JKAU: Comp. IT. Sci., **Vol. 8** No. 1, pp: 35 – 44 (1440 A.H. / 2019 A.D.)

Doi: 10.4197/Comp. 8-1.4

Feature Selection Approach for Improving the Accuracy of Software Bug Prediction

Emad Kaen and **Abdullah Algarni**

Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

ekaen0001@stu.kau.edu.sa

Abstract. We recently noticed the advancement and growth in the field of artificial intelligence and in its various branches such as Machine Learning (ML) and Deep Learning in various vital fields such as robotics, smart cars, smart cities, health care, software engineering and many other fields. Software bug prediction are one of the most important ML uses in software engineering. In addition, the feature selection is one of ML methods that aim to reduce a feature set that are used for building models. In this paper, we propose to use the Chi-Square feature selection method to calculate features importance, then to build a ML models, first by using top ten important features and second by using top five important features, based on three of well-known ML classifications algorithms, Support Vector Machine, Naïve Bayes and Linear Discriminant Analysis, with adding and exploring more about the effeteness of new metric of code smell intensity, the performance results of our approach against baseline achieved an improvements as average accuracy among nine datasets reaching up to 5.12%, 4.15% and 1% on the NB, SVM and LDA classifiers respectively.

Keywords: Machine Learning, Software Bug Prediction, Chi-Square.

1. Introduction

Machine learning has the ability of offering automatic learning techniques, it considered as a dominant branch of artificial intelligence and it can extract common patterns from realistic dataset based on the different learning behaviors making an accurate and sophisticate decisions. There are many types of ML classification algorithms that proved their efficiency in most of research experiments like Support Vector Machine, Naive Bayes, Decision Tree and many others. In the field of software engineering, there are many of areas and applications that ML can help to cover them such as software testing, software quality estimation, software reuse qualification, software measurement selection, project management and bug prediction models. The software bug prediction (SBP) is one of hottest research area in software engineering which focuses on quality assurance, especially on topics of effort-aware prediction, manipulating the data and machine learning-based prediction [1,2].The software bug prediction models can predict if the software have bugs or not by using different ML classification algorithms, that can help in evaluating the quality of software, reduce the effort and time taken in testing and maintenance, increasing the accuracy of software project from the traditional methods. Most research of the software bug prediction has focused on the method of supervised learning, which is mainly focus to use the categorize or labeled dataset in their learning algorithms of building bug prediction models. The concept of software bugs are known as programming error and most of the errors are coming from source code and software design. In another term, the software bug is also known as "software defects" that causes the software not performing its tasks as the programmer and customer needed. The SBP is such a method that mainly helps the domain of software testing and maintenance, it allows software engineers to assign the avaliable testing recources effectively for the defective instances, to improve the quality of software in the preliminary levels of development life cycle. For example, if the available testing resources that we have are represents only 25% of all our resources, thereafter the software engineers can center their attention by assigning the available testing resources to fix the more likely instances to defect prone. Thence, we can get a high software quality with reduced cost and can deploy the maintainable software at the given time, because these reasons the SBP today is a hot research topic in the field of software engineering [1,3]. We can build software bug prediction model by using historical bug data that gathered from old releases of the identical software projects. While the software metrics are used to measure the quality of software, there are different types of measurement that used for bug prediction and the most widely used are code and process metrics, the code metrics concerns with size and complexity of source code while the process concerns with code change, developer information and dependency analysis, there are various code metrics like line of code (LOC) metric, Cyclomatic Complexity, Halstead Metrics and others. The collected attributes from source code through software metrics it will be as inputs and it represents as features of software bug prediction. Interestingly, the modern measures of code smell severity proved itself in the software bug prediction models as an effective feature that increase the performance of SBP models, the code smells are known as an indicators of weak software design and implementation options, occurring as a results of software aging, or when the software is not properly designed from the beginning, exist some of complex or long classes, contains poor structured code likes the spaghetti code these only are a few examples of code smells that most likely to influence a software system and on the quality of produced source code [4]. For the evaluations measures of bug prediction performance there are many various evaluations and is ordinarily based on the data exploration from a confusion matrix, while this matrix declare how some model are classified their two categories of buggy and non-buggy compared in contrast to the actual classification in the original dataset. Based on confusion matrix we can get the others evaluation measures of performance such as precision, F-measure, G-measure, accuracy, area under the curve (AUC) and many others. One of the ML techniques used for enhancements is a Feature Selection, which is a step used in the level of data preprocessing and it aims to choose the best subset of features to use it in building the model of machine learning. Furthermore, the feature selection methods are used to improve the accuracy and reduce the complexity, it has proven itself in the practice to be as effective way in enhancing the efficiency of learning, [5,6]. Feature selection reduces feature space of datasets, removes irrelevant, redundant or noisy data. In addition, it can increase the performance models of SBP. For the methods types of feature selection, there are three main types that known as wrapper, filter, and hybrid. The features evaluation in wrapper methods uses learning algorithms. While in the filter methods are not depended on any types of learning algorithm and it implemented on the level of preprocessing, also they rely on all features of the training data. Finally the hybrid feature selectin gathering the characteristic of both method wrapper and filter [6].In our study we only center our attention on the filter feature selection method represented by Chi-square method (X^2) [7]. The evaluation of Chi-squared attribute it done by evaluates the worth of a feature by computing the value of the chi-squared statistic with respect to the class. The initial hypothesis H_0 is the assumption that the two features are unrelated, and it is tested by chi-squared formula [9].

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij}-E_{ij})2}{E_{ij}} \qquad [8]$$

Where Oij is the observed frequency and Eij is the expected (theoretical) frequency, asserted by the null hypothesis. The greater the value of X^2, the greater the evidence against the hypothesis H_0 is [8]. Different research studies use the Chi-square Feature selection [5, 10, 11].

The outline of the paper is as follows: reviews a few of related works in Section 2, the proposed approach in Section 3, results and discussion in Section 4 and finally the conclusion and future work.

2. Related Works

In the research of Z. Xu *et al*. [15], they made an empirical study to discover the impact of thirty-two types of feature selection methods on the software defect prediction performance. They have used three datasets from NASA and AEEEM in their studies for comparing the different methods of feature selection and to identify a set of worthy methods. Their results of feature selection methods displaying notable differences among the three datasets. In general, they found that the wrapper and filter based feature selection methods gives the finest performances and these kind of methods are tend to spend more time or select more number of features. Nearly all the clustering and some types of filter based methods can gives satisfying results with less time, fewer numbers of features and we can understand how methods works easily. Their research study have a significant value and through this kind of studies we can choose the suitable feature selection methods based on our statues and adding some of additional criteria.

Ye Xia *et al.* [16], they have used different methods of feature selection aiming to reduce the dimensionality and determining the most important software metrics. For the machine learning classifiers they used Naïve Bayes, Decision Tree (DT) and support vector machine, in the experiment they used the dataset of NASA and it shows a comparative result, giving that instead of using 22 or more metrics, only less than ten metrics can gain better prediction performance. Shivaji *et al.* [14] investigated the influence of multiple wrapper and filter feature selection techniques while the Chi-Square one of them on eleven software projects. For improving the bug prediction of the code change based. They conducted that if eliminating 90% from the original set of features, the feature selection techniques could improve the performance of models. In a research study of Kalai Magal *et al*.[13], authors have been studied and compared some of feature selection techniques likes information gain (IG), correlation-based feature selection (CFS) and gain ratio (GR) for Software Defect Prediction, they used a five types of machine learning classifiers to compare results and they conduct to the results of Random Forest classifier provides the high accuracy among them, they also gathered the correlation-based feature selection with the algorithm of random forest and they found a high percentage of results that get up to 98.3%. Fabio Palomba *et al.* [12] they evaluate the significance of using the severity of code smells measure, specifically in the term of code smell intensity by adding it as new feature with existing features of bug prediction models that based on both metrics of process and product, then comparing the outputs result of the new created models to explore the benefits of adding the intensity index opposed to four of latest state of art models in the baseline, which are without the intensity index is helpful to increase the performances. The outputs result of a new model displays that, when they added the code smell intensity as new feature (predictor), the accuracy increases and reaching to an improvement up to 21% by the F-Measure. They assessing the actual obtained information that given by the intensity feature with respect to the other different software metrics in the model and they note that the feature of intensity is a relevant feature for the process and product software metrics and it gives a positive and valuable contribution to the latest state of arts models of bug prediction.

3. Proposed Approach

The main steps of our proposed approach illustrated in the Fig.1, where each step will described in detail.

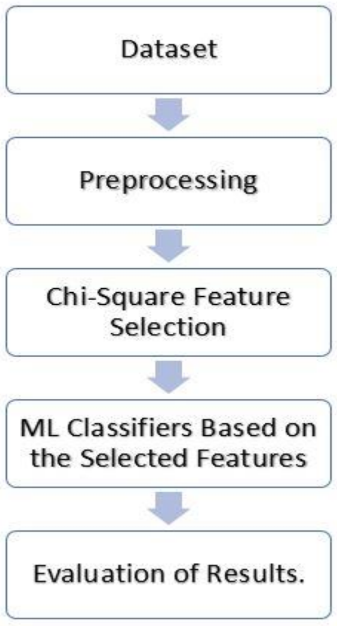
Dataset
Preprocessing
Chi-Square Feature Selection
ML Classifiers Based on the Selected Features
Evaluation of Results.

Fig. 1. Main levels of the proposed approach.

A. Dataset

Table 1 provides statistics about the number of instances and distribution of class labels of each dataset used in this study. The nine datasets that we used are shared and publically available [18], while they're originally collect the data from the dataset produced by Jureczko *et al*.[17] that contains 20 types of structural metrics as described in Table 2, while Palomba *et al*. [12] attached the new feature of code smell intensity by measuring the code smell severity and some of other features.

Table 1. Details of our experiments datasets.

System Name	Releases	Instances	Buggy Instances	Non-Buggy Instances
Apache Ant	Ant-1.3	125	20	105
	Ant-1.4	178	40	138
	Ant-1.5	293	32	261
	Ant-1.6	351	92	259
	Ant-1.7	745	166	579
Apache Camel	Camel-1.0	339	13	326
	Camel-1.2	608	216	392
	Camel-1.4	872	145	727
	Camel-1.6	965	188	777

Table 2. Summary of structural metrics for the datasets of each software project.

Abbrev	Feature	Type
WMC	Weighted Methods per Class	Numerical
DIT	Depth of Inheritance Tree	Numerical
NOC	Number of Children	Numerical
CBO	Coupling between Object classes	Numerical
RFC	Response for a Class	Numerical
LCOM	Lack of Cohesion in Methods	Numerical
CA	Afferent Couplings	Numerical
CE	Efferent Couplings	Numerical
NPM	Number of Public Methods	Numerical
LCOM3	Normalized version of LCOM	Numerical
LOC	Lines of Code	Numerical
DAM	Data Access Metric	Numerical
MOA	Measure Of Aggregation	Numerical

MFA	Measure of Functional Abstraction	Numerical
CAM	Cohesion Among Methods	Numerical
IC	Inheritance Coupling	Numerical
CBM	Coupling Between Methods	Numerical
AMC	Average Method Complexity	Numerical
MAX_CC	Maximum values of methods in the same class	Numerical
AVG_CC	Mean values of methods in the same class	Numerical

B. Pre-processing

1- Data Cleaning and Preparing

In this step of data preprocessing we had deleted the invaluable, redundant and undesirable features, We also delete the other new software metrics that added by the researchers [12] to the original data and we keep only the 20 structural metrics provided by Jureczko *et al.* + intensity metrics of Palomba *et al.* to become 21 features. In fact, the column that used for classify if the class is buggy or non-buggy, it is out of counting here because it is necessary to be present in all cases for the purpose of classification.

2- Data Partition

After we prepared the data, we divide it into two parts training and testing, we divide the data to be as 70% for training and 30% for testing in the all releases as the most works of machine learning researches.

3- Data Resampling

For the data resampling, we use the well-known k-Fold Cross-Validation while the k is equals to 10.

C. Chi-Square Feature Selection

The Chi-Square is a filter based feature selection method and it is simply used to calculates and determine how closely the observed data fit the expected data for all features set. We aim to use it and to reduce the current dimension of dataset that contains 21 features.

D. Machine Learning Classifiers

In this study, three of machine learning supervised classification methods are used, the Naïve Bayes (NB), Support Vector Machine (SVM), and linear discriminant analysis (LDA) methods to train the selected features data obtained by the Chi-Square feature selection and to generate models, then to test these generated models through testing data to evaluates the effeteness of models. Naïve Bayes is a machine-learning algorithm used for classification problem and it based on Bayesian theorem. Naïve Bayes offering high speed and accuracy in the large scale data processing, it assumes that the influence of a given class's attributes is independent and this assumption is called class conditional independent [16]. Support Vector Machine was introduced by Vapnik and Cortes (1995). It is a very popular classifier and it has been widely used in many of machine learning applications. It considered as one of the most effective classification methods, one of its advantages is minimizing the complexity of the large data set by break the dataset into many classes before decide the effected vectors that belong to each class. Thereby, the decision making of feature selection will be simplified. The Linear Discriminant Analysis is a well known ML technique that used for classification to predicting classes. One of its main advantages, compared to other ML classification algorithms, are that the prediction is easy and the model is interpretable. Also it is a very widespread method used for dimensionality reduction problems [19, 20].

E. Evaluation Measure of Results

In our study, we used the accuracy, which is a metric for evaluating classification of machine learning models. Informally, accuracy is the fraction of predictions our model got right [21]. Formally, the accuracy following this formula:

Total Number of Correct Predictions

$$\text{Accuracy} = \frac{\text{Total Number of Correct Predictions}}{\text{Total Number of Predictions}} \quad [21]$$

4. Results and Discussion

In this research study, we aiming to explore more about the effeteness of the code smell metric represented by the intensity index, reduce the total number of features on the training data, increase the predictions performance and the total number of truly predicted instances through applying the Chi-Square feature selection. We selected nine releases of two software systems namely as APACHE ANT and APACHE CAMEL among of all the available datasets, because it has a relatively comparable number of instances in each release and is equal to 4476 rows instances as total of all releases with 34 columns in each release which contains the different valuable software metrics (i.e. structural, code smell (intensity), scattering.). In the data cleaning step, we delete the undesirable features such as project.name, version, name, bug and is.buggy. We also delete the other new software metrics that added by the researchers [90] to the original data namely as fi.changes, ostrand, scattering, ana, acm, arl and acpd, we keep only the feature of intensity, reaching at the end to have only a 22 columns represents 21 features of structural metrics and intensity while the remaining column is for the purpose of instances classification, to represent the baseline of study. Then on each release of dataset we divide the dataset to be as 70% for training and the remaining 30% for testing, after that we use the set of training data to be as input for the next step of Chi-Square feature selection to make its calculation of features importance, firstly by select the K = 10, to select the top ten features with high importance value Subsequently, we use the set of 10 selected features from the training dataset to generate three different models by using SVM, NB and LDA classifiers that work perfectly for classification problem, in addition to using the 10-fold cross validation as a resampling of training data to select best model, then we test the selected model by using the remaining testing data to obtain the predictions of software buggy and non-buggy that represented as a confusion matrix to evaluate the accuracy of models. Secondly, by repeating same steps except selecting K of Chi-Square to equal 5 to select the top five features with high importance value and we follow the same steps again to acquire the results of new models accuracy. We note that the intensity features of code smell metrics it always selected by Chi-Square method and has the higher importance value among all of other features in all cases of the nine datasets. From the Table 3, we can note the results of accuracy on the nine datasets when we applied to baseline, top 10 selected features and top 5 selected features with classifiers of SVM, NB and LDA respectively, from the baseline, the LDA classifiers nearly gives best accuracy between all datasets also it has the best average accuracy among other classifiers that reach to 86.74% while SVM is 86.06% and NB is 81.56%, while the results of top ten selected features shows great improvements with the SVM classifier, which is improved by 2.32% and it reached to an accuracy of 88.38% overcoming the LDA that got 87.19% with improvements by 0.45% and NB that got 83.56% with great improvements by 2.00% than baseline. Eventually after we choose the top five features which is nearly means the quart number of baseline, we conduct to more improvements than baseline and top ten selected features. We note from the results of Table 3, the SVM reach to 90.21% of average accuracy with improvements by 4.15% than baseline and 1.83% than top ten features, while the second best classifiers is the LDA that reach to 87.74% and improved by 1.00% from baseline and 0.55% than top ten, finally the NB which is improves by 5.11% from the performance accuracy of baseline set of features and by 3.11% than top ten features. On the other side if we want to discover more about the number of cases that are truly predicated in each model. Table 4 summarize the results of all instances in the baseline, top ten and top five features on each dataset, we note the highest number of truly classified of baseline for all nine datasets achieved by the LDA classifiers, which is reach to 1142 instances, while the SVM 1132 and NB 1084 instances. When we look on the results of top ten selected features, we note the improvements on all classifiers that reach to be 1172 truly classified with SVM that means it increases by 40 more than same classifiers of baseline and for the NB classifiers it enhanced by 27 instances reaching up to 1111 of truly classified and lastly the LDA improved by 4 to reach at 1146 truly classified instances. The results of top five features achieve another better improvements than baseline, where the SVM reach to truly classify 1200 instances and better than baseline by 68 and better than top ten by 28, while the NB got 1137 and improved by 53 more than baseline and by 26 more than top ten and finally the LDA classifier is reach to 1151 truly classified which is means there are nine instances more than the features set of baseline and more than top ten features by five. Figure 2 summarized the number of truly classified instances for our approach when using top ten and top five selected features via Chi Square comparing to the baseline.

Table 3. Accuracy of baseline, top ten and top five selected features via chi square.(higher model accuracy of each dataset shown in bold).

Dataset Name	#n of instances	Baseline			K=10 Features			K=5 Features		
		SVM	NB	LDA	SVM	NB	LDA	SVM	NB	LDA
apache-ant-1.3	125	91.89%	81.08%	91.89%	91.89%	78.38%	91.89%	91.89%	89.19%	**94.59%**
apache-ant-1.4	178	77.36%	75.47%	79.25%	79.25%	83.02%	83.02%	**84.91%**	83.02%	81.13%
apache-ant-1.5	293	91.95%	86.21%	**95.40%**	91.95%	88.51%	**95.40%**	93.10%	91.95%	**95.40%**
apache-ant-1.6	351	89.42%	83.65%	86.54%	88.46%	82.69%	85.58%	89.42%	**90.38%**	89.42%
apache-ant-1.7	745	86.04%	82.88%	86.04%	**87.84%**	83.78%	85.59%	**87.84%**	84.23%	86.49%
apache-camel-1.0	339	97.00%	94.00%	97.00%	97.00%	96.00%	97.00%	97.00%	**98.00%**	97.00%
apache-camel-1.2	608	70.17%	64.64%	71.27%	85.08%	70.72%	71.82%	**87.29%**	72.38%	71.27%
apache-camel-1.4	872	86.97%	81.99%	88.89%	88.12%	83.14%	**89.27%**	87.36%	85.44%	88.89%
apache-camel-1.6	965	83.74%	84.08%	84.43%	85.81%	85.81%	85.12%	**93.08%**	85.47%	85.47%
Average Accuracy		86.06%	81.56%	86.74%	88.38%	83.56%	87.19%	**90.21%**	86.67%	87.74%
Improvements		-	-	-	2.32%	2.00%	0.45%	4.15%	5.11%	1.00%

Table 4. Comparison of truly classified instances for each classifier between baseline, top ten and top five (higher number of truly classified instances on each dataset shown in bold).

Dataset Name	#n of Rows	Baseline			K=10 Features			K=5 Features		
		SVM	NB	LDA	SVM	NB	LDA	SVM	NB	LDA
apache-ant-1.3	125	34	30	34	34	29	34	34	33	**35**
apache-ant-1.4	178	41	40	42	42	44	44	**45**	44	43
apache-ant-1.5	293	80	75	**83**	80	77	**83**	81	80	**83**
apache-ant-1.6	351	93	87	90	92	86	89	93	**94**	93
apache-ant-1.7	745	191	184	191	**195**	186	190	**195**	187	192
apache-camel-1.0	339	97	94	97	97	96	97	97	**98**	97
apache-camel-1.2	608	127	117	129	154	128	130	**158**	131	129
apache-camel-1.4	872	227	214	232	230	217	**233**	228	223	232
apache-camel-1.6	965	242	243	244	248	248	246	**269**	247	247
Truly Classified on all dataset		1132	1084	1142	1172	1111	1146	**1200**	1137	1151

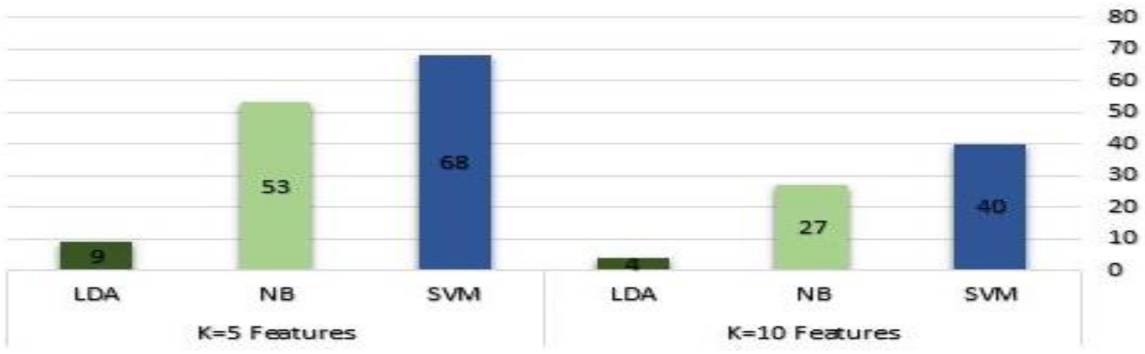

Fig. 2. The improvements of correct classifications of top ten and top five selected features via Chi Square comparing to the baseline (The prediction algorithms are ranked from right to left).

5. Conclusion and Future Work

In this paper, we note from the results of our proposed approach by using the Chi-Square feature selection first by select top ten important features and second by select top five features in building machine learning models of SVM, NB and LDA the valuable improvements in terms of accuracy after the feature selection, The proposed approach proved their effectiveness on all three classifiers, especially with the NB and SVM which are improved by 5.11% and 4.15%, also we achieved 90.21% as average accuracy of our experiments datasets by the SVM with reducing the total number of features from 21 to only 5 features, in addition to increasing the total number of truly predicted instances reaching to an improvements by 68 instances more than baseline with SVM classifier, on the other hand through our experiments we noticed the significant of adding the intensity index as new feature, which is selected as the first important feature by the Chi-Square method in all our cases of datasets.

Finally, for the future work we intend to expand our study by applying the proposed approach on more different machine learning classifiers such as neural network and deep neural network to investigate their applicability and performances, also to investigate more about another new software metrics than intensity of code smell to include them with existing baseline.

References

[1] **Li, Z., Jing, X. and Zhu, X.**, Progress on approaches to software defect prediction. *IET Software*, **12**(3): 161-175. doi:10.1049/iet-sen.2017.0148, 2018

[2] **Hall, T., Beecham, S., Bowes, D., et al.:** ‘A systematic literature review on fault prediction performance in software engineering’, *IEEE Trans. Softw. Eng.*, **38** (6): 1276–1304, 2012.

[3] **Kamei, Y.,** and **Shihab, E.**: ‘Defect prediction: accomplishments and future challenges’. *Proc. IEEE 23rd Int. Conf. Software Analysis, Evolution, and Reengineering*, pp: 33–45, 2016

[4] M. Fowler, Refactoring: improving the design of existing code. Addison-Wesley, 1999.

[5] **Ramaswami, M. and Bhaskaran, R.**, "*A study on feature selection techniques in educational data mining*," arXiv preprint arXiv:0912.3924, 2009.

[6] **Koller, D.** and **Sahami, M.**, “Toward optimal feature selection,” In: *Proceedings of the Thirteenth International Conference on Machine Learning*, pp: 284–292, 1996.

[7] **Liu, H. and Setiono, R.,** “Chi2: Feature selection and discretization of numeric attributes”, *Proc. IEEE 7th International Conference on Tools with Artificial Intelligence*, 338-391, 1995.

[8] **Novakovic, J.,** STRBAC P, Bulatovi c D. Toward optimal feature selection using ranking methods and classification algorithms. *Yugosl J Oper Res*,;**21**. ISSN: 0354e0243, EISSN: 2334-6043, 2011.

[9] **Blum, A.I.** and **Langley, P.**, “Selection of relevant features and examples in machine learning”, Artificial Intelligence, 97 (1997) 245-271.

[10] **Zaffar, M., Hashmani, M. A., Savita, K.S.** and **Rizvi, S. S. H.**, “A Study of Feature Selection Algorithms for Predicting Students Academic Performance” In*ternational Journal of Advanced Computer Science and Applications (IJACSA)*, **9**(5), 2018.

[11] Rachburee, N. and Punlumjeak, W., "A comparison of feature selection approach between greedy, IGratio, Chi-square, and mRMR in educational mining", *7th International Conference on Information Technology and Electrical Engineering*, pp: 420-424, DOI:10.1109/ICITEED.2015.7408983, 2015.

[12] **Palomba, F., Zanoni, M., Fontana, F. A., Lucia, A. D.** and **Oliveto, R.**, "Toward a smell-aware bug prediction model," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[13] **Kalai, M. R.** and **Jacob, S G.**, *Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques*, pp. 0975 – 8887, 2015.

[14] **Shivaji, S., Whitehead, E. J., Akella, R.** and **Kim, S.,** Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, **39**(4): 552-569, 2013.

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

[16] **Xia, Y., Yan, G.** and **Si, Q.**, "A Study on the Significance of Software Metrics In Defect Prediction," *IEEE Sixth International Symposium On Computational Intelligence and Design*, pp: 343-346, 2013.

[17] **Jureczko, M.** and **Madeyski, L.,** "Towards identifying software project clusters with regard to defect prediction," In: *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ser. PROMISE '10. New York, NY, USA: ACM*, pp: 9:1–9:10, 2010. [Online]. Available: http: //doi.acm.org/10.1145/1868328.1868342

[18] https://figshare.com/articles/Toward_a_Smell-aware_Bug_Prediction_Model/4542709

[19] https://www.displayr.com/linear-discriminant-analysis-in-r-an- introduction/

[20] **Tharwat, A., Gaber, T., Ibrahim, A.** and **Hassanien, A.,** Linear discriminant analysis: A detailed tutorial. *Ai Communications*. **30**:169-190, 10.3233/AIC-170729, 2017.

[21] https://developers.google.com/machine-learning/crash-course/classification/accuracy

أسلوب تحديد الخصائص لتحسين الدقة من التنبؤ بالأخطاء البرمجية

عماد نبيل كائن و **عبدالله مهدي القرني**

كلية الحاسبات وتقنية المعلومات، جامعة الملك عبدالعزيز، جدة، المملكة العربية السعودية

ekaen0001@stu.kau.edu.sa

المستخلص. لاحظنا مؤخرًا التطور والنمو في مجال الذكاء الاصطناعي وفي فروعه المختلفة، مثل التعلم الآلي والتعلم العميق في مختلف المجالات الحيوية، مثل الروبوتات، والسيارات الذكية، والمدن الذكية، والرعاية الصحية، وهندسة البرمجياتن وغيرها الكثير من المجالات. وتعد نماذج التنبؤ بالأخطاء البرمجية أحد أهم استخدامات التعلم الآلي في هندسة البرمجيات. وبالإضافة إلى ذلك، فإن تحديد الخصائص هو أحد طرق تعلم الآلة والتي تهدف إلى تقليص مجموعة الخصائص أو صفات البرمجيات، والتي من خلالها يمكننا التصنيف بوجود الأخطاء البرمجية من عدمه. ومن خلال بحثنا هذا نقترح استخدام طريقة تحديد الخصائص بمربع كاي وتقييمها على ثلاثة خوارزميات معروفة جيدًا في مجال تعلم الآلة، وهي: آلة المتجهات الداعمة، وبايز البسيط، وتحليل التمييز الخطي، باستخدام مجموعة من البيانات المفتوحة المصدر معرفة باسم "أباتشي"، مضافًا إليها المقياس القيم الجديد، والمسمى بكثافة رائحة الكود، وهي تعرف بأنها علامات داخل الكود تدل على أن هناك خلل في كود البرنامج أو في تصميمه، وذلك بهدف تحسين الدقة للتنبؤ من الأخطاء البرمجية وزيادة عدد التصنيفات الصحيحة. نتائج طريقتنا حققت تحسينات تصل إلى ٥.١٢ ٪، ٤.١٥ ٪، ١ ٪ على دقة الأداء من مصنفات بايز البسيط، وآلة المتجهات الداعمة، وتحليل التمييز الخطين على التوالي. وتم تخفيض مجموعة الخصائص، والتي مثلت تقريبًا الربع من عدد مجموعة الميزات الأصلية، وتمت زيادة عدد الحالات التي تم توقعها بشكل صحيح، وقد كانت أفضل النتائج التي تم الحصول عليها خلال مصنف آلة المتجهات الداعمة، ووصلت إلى زيادة تقدر بثمان وستون حالة توقع صحيح باستخدام الخصائص المحددة الفرعية عن الحالات الصحيحة باستخدام مجموعة الخصائص كاملة.

الكلمات المفتاحية: التعلم الآلي، التنبؤ بأخطاء البرمجيات، مربع كاي.