Advances and Applications in Mathematical Sciences
Volume 21, Issue 6, April 2022, Pages 3093-3103

A MODEL FOR AUTOMATED BUG CLASSIFICATION USING MACHINE LEARNING

SARITHA DOPPALAPUDI, SRILATHA BADDURI, NAVYA BOMMU, SRI LAVANYA KAPAROUTHU and N. MD. JUBAIR BASHA

Department of CSE
Kallam Haranadhareddy Institute
of Technology, Guntur, AP, India
E-mail: sarithadoppalapudi17@gmail.com
srilathareddybadduri@gmail.com
navyareddybommu@gmail.com
srilavanya541@gmail.com
jubairbasha@gmail.com

Abstract

Android malware prediction becomes an essential activity in the formation and implementation of coding systems. Deficiency prediction can be a significant essential for quality assurance techniques in the early stages of the computing system development cycle and has been explored loosely in the last 20 years. The central forecast for defective modules in establishing a standardized protocol can make it possible for the event team to quickly and effectively offer top-of-the-range goods in a limited time. Various machine learning approaches may be innovative because of the confirmation of defective modules, which are used to the prediction of codes defects in NASA's data set JM1 by identifying hidden patterns between the coding systems attributes throughout this study. This work proposes a new planned model that promotes the standardization of this XG Boost model by automatically boosting its parameter, especially estimator, learning rate, max depth, and sub-sample. The experimental results reveal that the intended models have inaccurate coding program development method defect prediction levels to improve the code quality.

1. Introduction

Bug classification and prediction is one of the main issues in software testing. Various efforts have been carried out in the past. But still the

2020 Mathematics Subject Classification: 68-XX.

Keywords: bug classification, XGBoost, Machine Learning, malware prediction.
Received July 20, 2021; Accepted September 22, 2021

research issues remain in various stages. Saiqa Aleem et al. [1] discussed the success of the software project nowadays. The biggest headache of a project manager is faults or bugs. Bad code design and implementation cause these bugs. M. Farida Begum et al. [3] A team that works on a project comprising 5 to 6 developers, some have the expertise, and some are new, is the primary problem in default-free code. Now the new developer does not know what problems can emerge in real-life settings with this code.

They are therefore only carrying out this project without worrying about future bugs. Subsequently, this application will be disseminated across users. They will experience in a non-uniform environment that the Surbhi Parnerkar [4] proposed bugs that affect application rating, customer engagement, and performance. Rashid et al. [5] noted that "we had spent a lot of work and resources fixing issues. Sometimes these vulnerabilities or defects can be seen, and hackers will use our website or app and sometimes compromise us with crucial information or money. This exposes the software industry's dilemma and the need to predict software flaws in its development phase the procedures to ensure safety safeguards before these effects occur".

The major problem in the software business is to design an application that is 100% bug-free. This problem is challenging for software developers to achieve, even if it is tested by Markland et al. [6]. Shruthi Puranika et al. [7] suggested that any application built by a person is essentially not an automated process because defects are prevalent or natural. Software developer organizations focus on the early detection of defects through many checks and testing techniques. To overcome this problem, we, therefore, reviewed that different ways are based on machine learning.

The rest of the paper is discussed in various sections. Section 1 deals with the introduction. Section 2 presents the related work and various issues works carried in the past works. Section 3 presents a planned model for automated bug classification using machine learning approach. Section 4 discusses the implementation of the methodology. Section 5 presents the results and its discussion. Section 6 concludes the paper.

2. Related Work

Various ways to bug classification have been offered. In the past, various

researchers have discussed many approaches. The problems relating to the prediction of defects still exist in this direction.

Zhaowei [8] "compared most approaches to machine learning in this method, including supervised and unchecked learning. WEKA experiment tool and PROMISE - NASA data collection is used for model training". Vignesh et al. [9] have implemented a CNN and Long Short-Term Memory (LSTM) model for precise detection. Neetu Goyal et al. [10] proposed using the historical data set using the Supervised Learning algorithm, primarily logistical reverse, Naif Bayes and Decision Tree. The technique of *K*-Fold cross-validation. Mohammad et al. [11] concentrated on detecting and removing external products, followed by reducing dimension. The classification system identifies faulty code from correct code by using deep Bug Framework.

Amod Kumar et al. [13] added a tool or frame called a defect detector framework with different compilers and languages, such as javac, GCC, visual study. Meiliana et al. [14] "developed a strategy employing the least and correct number of metrics performed simultaneously using marginal R square values. Uses selected Eclipse JDT Core dataset." Keita Mori et al. [15] suggested a one-class SFP (Software Fault Prediction) Model with One-Class SVM. Ali Ouni et al. [16] focused on web applications using machine learning vulnerability prediction. Validation of inputs and sanitation we generate characteristics. It calculates static rear sections for each sink. We based the program analysis on the web program's control flow charts, control dependency charts, and system dependency charts. Uma Subbiah et al. [17] advised that defects be first classified based on severity and component attributes according to their priority. It employs *X* means the Bayes Net Classifier Clustering Algorithm.

Supervised learning was proposed by Ashima Kukkar et al. [18]. KC1, MC1, AR1, AC6 and compares datasets. The results of naïve model Bays and j48 (Decision Tree Classifier) MC2. Awni et al. [19] Supervised Learning on 10 NASA Data Sets mainly classified using Bagging, SVM, Decision Tree (DS), and Random Forest (RF) classification tools. Fei Wu et al. [20] advocated that the data be gathered via an open-source software in which we translated data into object-oriented metrics.

The above literature work involves the use of techniques, such as logistic regression and decision trees. They provide less precision and presume various data sets that create prejudices if we do not meet the assumptions. I later introduced ensemble techniques, such as Random Forest, for categorization that enhance decision-making processes. However, we employed boosting approaches such as Ada Boost for classification to prevent bias from generation but still suffer from low accuracies.

3. A Model for Automated Bug Classification using Machine Learning

The Random Forest works very well for the Software Bug prediction. However, other newly arrived algorithms called XG Boost can also boost by offering a method for building a classification model with more accuracy than Random Forest.

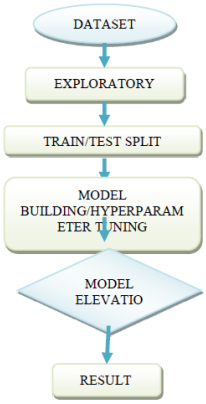


Figure 1. A Model for automated Bug Classification.

4. Implementation

The proposed methodology may be implemented by using the following concepts.

(A) Logistic Regression:

Logistic algorithm is an approach for machine learning used to classify problems. It is a predictive analytical tool based on the idea of probability.

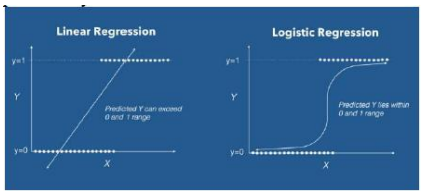


Figure 2. Linear Regression vs Logistic Regression Graph.

Here, call a logistic reversal a linear reversal model, however, "the logistic reversal uses a more complicated feature, which is characterized as the 'sigmoid function' or also called 'logical function, rather than a linear function. The logistic regression hypothesis restricts the cost function from 0 to 1. Thus linear functions do not reflect it since it may have a value larger than one or less than 0, which is not possible in the logistic regression hypothesis".

$$0 \le h_\theta(x) \le 1.$$

Logistic regression hypothesis expectation

(B) Random forest: "Random Forest" is an algorithm for the conventional learning machine that is part of the controlled learning experience. We may use it for classification and regression tasks in ML. It builds on the concept of entity learning, a process through which we can combine different classification systems to solve a complicated problem and improve model performance. As the name says, 'Random Forest is a classifier containing a range of decision trees on various data sub-sets and takes the average to raise the predicted precision of the data set." Instead of depending on a decision-making tree, we expect the random forest to provide a high number of votes from each tree. The rising quantity of forest trees leads to more accuracy and decreases over-fitting. The following graph explains how the algorithm Random Forest works.

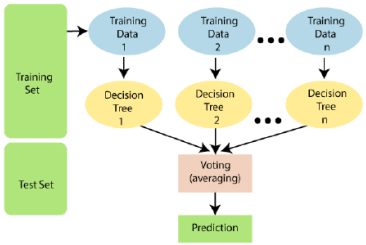


Figure 3. Prediction Process.

(C) ADA BOOST Algorithm: Adaptive Boosting, the brief is a boosting method used in machine learning as an ensemble method. It is called Adaptive Boosting because we reassign the weights of each instance to mistakenly identified cases with higher weights. Boosting is used to reduce both bias and variation in supervised learning. It works on the concept that students are sequentially cultivated. Except for the first, every succeeding student comprises previously grown students.

Simply put, turn weak students into strong ones. Adaboost method operates on the same idea as boosting however, the working process differs slightly.

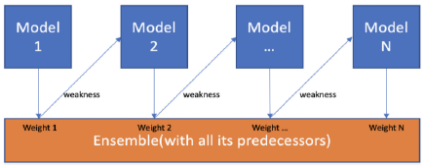


Figure 4. Ensemble Process.

(D) XGboost Algorithm: XGBoost stands for the boosting of eXtreme Gradient. Recently, it has been popular and dominates applied machines and Kaggle contests on structured data because of its scalability. XGBoost is an extension to gradation-enhanced decision-making bodies (GBM).

XGBoost Features

(i) **Regularized Learning:** The time of formalization minimizes overall ultimate weights gained to avoid high fitness. The standardized goal selects a system with primary and accurate parameters.

(ii) **Gradient Tree Boosting:** It cannot optimize the model for a tree set with standard optimization.

(iii) **Shrinkage and Column Subsampling:** It employs further two further strategies to prevent over-fitting besides the regularized aim. The first technique is Friedman's shrinkage. Recent weights applied with the factor μ after each step of the tree boost. Shrinkage scales Shrinkage lowers each tree's impact and leaves space for subsequent branches, in line with the evolutionary computing learning rate, enhance the model.

The second is the column for sub-sampling (feature). We have utilized it

in the Random Forest. The sub-sampling column prevents far more than the standard sub-sampling line. Using sub-sample columns also speeds up the calculation of parallel algorithms.

(E) **Decision Tree:** The Forest Algorithm is a supervised learning method that can be used for classification problems and regression, but most typically for classification problems. The classification is tree-structured, where the core nodes represent the functions of the dataset, where the trees reflect the rules of selection, and where every node in the tree is the result. Decision Tree contains two nodes: the Deciding Terminal and the Tree. It employs decision nodes and have many branches, whereas leaf nodes represent the output and have no branching.

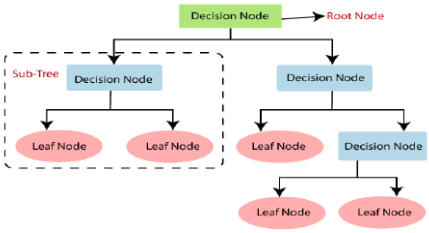
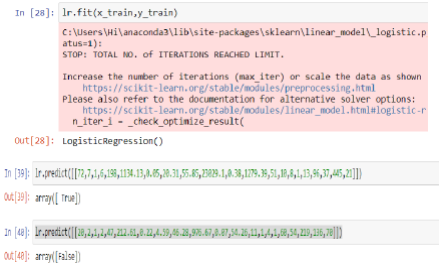


Figure 5. Decision Tree Process.

5. Discussion of Results

a. Logistic Regression:



b. Decision Tree:

```
In [43]: tree.fit(x_train,y_train)

Out[43]: DecisionTreeClassifier()
```

```
In [51]: tree.predict([[72,7,1.6,398,1134.13,0.05,20.31,55.85,23029.1,0.28,1279.28,53,10,8,1,13,90,37,445,21]])
Out[51]: array([ True])

In [52]: tree.predict([[20,2,1,2,47,212.63,0.22,4.59,44.28,676.67,0.07,54.28,11,1,4,1,60,54,219,136,70]])
Out[52]: array([ True])
```

c. Random Forest:

```
In [55]: rf.fit(x_train,y_train)

Out[55]: RandomForestClassifier()
```

```
In [64]: rf.predict([[20,2,1.8,2.5,6,1,5.8,2.3,65,2.3,2,1.5,63,52,45,4,5,6,3,2.3,5]])
Out[64]: array([ True])
```

d. Ada Boost:

```
In [67]: ad.fit(x_train,y_train)

Out[67]: AdaBoostClassifier()
```

```
In [75]: ad.predict([[20,2,1,1,47,212.63,0.22,4.59,45.28,676.67,0.07,54.28,11,1,4,1,60,54,219,136,70]])
Out[75]: array([ True])
```

e. XGBoost (Gradient Boosting Classifier):

```
[08:18:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linea
r is now deprecated in favor of reg:squarederror.

Out[92]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1,
              objective='reg:linear', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Representation of Accuracy Levels

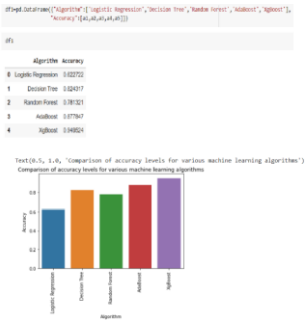


Figure 6. Comparison of accuracy levels for various algorithms.

6. Conclusion

The preprocessed information with characteristic scaling for more significant mining and choice has automatically handled sophisticated imbalance in data sets using the SMOTE approach of victimization. The proposed methodology of machine learning - logistic regression, Decision Tree, Random Forest, ada boost, and XG Boost as NASA-JM1 datasets. Soon a new model standardized the dominant XGBoost model by adjusting its unique N estimator, learning rate, max depth, and sub-sample parameters. The findings were compared, and the model also outperformed each other. Many entirely various methodologies are units employed for software system detect prediction. On broad public data set JM1, from the PROMISE warehouse, employs Machine Learning Models area unit call Tree, logistic regression, Random Forest, Ada Boost, XG Boost. The applied algorithms show more incredible average JM1 data set accuracy rates in XG models to speed up learning and give higher average data set performance. The performance figures suggest that the intended models are intelligent in the prediction of software system defects. It helps to identify the categorization of bug severity and the priority victimization of deep learning approaches. It might involve alternative unit approaches of metric capacity and provide a comprehensive comparison between them. As a part of future work, the metrics of software systems within the learning process are an approach to increase the prediction model accuracy.

References

[1] Saiqa Aleem, Luiz Fernando Capretz and Faheem Ahmed, Benchmarking machine learning technique for software defect detection, IJSEA 6(3) (2015).

[2] Jayati Deshmukh, Annervaz KM, Sanjay Podder, Shubhashis Sengupta and Neville Dubash, Towards Accurate Duplicate Bug Retrieval using Deep Learning Techniques, IEEE, 2017.

[3] S. Delphine Immaculate, M. Farida Begam and M. Floramary, Software Bug Prediction Using Supervised Machine Learning Algorithms, IEEE, (2019).

[4] Surbhi Parnerkar, Ati Jain and Vijay Birchha, An Approach to Efficient Software Bug Prediction using Regression Analysis and Neural Networks, IJIRCCE, (2015).

[5] Rashid, Mamoon and Lovepreet Kaur, Finding Bugs in Android Application using Genetic Algorithm and a priori Algorithm, Indian Journal of Science and Technology 9(23) (2016), 1-5.

[6] Markland J. Benson, Toward Intelligent Software Defect Detection NASA: (2019).

[7] Shruthi Puranika, Pranav Deshpande and K. Chandrasekaran, A Novel Machine Learning Approach for Bug Prediction, Science Direct, (2016).

[8] Lin Chen, Bin Fang and Zhaowei Shang, Software fault prediction based on One-Class SVM, IEEE, (2016).

[9] M. Vignesh and K. Kumar, Web Application Vulnerability prediction using machine learning, IJSER, (2017).

[10] Neetu Goyal, Naveen Aggarwal, and Maitreyee Dutta, A Novel Way of Assigning Software Bug Priority Using Supervised Classification on Clustered Bugs Data, Springer, (2015).

[11] Mohammad and Zubair Khan, Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study, JSEA, Meenakshi, Dr. Satwinder Singh, Software Bug Prediction using Machine Learning Approach, IRJET, (2019).

[12] Alsaeedi, Abdullah and Mohammad Zubair Khan, Software defect prediction using supervised machine learning and ensemble techniques: a comparative study, Journal of Software Engineering and Applications 12(5) (2019), 85-100.

[13] Amod Kumar and Ashwni Bansal, Software Fault Proneness Prediction Using Genetic Based Machine Learning Techniques, IEEE, (2019).

[14] Meiliana, Syaeful Karim, Harco Leslie Hendric Spits Warners, Ford Lumban Gaol, Edi Abdurahman, Benfano Soewito, Software Metrics for Fault Prediction Using Machine Learning Approaches, IEEE, (2017).

[15] Keita Mori and Osamu Mizuno, An Implementation of Just-In-Time Fault-Prone Prediction Technique Using Text Classifier, IEEE, (2015).

[16] Ali Ouni, Marwa Daagi, Marouane Kessentini, Salah Bouktif and Mohamed Mohsen Gammoudi, A Machine Learning-Based Approach to Detect Web Service Design Defects, IEEE, (2017).

[17] Uma Subbiah, Muthu Ramachandran and Zaigham Mahmood, Software Engineering Approach to Bug Prediction Models using Machine Learning as a Service (MLaaS), IEEE, (2019).

[18] Ashima Kukkar, Rajni Mohana, Anand Nayyar, Jeamin Kim, Byeong-Gwon Kang and Naveen Chilamkurti, A Novel Deep- Learning-Based Bug Severity Classification Using Convolutional Neural Networks and Random Forest with Boosting, IEEE, (2019).

[19] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan and Fatima Alsarayrah, Software Bug Prediction using Machine Learning Approach, Research Gate, (2018).

[20] Fei Wu, Xiao-Yuan Jing, Ying Sun, Jing Sun, Lin Huang, Fangyi Cui and Yanfei Sun, Cross-Project and Within-Project Semi-supervised Software Defect Prediction: A Unified Approach, IEEE, (2018).