A Novel Software Quality Characteristic Recommendation Model to Handle the Dynamic Requirements of Software Projects that Improves Service Quality and Cost

Kamal Borana[1], Meena Sharma[2], Deepak Abhyankar[3]
Research Scholar, Computer Engineering Department-Institute of Engineering and Technology,
Devi Ahilya Vishwavidyalaya, Indore, India[1]
Professor, Computer Engineering Department-Institute of Engineering and Technology,
Devi Ahilya Vishwavidyalaya, Indore, India[2]
Software Engineer, School of Computer Science & IT, Devi Ahilya Vishwavidyalaya, Indore, India[3]

***Abstract*—The software is created and constructed to address particular issues in the applied field. In this context, there is a need to be aware of the crucial characteristics to assess the quality of software. But not all software requires checking all the quality-of-service parameters, resulting in effort loss and time consumption. Therefore, it is required to develop software quality characteristics recommendation model to address and resolve the issue. The proposed work involved in this paper can be subdivided into three main parts (1) a review of popular software quality models and their comparison to create a complete set of predictable, and (2) the design of an ML-based recommendation model for recommending the software quality model and software quality characteristics (3) performance analysis. The proposed recommendation system utilizes the different software quality of service attributes as well as the software attributes where these models are suitably applied to satisfy the demands. Profiling of applications and their essential requirements have been performed Based on the different quality of service parameters and the requirements of applications. These profiles are learned by machine learning algorithms for distinguishing the application-based requirement and recommending the essential attributes. The implementation of the proposed technique has been done using Python technology. The simulation aims to demonstrate how to minimize the cost of software testing and improve time and accuracy by utilizing the appropriate quality matrix. Finally, a conclusion has been drawn and the future extension of the proposed model has been reported.**

Keywords*—Recommendation system; software quality model; ML (Machine Learning); quality matrix; software quality characteristics

I. Introduction

Machine learning provides ease in several real-world applications; in addition, improves the capacity and capability of existing research and methodologies.ML techniques are also used to improve and optimize different process models for improving the cost of employment and productivity[1][2].In this context, software quality evaluation is one of the essential steps. There are several different software quality measuring models currently utilized. These models include several different quality measuring characteristics.

Software quality is an emerging research area in the field of software engineering. The work presented here is relevant to the research around software quality models which gives a better understanding and knowledge of software quality attributes in Software quality models. Achieving software quality assurance requires the use of software quality models [3]. These quality attributes might be used to describe the software's quality. It might be difficult to decide which of the excellent models to utilize [4]. In Addition, software quality models are used for the global assessment of the software product. Therefore, the proposed issue of applying and selecting the appropriate quality matrix is defined here as the recommendation problem. The recommendation engines are the machine learning technique for evaluating the problem's current scenario and suggesting the most suitable solutions for the given set of problems [5].

There are different kinds of recommendation systems available, which will also work as the information filter to reduce the less relevant data and optimize the ranking of the desired set of information [6][7]. The proposed software quality characteristics recommendation model includes the technique of machine learning to learn when, where, and which software characteristic is appropriate for evaluation based on profiling of the software quality of service requirements. In this context, the proposed work is subdivided into the following essential task:

- Examination of different software quality characteristics and models.
- Implement and design a content-based recommendation model to suggest the appropriate quality matrix.
- Study the impact of the software quality characteristic recommendation model over the existing models.

In this section, an outline of the proposed concept for the software quality measuring model is provided. The next section involves the study of different software quality estimation models. Further, the proposed recommendation model has been formulated and its implementation plan will be explained. In addition, based on the implemented model simulations have been carried out and their performance will be reported. Finally, the conclusion has been reached and their future extension plan will be proposed.

The paper is organized as follows: Section II reviews software quality models based on basic and tailored criteria which give a solid foundation in attribute selections. Section III briefly introduces some essential attributes and a proposed software quality characteristics recommendation model is introduced which uses machine learning for recommending the appropriate characteristics for software quality estimation. Section IV describes the used Machine learning algorithms in the proposed model. Section V gives an analysis and validation of the results. Finally, this work is concluded in Section VI where possible future research directions are indicated.

II. Review of Literature

In this study, two types of software quality models are considered. The first one is based on basic software quality models and the second one is based on the tailored software quality models which are described in the below section.

A. Basic Software Quality Models

Each of the several software quality models consists of a number of different attributes. Basic quality models denote those models which were developed until 2000. This study needs to explore some essential software quality models and each model has various qualities.

1) McCall's quality model: It exposes three task domains.

a) Operation in a product refers to its capacity to be easily understood, and able to deliver the desired results. It addresses criteria for correctness, dependability, effectiveness, integrity, and usability.

b) Revision in a product is the ability to endure modifications, error rectification, and system adaptation. It includes testability criteria, maintainability, and flexibility.

c) When a product is in a transition phase, it means it can accommodate distributed processing in new environments with rapidly changing technology.

The goal was on the relationship between metrics and quality characteristics [3]. The problem is that it is dependent on Yes and No responses, there is no accuracy in the results.

2) Boehm's quality model: It adds maintainability to McCall's model [8]. High-level factors are as follows:

a) Utility describes efficient, reliable, and easy to use.

b) Maintainability describes the ability to modify, testable, and features of understanding.

3) Dromey's quality model: Three models have been given by Dromey, i.e., the Requirement model, the design model, and the implementation model. The product properties are given below:

a) Correctness assesses if certain principles are broken, along with usability and reliability.

b) Measures the effectiveness of a component's deployment in terms of usability, maintainability, efficiency, and reliability.

c) Descriptive evaluates the description of a component, about maintainability, reusability, portability, and usability.

d) Despite the design quality model considering the development process, architectural integrity is not fully attention Testability is implicitly included. None of the domain-specific properties are discussed. Furthermore, one drawback of the said model is allied with reliability and maintainability, as judging them before the software system is functioning is not practical. [8] [9]

4) FURPS quality model: The elements of the FURPS model that are considered [8] [9] are:

a) Functionality encompasses capability sets, security, and feature sets.

b) Usability includes stability in the user interface view, help (online), user documentation, and materials required in training.

c) Reliability focuses on the mean time between failures (MTBF), frequency and strictness of the failure, accuracy, recoverability, and predictability.

d) Functional needs like efficiency, speed, availability, throughput, accuracy, resource utilization, reaction time, recovery time, and are constrained by performance.

e) Testability, extensibility, adaptability, maintainability, and compatibility are all aspects of supportability. Its failure to consider software portability is one drawback. The model does not include any attributes that are domain specific.

5) ISO 9126 quality model: The McCall and Boehm models served as the basis for the ISO model [10][11][12]. It works on four parts quality Model, quality in use metrics, internal quality attribute, and external quality attribute. The attention of that model is to an exploration of attributes into 6 independent characteristics which are reliability, usability, efficiency, functionality, maintainability, and portability. Now attributes are further split into internal quality attributes, which refer to system features that can be assessed without incorporating, and external quality attributes, which refer to assessment by observation while it is being carried out. [13]. This model addresses effectiveness, security, and satisfaction [14].

6) ISO 25010 quality model: The modernized version of ISO 9126 is ISO 25010. This approach divides quality into eight smaller sub-characteristics. The ISO-9126 Model serves as the sole foundation for the set of standards. The model adds new features including compatibility and

security. As an extension of portability, it employs the term transferability while conducting its operation.

B. Tailored Software Quality Models

Tailored software quality models were built from the fundamental (basic) software quality models. This model was made with certain individual components. There are various tailored software quality models are there, and selected tailored software quality models are presented below.

1) BERTOA model: It defines the quality attributes for the assessment of Commercial Off-The-Shelf Components. The application of the model is to build Complex software.

2) GEQUAMO model: The breakdown of the sub-layers in this model, known as GEQUAMO (Generic, Multi-layered, and Customizable Model) [15], allows for the flexible inclusion of user requirements. End users can create their models with the aid of that model.

3) ALVARO model: The methodology used in the Alvaro model is essentially used to certify software components and identify quality components. The model involves a framework that may be divided into four sections: components related to the model quality, framework for technical certification, certification process, and framework with metrics. For quality assessment and technical certification, all components are utilized.

4) RAWASHDEH model: Rawashdeh's model is conquered by the Dromey and ISO 9126[16] models. It addresses the genuine requirements of various users. To produce high-quality products four processes are suggested by the said model.

a) Selecting a limited group of quality characteristics, applied using a top-down method, dividing each characteristic into several subordinate characteristics.

b) Examine how internal and external measures differ. This includes characteristics such as requirements or lines of code, as well as external metrics, behavior during testing procedures, and components.

c) Quality attributes for each user must be identified.

d) Any new quality model can be built from ISO 9126, and the Dromey model.

In addition to the above software models S. S. Kamaruddin et al [6] provide a feature subset selection approach to choose the right attributes for software quality assessment to address this dynamic software quality assessment problem. The current models for evaluating the quality of software do not permit dynamic assessment. As new quality attributes surface, they can be incorporated into the model in dynamic software quality evaluation. To establish dynamic software quality evaluation, they concentrated on creating an intelligent technology that can learn and include new quality criteria into the model. Additionally, S. S. Kamaruddin et al [7] introduce a filter-wrapper-based feature ranking method that can learn from and order quality attributes depending on fresh data from software quality assessment instances. The Most Priority of Feature (MPF) score method and the software quality attribute weights learning algorithm make up the suggested feature ranking strategy. The issue of repetition in the rankings of the software quality attribute is not addressed by the present ranking methodologies. The redundancy problem is solved by the suggested method by selecting characteristics with good classification accuracy utilizing classifiers.

III. Proposed Methodology

In this section, a proposed software quality characteristics recommendation model is introduced which uses machine learning for recommending the appropriate characteristics for software quality estimation. The proposed model is aimed at reducing the quality estimation time overhead. But according to the available different quality estimation matrices as discussed in the above section, it has been observed some essential characteristics which are followed by entire models these characteristics are described in Table I.

TABLE I. Properties of Basic and Tailored Software Quality Models

Properties	Tailored models	Basic models
Functionality	Yes	No
Maturity	Yes	No
Resource-utilisation	Yes	No
Testability	Yes	No
Compliance	Yes	No
Understandability	Yes	No
Usability	Yes	No
Learnability	Yes	No
Reliability	No	Yes

Therefore, the characteristics available in Table I have been included in our quality matrix. In addition, based on the requirements a list of questions is also included that help to decide the additional quality characteristics requirements.

Key Questions have been prepared with the consultation of IBM India Pvt. Ltd.

Q1. Is the model involving any calculations?

If the software is being developed for performing any calculation, then the following characteristics need to be included:

1. Accuracy
2. Correctness
3. Efficiency

Q2. Is the model involve security, privacy, and communication modules?

If the software involves communication and data security, then the following properties need to be considered.

1. Integrity

2. Fault Tolerance
3. Time Behaviour

Q3. Is the software involved in data collection and analysis?

If yes, then the following characteristics must involve:

1. Human Engg.
2. Analysability

Q4. Is the software utilized by a person who has a new or non-technical background?

If yes, then need to consider the followings:

1. Recoverability
2. Suitability
3. Attractiveness
4. Operability

Q5. Is the software needed to change, modify, or scale shortly?

If yes, then need to consider the followings:

1. Adaptability
2. Changeability
3. Flexibility
4. Modifiability
5. Reusability
6. Operability
7. Suitability

Q6. Does the software need to deploy in multiple places/multiple machines/multiple clients with the same or different configurations?

If yes, then need to consider the followings:

1. Flexibility
2. Installability
3. Maintainability
4. Portability
5. Transferability
6. Configurability
7. Compatibility
8. Reusability
9. Interoperability

Q7. Is software deployed in resource-constrained scenarios?

If yes, then the followings need to include:

1. Stability
2. Resource Utilisation
3. Self Contained
4. Replaceability
5. Manageability

Q8. Is software having many modules which require assistance?

If yes, then the followings need to include:

1) Supportability: To decide the suitable quality of service the proposed model has been demonstrated in Fig. 1. The flowchart of the proposed software quality characteristics recommendation model is presented here in Fig. 2. Additionally, the key components are described in Table I. The proposed model accepts two inputs, namely the project source code and the prepared questionnaire. These questionnaires are prepared based on the activities involved in the project development life cycle. Based on these questionnaires the dataset has been prepared. The Highlight of the dataset is defined in Table II.

The given Table II provides a limited number of dataset instances but provides a structure of the dataset which is used for training. By using a similar method, prepared a total of $2^{n-1} + 1 = 2^{8-1} + 1 = 129$ instances. In this situation for making training with a supervised machine learning algorithm, it is required to decide the class labels of these instances. To calculate class labels of instances, let a project quality requirement can be satisfied with an initial set of software quality characteristics as given in Table I.

This table is denoted as A1. Additionally, the answers to the questions can be denoted as:

$$A_n = \{A1, A2\ldots, A8\} \quad (1)$$

Each true answer to the questions includes a set of quality characteristics, where A1 is always constant and A1 = True (T). Therefore, the set of attributes for a unique class label can be calculated using:

$$C = A1 \cup Ai \quad (2)$$

Where Ai is the set of characteristics where i'th question's answer is true.

Some examples of unique class calculations have been given in Table II. Additionally, the dataset has a similar number of classes to predict. The meaning of the class label is defined in Table III.

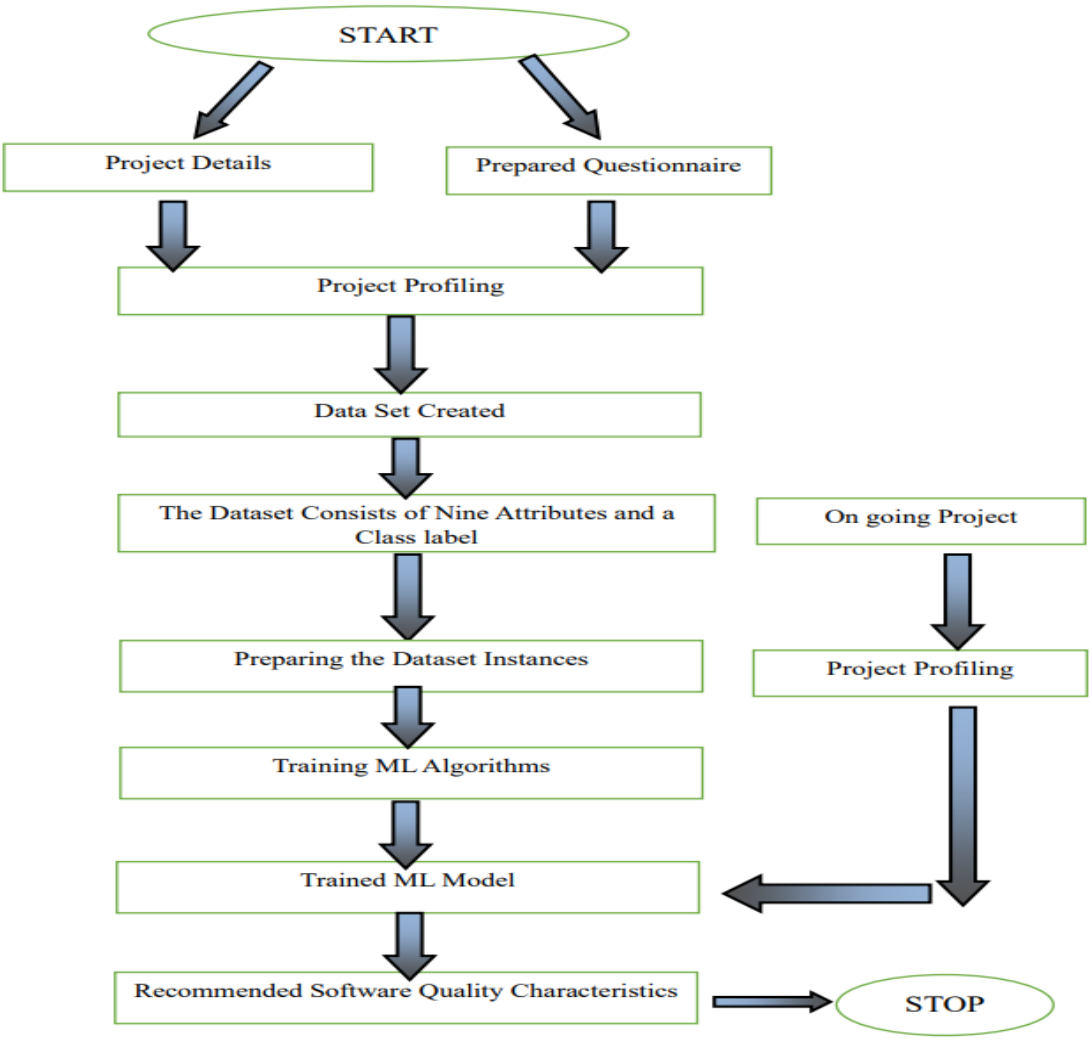


Fig. 1. Flow chart of proposed software quality characteristics recommendation model.

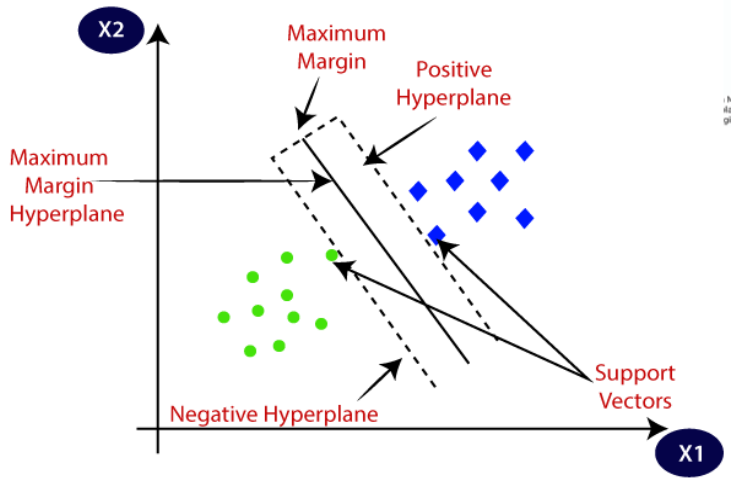


Fig. 2. Support Vector Machine (SVM) classifier diagram.

TABLE II. EXAMPLE OF DATASET USED

A1	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Class
T	F	F	F	F	F	F	F	F	C1
T	T	F	F	F	F	F	F	F	C2
T	F	T	F	F	F	F	F	F	C3
T	F	F	T	F	F	F	F	F	C4
T	F	F	F	T	F	F	F	F	C5
T	F	F	F	F	T	F	F	F	C6
T	F	F	F	F	F	T	F	F	C7
T	F	F	F	F	F	F	T	F	C8
T	F	F	F	F	F	F	F	T	C9

TABLE III. EXAMPLES OF CLASS LABEL SEQUENCES

Class	Means
C1	$A1$
C2	A1 ∪ A_1 if $A_1 = true$
C3	A1 U A_2 if A2 = *true*

The mapping for each class label is prepared in the dataset. In the proposed model, the name given for this process is the profiling of historical projects. Now, it is needed to train machine learning algorithms on the prepared profiles. In this context, five widespread ML algorithms have been used such as K nearest neighbor (KNN), Support vector machine (SVM), artificial neural network (ANN), C4.5 decision tree, and Bayesian.

IV. USED ML ALGORITHMS

A. Support Vector Machine (SVM)

One or more hyperplanes are created by the SVM classifier for regression and classification [17]. The line is used to divide the 2D linearly separable data. The line function is:

$$y = ax + b \quad (3)$$

Here in Equation (3) variable x is renamed with x_1 and y is renamed with x_2 then the line Equation can be defined as:

$$ax_1 - x_2 + b = 0 \quad (4)$$

Now, if define $x = (x_1, x_2)$ and $w = (a, -1)$, then:

$$w \cdot x + b = 0 \quad (5)$$

This is the hyperplane equation. After obtaining the hyperplane, it can be used to create predictions. Hypothesis function h is called a Hypothesis function shown in below Equation (6).

$$h(x_i) = \begin{cases} +1 & if\, w \cdot x + b \geq 0 \\ -1 & if\, w \cdot x + b \leq 0 \end{cases} \quad (6)$$

In addition to this SVM, the classifier can also be explored by the below flowchart.

The below Fig. 3 explores the support vector machine and talks about classification and regression. With the help of the SVM algorithm, we may swiftly categorize new data points in the future by determining the optimal line or decision border of n-dimensional space. A hyperplane is the name given to this optimal decision boundary.

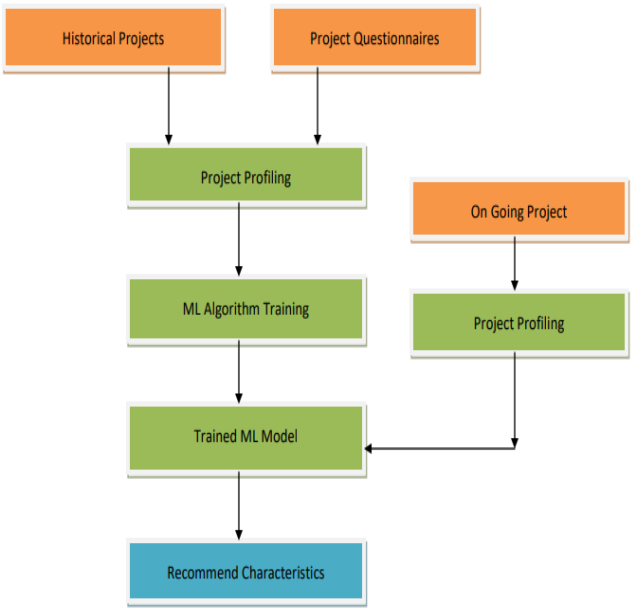


Fig. 3. Proposed software quality characteristic recommendation model.

B. Naive Bayes

A probabilistic classifier is the Naive Bayes classification [18]. The Bayes theorem can be used to derive this. Need to train the Naive Bayes algorithm as supervised learning using observations from nature. Two types of probabilities are given below: Posterior Probability P (H/X) and Prior Probability P (H), Where H is an assumption and X is data. Thus, Baye's Theorem stated: In the below Fig. 4 explore the flow chart of Naive Bayes classification in which every iteration according to the probability value of attributes is updated.

$$P\left(\frac{A}{B}\right) = \frac{P\left(\frac{B}{A}\right)P(A)}{P(B)} \quad (7)$$

C. K-Nearest Neighbor Classification

The KNN is a traditional tool for both classification and prediction applications [19]. It is a lazy learning classifier. The KNN method has three key parts. First, it calculates the distance between the sample under consideration and each training sample. To calculate the distances mostly Euclidean distance will be used. Euclidean is described by:

$$d(e, f) = \sqrt{\sum_{i=1}^{N}(f_i - e_i)^2} \quad (8)$$

Where q is the query vector and p is the dataset samples.

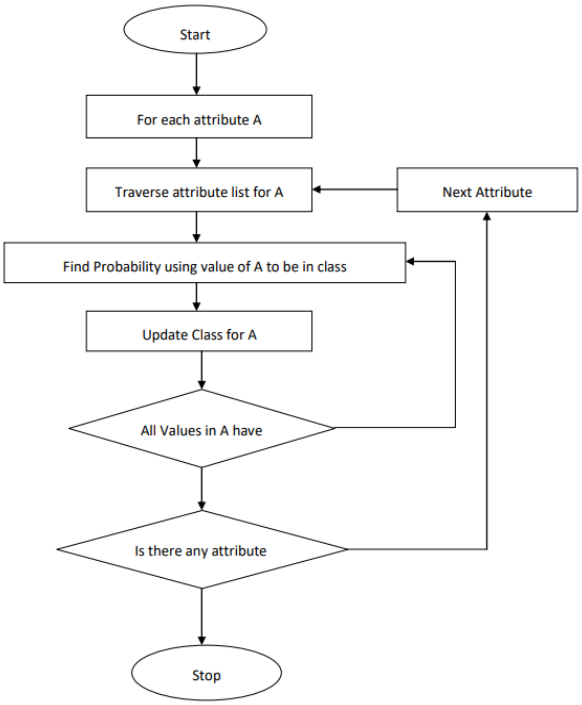


Fig. 4. Flow chart of naive bayes classification.

The method assigns a class label to the sample in question after measuring the distance between the two samples. Based on the k nearest samples from the training sample, the class label is assigned. In this case, the designer will supply the integer k. Not a lot of data is needed for the k-NN to learn. It is the algorithm's main benefit.

In the below Fig. 5 explores the flow chart of the KNN Classifier which is based on the Euclidean distance.

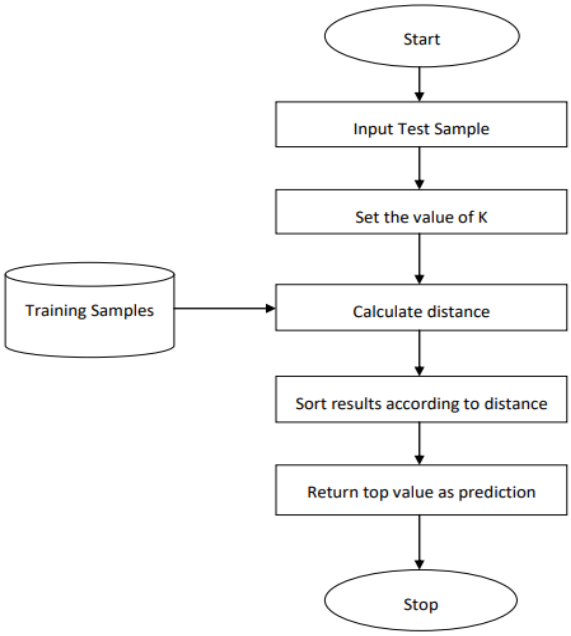


Fig. 5. Flow chart of KNN classifier.

D. Artificial Neural Network (ANN)

In processes where complicated multivariable, non-direct relationships between information and yield factors are present, ANN is becoming a popular showing tool [20].

ANN is primarily a data handling discipline, with its underpinnings in the operational principles of organic neural systems. It is like our cerebrum, which gets the data, translates, and gives the yield. Many preparation elements are known as hubs or neurons make up an ANN. These components are highly coupled with one another and serve as a system to produce standard results. The new features of ANN include its ability to understand instances and infer conclusions from precise information. Additionally, the speed and precision levels are unmatched. The charming element of ANN is no past information or science behind the procedure is required and thus they are alluded to as discovery displays. Also, ANN models can go up against every single semantic variable or parameter that can't be estimated, and ordinary demonstrating techniques are in this manner unacceptable and may neglect to give reasons. Neural networks could classify patterns for which they have not been taught, as well as a high tolerance for noisy input. The neural network's internal weights, which are applied by the transactions employed during the learning process, are a primary issue of the training phase. The predicted output is added to each training transaction for the neural network. The concept of ANN can be visualized below in Fig. 6:

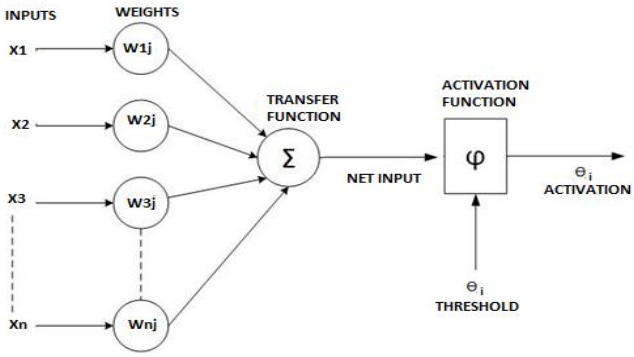


Fig. 6. Mathematical model of artificial neural network.

E. C4.5 Decision Tree

The program must produce decision rules. This is an expansion of the well-known ID3 decision tree [21]. To obtain data partition, entropy as well as information gain is required in the program. Additionally, the highest information gain attribute is selected to create a tree. Partitioned sub-lists are used by the C4.5 method to shape a full decision tree. The algorithm considers the restrictions of the next set. If samples in the dataset cover a similar class, then it forms a leaf node as a decision tree.

1) If the computation of IG (information gain) is not attainable then it creates a node higher up, then the expected value of the class is used by the tree.

2) If a previously undetected class is met, the decision node has been created from a target value.

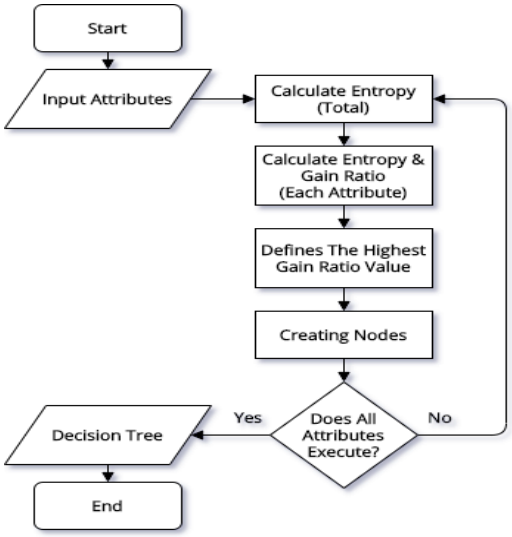


Fig. 7. Flow chart of decision tree (C4.5 or J48) algorithm.

Before presenting the decision tree's phases, it is necessary to acknowledge the information acquired. Thus, compelled to discuss entropy first, considering that the aftereffect decision tree classifies data into two classes, i.e., P (+ve) and N (-ve). The binary classification of entropy S is given by E(S):

$$E(S) = -P(+ve)\log_2 P(+ve) - P(-ve)\log_2 P(-ve) \quad (9)$$

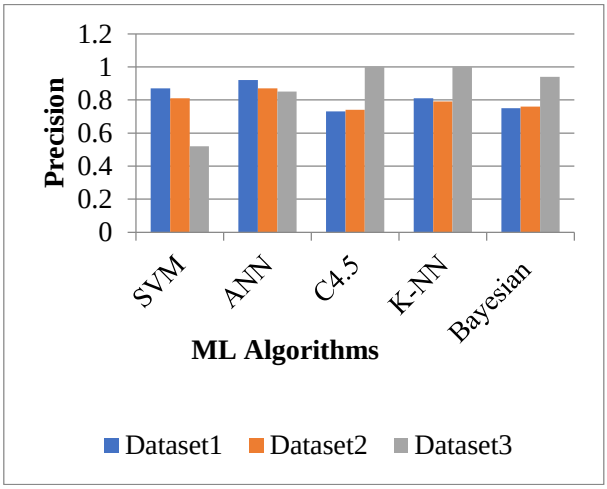
The above Figure 7 is exploring the step-by-step process of the C4.5 decision tree algorithm. The best potential characteristic to separate tree branches must be chosen to reduce the depth of the tree when traversing it. It can be seen that the best choice will be the property with the least entropy. As a necessary decrease in entropy in connection to each characteristic during splitting, the information gain can be described. The IG (information-gain) calculates, Gain (E, A) of an attribute A using equation number (10).

$$Gain(E, A) = Entropy(s) - \sum_{n=1}^{v} \frac{E_v}{E} XEntropy(E_v) \quad (10)$$

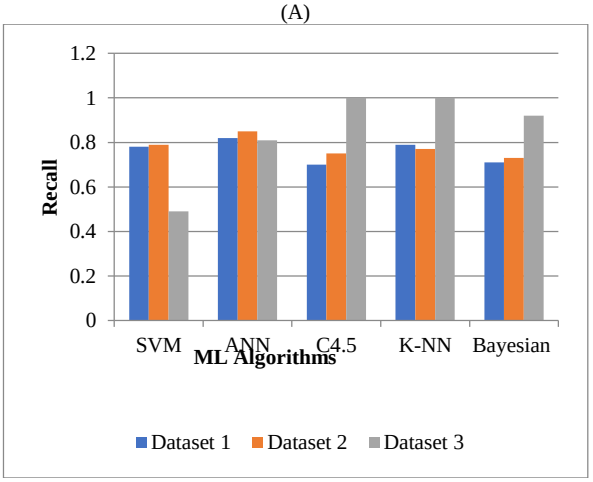
The system accepts the ongoing project activities-based questions after the algorithm has been trained with the profiled attributes of the projects. Based on the trained machine learning and the input current project questioners the model predicts the most suitable characteristics for the project.

V. Experiments and Result Analysis

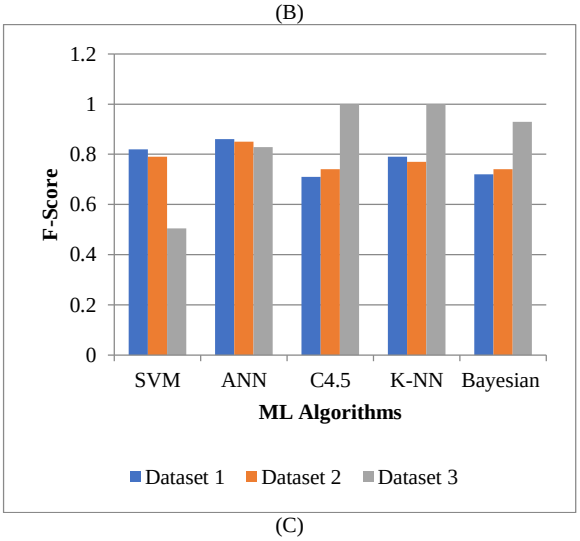
Three datasets of the machine learning algorithm's classification problem are used in this experiment. Here the dataset obtained from Kaggle [22] is denoted as Dataset 1, the dataset obtained from GitHub [23] is given as Dataset 2 and the prepared dataset is denoted as Dataset 3. The precision, recall, and f-score are calculated based on class wise, and then the mean of the performance is calculated. The performance of the model is also reported using the bar graph in Fig. 8. Fig. 8(A) demonstrates the precision of the proposed model, 8(B) shows the recall, 8(C) shows the f-score and finally Fig. 8(D) shows the accuracy of the presented model. The performance demonstrates the comparison of five different machine learning algorithms for predicting the quality characteristics as a recommendation of the quality characteristics. According to the obtained performance of the model, support vector machine (SVM) and artificial neural network (ANN) is providing higher performance as compared to C4.5, KNN, and Bayesian. But the Support vector machine has a higher amount of time complexity. Thus, it is suggested to be utilizing the ANN as the final machine learning algorithm which is suitably worked with the proposed recommendation system.



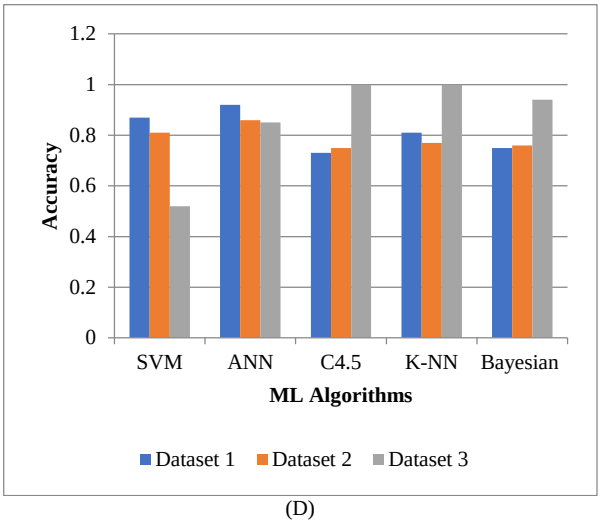
(A)



(B)



(C)



(D)

Fig. 8. (A). Precision of the proposed recommendation model, (B). Recall of the proposed recommendation model, (C). F Score of the proposed recommendation model, (D). Accuracy of the proposed recommendation model.

VI. Conclusion and Future Work

Concerning the criteria of the project, the presented investigation aims to produce an intelligent model that offers software quality characteristics. These recommendations are based on the involved activities in the project. These activities are summarized as questionnaires during different software development process life cycles. The project source code and documentation are utilized with the proposed recommendation system to analyze. The past project's quality assurance characteristics were utilized to develop an ML model to predict the desired attributes. The ML algorithm needs to train with a limited set of sequences for predicting the possible attributes to evaluate the software. Here, the suggested model has been evaluated using five machine-learning methods across three distinct datasets. Accuracy, F-score, precision, and recall are used to evaluate performance. A suggested model demonstrates higher prediction accuracy which means that model successfully be able to serve as the software quality characteristics recommendation model. Additionally, the model is also helpful to lower expenses and increase the revenue of software development companies.

Soon the following future extension has been proposed for investigation:

1) Apply real-world data to evaluate the performance of the proposed model.

2) Prepare detailed questionnaires for each stage of software development which influences the characteristics of the software quality testing matrix.

3) To enhance the model performance, can apply deep learning techniques.

References

[1] Sircar, K. Yadav, K. Rayavarapu, N. Bist, H. Oza, "Application of machine learning and artificial intelligence in oil and gas industry", Petroleum Research 6 (2021) 379-391.

[2] Siebert, J., Joeckel, L., Heidrich, J., Trendowicz, A., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., Aoyama, M.: Construction of a quality model for machine learning systems, Software Quality Journal, 30:307–335, 2022.

[3] M. Bhushan, S. Goel, "Improving software product line using an ontological approach", Sadhana, 41, 1381–1391, 2016.

[4] J. P. Miguel, D. Mauricio, G. Rodríguez, "A Review of Software Quality Models for The Evaluation of Software Products", International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, November 2014.

[5] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh "Recommendation systems: Principles, methods and evaluation", Egyptian Informatics Journal Volume 16, Issue 3, November 2015, Pages 261-273.

[6] S. S. Kamaruddin, J. Yahaya, A. Deraman, R. Ahmad, "Feature Subset Selection Method for Dynamic Software Quality Assessment", 5th Malaysian Software Engineering Conference (My SEC), 2011.

[7] S. S. Kamaruddin, J. Yahaya, A. Deraman, R. Ahmad, "Filter-Wrapper based Feature Ranking Technique for Dynamic Software Quality Attributes", Knowledge Management International Conference (KMICe) 2012, Johor Bahru, Malaysia, 4 – 6 July 2012.

[8] IEEE STD 610.12-1990 "IEEE Standard Glossary of Software Engineering-Terminology", http://web.ecs.baylor.edu/faculty/grabow/Fall2013/csi3374/secur Standards/IEEE610.12.pdf, 1990.

[9] Al-Badareen A. Bassam, "Software Quality Evaluation: User's View," International Journal of Applied Mathematics and Informatics, Issue 3, Volume 5, pp 200 207, 2011.

[10] ISO/IEC 9126-1: Software Engineering - Product Quality- Part 1: "Quality Model, International Organization for Standardization," Switzerland, 2001.

[11] ISO/IEC 9126-2: Software Engineering - Product Quality- Part 2: "External Metrics International Organization for Standardization," Switzerland, 2002.

[12] ISO/IEC 9126-3: Software Engineering - Product Quality- Part 3: "Internal Metrics, International Organization for Standardization," Switzerland, 2003.

[13] A. Alvaro, E. S. de Almeida, S. R. de Lemos Meira, "A Software Component Quality Framework,"ACM SIGSOFT SEN 35, 1 Mar. 2010.

[14] B. W. Boehm, J. R. Brown, M. Lipow, "Characteristics of Software Quality," North Holland, (1978).

[15] C. Jones, "Strengths and Weaknesses of Software Metrics," Version 5, March 22, 2006.

[16] J.A. McCall et al, "Factors in Software Quality," Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977.

[17] Cristianini, N., & Shawe-Taylor, J. (2000). Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press.

[18] Duda., R. O., Hart, P. E., & Stork, D. G. (2001). Pattern Classification. Wiley-Interscience.

[19] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[20] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[21] Lior Rokach, Oded Maimon (2008). Data Mining with Decision Trees: Theory and Applications.

[22] https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction

[23] https://github.com/feiwww/GHPR_dataset