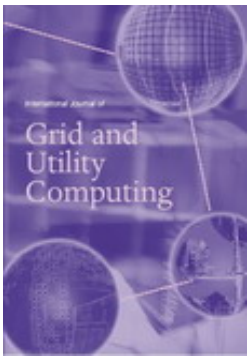


International Journal of Grid and Utility Computing

ISSN online: 1741-8488 - ISSN print: 1741-847X
https://www.inderscience.com/ijguc

Developing software predictive model for examining the software bugs using machine learning

Swati Singh, Monica Mehrotra, Taran Singh Bharati

DOI: 10.1504/IJGUC.2023.10060445

Article History:

Received:	26 January 2023
Last revised:	05 July 2023
Accepted:	09 July 2023
Published online:	19 February 2024

Developing software predictive model for examining the software bugs using machine learning

Swati Singh*, Monica Mehrotra and Taran Singh Bharati

Jamia Millia Islamia,
Delhi, India
Email: swatisingh.aug@gmail.com
Email: mmehrotra@jmi.ac.in
Email: tbharti@jmi.ac.in
*Corresponding author

Abstract: Software faults prediction is an emerging research area in the software engineering. It is an important issue for IT industry and professionals. We need prior information of an application for faults or faulty modules in traditional approach to determine software faults. If we use machine leaching techniques then we can easily automate the models enabling application software to knowingly predict and recover the application software faults. This capability type features helps in developing the application software to execute more productively and minimise faults, cost and time. In the scenario of this research, we are considering the software appropriate models that predicted development models using subsets of artificial intelligence-based approaches. Besides, we utilise noticeable benchmark techniques for evaluation of performance for software predictive models. However, researchers and software exponents can accomplish independent perception from this research and can pick out automated tasks for their deliberated application.

Keywords: machine learning; software predictive model; software faults.

Reference to this paper should be made as follows: Singh, S., Mehrotra, M. and Bharati, T.S. (2024) 'Developing software predictive model for examining the software bugs using machine learning', *Int. J. Grid and Utility Computing*, Vol. 15, No. 1, pp.44–52.

Biographical notes: Swati Singh is pursuing her PhD degree in Department of Computer Science, Jamia Millia Islamia (A Central University), New Delhi. She received her Post-Graduate degree of MTech in Information Technology from Banasthali Vidyapith, Jaipur (Rajasthan). She completed her BTech degree in Computer Science from Guru Gobind Singh Indraprastha University (GGSIPU), Delhi. She is UGC-NET as well as GATE qualified. Her research interests include software reliability, software growth modelling and machine learning.

Monica Mehrotra is presently working as Head of the Department of Computer Science, Jamia Millia Islamia (Central University). She completed her PhD degree in August 2007 from Jamia Millia Islamia. She has over 25 years of teaching experience. Her research interests include data mining, information retrieval and social network analysis. She has won 'Excellent Researcher Award (female)' in 2nd International Academic and Research Excellence Awards (IARE'2020) ceremony organised by GISR foundation on 3rd October 2020. She is a Member of IEEE.

Taran Singh Bharati is presently working as an Associate Professor in the Department of Computer Science at Jamia Millia Islamia (Central University). He completed his PhD degree in Network Security from Jamia Millia Islamia. He has over 20 plus years of teaching experience. His research interests include cyber security, theoretical computer science, compilers, IoT, data science and big data. He has various research papers in reputed journals under his name.

1 Introduction

In the field of software development, there are more challenges for developer and users both for analysing, maintaining and managing the software applications. Moreover, industry 4.0 revolution one of the best modern time techniques for observation of regular transformation of software development basis of large amount of automating software technologies (Bolat and Temur, 2019). According to observation quantity, quality and programming complexity is consistently increasing leading towards a down fall of software engineering with defects in software development from starting phase. It is critical to assemble exceptionally steady and trustworthy programming frameworks to offer better assistance (Bhandari and Gupta, 2018). It's more important to classify software faults in actual scenario, else

cost of searching faults and hiding efforts inside application that will also increase very fast. Software fault prediction motivates the development of automated software faults prediction models which can predict software faults (Malhotra, 2015). If software developer identifies the software defects before releasing the software, it could be very helpful for in allocating and fixing the defects. Machine learning approach is more effective for researchers and developer's community to solve software fault prediction (Wang et al., 2021). For findings of latest software defects, we can apply machine learning algorithms for making effective outcomes to the consumers (Singh and Mehrotra, 2021). However the disadvantage being a slow training process and dependence on the nature of the data set used for training (Singh et al., 2022). For the study in this research area many relevant machine learning techniques which are very recent classification were found (Yohannese and Li, 2017). Classifications of such machine learning approach are applied over several original application repositories which are concerned to software fault prediction applications. But we were not able to validate the correctness of feature in terms of quality of data. Therefore we used machine learning approach for enhancing the correctness of fault data sets and removing the unnecessary features (Hudaib et al., 2015). The goal of research is observing some machine learning classifiers techniques and automate the techniques to resolve software faults that were there in the software. Some real application related data set were taken for testing and finding the correctness of software faults prediction. Where we have applied various relevant machine learning algorithms and made software recovery predictive modules. The use of a technology that informs testers of the files that are most likely to malfunction might also have some unfavourable outcomes. Developers and testers could rely too much on the tool, failing to take into account actually flawed files that the tool missed. This false sense of security might cause flaws to be discovered at later stages of development, which, as previously said, will incur additional costs to fix. Even in the field of communications, a number of research have been done to look into the source code's defect-prone modules. Almost all computer programs have bugs that were introduced when the code was being written. Some of these flaws are eventually found, e.g., through quality control testing. The cost of addressing faults increases when they are found and addressed later in the development process. Therefore, it is crucial to identify the shortcomings as soon as feasible. Testing is one method of spotting errors or faults. Software testing is described as 'the process of executing a program with the intent of finding errors' by the researcher. Unit testing, function testing, system testing, regression testing and integration testing are just a few of the testing layers that are used. Unit tests are the first level of testing, and they examine a product's functionality. Machine Learning deals with the systems that can be learned from data. Machine learning works in two phases. The initial phase is the model building and the second is categorisation or classification using the new data set. The training set is used to build a model which then uses this trained model as an input for a classifier. Below diagram depicts the process of machine learning approach (Challagulla et al., 2008). Machine Learning algorithm can be distinguished based on the input data and the expected outcome of the algorithm.

Supervised learning: It is a type of learning which deals with a known labeled data set to make predictions (García et al., 2015). It is called supervised learning because there is a corresponding outcome with each input value in the algorithm. The main objective is to reduce the error between the predicted outcome of the model and the expected outcome. For example, Prediction and forecasting of weather.

Unsupervised learning: It is a type of learning which deals with unlabeled data set to make predictions. It is called unsupervised learning because for every input value there is no corresponding output. The main objective of this algorithm is to predict outcome for each input values and thus the output predicted is the desired outcome. For example, Image of cats and dogs of different types.

Decision tree: It is one of the most commonly used learning methods in the field of machine learning (Hassouneh et al., 2021). Decision tree is a classification method whose primary aim is to represent in an understandable form. Decision tree is based on attribute vectors which comprises of set of classification attributes along with a category attribute that assigns the entry of info to a class. Formation of tree is defined by iteratively splitting the info attribute into the existing classes and this iteration continues till the criteria is met. The tree like representation helps the users to understand and process the information as the visual representation is easier and faster to understand.

KNN: KNN is known as instance-based learning. As the name suggests, this algorithm waits for an instance to act upon. This is often referred as lazy learning method without the task, no action is performed on the given data set. It is opposite to decision tree in terms of learning technique as in decision tree, a structure is generated based on the data with any instruction to perform task. It is based on the phenomenon to learn from the similar situations/problem and thereby provides solution based on known solution of the similar problems. Owing to this characteristic, it is also known as nearest neighbor learning (Hudaib and Fakhouri, 2016).

Naïve Bayes: Naïve Bayes classifier is a probabilistic machine learning model. Naïve Bayes Classifier is based on Bayes Theorem.

SVM: It is supervised learning classifier. It is a discriminative classifier defined by separating hyperplane. The data points that are nearest to the hyperplane are called as support vectors. SVM is used for both classification and regression problem. Different SVM uses different kernel functions. The various types of kernel functions are linear, polynomial, radial basis function and sigmoid (Jureczko and Madeyski, 2010).

Logistic regression: It is named logistic regression because the algorithm used logistic function or sigmoid function as the core of method. This function is an S-shaped curved that take real values only between 0 and 1. The output must be categorical or discrete value, either yes or no, 0 or 1,

true or false. The concept of threshold value used in this classifier (Tanwar and Kakkar, 2019). This technique is used in classification for predicting the categorical dependent variable using given set of independent variables (Shepperd et al., 2013).

Random forest: It is used for both classification and regression. It is based on concept of ensemble learning. Itcombines multiple classifiers to solve problem and improve performance of model. It resolves the issue of overfitting in decision trees by averaging the decision trees results. It creates decision trees on the data set and then result are predicted from these multiple decision trees (Shihab et al., 2013). Finally, the voting classifier is applied for every result to find the maximum result.

The remainder of this paper is paraphrased as follows: Section 2 discusses the related work. Section 3 discusses the system description, the result and discussion is done in Section 4 and the conclusion and future directions are discussed in Section 5.

2 Related work

Much research is done in this field and there is a lot to be carried out further so as to develop a system that is capable enough to provide accurate results. As the processing techniques are developing day by day, it becomes necessary for us to choose the most appropriate technique of all, which is termed as most efficient of all the techniques present. For this, we need to study about the techniques to find out the best suitable technique for our work and then we need to study the research work which has already been accomplished in this field to decide what advances can be bought in this area.

From January 2014 to April 2017, Li et al. (2018) analysis and discussion of nearly 70 pertinent defect prediction papers. They distilled the chosen papers into four categories:data manipulation, machine learning algorithms, effort-aware prediction and empirical studies. Rathore and Kumar (2018) looked through multiple digital libraries to locate pertinent works that were released in the public domain between 1993 and 2017. A comprehensive review that identified and evaluated the research published between 2000 and 2018 was carried out by Li et al. (2020). Therefore, their meta-analysis demonstrated that supervised and unsupervised models for both CPDP and WPDP models are equivalent. The SDP survey was given by Akimova et al. (2021) based on deep learning papers that were divided into approaches for defect prediction, software metrics and data quality issues. For each class, the various approaches' taxonomic classifications and observations were given. From 1990 until June 2019, Pandey et al. (2021) evaluated several statistical techniques and machine learning research for software defect prediction.

Maintainability and localisation of software fault are represented as modules or software system can be modified to the exact faults, enhance performance, testing and software development approach or adapted to change platform (Elmidaoui et al., 2019). The software projects minimising the cost, time and maintenance effort through models predicting defects (Riaz et al., 2009). On the off chance that extremely less number of failures happen during programming execution time then we can ensure the good quality of software (Malhotra and Kamal, 2019). Duration of software development process has put large impact by classification of defects on software modules. But in real case it's hard, because if developer changes the internal code of application and related another module also it fails to updates new version applications. Hence, making software more feasible to faults (Oman and Hagemeister, 1994). According to this study software fault proneness is promoting automated service features every day (Anderson et al., 1985). The author presented the comparative analysis of some algorithm like, DT, ANN, SVM, NN, etc., to predict fault modules (Nisa and Ahsan, 1985). Author works with 3 NASA projects and they have taken data for analysing the performance of Decision Tree and getting highest accuracy than normal classifier. KPWE is best technique of solving the baseline faults. It is combination of two techniques representing defect prediction framework (Ahsan and Wotawa, 2011). To predict the software defect modules representation of comparative analysis of some widely used algorithms is presented. By experiment high accuracy is obtained in Random Forest between classifiers (Radjenović et al., 2013). Developed the model for feature selection and classifier approach to examine the benefits of feature selection for cross-product fault prediction. These techniques can increase the capabilities of software fault analysis (Malhotra, 2014). There are some applications which are automatically learning the customer expectation over multiple applications through semantic contexts (Xu et al., 2019). The case base reasoning techniques give capability to developed model for fault prediction inside suitable system applications (Moeyersoms et al., 2015). Study of recent research papers related to software bugs or defects, we find out some better machine learning, artificial intelligence-based approach than statistical or traditional-based approach for software defect prediction (Yu et al., 2019). Thus, we can take open-source data set for maintaining the consistency, authenticity and reliability, etc. (Hotzkow, 2017). Finding and eliminating insignificant elements from data can bring a significant performance change in learning algorithms and bring down the computational time (Turabieh et al., 2019).

3 System description

3.1 Data collection

For this paper, we have collected a publicly available PROMISE Software Engineering Database which has been widely used in previous researches as well. The data had 22 attributes. Inside Table 1, we shown the various attributes for software bugs data. Using machine learning algorithms, we have predicted the software defects through software fault prediction model.

Table 1 Attribute list

S.N.	*Attributes name*	*Types*
1	Line of code	McCabe
2	Cyclomatic complexity of code	McCabe
3	Crucial Complexity of code	McCabe
4	Complexity of design code	McCabe
5	Operands and Operators in code	Halstead
6	Volume of code	Halstead
7	Program length	Halstead
8	Difficulty of code	Halstead
9	Code Intelligence	Halstead
10	Code effort	Halstead
11	Code time estimator	Halstead
12	Code line count	Halstead
13	Code comments count	Halstead
14	Code blank line count	Halstead
15	Input/output code and comments	Miscellaneous
16	Unique Operators	Miscellaneous
17	Unique operands	Miscellaneous
18	No. of operators in code	Miscellaneous
19	No. of operands in code	Miscellaneous
20	No. of branches in code	Miscellaneous
21	b: numeric	Halstead
22	Default code	Boolean value

3.2 Classification techniques used

The subset of artificial intelligence, i.e., machine learning has some effective algorithms which play a vital part in the field of programming software. From last few years, Machine learning techniques are most operational approach for solving the real-world problems with high performance. We have found in the study of some research papers, the approaches of machine learning which are used most of the time are assembling the software fault prediction models like K-Nearest Neighbours, Support Vector Machine, Naïve Bayes, Random Forest, etc. (Tumar et al., 2020). Different AI classifiers strategies were used in this paper such as DT, KNN, NB, LG, GNB, RF and SVM. These techniques helped in automating the task of resolving faults that were there in the software. The data set was trained and tested for finding the correctness of software faults prediction models.

S.N.	*Formula*
Accuracy	$\left(T_p + T_n\right)/\left(T_p + F_p + T_n + F_p\right)$
Precision	$T_p/\left(T_p + F_p\right)$
Recall	$T_p/\left(T_p + F_n\right)$
F1	2* (Recall*Precision)/(Recall + Precision)
TNR	$T_n/\left(T_n + F_p\right)$

3.3 Performance measurement

Firstly, we have constructed the predictive models after that we test the fault models on different machine learning algorithms. These models are applicable for predicting the fault on any executable software. In this experiment, we observed that machine learning prediction models, using some classification algorithms that are based on distinct statistical approach (Montani and Anglano, 2008). Those are defined below in Table 2 with mathematical statistics.

Table 2 Software defect analysis

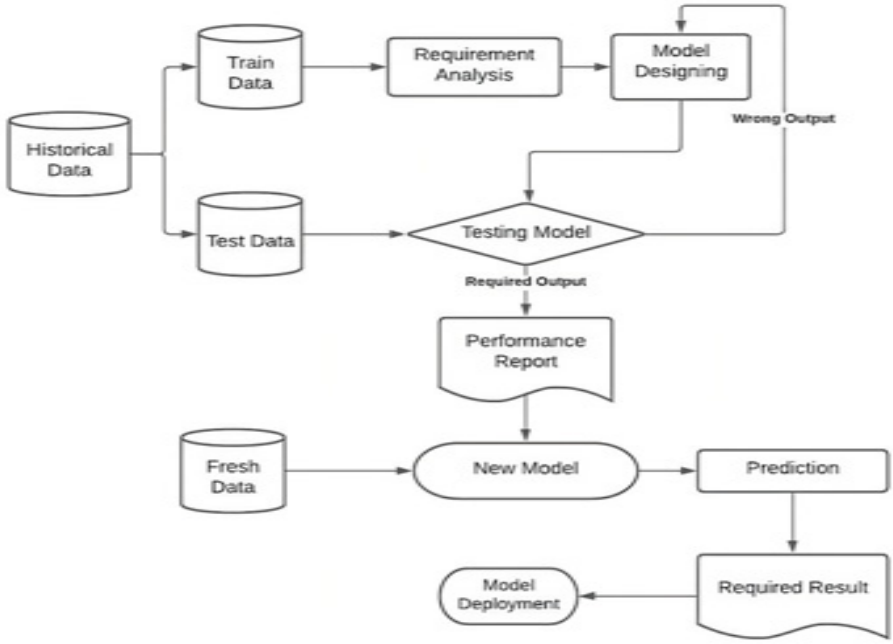
S. No.	*ML Techniques*	*Accuracy under developed model*
1	DT	99.9%
2	KNN	98.4%
3	LG	98.26%
4	NB	97.91%
5	RF	99.82%
6	SVM	99.73%

3.4 Experimental setup procedure

This section tells us about procedure of experiment and software defect predictive development models.

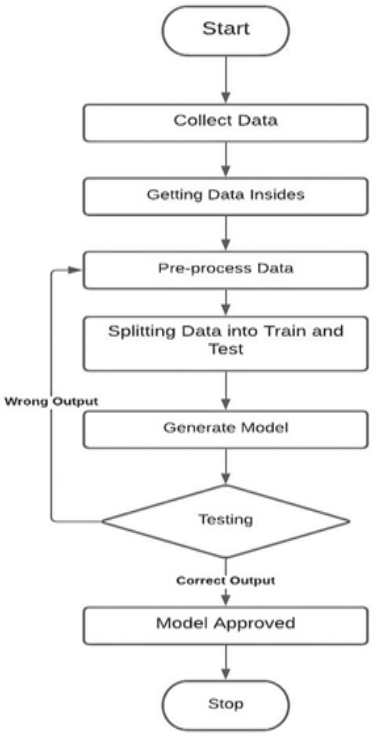
The development of proposed models assembling the duration of primary stage of software life cycle and these developed models indicates to combine the development process and using implementation in form of inputs which are required for analysis predictive model for developments. The main phases of software defect predictive development model is testing, and physical design phase which are analysing defects assigned by study based automated fault recovery of system applications. And these are leading to predicts bugs and collect all require information about faulty modules of the system applications. Representing the proposed model's components in Figure 1.

Figure 1 Proposed models for predictive development



The above figure displays the brief approach behind developing this model. In order to develop this model we firstly take widely used open source data sets from the repository then divide the whole data sets into training and testing data set. Then, describe the training data sets and gets the basic insights of the data sets then pre-process it so that effective model is obtained. Upon finding the requirement for this model, we design the appropriate model. After this, testing of the model on training data sets is performed if model is giving appropriate result then we prepare performance report of the model otherwise we retrain the model by making few changes on the model. We then further test model on fresh data in order to check if it is still giving desired result and then the model is ready. According to experiments, we have taken defects data sets for constructing software defect model. After which the data processing techniques were implemented over defect data sets by using statistical analysis, we found out high correlation in data for example min-max, feature extraction, normalisation and missing value, etc. Figure 2 is showing the complete details of the proposed model on which software defect predictive model is based.

Figure 2 Defect model flowchart



4 Result and discussion

The results of various classification techniques obtained for the new developed model are represented in Table 3. They are based on analysis of software defects with respect to systematic mapping.

Table 3 Performance of proposed model under decision tree algorithm

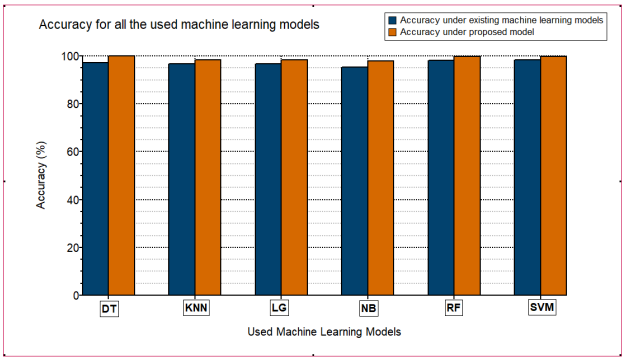
	Precision	*Recall*	*F1-score*	*Supports*
Redesign	1.00	1.00	1.00	375
Successful	1.00	1.00	1.00	1875
Accuracy			1.00	2250
Macro Average	1.00	1.00	1.00	2250
Weighted Average	1.00	1.00	1.00	2250

In this section, machine learning classifiers are used to study the correlations among the constructs in the proposed model. Thus, various classification algorithms such as Logistic Regression (LG), support vector machine (SVM), Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF) and K-Nearest Neighbours (KNN) classifiers are employed. The analysis using these algorithms is carried out using scikit module of python in Jupyter-Notebook by applying the *k*-fold cross-validation technique on data set. Here, we are considering two cases, viz., *Case 1*: Affects performance metrics when using existing machine learning models and Case 2: Affects performance metrics when using proposed predictive models. Case 1: Affects performance metrics when using existing machine learning models: Accuracy of existing models viz., DL, KNN, LG, NB, RF and SVM are 97.20%, 96.80%, 96.63%, 95.23%, 98.01% and 98.29%, respectively.

Case 2: Affects performance metrics when using proposed predictive models: Our results show that accuracy of existing models viz., DL, KNN, LG, NB, RF and SVM are having 99.9%, 98.4%, 98.26%, 97.91%, 99.82% and 99.73%, respectively.

Figure 3 shows that existing machine learning models give better accuracy when using proposed predictive model.

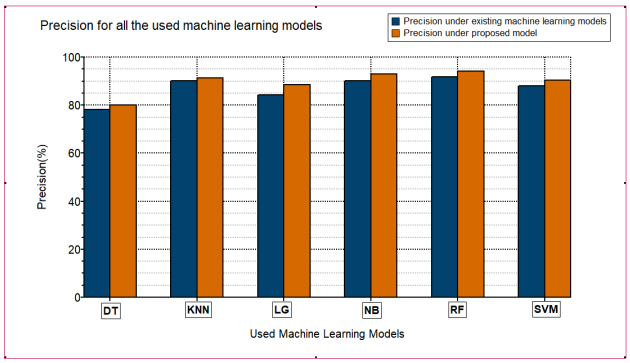
Figure 3 Accuracy of existing models under proposed predictive model (see online version for colours)



Case 1: Affects performance metrics when using existing machine learning models: precision of existing models viz., DL, KNN, LG, NB, RF and SVM are 78.12%, 90.04%, 84.33%, 90.04%, 91.7% and 88.10%, respectively.

Case 2: Affects performance metrics when using proposed predictive models: Our results show that accuracy of existing models viz., DL, KNN, LG, NB, RF and SVM are having 80.12%, 91.30%, 88.50%, 93.0%, 94.19% and 90.27%, respectively. Figure 4 shows that existing machine learning models give better precision when using proposed predictive model.

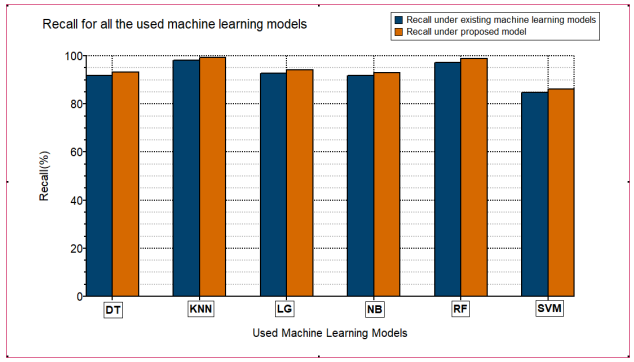
Figure 4 Precision of existing models under proposed predictive model (see online version for colours)



Case 1: Affects performance metrics when using existing machine learning models: Recall of existing models viz., DL, KNN, LG, NB, RF and SVM are 91.9%, 98.1%, 92.70%, 91.9%, 97.20% and 84.8%, respectively.

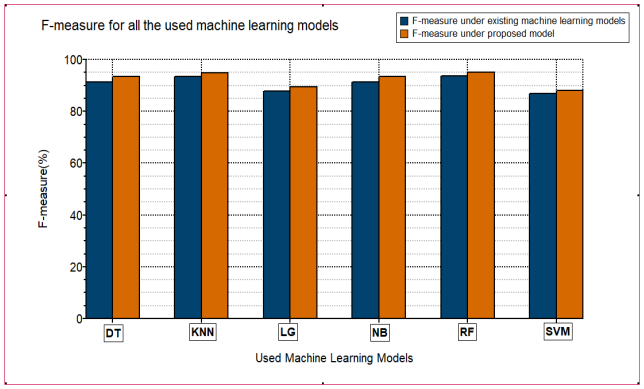
Case 2: Affects performance metrics when using proposed predictive models: Our results show that recall of existing models viz., DL, KNN, LG, NB, RF and SVM are having 93.15%, 99.24%, 94.20%, 92.89%, 98.90% and 86.06%, respectively. Figure 5 shows that existing machine learning models give better recall when using proposed predictive model.

Figure 5 Recall of existing models under proposed predictive model (see online version for colours)



Case 1: Affects performance metrics when using existing machine learning models: F-measure of existing models viz., DL, KNN, LG, NB, RF and SVM are 91.2%, 93.5%, 87.70%, 91.3%, 93.60% and 86.9%, respectively. *Case 2*: Affects performance metrics when using proposed predictive models: Our results show that F-measure of existing models viz., DL, KNN, LG, NB, RF and SVM are having 93.40%, 94.92%, 89.45%, 93.43%, 95.05% and 88.11%, respectively. Figure 6 shows that existing machine learning models give better F-measure when using proposed predictive model.

Figure 6 F-measure of existing models under proposed predictive model (see online version for colours)



We worked on automated software fault recovery by proposing predictive model and observed the defective software and not faulty software both. Defective models are more critical models compared with non-faulty models. But in our model experiments, we consider the cross-validation approach to evaluate the capability of some classification approach. In this process, we regulate the variables for the software defect models. And performed several distinct data processing techniques which were able to improve the model accuracy and representing classification models consistency. Some of the experimented results are shown in Tables 4 below performance wise.

Table 4 Performance of proposed model under *k*-nearest neighbour algorithm

	Precision	*Recall*	*F1-score*	*Supports*
Redesign	0.966	0.955	0.954	366
Successful	0.999	0.999	0.992	1884
Accuracy			0.984	2250
Macro Average	0.971	0.971	0.971	2250
Weighted Average	0.984	0.984	0.984	2250

We have performed the decision tree algorithm on developed model and we are getting accuracy of 0.999555 while 0.984 in case of *k*-nearest neighbour algorithm.

4.1 Complexity evaluation

During the preprocessing phase, we get the insights of data like frequency count of different complexities presents in the code in our case we obtained two values successful and redesign while training the model (Catal and Diri, 2009).

From these values we get to know the number of software that were successful and number of software that needs to be redesigned. Complexity was calculated for our new developed model which gave us the frequency of successful software designs and re-designs that were needed. Figure 7 shows a complexity evaluation between redesigns and successful designs.

Figure 7 Complexity evaluation graph b/w frequency vs. successful-redesign (see online version for colours)

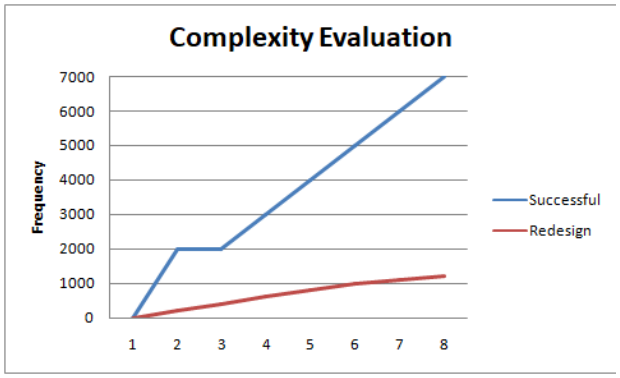


Figure 8 display the relation between the number of bugs in the code and the volume of code. From this figure, we observed that chances of bug are directly proportional to the volume of code up to some extends then if the volume of code is much higher than the chances of bug in code is less.

Figure 8 Visualisation of volume bugs

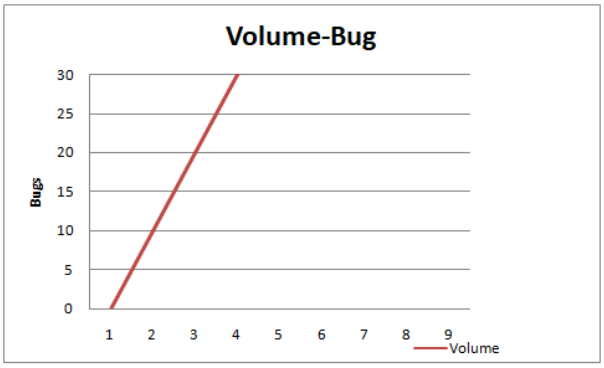
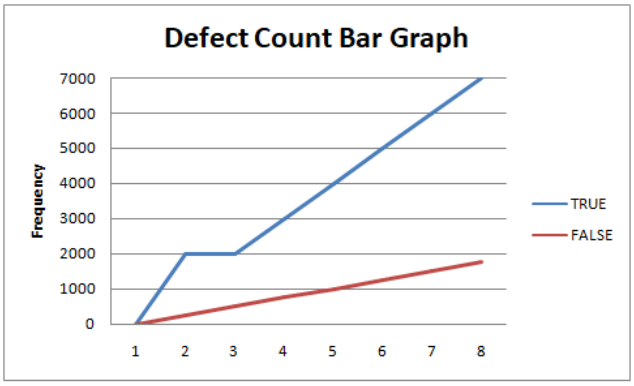


Figure 9 also comes under the pre-processing phase of model training. From this graph, we get the insights of data like frequency count of different defects presents in the data. In above graph we observed two values true and false where true count represents the number of default software presents in the data sets while the false count in the graph shown the number of correct software presents in the data sets.

Figure 9 Visualisation of defects counts (see online version for colours)



5 Conclusion and future work

Studies of this research, we present an automated software technique for software defect predictive models on the software life cycle. The principal objective of this research is evaluating capability of some machine learning classification approach that predict defect modules using open source data-sets. The outcomes of experiment using different attribute represented ability and proficiency of newly developed models to recognise the defects and upgrade the software standard. This model helps in detecting faults through collection of actual software setup data from aimed applications. The proposed techniques uses software fault recovery in software and upgrading through machine learning approach making the software prediction model better in retrieving faults offering more functionality. The future plan is to authenticate the efficiency of software defect prediction by experimenting with new data sets and implementing more classifications algorithm. Testing the tool set against metrics from publicly accessible data corpora like the NASA and Eclipse data sets would also be beneficial. Given that the results of previous studies are largely dependent on the underlying data and that generalising between two different projects is a challenging task, this may be the only objective approach to compare the prediction performance of the tool to those of other studies.

References

Ahsan, S.N. and Wotawa, F. (2011) 'Fault prediction capability of program file's logical-coupling metrics', *Proceedings of the Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp.257–262.

Akimova, E.N., Bersenev, A.Y., Deikov, A.A., Kobylkin, K.S., Konygin, A.V., Mezentsev, I.P. and Misilov, V.E. (2021) 'A survey on software defect prediction using deep learning', *Mathematics*, Vol. 9 Doi: 10.3390/math9111180.

Anderson, T., Barrett, P.A., Halliwell, D.N. and Moulding, M.R. (1985) 'Software fault tolerance', *IEEE Transactions on Software Engineering*, Vol. 12, pp.1502–1510.

Bhandari, G.P. and Gupta, R. (2018) 'Machine learning based software fault prediction utilizing source code metrics', *Proceedings of the IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, pp.40–45.

Bolat, H.B. and Temur, G.T. (2019) *Agile Approaches for Successfully Managing and Executing Projects in the Fourth Industrial Revolution*, IGI Globa.

Catal, C. and Diri, B. (2009) 'A systematic review of software fault prediction studies', *Expert Systems with Applications*, Vol. 36, pp.7346–7354.

Challagulla, V.U.B., Bastani, F.B. and Yen, I-L. and Paul, R.A. (2021) 'Empirical assessment of machine learning based software defect prediction techniques', *International Journal on Artificial Intelligence Tools*, Vol. 17, pp.389–400.

Elmidaoui, S., Cheikhi, L. and Idri, A. (2019) 'Towards a taxonomy of software maintainability predictors', *New Knowledge in Information Systems and Technologies*, Vol. 1, pp.823–832.

García, S., Luengo, J. and Herrera, F. (2015) *Data Preprocessing in Data Mining*, Springer.

Hassouneh, Y., Turabieh, H., Thaher, T., Tumar, I., Chantar, H. and Too, J. (2021) 'Boosted whale optimization algorithm with natural selection operators for software fault prediction', *IEEE Access*, Vol. 9, pp.14239-14258.

Hotzkow, J. (2017) 'Automatically inferring and enforcing user expectations', *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp.420–423.

Hudaib, A., Al-Zaghoul, F.F., Saadeh, M. and Saadeh, H. et al. (2015) 'ADTEM-architecture design testability evaluation model to assess software architecture based on testability metrics', *Journal of Software Engineering and Applications*, Vol. 8, pp.647–655.

Hudaib, A.A. and Fakhouri, H.N. (2016) 'An automated approach for software fault detection and recovery', *Communications and Network*, Vol. 8, pp.158–169.

Jureczko, M. and Madeyski, L. (2010) 'Towards identifying software project clusters with regard to defect prediction', *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pp.1–10.

Li, N., Shepperd, M. and Guo, Y. (2020) 'A systematic review of unsupervised learning techniques for software defect prediction', *Information and Software Technology*, Vol. 51, pp.106–287.

Li, Z.., Jing, X.Y. and Zhu, X. (2018) 'Progress on approaches to software defect prediction', *IET Software*, Vol. 12, pp.161–175.

Malhotra, R. (2014) 'Comparative analysis of statistical and machine learning methods for predicting faulty modules', *Applied Soft Computing*, Vol. 21 pp.286–297.

Malhotra, R. (2015) 'A systematic review of machine learning techniques for software fault prediction', *Applied Soft Computing*, Vol. 27, pp.504–518.

Malhotra, R. and Kamal, S. (2019) 'An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data', *Neurocomputing*, Vol. 343, pp.120–140.

Moeyersoms, J., De Fortuny, E.J., Dejaeger, K., Baesens, B. and Martens, D. (2015) 'Comprehensible software fault and effort prediction: a data mining approach', *Journal of Systems and Software*, Vol. 100 pp.80–90.

Montani, S. and Anglano, C. (2008) 'Achieving self-healing in service delivery software systems by means of case-based reasoning', *Applied Inteliigence*, Vol. 28, pp.139–152.

Nisa, I.U. and Ahsan, S.N. (1985) 'Fault prediction model for software using soft computing techniques', *International Conference on Open Source Systems and Technologies (ICOSST)*, pp.78–83.

Oman, P. and Hagemeister, J. (1994) 'Construction and testing of polynomials predicting software maintainability', *Journal of Systems and Software*, Vol. 24, pp.251–266.

Pandey, S.K., Mishra, R.B. snd Tripathi, A.K. (2021) 'Machine learning based methods for software fault prediction: a survey', *Expert Systems with Applications*, Vol. 172. Doi: 10.1016/j.eswa.2021.114595.

Radjenović, D., Heričko, M., Torkar, R. and Živkovič, A. (2013) 'Software fault prediction metrics: a systematic literature review', *Information and Software Technology*, Vol. 55, pp.1397–1418.

Rathore, S.S. and Kumar, S. (2018) 'A study on software fault prediction techniques', *Artifical Intelligence*, Vol. 51, pp.255–327.

Riaz, M., Mendes, E. and Tempero, E. (2009) 'A systematic review of software maintainability prediction and metrics', *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pp.367–377.

Shepperd, M., Song, Q., Sun, Z. and Mair, C. (2013) 'Data quality: some comments on the NASA software defect datasets', *IEEE Transactions on Software Engineering*, Vol. 39, pp.1208–1215.

Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W.M., Ohira, M., Adams, B., Hassan, A.E. and Matsumoto, K-I. (2013) 'Studying re-opened bugs in open source software', *Empirical Software Engineering*, Vol. 18, pp.1005–1042.

Singh, S. and Mehrotra, M. (2021) 'Prediction models for software reliability: an insight', *Design Engineering*, pp.15638–15654.

Singh, S., Mehrotra, M. and Bharti, T.S. (2022) 'A comparison of 4-parameter mathematical logistic growth model with other srgm based on bugs appearing in the software', *Proceedings of the IOT with Smart Systems*, Vol. 2, pp.409–507.

Tanwar, H. and Kakkar, M. (2019) 'A review of software defect prediction models', *Data Management, Analytics and Innovation: Proceedings of ICDMAI*, pp.89–97.

Taradeh, M., Mafarja, M., Heidari, A.A., Faris, H., Aljarah, I., Mirjalili, S. and Fujita, H. (2019) 'An evolutionary gravitational search-based feature selection', *IEEE Access*, Vol. 497, pp.219–239.

Tumar, I., Hassouneh, Y., Turabieh, H. and Thaher, T. (2020) 'Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction', *IEEE Access*, Vol. 8, pp.8041–8055.

Turabieh, H., Mafarja, M. and Li, X. (2019) 'Iterated feature selection algorithms with layered recurrent neural network for software fault prediction', *Expert Systems with Applications*, Vol. 122, pp.27–42.

Wang, S. and Wang, J., Nam, J. and Nagappan, N. (2021) 'Continuous software bug prediction', *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp.1–12.

Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y. and Zhang, T. (2019) 'Software defect prediction based on kernel PCA and weighted extreme learning machine', *Information and Software Technology*, Vol. 106, pp.182–200.

Yohannese, C.W. and Li, T. (2017) 'A combined-learning based framework for improved software fault prediction', *Atlantis Press BV*, Vol. 10, pp.647–655.

Yu, Q., Qian, J., Jiang, S., Wu, Z. and Zhang, G. (2019) 'An empirical study on the effectiveness of feature selection for cross-project defect prediction', *IEEE Access*, Vol. 19, pp.35710–35718.