International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/IJSRCSEIT

A Study Using Machine Learning Techniques to Predict Software Quality

B. Sivaji *[1], D. Santhosh[2]
MCA Student, Madanapalle Institute of Technology & Science, Madanapalle, India

ABSTRACT

Article Info

Publication Issue :

Volume 8, Issue 4
July-August-2022

Page Number : 316-323

Article History
Accepted: 10 August 2022
Published: 30 August 2022

Software quality estimation is an activity needed at various stages of software development. It may be used for planning the project`s quality assurance practices and for benchmarking. In earlier previous studies, two methods (Multiple Criteria Linear Programming and Multiple Criteria Quadratic Programming) for estimating the quality of software had been used. Also, C5.0, SVM and Neutral network were experimented with for quality estimation. These studies have relatively low accuracies. In this study, we aimed to improve estimation accuracy by using relevant features of a large dataset. We used a feature selection method and correlation matrix for reaching higher accuracies. In addition, we have experimented with recent methods shown to be successful for other prediction tasks. Machine learning algorithms such as XGBoost, Random Forest, Decision Tree, Logistic Regression and Naïve Bayes are applied to the data to predict the software quality and reveal the relation between the quality and development attributes. The experimental results show that the quality level of software can be well estimated by machine learning algorithms.

Keywords : Estimation, Machine Learning, Software Quality, Extreme Gradient Decent, Boosting.

I. INTRODUCTION

Software applications may contain defects, originating from requirements analysis, specification and other activities conducted in the software development. Therefore, software quality estimation is an activity needed at various stages. It may be used for planning the project based quality assurance practices and for benchmarking. In addition, the number of defects per unit is considered one of the most important factors that indicate the quality of the software.

The quality of a software product can be defined as the measure of performance of a system on which the software is implemented in terms of execution time, memory capacity utilized and probability of errors, etc. In addition to this, the amount of effort contributed by the software developer also represents a key factor while assessing the quality of a software product. The quality of a software product can be considered to be internal as well as external. The internal quality of a software can be assessed in course of development during software development life cycle (SDLC); whereas, the external quality can be measured during its implementation and can be assessed with respect to its level of functionality. The external quality also depends upon its internal quality. In order to assess the external quality of a software product, quality models can be devised that represent a function of the internal quality attributes. In order to achieve this, first of all the internal attributes must be identified and then the relationship existing between the internal and external quality attributes must be identified. A number of software quality prediction models have been proposed by various authors. However, the machine learning approach to devising such a model appears to be more popular and more effective as claimed by the authors. We have been motivated by this aspect to carry out a review of the machine learning approaches for software quality prediction models the quality of a software product can be defined as the measure of performance of a system on which the software is implemented in terms of execution time, memory capacity utilized and probability of errors, etc. In addition to this, the amount of effort contributed by the software developer also represents a key factor while assessing the quality of a software product. The quality of a software product can be considered to be internal as well as external. The internal quality of a software can be assessed in course of development during software development life cycle (SDLC); whereas, the external quality can be measured during its implementation and can be assessed with respect to its level of functionality. The external quality also depends upon its internal quality. In order to assess the external quality of a software product, quality models can be devised that represent a function of the internal quality attributes. In order to achieve this, first of all the internal attributes must be identified and then the relationship existing between the internal and external quality attributes must be identified. A number of software quality prediction models have been proposed by various authors. However, the machine learning approach to devising such a model appears to be more popular and more effective as claimed by the authors. We have been motivated by this aspect to carry out a review of the machine learning approaches for software quality prediction models the quality of a software product can be defined as the measure of performance of a system on which the software is implemented in terms of execution time, memory capacity utilized and probability of errors, etc. In addition to this, the amount of effort contributed by the software developer also represents a key factor while assessing the quality of a software product. The quality of a software product can be considered to be internal as well as external. The internal quality of a software can be assessed in course of development during software development life cycle (SDLC); whereas, the external quality can be measured during its implementation and can be assessed with respect to its level of functionality. The external quality also depends upon its internal quality. In order to assess the external quality of a software product, quality models can be devised that represent a function of the internal quality attributes. In order to achieve this, first of all the internal attributes must be identified and then the relationship existing between the internal and external quality attributes must be identified. A number of software quality prediction models have been proposed by various authors. However, the machine learning approach to devising such a model appears to be more popular and more effective as claimed by the authors. We have been motivated by this aspect to carry out a review of the machine learning approaches for software quality prediction models The quality of a software product can be defined as the measure of performance of a system on which the software is implemented in terms of execution time, memory capacity utilized and probability of errors, etc. In addition to this, the amount of effort contributed by the software developer also represents a key factor while assessing the quality of a software product. The quality of a software product can be considered to be internal as well as external. The internal quality of a software can be assessed in course of development during software development life cycle (SDLC); whereas, the external quality can be measured during its implementation and can be assessed with respect to its level of functionality. The external quality also depends upon its internal quality. In order to assess the external quality of a software product, quality models can be devised that represent a function of the internal quality attributes. In order to achieve this, first of all the internal attributes must be identified and then the relationship existing between the internal and external quality attributes must be identified. A number of software quality prediction models have been proposed by various authors. However, the machine learning approach to devising such a model appears to be more popular and more effective as claimed by the authors. We have been motivated by this aspect to carry out a review of the machine learning approaches for software quality prediction mode the quality of a software product can be defined as the measure of performance of a system on which the software is implemented in terms of execution time, memory capacity utilized and probability of errors, etc. In addition to this, the amount of effort contributed by the software developer also represents a key factor while assessing the quality of a software product. The quality of a software product can be considered to be internal as well as external. The internal quality of a software can be assessed in course of development during software development life cycle (SDLC); whereas, the external quality can be measured during its implementation and can be assessed with respect to its level of functionality.

The external quality also depends upon its internal quality. In order to assess the external quality of a software product, quality models can be devised that represent a function of the internal quality attributes. In order to achieve this, first of all the internal attributes must be identified and then the relationship existing between the internal and external quality attributes must be identified. A number of software quality prediction models have been proposed by various authors. However, the machine learning approach to devising such a model appears to be more popular and more effective as claimed by the authors. We have been motivated by this aspect to carry out a review of the machine learning approaches for software quality prediction models.

II. RELATED WORKS

A Review of Machine Learning Techniques for Software Quality Prediction: Successful implementation of a software product entirely depends on the quality of the software developed. However, prediction of the quality of a software product prior to its implementation in real-world applications presents significant challenges to the software developer during the process of development. A limited spectrum of research in this area has been reported in the literature as of today. Most of the researchers have concentrated their research work on software quality prediction using various machine learning techniques.

An experimental study for software quality prediction with machine learning methods: Software quality estimation is an activity needed at various stages of software development. It may be used for planning the project's quality assurance practices and for benchmarking. In earlier previous studies, two methods (Multiple Criteria Linear Programming and Multiple Criteria Quadratic Programming) for estimating the quality of software had been used Also, C5.0, SVM and Neutral network were experimented with for quality estimation.

A survey on machine learning techniques used for software quality prediction: In the present software development scenario, software quality prediction has become significantly important for successful implementation of the software in real world application and enhances the longevity of its functionality. Moreover, early identification of anticipated fault prone software modules in the process of development of software is crucial in saving efforts involved in this process. Machine learning techniques are considered to be the most appropriate techniques for software quality prediction and a large spectrum of research work has been conducted in this direction by several authors. In this paper, we conduct an extensive survey on various machine learning techniques like fuzzy logic, neural network, and Bayesian model, etc. used for software quality prediction along with an analytical justification for each of the proposed solutions.

Study of the impact of hardware fault on software reliability: As software plays increasingly important roles in modern society, reliable software becomes desirable for all stakeholders. One of the root causes of software failure is the failure of the computer hardware platform on which the software resides. Traditionally, fault injection has been utilized to study the impact of these hardware failures. One issue raised with respect to the use of fault injection is the lack of prior knowledge on the faults injected, and the fact that, as a consequence, the failures observed may not represent actual operational failures.

A survey on machine learning techniques used for software quality prediction: In the present software development scenario, software quality prediction has become significantly important for successful implementation of the software in real world application and enhances the longevity of its functionality. Moreover, early identification of anticipated fault prone software modules in the process of development of software is crucial in saving efforts involved in this process. Machine learning techniques are considered to be the most appropriate techniques for software quality prediction and a large spectrum of research work has been conducted in this direction by several authors. In this paper, we conduct an extensive survey on various machine learning techniques like fuzzy logic, neural network, and Bayesian model, etc. used for software quality prediction along with an analytical justification for each of the proposed solutions.

III. METHODS AND MATERIAL

Proposed system:
In this study, we aimed to improve estimation accuracy by using relevant features of a large dataset. We used a feature selection method and correlation matrix for reaching higher accuracies. In addition, we have experimented with recent methods shown to be successful for other prediction tasks. Machine learning algorithms such as XgBoost, Random Forest, Decision Tree, Logistic Regression and Naïve Bayes are applied to the data to predict the software quality.

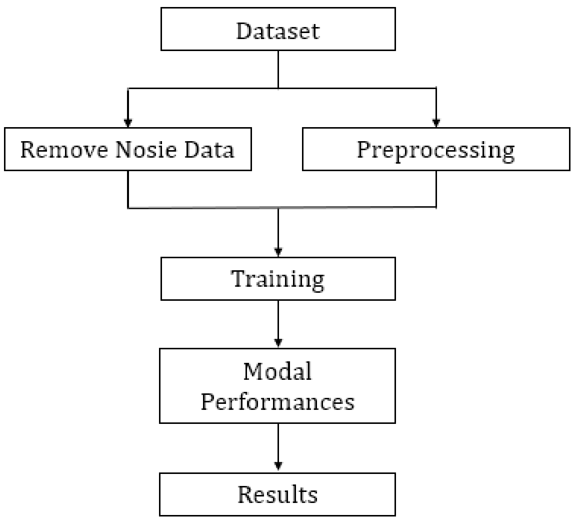


Figure 1: Block diagram

IV. IMPLEMENTATION

The project was carried out using the algorithms listed below.

XGBoost Classifier:

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

Bagging: Now imagine instead of a single interviewer, now there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

Random Forest Classifier:

First, Random Forest algorithm is a supervised classification algorithm. We can see it from its name, which is to create a forest by some way and make it random. There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result. But one thing to note is that creating the forest is not the same as constructing the decision with information gain or gain index approach.

The author gives four advantages to illustrate why we use Random Forest algorithm. The one mentioned repeatedly by the author is that it can be used for both classification and regression tasks. Overfitting is one critical problem that may make the results worse, but for Random Forest algorithm, if there are enough trees in the forest, the classifier won't overfit the model. The third advantage is the classifier of Random Forest can handle missing values, and the last advantage is that the Random Forest classifier can be modeled for categorical values.

There are two stages in Random Forest algorithm, one is random forest creation, the other is to make a prediction from the random forest classifier created in the first stage.

STEPS:

1. Randomly select "K" features from total "m" features where k << m

2. Among the "K" features, calculate the node "d" using the best split point

3. Split the node into daughter nodes using the best split

4. Repeat the a to c steps until "l" number of nodes has been reached

5. Build forest by repeating steps a to d for "n" number times to create "n" number of trees

Decision Tree Classifier:

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal.

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/internal node, based on which the tree splits into branches/ edges. The end of the branch that doesn't split anymore is the decision/leaf, in this case, whether the passenger died or survived, represented as red and green text respectively.

Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can't ignore the simplicity of this algorithm. The feature importance is clear and relations can be viewed easily. This methodology is more commonly known as learning decision tree from data and above tree is called Classification tree as the target is to classify passenger as survived or died. Regression trees are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

So, what is actually going on in the background? Growing a tree involves deciding on which features to choose and what conditions to use for splitting, along with knowing when to stop. As a tree generally grows arbitrarily, you will need to trim it down for it to look beautiful. Let's start with a common technique used for splitting.

Logistic Regression:

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable (target) is categorical.

For example,

To predict whether an email is spam (1) or (0)

Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.

From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

Types of Logistic Regression

1. Binary Logistic Regression

The categorical response has only two 2 possible outcomes. Example: Spam or Not

2. Multinomial Logistic Regression

Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5

Bagging Classifier:

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

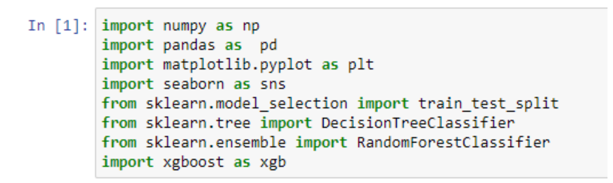
Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples (or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out. Bagging reduces Overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. ... The base estimator to fit on random subsets of the dataset.

V. RESULTS AND DISCUSSION

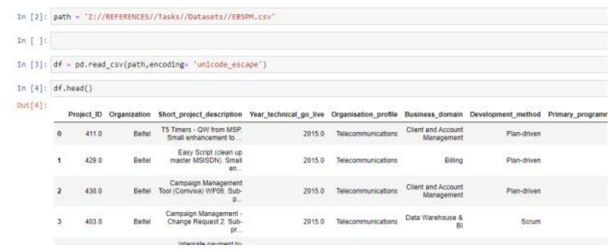
The following screenshots are depicted the flow and working process of project.

Importing Libraries:



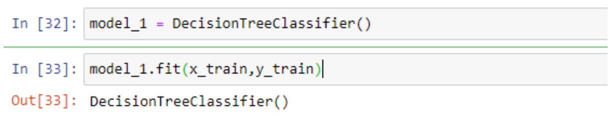
```
In [1]: import numpy as np
        import pandas as  pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        import xgboost as xgb
```

Data Loading:



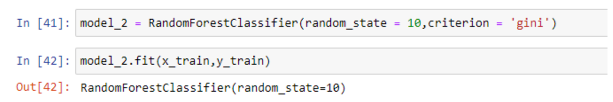
```
In [2]: path = 'Z://REFERENCES//Tasks//Datasets//EBSPM.csv'

In [ ]:

In [3]: df = pd.read_csv(path,encoding= 'unicode_escape')

In [4]: df.head()

Out[4]:
   Project_ID Organization Short_project_description      Year_technical_go_live Organisation_profile Business_domain        Development_method Primary_programm
0  411.0      Beltel       T5 Timers - QW from MSP.       2015.0                 Telecommunications   Client and Account     Plan-driven
                           Small enhancement to ...                                                   Management
1  429.0      Beltel       Easy Script (clean up          2015.0                 Telecommunications   Billing                Plan-driven
                           master MSISDN). Small
                           en...
2  430.0      Beltel       Campaign Management            2015.0                 Telecommunications   Client and Account     Plan-driven
                           Tool (Comviva) WP05. Sub-                                                  Management
                           p...
3  403.0      Beltel       Campaign Management -          2015.0                 Telecommunications   Data Warehouse &       Scrum
                           Change Request 2. Sub-                                                     BI
                           pr...
                           integrate payment by
```

Decision Tree Model Building:



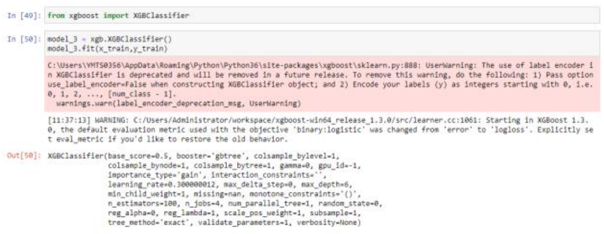
```
In [32]: model_1 = DecisionTreeClassifier()

In [33]: model_1.fit(x_train,y_train)
Out[33]: DecisionTreeClassifier()
```

Random Forest Model Building:



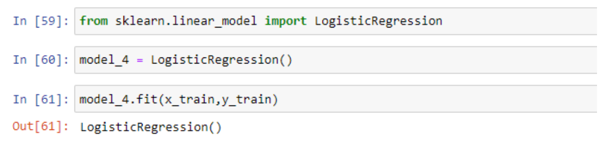
```
In [41]: model_2 = RandomForestClassifier(random_state = 10,criterion = 'gini')

In [42]: model_2.fit(x_train,y_train)
Out[42]: RandomForestClassifier(random_state=10)
```

XGBoost Model Building:



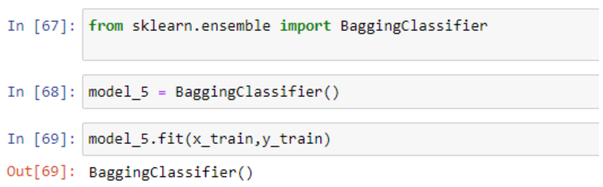
```
In [49]: from xgboost import XGBClassifier

In [50]: model_3 = xgb.XGBClassifier()
         model_3.fit(x_train,y_train)

         C:\Users\VMTS0356\AppData\Roaming\Python\Python36\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label encoder i
         n XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option
         use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e.
         0, 1, 2, ..., [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)

         [11:37:13] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.
         0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
         t eval_metric if you'd like to restore the old behavior.

Out[50]: 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, random_state=0,
                       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                       tree_method='exact', validate_parameters=1, verbosity=None)
```

Logistic Regression Model Building:



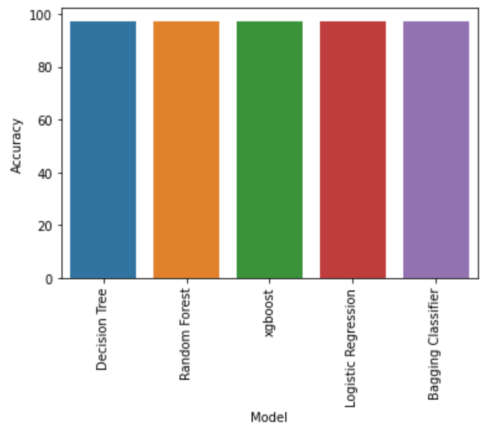
```
In [59]: from sklearn.linear_model import LogisticRegression

In [60]: model_4 = LogisticRegression()

In [61]: model_4.fit(x_train,y_train)
Out[61]: LogisticRegression()
```

Bagging Classifier Model Building:



```
In [67]: from sklearn.ensemble import BaggingClassifier

In [68]: model_5 = BaggingClassifier()

In [69]: model_5.fit(x_train,y_train)
Out[69]: BaggingClassifier()
```

Graphs:



VI.CONCLUSION

In this application, we used supervised Machine Learning models to predict the quality of the software. We used five ML algorithms to predict software quality they are Random Forest Classifier, Decision Tree Classifier, XGBoost Classifier, Logistic Regression and Bagging Classifier. All five algorithms performs well with good accuracies.

VII. REFERENCES

[1]. N.Kalaivani, Dr.R.Beena, International Journal of Pure and Applied Mathematics Volume 118 No. 20 2018, 3863-3873 ISSN: 1314-3395.

[2]. He, Peng, et al. "An empirical study on software defect prediction with a simplified metric set." Information and Software Technology 59 (2015): 170-190.

[3]. Yu, Xiao, et al. "Using Class Imbalance Learning for Cross-Company Defect Prediction." 29th International Conference on Software Engineering and Knowledge Engineering (SEKE 2017). KSI Research Inc. and Knowledge Systems Institute, 2017.

[4]. D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?." Software Quality Journal, 26(2), 2018, pp. 525-552.

[5]. X. Wang, Y. Zhang, L. Zhang and Y. Shi, "A Knowledge Discovery Case Study of Software Quality Prediction: ISBSG Database," 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Toronto, ON, 2010, pp. 219-222.

[6]. X. Wang, Y. Zhang, L. Zhang and Y. Shi, "A Knowledge Discovery Case Study of Software Quality Prediction Based on Classification Models: ISBSG Database," The 11th International Symposium on Knowledge Systems Sciences (KSS 2010), 2010.

[7]. S. Sohrabi, O. Udrea, and A. V. Riabov, "Hypothesis Exploration for Malware Detection Using Planning," Twenty-Seventh AAAI Conf. Artif. Intell., pp. 883– 889, 2013.

[8]. Zimmermann, Thomas, et al. "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process." Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009.

[9]. Amasaki, S., Takagi, Y., Mizuno, O., Kikuno, T.: Constructing a bayesian belief network topredict final quality in embedded system development. IEICE Trans. Inf. Syst.8(6), 1134–1141(2005).

[10]. Idri, A., Abra, A.: A fuzzy logic based measures for software project similarity: validation andpossible improvements. In: Proceedings of 7th International Symposium on Software Metrics,pp. 85–96. IEEE, England, UK (2001).

[11]. Pattnaik, S., Pattanayak, B.K.: A survey on machine learning techniques used for software quality prediction. Int. J. Reasoning-Based Intell. Syst. 12(1/2), 3–14 (2016).

[12]. Kapur, P.K., Khatri, S.K., Goswami, D.N.: A generalized dynamic integration software reliability growth model based on neural network approach. In: Proceedings of International Conference on Reliability, Safety and Quality Engineering, pp. 831–838 (2008).

[13]. Wagner, S.: A bayesian network approach to assess and predict software quality using activity-based quality model. Inf. Softw. Technol. 52(11), 1230–1241 (2010).

[14]. Ahmed, M.A., AL-Jamini, H.A.: Machine learning approaches for predicting software maintainability: a Fuzzy based transparent model. IET Softw. 7(6), 317–326 (2013).

[15]. Pattnaik, S., Pattanayak, B.K., Patnail, S.: Prediction of software quality using neuro-fuzzy model. Int. J. Intell. Enterp. (IJIE) 5(3), 292–307 (2018).

Cite this article as :

B. Sivaji, Mr. A. Damodar Reddy, "A Study Using Machine Learning Techniques to Predict Software Quality", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 8 Issue 4, pp. 316-323, July-August 2022.

Journal URL : https://ijsrcseit.com/CSEIT228465