their decreasing ranks. As per its relevance, the first one-third of the ranked features shows that WMC, LCOM, LCOM3, RFC, and CBO, are being consistently selected by almost all rankers. In these, WMC and RFC are common output, irrespective of the ranker. On the other hand, the rankers find MFA, MOA, IC, CBM, DAM, NOC, Ca, and Ce to be of lesser relevance, than the much debated LOC metrics. Thus, it follows that the C&K metrics are most relevant OO metrics, in addition to AMC the latter which is originally due to Tang, Kao & Chen [25]. In view of the above inferences, it is suggested that web developers may contemplate on metrics representing complexity, cohesion and coupling, to control the defects in the software.

Table 1. The ordered list of the ranked metrics using CS, GR, OneR, ReliefF, SU and IG ranking algorithms

No.	CS	GR	OneR	ReliefF	SU	IG
1.	LCOM3	AMC	LCOM3	CAM	AMC	LCOM3
2.	AMC	CBO	LCOM	LCOM3	CBO	AMC
3.	CBO	DIT	WMC	DAM	LCOM	CBO
4.	RFC	LCOM	RFC	NPM	RFC	RFC
5.	LCOM	RFC	LOC	WMC	WMC	LCOM
6.	WMC	WMC	CBO	RFC	LCOM3	WMC
7.	NPM	NPM	Ce	LOC	DIT	NPM
8.	DIT	LOC	AMC	CBO	NPM	DIT
9.	Ce	Ce	CBM	Ce	LOC	LOC
10.	LOC	LCOM3	IC	Ca	Ce	Ce
11.	DAM	DAM	MFA	NOC	DAM	DAM
12..	CAM	CAM	Ca	AMC	CAM	CAM
13.	NOC	NOC	NOC	LCOM	IC	NOC
14.	Ca	IC	DAM	DIT	NOC	Ca
15.	MOA	Ca	NPM	MOA	MOA	MOA
16.	IC	CBM	MOA	IC	MFA	IC
17.	CBM	MFA	DIT	CBM	CBM	CBM
18.	MFA	MOA	CAM	MFA	Ca	MFA

Having ranked the features, we shown in Figure 2, the variation in the AUC values as a function of ranked features evaluated using various ML techniques. Common to all ranker algorithms, we find that the classifiers find an abrupt initial increase in the AUC value, which then gradually saturates to a finite value. Although, this variation is found to be different for different classifiers, the overall trend appears to be logarithmic in nature, except for NB classifier. It is also obvious from Figure 2, a non-analytical dependence of AUC showing two plateau-like regions pertaining to 6-10 features and 10-18 features. However, the magnitudes of these two plateau regions were found to be less than 4%, irrespective of the ranker and the classifier.

A detailed analysis of the Figure 3 with respect to the AUC value calculated using all 18 metrics shows that the first ranked metric, irrespective of the ranker and classifier contributes more than 70% to the total performance. This is found to be highest for the BAG technique ($\simeq$ 88%) in combination with the CS/IG/OneR algorithms, followed by MLP technique ($\simeq$ 84%) using GR. This evidences to a strong interdependence between the ranker and classifier. With subsequent addition of metrics to determine the AUC, a slow and steady increment is calculated, which saturates around 5-6 metrics, for most of the ranker-classifier combinations.

Considering that the most relevant features set can be empirically obtained using the relation $\log_2(N)$; where N=18, we obtain the threshold as 4.16. Accordingly, considering the first five metrics as important, we find the AUC predicted by most of the six classifiers to be on an average of 0.72, which is close to the value determined by the classifiers with all features as input. However, we also note that NB classifier with feature set ranked using OneR stands odd, with AUC being 0.58. This is an important result, in view that NB classifiers tend to yield low performance values when the input features are correlated. In situations where correlation exists, the metrics are voted for twice in the model, thereby over-inflating their

importance. Hence, we argue that the OneR ranker algorithm offers a biased set of features, which is then found little suitable for defect prediction in web applications.

However, by virtue of the curve as shown in Figure 3, and neglecting the ±4% variation between the two plateau regions, one can deduce a critical point which can be considered as threshold. If one defines threshold as the value above which the change in the AUC value is minimal, then we find about 6-10 features render a reliable input for defect prediction modeling. This amounts to 33%-55% of the total number of features considered. The lower limit of our threshold range in fact nearly coincides (within ±2%) with the earlier prescription of using $\log_2(N)$ as the threshold. We note that in a earlier work {reference}, the authors have proposed the number of relevant features, following ranking algorithms to be ≃ 80% of the total set of features.

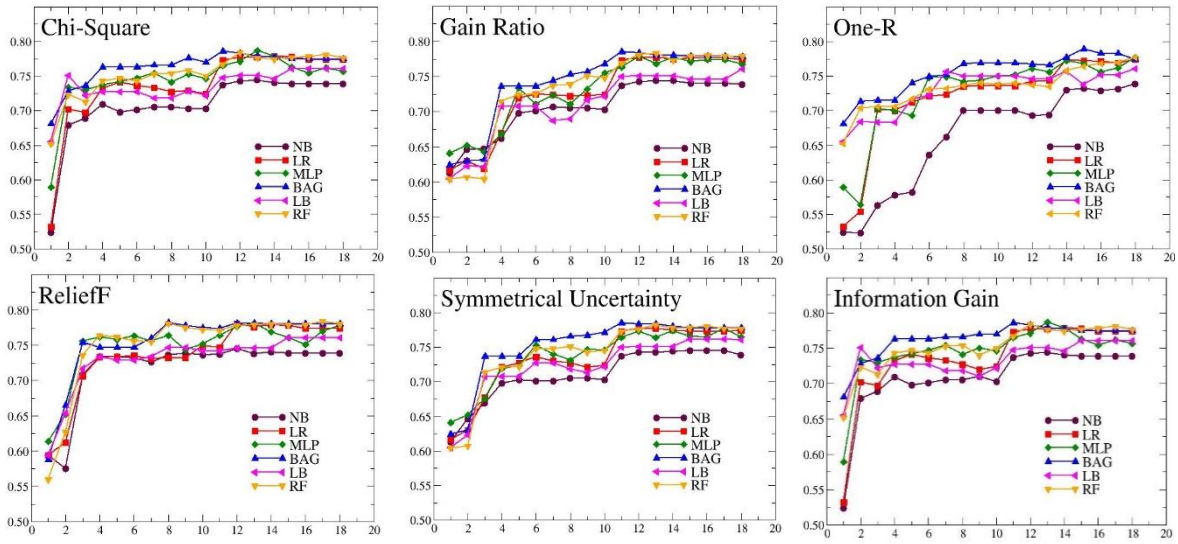

Figure 2. Plot of AUC values as a function of features ranked by CS, GR, One-R, ReliefF, SU, IG and evaluated using various machine learning techniques

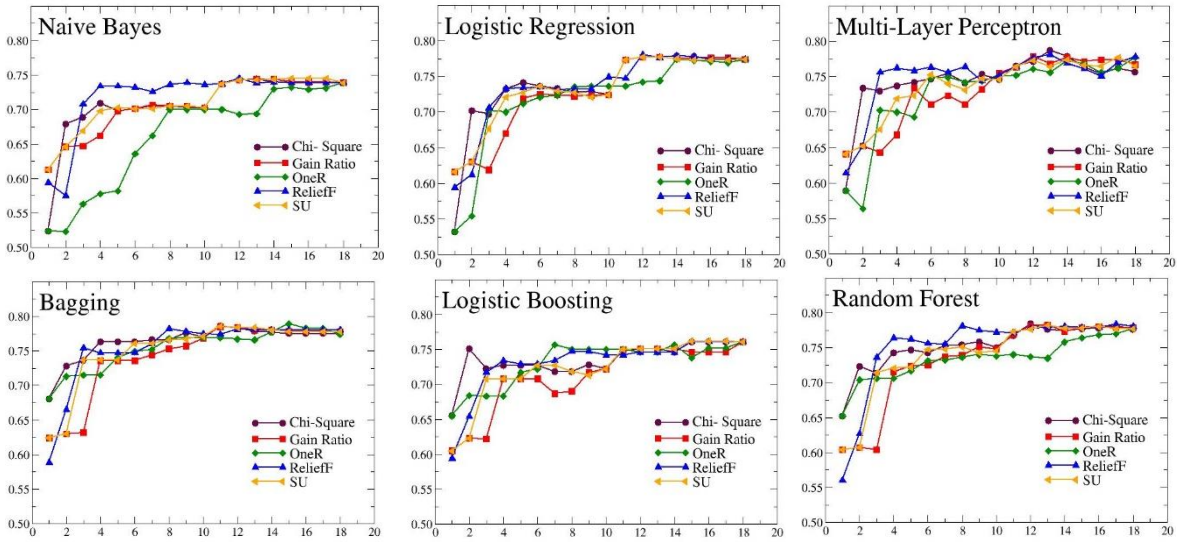

Figure 3. Plot of the AUC values as a function of ranked features using NB, LR, MLP, BAG, LB and RF machine learning technique for a given ranker algorithm

In analogy to the result, we derive from Figure 2 it becomes equally interesting to study the extent to which a given classifier would determine the performance with the ranked metrics. For the same, we show in

Figure 3, the AUC values as a function of ranked features using various ML techniques. Note that the data is essentially a re-plot of Figure 2. We find that for a threshold value of six, the choice of the ranker appears less significant to the LR, LB and BAG classifiers, with variance being less than 2%. On the other hand, few ML techniques appear to be dependent on the ranker to yield better AUC values. For instance, the best AUC values for NB, MLP and RF classifiers are obtained when the features are ranked by ReliefF. It is also interesting to note that the variation of the AUC as a function of ranked features determined by BAG and LR does not distinguish much on the choice of the ranker algorithm. However, for NB, MLP, RF and LB, the AUC variation is highly non-analytical.

On empirical grounds we find that about first half of the features ranked is a safe choice for precision, given a variation in the accuracy as ±2%. However, the earlier proposed threshold as $\log_2(N)$ [5], does indeed provide a quality value within a higher uncertainty of ±4%. In fact, we find that except NB, all classifiers are found to be reasonably good in providing a good consistent prediction model. Overall, we find that MLP and BAG are the most reliable classifiers. While the AUC values determined with the help of BAG are lesser prone to the choice of the rankers, MLP appears to specific to certain choice of rankers. For the best possible prediction, we find ReliefF, CS and IG as the best ranker algorithms for both BAG and MLP classifiers.

3.1. Threats to validity

Threats to build legitimacy center around the bias of the measures used to assess the prediction performance. In the work, we utilize AUC as the assessment measure. In any case, other thorough measures, such as the F-measure and g-measure can likewise be considered. Threats to internal validity allude to the bias on the choice of defect prediction techniques and also include feature selection strategies. In this work, we have included six classifiers because of their popularity in software defect prediction models for skewed data. Besides, we also included six feature selection methods to make our experimental examinations more conclusive. Threats to external validity principally concern the speculation of the exploratory outcomes. In spite of the fact that the datasets utilized in this work have been broadly contemplated in defect prediction, despite everything one guarantees little that the conclusions can be summed up to other projects. Nevertheless, our work gives a nitty-gritty trial portrayal, including parameter settings, which along these lines different specialists can easily replicate this empirical study on new datasets.

4. CONCLUSION

A large feature set as input to machine learning pose several problems which are inherited to its correlated behavior. Thus, while ranker algorithms are being used, a threshold to the input parameters becomes indispensable. In this view, towards determining a feature set threshold value, we performed an empirical investigation on Apache Click dataset using six feature ranking techniques (CS, GR, IG, ReliefF, OneR, and SU) and six ML techniques (NB, LR, MLP, RF, BAG, and LB). We find that ≃ 33%-50% of the features are necessary and sufficient, to yield a reasonable performance measure. Below the threshold, we find a strong variation in the AUC values reflecting a strong dependence of the ranker and ML techniques. The much accepted $\log_2(N)$ value is, however found to represent the lower limit to the threshold range. Interestingly, the features representing complexity, cohesion and coupling ranks as the most important as these are consistently ranked by all techniques.

In the future, we need to investigate the generality of our empirical results by considering more data-sets from various open-source community and commercial enterprises. Further, having noticed that there is link between the ranker algorithms and classifiers in determining the performance measure, it would be interesting to investigate whether there exists any particular threshold value for such given ranker-classifier combination over a range of varying data-sets.

REFERENCES

[1] S. Shivaji, E. James Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552-569, April 2013.
[2] B. Seijo-Pardo, V. Bolón-Canedo, and A. Alonso-Betanzos, "Testing different ensemble configurations for feature selection," *Neural Processing Letters*, vol. 46, no. 3, pp. 857-880, 2017.
[3] K. Muthukumaran, A. Rallapalli, and N. L. B. Murthy, "Impact of feature selection techniques on bug prediction models," *In Proceedings of the 8th India Software Engineering Conference*, pp. 120-129, 2015.
[4] V. Bolón-Canedo and A. Alonso-Betanzos "Ensembles for feature selection: A review and future trends," *Information Fusion*, vol. 52, pp.1-12, 2019.

[5] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing, software metrics for defect prediction: An investigation on feature selection techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579-606, 2011.
[6] H. Wang, T. M. Khoshgoftaar, J. V. Hulse, and K. Gao, "Metric selection for software defect prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 2, pp. 237-257, 2011.
[7] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," *2010 Ninth International Conference on Machine Learning and Applications*, pp. 135-140, 2010.
[8] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276-1304, 2012.
[9] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, Jan. 2007.
[10] M. R. Alhamidi and W. Jatmiko "Optimal feature aggregation and combination for two-dimensional ensemble feature selection," *Information*, vol. 11, no. 1, p. 38, 2020.
[11] B. Seijo-Pardo, V. Bolón-Canedo, and A. Alonso-Betanzos, "On developing an automatic threshold applied to feature selection ensembles," *Information Fusion*, vol. 45, pp. 227-245, 2019.
[12] S. Barak and M. Modarres, "Developing an approach to evaluate stocks by forecasting effective features with data mining methods," *Expert Systems with Applications*, vol. 42, no. 3, pp.1325-1339, 2015.
[13] M. Ashraf, G. Chetty, and D. Tran, "Feature selection techniques on thyroid, hepatitis, and breast cancer datasets," *International Journal on Data Mining and Intelligent Information Technology Applications*, vol. 3, no. 1, p. 1, 2013.
[14] M. Leach, "Parallelising feature selection algorithms," *University of Manchester*, ManchesterR, 2012.
[15] M. Shepperd and G. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Transactions on Software Engineering*, vol. 27, no. 11, pp. 1014-1022, Nov. 2001.
[16] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380-391, May 2005.
[17] H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 312-322, 2014.
[18] X. Y. Jing, S. Ying, Z. Zhang, S. Wu, and J. Liu, "Dictionary learning based software defect prediction," *Proceedings of the 36th International Conference on Software Engineering*, pp. 414-423, 2014.
[19] K. Herzig, "Using pre-release test failures to build early post-release defect prediction models," *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 300-311, 2014.
[20] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pp. 789-800, 2015.
[21] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semi-supervised learning with dimension reduction," *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 314-317, 2012.
[22] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 309-320, 2016.
[23] d. bowes, t. hall, and j. petrić, "software defect Prediction: Do different classifiers find the same defects?," *Software Quality Journal*, vol. 26, no. 2, pp.525-552, 2018.
[24] 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.
[25] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," *2007 IEEE International Conference on Information Reuse and Integration*, pp. 667-672, 2007.
[26] R. Malhotra, N. Pritam, K. Nagpal, and P. Upmanyu, "Defect collection and reporting system for git based open source software," *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)*, pp. 1-7, 2014.
[27] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994.
[28] M. Lorenz and J. Kidd, "Object-oriented software metrics: A practical guide," *Prentice Hall*, United States, vol. 131, 1994.
[29] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4-17, Jan. 2002.
[30] K. Kira and L. Rendell, "A practical approach to feature selection," *Proceedings of the Ninth Interna tional Workshop on Machine Learning*, pp. 249-256, 1992.
[31] M. Robnik-Sikonja and I. Kononenko, "Theoretical and empirical analysis of ReliefF and RReliefF," *Machine Learning*, vol. 53, no. 1-2, pp. 23-69, 2003.
[32] T. Fawcett, "ROC graphs: Notes and practical considerations for researchers," *Machine learning*, vol. 31, no. 1, pp. 1-38, 2004.

[33] M. Hall, E. Frank, G. Holmes, B. P. P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *CM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10-18, 2009.
[34] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pp. 388-391, 1995.
[35] D. W. Aha, D. Kibler, and M. K. Albert, "Instance based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37-66, Jan. 1991.
[36] J. P. Callan, T. E. Fawcett, and E. L. Rissland, "An adaptive approach to case-based search," *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, vol. 12, pp. 803-808, Aug. 1991.
[37] R. Martin, "OO design quality metrics-an analysis of dependencies," *Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOP-SLA*, vol. 12, no. 1, pp. 151-170, 1994.
[38] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen, "An empirical study on object-oriented metrics," *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*, pp. 242-249, 1999.
[39] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society*, vol. 36, no. 2, pp. 111-147, 1974.