ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20211029115834
Original scientific paper

An Experimental Study on Attribute Validity of Code Quality Evaluation Model

Tianze GUO, Hanli BAI*, Yunzhan GONG, Yawen WANG, Dahai JIN

Abstract: Regarding the practicality of the quality evaluation model, the lack of quantitative experimental evaluation affects the effective use of the quality model, and also a lack of effective guidance for choosing the model. Aiming at this problem, based on the sensitivity of the quality evaluation model to code defects, a machine learning-based quality evaluation attribute validity verification method is proposed. This method conducts comparative experiments by controlling variables. First, extract the basic metric elements; then, convert them into quality attributes of the software; finally, to verify the quality evaluation model and the effectiveness of medium quality attributes, this paper compares machine learning methods based on quality attributes with those based on text features, and conducts experimental evaluation in two data sets. The result shows that the effectiveness of quality attributes under control variables is better, and leads by 15% in AdaBoostClassifier; when the text feature extraction method is increased to 50 - 150 dimensions, the performance of the text feature in the four machine learning algorithms overtakes the quality attributes; but when the peak is reached, quality attributes are more stable. This also provides a direction for the optimization of the quality model and the use of quality assessment in different situations.

Keywords: Code metrics; Feature extraction; Machine learning; Quality evaluation

1 INTRODUCTION

The software has penetrated all areas of social life. A software project with unstable quality may lead to failure, paralysis, and even to the huge catastrophic consequences of the entire system [1]. Therefore, it is necessary to conduct code quality assessments during the software development process.

Software quality assurance is a systematic activity that runs through the entire software development process. If a clear assessment of the quality can be given, the pros and cons of different versions of the software and the degree of improvement can be given through comparison, which provides valuable references to the industry personnel [2]. Software quality is broadly defined as "the software produced meets express and implied requirements" [3]. For this reason, as a user, the artificial definition becomes a factor that needs to be involved in the software quality evaluation method.

Our research mainly focuses on the quality of software code. The goal of software quality evaluation is to use metrics to calculate and ultimately transform into quantitatively evaluated quality indicators, which allows us to track the reliability of a specific software project simultaneously. The basic metrics of software code mainly includes the number of modules, the number of code lines, the number of comment lines, the ratio of code comments, the fan-in and fan-out of functions, the instability factors, the number of abstract classes, the number of classes, the depth of function calls, complexity, the repetition rate, etc. The quality model is a composite conversion of the basic code measurement results, that is, to transform the basic measurement calculated by the measurement tool of source-code to a composite one using a mathematical model to obtain the artificially defined quality attributes. These quality attributes are defined by many experts in the field, which can better describe the situation of the software from various aspects, and at the same time provide relatively traceable standards for the evaluation of software quality. Many well-known quality models have been proposed in the literature related to software quality, such as McCall's [4], Boehm's [5], ISO/IEC 9126 [3], Dromey's [6], SQuaRE [7], QMOOD [8], SQO-OSS [9] and many other less well-known models. Most of the early quality models only emphasized abstract views on quality factors, which did not prove the effect of applying them in practice. Although in continuous model research, people try to solve these problems using integrated tools and specific techniques to merge basic metrics to a higher level of abstraction [10], quality is a multi-faceted concept and still requires some form of standardization and enumeration. Therefore, it is also necessary to confirm whether the existing quality model's evaluation of the code can more accurately describe the overall situation of the code, to facilitate the evaluation of the software quality.

Software quality includes many aspects, and software defects are one of the important indicators. Typically, the way to evaluate problems in a software project is based on data acquired during the history of software development and defects discovered, with the help of methods such as machine learning. Therefore, this paper aims at the problem that the performance of the quality evaluation model in the evaluation of code quality cannot be quantitatively measured. Based on the sensitivity of the quality evaluation model to code defects, a machine learning-based quality evaluation attribute validity verification method is proposed. Perform horizontal and vertical comparisons in multiple machine learning algorithms to verify the performance of the quality evaluation model in different situations, and provide support for the future research direction of the quality model and the guidance for the application of quality evaluation.

This paper firstly focuses on a review of the existing research work on the quality evaluation model. Specifically, this type of method analyses the basic measurement elements of the software code and then performs compound calculations using formulas to convert the basic measurement into defined indicators that describe the condition of the software. At present, there are many such models. The attribute definitions of each are varied, and the conversion formulas of the metric are also different. Therefore, this paper mainly chooses one of the most commonly used models QMOOD for research and conducting evaluation subsequent experiments later. After selecting the model, this paper uses the source-code measurement tool to calculate the basic measurement and corresponds the measurement result to the parameters in the model using the corresponding mapping method. Finally, the quality attributes that the experiment needs would be calculated. As for the selection of measurement tools this paper mainly selects open-source measurement tools for java language code and explains the relevant conditions of the measurement tools, and gives the correspondence between the mapping parameters.

After determining the measurement tools and models, this paper will conduct comparative experiments based on the method of controlling variables to study the accuracy of the quality evaluation model for the evaluation of code quality. This paper compares the machine learning method based on quality attributes with the machine learning method based on text feature extraction and conducts experimental evaluation on the same data set. Comparing two different feature extraction methods as input, the training performance of the same type of machine learning model for code quality evaluation shows the effectiveness of the quality attribute on the characteristics of the software code itself. In the experiment, the sample code in the control data set, the splitting method of the training set and the test set, and the machine learning model are kept consistent. The defect situation is used as the output to compare the accuracy of the final trained model. And it conducts in-depth experiments to discuss the pros and cons of the quality evaluation model in the evaluation of code quality in different situations to provide data support for the selection of evaluation methods in varying situations.

First, this paper summarizes the methods of software quality evaluation and selects the commonly used quality models to illustrate and implement them in experiments. Secondly, it concludes the existing source-code measurement tools and applies them in experiments. Then it summarizes and implements the commonly used models and feature extraction methods of machine learning. The experimental part verifies the usability and accuracy of the quality assessment by comparing the results of experimental data. Finally, the full text is summarized in combination with related research, and the directions that are worth paying attention to in the future are preliminarily discussed.

2 SOFTWARE QUALITY ASSESSMENT

2.1 Existing Research on Quality Assessment and the Goals of This Paper

Software quality is an important field of software testing. The quality of software products is measured according to their ability to meet the goals of developers and user needs. The quality evaluation mechanism relies on a quality model, which usually defines the constituent factors and evaluates and summarizes the pros and cons of individual metrics. So the judgment of whether the quality evaluation mechanism can better describe the software quality, meanwhile, whether the quality attributes can better generalize the code and apply it to the evaluation of the code quality will become problems that need to be urgently confirmed.

As mentioned above, the quality of the code can be described quantitatively by a well-defined model, and each model will give a corresponding quality index. In recent years, scholars have studied the integrated multi-version quality measurement framework [11]. The inquiry explains the relevance of the quality model evaluation results to defect information, change information, and other data. But it cannot be proven whether the quality evaluation results can describe the code situation more comprehensively and accurately or whether they can be used to evaluate the quality of the code. This paper will choose a more commonly used and excellent model in previous studies for research and apply it to prove the accuracy, representativeness, and usability of the quality evaluation results, which also explores the help of quality attributes in evaluating code defects to fill in the gap here.

2.2 The Selection and Association of Quality Models and Software Metrics

Since the quality model provides a basis for defining the association between a set of quality attributes and measurement elements, it is needed to select a model to evaluate quality at first. Existing scholars divide software quality models into two categories: basic models and customized models. Generally speaking, the project team can choose an existing ready-to-use model or use a custom design to meet specific goals. The only prerequisite is that it can be put into operation as a function of mapping specific attributes and code metrics. Through the investigation of various studies, such as [12-14] using ISO quality standards as a reference point, the system characteristics are mapped into a subset of quality characteristics. Among them, ISO-square is a real model of the basic model. Compared with the customized model, it has some limitations and cannot be shaped according to specific needs or used for certain specific products. Of course, the quality model can also be customized to meet the requirements of a specific environment [15]. In this view, people can use the composition parameters of the existing model and define new weights for them to obtain new formulas for related attributes. If necessary, it is possible to replace the original component metrics with a new set of metrics.

Table 1 QMOOD quality model [8]

Quality Attributes	Index Computation Equation
Reusability	−0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * DesignSize
Flexibility	0.25 * Encapsulation − 0.25 * Coupling + 0.5 * Composition + 0.5 *Polymorphism
Understand-Ability	−0.33 * Abstraction + 0.33 * Encapsulation − 0.33 * Coupling + 0.33 * Cohesion − 0.33 * Polymorphism − 0.33 * Complexity − 0.33 * DesignSize
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * DesignSize + 0.22 * Hierarchies
Extendibility	0.5 * Abstraction − 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

With the research of the previous work, this paper chooses QMOOD with better performance as the quality model for experimental research. As shown in Tab. 1, QMOOD [8] directly shows the mechanism of converting a set of metrics as parameters into six quality characteristics.

3 USE SOFTWARE MEASUREMENT TOOLS FOR SOURCE-CODE MEASUREMENT

3.1 Overview of Measurement Tools

Mining the historical research of software measurement and setting up novel measurement elements with a strong correlation with software quality is the key to constructing a high-quality model. Existing research divides the metrics into two categories. The first category focuses on the code size and internal complexity of program modules; the second category focuses on the analysis of the software development process, from the analysis of code modification characteristics, developers, the perspectives of experience, inter-module dependencies, and project team organizational structure to design the metric.

The software measurement tool is a program that extracts several attributes of source-code entities and converts general measurement definitions into corresponding values. There are countless open source tools, free software, and commercial applications, such as C and C++ Code Counter (CCCC), Chidamber & Kemerer Java Metrics (CKJM), Dependency Finder, Sonar, Source Monitor, JHawk, IBM Rational Logiscope, and McCabe QA. They are different in various aspects, such as language support, measurement support, open-source/closed-source, applicable software size, output file format, user interface, etc. Of course, you can also choose to develop your tools, or you can choose existing tools according to your preferences. Among them, many tools also provide the option of exporting metrics to XML/CSV files for subsequent processing using spreadsheet programs or in a database environment.

3.2 Selection and Application of Measurement Tools

Once the tool can count code metrics and the results are imported into the database, the quality attributes can be enumerated as a function of the basic metrics specified by the selected quality model, as described in section 2.2.

This paper chooses the more convenient and commonly used CKJM tool on the market to measure the source code. It is an open-source command-line measurement calculation tool used to extract source-code measurement for the selected input code. This tool can count nineteen sizes and structure metrics for each class. CKJM deals with class files, so as a prerequisite, the project should be fully compiled to enable the tool to start statistics. To this end, all source-code in the data set is compiled and passed, and command-line scripts for building software are required. The sample size of the data set which the paper used in the experiment is very large. Therefore, a small command-line batch script must be written. In addition, the path of all directories containing .class files must be specified. Metrics are planned to be redirected to a .csv file, which is an option provided by the tool itself. Once all the metric data are obtained, they will be used as input parameters for the selected quality model. This paper uses the latest version of CKJM-extended 2.0 (Chidamber and Kemerer Java Metrics and many other metrics) for experiments in the subsequent experimental part, which is an enhanced version relative to the original CKJM version [16]. After completing the preliminary basic work, batches call scripts from the command line to start CKJM for each running script, one sample source code at a time. After collecting the output in the CSV file, it can be easily imported into any spreadsheet program for further analysis.

It should be noted that the indicator names provided by CKJM are not the same as those described in the QMOOD model, but many similarities have been identified. For example, in a previous job, by using specific code metrics (such as CBO, DIT, WMC, and NOC, etc.) as the input parameters of the QMOOD quality model, some quality attributes of the software (such as reusability, flexibility, scalability and so on) are quantified [17]. In addition, according to Bansiya et al., in the lowest-level model design, the indicators used to evaluate design attributes may change, or a set of different design indicators can be used to evaluate quality attributes [8]. Therefore, the mapping of CKJM tool measurement data to QMOOD indicators and equivalent substitute indicators has been recorded in Tab. 2 [17]. In the follow-up experimental research of this paper, this paper plans to convert these metrics extracted from the source code into quality indicators, and use six quality attributes as the result data of the evaluation software to describe the code situation, which can then be used as a feature extraction method for the code in training machine learning models. And the follow-up research uses experiments to compare whether the quality attributes are more comprehensive and accurate than the basic metric description of the code.

Table 2 QMOOD properties & equivalent substitute metrics [17]

Design Property / QMOOD	Design Metric / QMOOD	Equivalent metric substitute(in this paper)
Coupling	Direct Class Coupling (DCC)	Coupling between object classes (CBO)
Cohesion	Cohesion Among Methods of Classes (CAM)	Same as in QMOOD
Messaging	Class Interface Size (CIS)	Number of public Methods (NPM)
Design Size	Design Size in Classes (DSC)	Number of Classes (NOC)
Encapsulation	Data Access Metric (DAM)	Same as in QMOOD
Composition	Measure of Aggregation (MOA)	Same as in QMOOD
Polymorphism	Number of Polymorphic Methods (NOP)	Coupling Between Methods (CBM)
Abstraction	Average Number of Ancestors(ANA)	Inheritance Coupling (IC)
Complexity	Number of Methods (NOM)	Weighted methods per Class (WMC)
Hierarchies	Number of Hierarchies (NOH)	Depth of Inheritance Tree (DIT)
Inheritance	Measure of Functional Abstraction (MFA)	Same as in QMOOD

4 MACHINE LEARNING MODEL AND FEATURE EXTRACTION

Nowadays, machine learning algorithms have become a normal and advanced method for solving fuzzy problems. In other words, machine learning can be used to solve problems where the exact results are not known, where what is good and what is bad is not strictly defined, or where it is strictly defined and costs too much. This paper uses the method named machine learning to solve such problems, which can be improved by obtaining standards or models that conform to the judgments of experts and can also improve the accuracy and the efficiency of the large-scale evaluations. This paper uses quality attributes to evaluate code defects. Code defect is an important aspect of code quality. Since code quality is similar to a doctor's diagnosis, each doctor may evaluate it differently after seeing the same patient. And doctors will use different methods with different conclusions. Another example is the evaluation of film releases. Each different group of people gives different evaluation criteria for watching different movies, and it is also difficult to define personal opinions. There is no unified standard like this, which is relatively subjective and can be understood as a strong personification. If you want to obtain a more recognized model, machine learning algorithms can be used. For unknown results, when judging and evaluating based on existing experience, machine learning models are usually one of the most effective and commonly used methods in today's methods. Training valid data sets, obtaining models that are closer to human standards (reality), resulting in more accurate evaluation results than the analysis of industry insiders, can save costs and improve efficiency when confirming the usability of models.

The traditional way of checking the code is mainly carried out by a manual walk-through. After the machine learning algorithm is used, the marked input and output can be used as the training set, and the model can be called for learning so that the actual y and the proposed y^ tend to be consistent. After the model is determined, the test set can be used for the accuracy judgment, and it can be applied to the evaluation and analysis of all similar projects. The model obtained by using machine learning algorithms will be more efficient and reasonable than expert evaluation or empirical judgment. The follow-up comparison experiments in this paper mainly use the results of quality evaluation using QMOOD and traditional text feature extraction as input and control variables. Then the performance of the trained model is compared to judge the accuracy, comprehensiveness, and usability of the QMOOD quality evaluation data.

4.1 Feature Extraction Method

In machine learning, the input and output need to be determined before training the model. For the data and the source code as the sample, the first thing to do is to extract the features of the code and extract the key features that can represent the code to make the input matrix. This is the first step in machine learning to transform the input matrix. Among them, the commonly used feature extraction method for the source code is TF-IDF scoring. TF-IDF (term frequency-inverse document frequency) is a commonly used weighting technique for information retrieval and text mining. It is a statistical method used to evaluate a word for a document set or the importance of one document in a corpus. The importance of a word increases in proportion to the number of times it appears in the document, but at the same time, it decreases in inverse proportion to the frequency of its appearance in the corpus.
(1) TF is used for the term frequency: it means the frequency of the term (keyword) in the text.

This number is usually normalized (usually the number of occurrences of the word divided by the total number of words in the paper) to prevent it from being biased towards longer documents.

Formula:

$$tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

Ie:

$$\mathrm{TF}_{\omega} = = \frac{\text{The number of occurrences of } \omega \text{ in acertain type of entry}}{\text{The number of entries in this category}} \tag{2}$$

where $n_{i,j}$ is the number of times the word appears in the file d_j, and the denominator is the sum of the number of times all words in the file d_j appearance.
(2) IDF is InverseDocumentFrequency: The IDF of a particular word can be obtained by dividing the total number of documents by the number of documents containing the word and then taking the logarithm of the obtained quotient.

If fewer documents are containing the term t, the larger is the IDF, which means that the term has a good ability to distinguish categories.

Formula:

$$idf_i = \frac{|D|}{\left|\{j: t_i \in d_j\}\right|} \tag{3}$$

Among them, |D| is the total number of files in the corpus. $\left|\{j: t_i \in d_j\}\right|$ Represents the number of files containing the term t_i (ie, the number of files with $n_{i,j} \neq 0$). If the word is not in the corpus, it will cause the denominator to be zero, so in general, use $1+\left|\{j: t_i \in d_j\}\right|$.

Ie:

$$\mathrm{IDF} = = \log\left(\frac{\text{The sum of documents in the corpus}}{\text{The number of documents contain in the term } \omega + 1}\right) \tag{4}$$

The reason for adding 1 to the denominator is to prevent the denominator from being 0;
(3) TF-IDF is actually: TF × IDF.

A high word frequency in a particular document and a low document frequency of the word in the entire document collection can produce a high-weight TF-IDF. Therefore, TF-IDF tends to filter out common words and keep important words.

Formula:

$$\mathrm{TF-IDF} = \mathrm{TF} \times \mathrm{IDF} \tag{5}$$

TF-IDF is often used in machine learning as a feature extraction method of input text for keyword extraction and text summarization. Therefore, in the follow-up experiments, this paper will apply the TF-IDF algorithm to score the source code and convert the data set into an input matrix that can be recognized by the model.

4.2 Machine Learning Model Selection

Existing commonly used machine learning models have been integrated into some very powerful machine learning libraries provided by Python third parties. In the follow-up of this paper, this paper mainly chooses several commonly used models in sklearn [18] for comparison experiments. It covers all aspects from data preprocessing to training models. The actual use of sci-kit-learn can greatly save us time to write authorcode and reduce the amount of our code so that it will have more energy to analyze the data distribution, adjust the model and modify the hyperparameters. There are several commonly used models in sklearn:

(1) SVC: Its full name is SVM for Classification. SVM is a support vector machine (support vector machine), which is a classification algorithm, but regression can also be done. Different models can be made using the input data (if the input label is a continuous value, it is used for regression; if the input label is categorical value, SVC () is used for classification). In seeking to minimize the structured risk, the generalization ability of the learning machine can be improved, and the experience risk and confidence range can be minimized, to achieve the goal of obtaining good statistical laws even with small statistical sample size. Generally speaking, it is a two-class classification model. Its basic model is defined as the linear classifier with the largest interval in the feature space. That is, the learning strategy of the support vector machine is to maximize the interval, which can finally be transformed into a convex quadratic solving planning problem. The SVM algorithm in sklearn is implemented in the package sklearn.svm.

Formula:

$$f(x) = \operatorname{sign}\left(\omega^{*} \cdot x + b^{*}\right) \tag{6}$$

(2) MLPClassifier: Multilayer Perceptron (MLP) is also called Artificial Neural Network (ANN) [18]. In addition to the input and output layers, there can be multiple hidden layers between them. The simplest MLP only contains a hidden layer, which is a three-layer structure, as shown in Fig. 1.

As can be seen from the above figure, the layers of the multi-layer perceptron are fully connected. The bottom layer of the multi-layer perceptron is the input layer, the middle is the hidden layer, and the last is the output layer. The most basic problem to be solved by the neural network is the classification problem. Passing the feature value into the hidden layer and training the parameters of the neural network (*W*, weight; *b*, bias) using the data with the result, so that the output value consistent with the result is given, and it can be used to predict the new. The three-layer MLP mentioned above can be summed up with the formula that the function *G* is softmax:

$$f(x) = G\left(b^{(2)} + W^{(2)}\left(s\left(b^{(1)} + W^{(1)}x\right)\right)\right) \tag{7}$$

Therefore, all the parameters of MLP are the connection weights and biases between each layer, including W^1, b^1, W^2, and b^2;

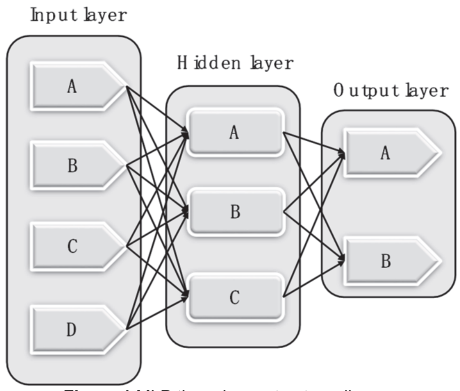

Figure 1 MLP three-layer structure diagram

(3) RandomForestClassifier [19]: Randomforest is a very representative Bagging ensemble algorithm. All its base evaluators are decision trees. The forest composed of classification trees is called Random Forest Classifier, and the forest integrated by regression trees is called Random Forest Regressor. It is a meta-estimator, suitable for multiple decision tree classifiers on each sub-sample of the data set, and uses the average to improve prediction accuracy and control overfitting. The sub-sample size is always the same as the original input sample size, but if bootstrap = True (the default value), the replacement will be used to draw the sample. This algorithm is implemented in the sklearn.ensemble. RandomForestClassifier method in sklearn.

Formula:

$$H(x) = arg \max_{\gamma \in Y} \sum_{t=1}^{T} \prod \left(h_t(x) = \gamma\right) \tag{8}$$

The essence of the random forest algorithm is a classifier ensemble algorithm based on decision trees, in which each tree relies on a random vector. All vectors in the random forest are independent and identically distributed. Random Forest is to randomize the column variables and row observations of the data set, generate multiple classification numbers, and finally summarize the results of the classification tree. Compared with the neural network, it reduces the number of calculations and improves prediction accuracy. The algorithm is not sensitive to multivariate collinearity and is more robust to missing and unbalanced data, and can well adapt to up to thousands of explanatory variables data set.

(4) AdaBoostClassifier: Adaboost is a common boosting learning model. Boosting is a machine learning technology that can be used for regression and classification problems. It generates weak prediction models (such as decision trees) at each step, and the weight is added to the overall model; if the generation of the weak prediction model at each step is based on the gradient of the loss function, then it is called gradient boosting. The significance of this technology is that if a problem has a weak predictive

model, then a strong predictive model can be obtained by upgrading the technology. Adaptive Boosting is just an iterative algorithm. In each iteration, a new learner is generated on the training set, and then the learner is used to predict all samples to evaluate the importance of each sample. The Adaboost algorithm uses the linear combination of the base classifiers as a strong classifier, and at the same time gives the base classifier with a smaller classification error rates a large weight, and gives the base classifier with a larger classification error rate a small weight. The linear combination constructed is:

$$f\left(x\right)=\sum\nolimits_{t=1}^{T}\alpha_t h_t\left(x\right) \tag{9}$$

The final classifier is to perform the sign function conversion based on the linear combination, and get the formula:

$$G\left(x\right)=\operatorname{sign}\left(f\left(x\right)\right)=\operatorname{sign}\left(\sum\nolimits_{t=1}^{T}\alpha_t h_t\left(x\right)\right) \tag{10}$$

QuadraticDiscriminantAnalysis: Discriminant analysis in sklearn mainly includes two types, LinearDiscriminantAnalysis and QuadraticDiscriminantAnalysis. Linear discriminant analysis is a classification model that selects a projection hyperplane in the *k*-dimensional space to make the distance between the projections of different categories on the hyperplane as close as possible, and the distance between the projections of different categories as far as possible. In LDA, it assumes that each category of data obeys Gaussian distribution and has the same covariance matrix ∑. QuadraticDiscriminantAnalysis is similar to LDA. The difference is that it can form a nonlinear boundary and the Gaussian distributions to which different classes belong have different covariance matrices. Therefore, the formula [20] is obtained:

$$\delta_k\left(x\right)=-\frac{1}{2}\log\left|\sum\nolimits_k\right|-\frac{1}{2}\left(x-\mu(k)\right)^{\mathrm{T}}\sum\nolimits_k^{-1}\left(x-\mu(k)\right)+\log\pi_k \tag{11}$$

5 COMPARATIVE EXPERIMENT

5.1 Experimental Design

(1) Experimental method: The experiment in this paper adopts the controlled variable method to conduct comparative experiments, that is, control irrelevant variables to be consistent, reduce the influence of other factors on the results, and finally obtain the performance of the unique variable from the difference of the results.
(2) Irrelevant variables: data set (composed of the sample, training set, and test set), input matrix format (dimension), machine learning model.
(3) Experimental process:

- This paper firstly selects an open-source data set. The data set sample should be the source code and contain the code quality information. Therefore, this paper selects the appropriate part from the defects4j data set used in similar researches as the experimental data set. The data set contains multiple code files written in java language, which meets the requirements of the code measurement tool CKJM. At the same time, TF-IDF can also be used for feature extraction;
- There is 172 pieces of defect information in all samples in the open-source set. Therefore, to balance the positive and negative sets, this paper extracts 172 samples of defect information as positive samples, and randomly selects the other 172 from the remaining samples as negative samples, forming the follow-up data set of the experiment in this paper. Then according to the selection of 6:4, the training set and the test set are allocated.
- Next, according to the script mentioned in section 3.2 of the previous paper. The CKJM tool is used to count the measurement results of all code files in the data set, and convert them into the corresponding quality attributes using the QMOOD model to obtain six quality evaluation results of QMOOD for each sample. Take these six results as the six-dimensional input matrix of each sample, that is, describe the code using these quality attributes.
- Similarly, this paper uses the commonly used feature extraction method TF-IDF to extract all samples in the same data set and convert them into a six-dimensional input matrix as input.
- Put two input matrices of the same format into the same five models for training, and then compare the accuracy of the models using the test set. Finally, a conclusion is drawn by comparing the performance differences of the model with two different input situations.

5.2 Basic Experimental Results

To increase the persuasiveness of the experiment and the amount of experimentation, this paper has selected five additional models including SVC, MLPClassifier, RandomForestClassifier, AdaBoostClassifier, and QuadraticDiscriminantAnalysis in addition to the five commonly used machine learning models studied in Section 4.2. These ten models are commonly used learning models in the sklearn library. The data set used in this paper contains 344 code files, which includes 172 samples of defect information, 206 samples in the training set, and 138 samples in the test set. The input matrices of the two training models are 6 - dimensional, and the QMOOD quality evaluation results and the TF-IDF feature extraction are used for conversion respectively. The performance of the models trained in the two ways is shown in Tab. 3. Because all the results exceed 73 ÷ 138 = 52.9%, it shows that the training of the model is effective.

Table 3 The accuracy of the results obtained by different inputs in the model

Input / Model	QMOOD	TF-IDF
SVC	0.688406	0.666667
MLPClassifier	0.681159	0.659420
RandomForestClassifier	0.768116	0.702899
AdaBoostClassifier	0.775362	0.623188
QuadraticDiscriminantAnalysis	0.666667	0.644928
BaggingClassifier	0.717391	0.710144
KNeighborsClassifier	0.717391	0.666667
PassiveAggressiveClassifier	0.710144	0.586957
Perceptron	0.724638	0.688406
SGDClassifier	0.717391	0.702899

Therefore, it is not difficult to see from the results that QMOOD as input data performs better than the input model obtained by TF-IDF in the 10 models. In addition, the RandomForestClassifier and AdaBoostClassifier models perform more prominently.

At the same time, this experiment is to further study whether the QMOOD quality model can better describe the code and evaluate the defects relative to the measurement results, that is, the effectiveness of the quality attributes. CKJM also explored the statistical results of the unified data set as input and compared the performance of the same data set in 10 models under the QMOOD model evaluation and CKJM measurement statistical results as the input data acquisition method. The results are as follows in Tab. 4. This paper found that the QMOOD evaluation result as an input performs better in the 10 models than the basic metric training result obtained by CKJM.

Table 4 The accuracy of the model trained by CKJM extracting input

Input / Model	CKJM
SVC	0.681159
MLPClassifier	0.666667
RandomForestClassifier	0.724638
AdaBoostClassifier	0.731884
QuadraticDiscriminantAnalysis	0.536232
BaggingClassifier	0.688406
KNeighborsClassifier	0.644928
PassiveAggressiveClassifier	0.579710
Perceptron	0.666667
SGDClassifier	0.695652

Among them, each row under QMOOD, CKJM, and TF-IDF represents the accuracy of the data set using three different methods to obtain the input matrix to train the model. All results are the average of the results of multiple runs to eliminate noise.

5.3 In-Depth Experimental Results

In addition, to give more guidance on the choice of quality assessment methods, this paper added the dimensions extracted by TF-IDF in addition to the above experiment, that is, took the feature words with a wider range of importance rankings in the code, and vertically explored the changes in the accuracy of each model when the dimensionality of the feature matrix was higher.

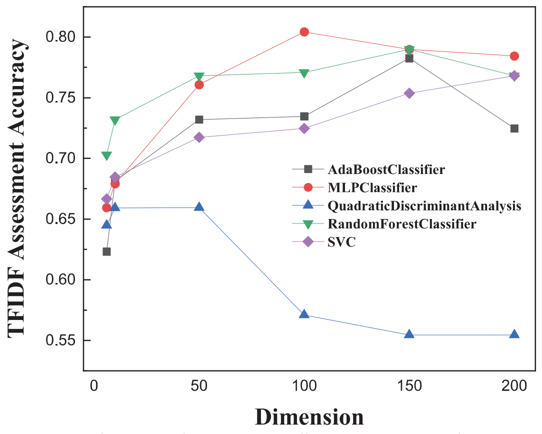

Figure 2 Changes of 5 models in different dimensions of TF-IDF

After the selected dimension of the text feature extraction method becomes higher, it is horizontally compared with the QMOOD quality evaluation results to verify the effectiveness under different extraction conditions, and then provide evidence-based guidance for subsequent people to choose the method when applying.

(1) Longitudinal comparison: This paper selected a different number of text feature words, that is, the performance of training TF-IDF in different dimensions, including 10, 50, 100, 150, and 200 dimensions to conduct experiments, to explore the previous paper. The vertical performance trends of the five key machine learning models obtained are transformed into a line chart as shown in Fig. 2.

It is obvious that the four models include MLPClassifier, RandomForestClassifier, AdaBoostClassifier, and QuadraticDiscriminantAnalysis rises as the dimensionality increases, and then falls after reaching the extreme value. Of course, the extreme values of different models are also different; the SVC model has been rising, obviously under 200 dimensions. The accuracy rate is not the extreme value of this model.

(2) Horizontal comparison: This paper selects QMOOD quality evaluation and TF-IDF text features in the above process and compares the performance in each dimension based on these five machine learning models. Because there are six quality attributes of QMOOD and they will not change with the change of dimensions, then take the data from the previous step, only change the number of feature extractions of TF-IDF. Furthermore, it explores the practicability of the quality assessment model in various situations and the pros and cons of methods based on text features. Compare the histogram tactics of 6, 10, 50, 100, 150, and 200 dimensions respectively as shown in Fig. 3 to Fig. 8.

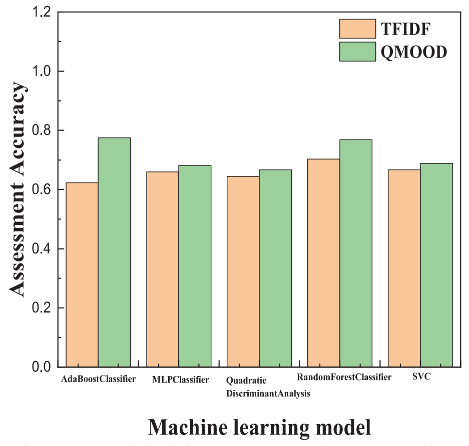

Figure 3 Comparison of QMOOD and TF-IDF of 5 models in 6 dimensions

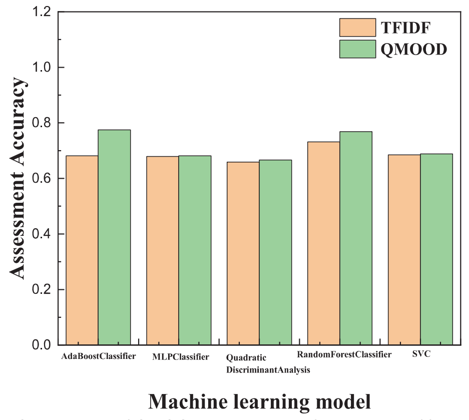

Figure 4 Comparison of QMOOD and TF-IDF of 5 models in 10 dimensions

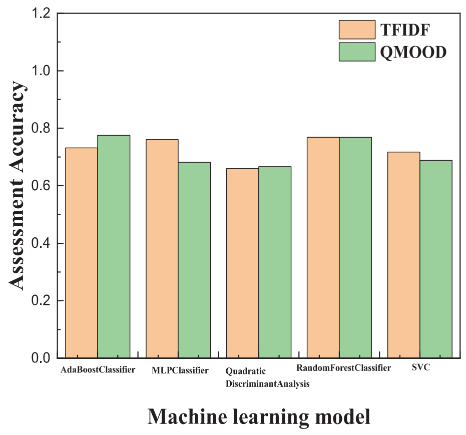

Figure 5 Comparison of QMOOD and TF-IDF of 5 models in 50 dimensions

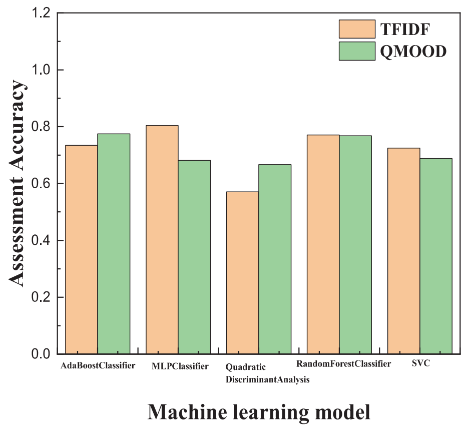

Figure 6 Comparison of QMOOD and TF-IDF of 5 models in 100 dimensions

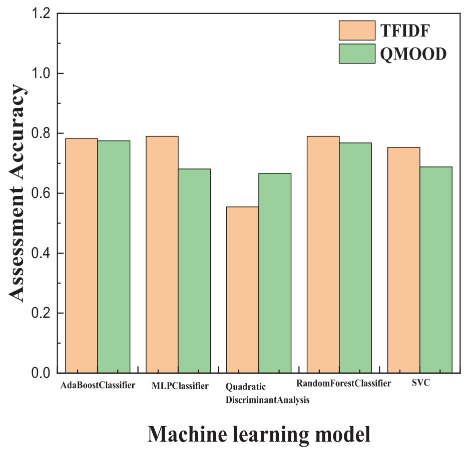

Figure 7 Comparison of QMOOD and TF-IDF of 5 models in 150 dimensions

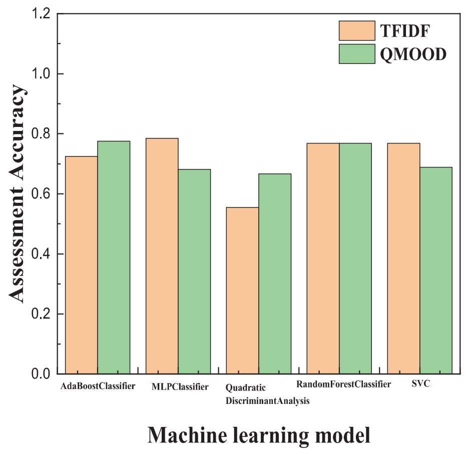

Figure 8 Comparison of QMOOD and TF-IDF of 5 models in 200 dimensions

To verify the reliability of the method in more practical applications and provide better experimental data support, this paper selects another set of data and also conducts experiments following the above process. There are a total of 600 code files in this set of data, including 120 defect information, the experimental results obtained are shown in Fig. 9 to Fig. 15.

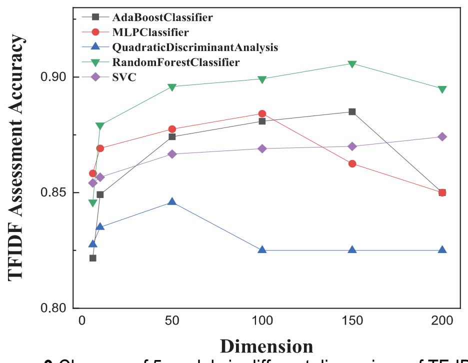

Figure 9 Changes of 5 models in different dimensions of TF-IDF

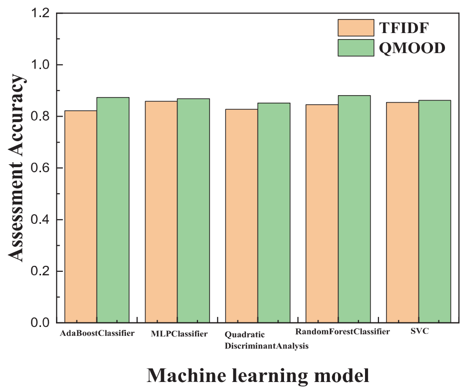

Figure 10 Comparison of QMOOD and TF-IDF of 5 models in 6 dimensions

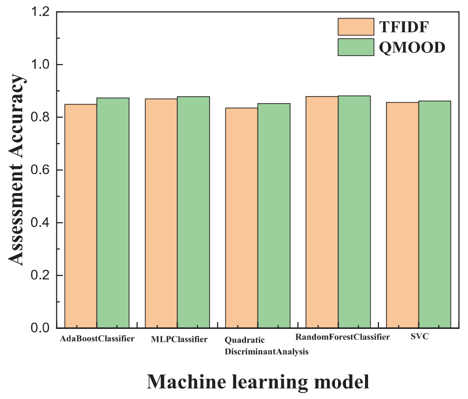

Figure 11 Comparison of QMOOD and TF-IDF of 5 models in 10 dimensions

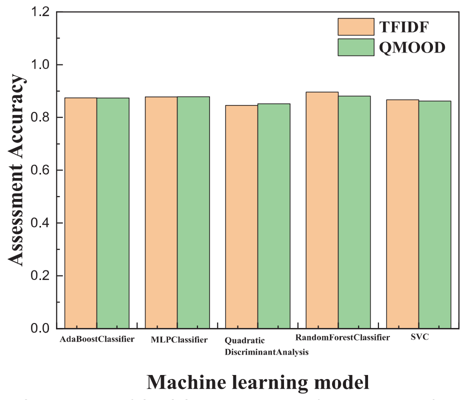

Figure 12 Comparison of QMOOD and TF-IDF of 5 models in 50 dimensions

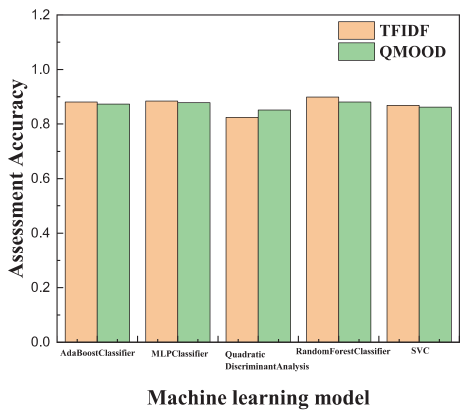

Figure 13 Comparison of QMOOD and TF-IDF of 5 models in 100 dimensions

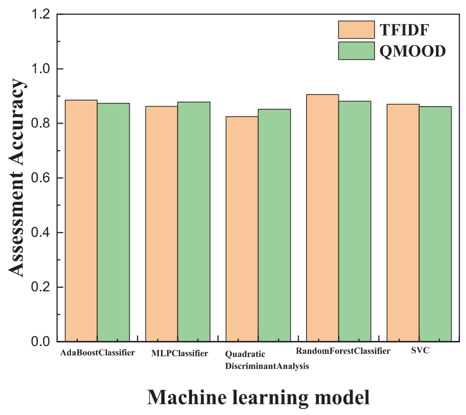

Figure 14 Comparison of QMOOD and TF-IDF of 5 models in 150 dimensions

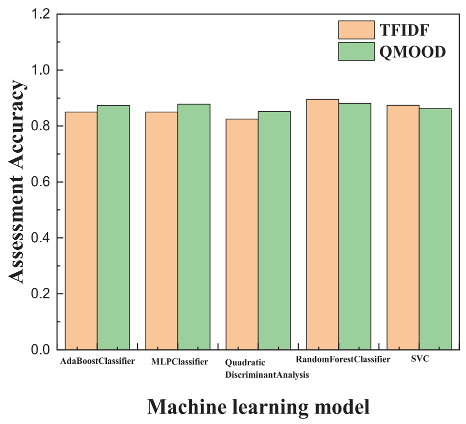

Figure 15 Comparison of QMOOD and TF-IDF of 5 models in 200 dimensions

The author observed that the results obtained by the two data sets are the same. In the control variables, QMOOD performed better than TF-IDF; in the vertical comparison, TF-IDF showed the same changes in the five machine learning models when the dimensionality changed; except for the horizontal comparison, the MLPClassifier model was in the second place. In the results obtained from the group data set, as the dimensionality changes from the extreme value, there is a situation where QMOOD again achieves overtake in high dimensions. It can be seen that the machine learning model of MLPClassifier is unstable when the text features are in high dimensions.

5.4 Experimental Conclusions and Suggestions

According to the experimental results in Section 5.2 and 5.3, it is not difficult to find the following conclusions:

(1) In the basic experiment of control variables, that is, at the same latitude, among the 10 selected models, the model trained with the QMOOD evaluation result as the input performed better than the model obtained by TF-IDF and CKJM. Therefore, this paper can explain that the model obtained when the QMOOD evaluation result is the only variable as the input is better than the model obtained by TF-IDF, and the accuracy of the evaluation is improved by at least 2%. It can be concluded based on machine learning that QMOOD quality attributes can better describe the code itself, can more comprehensively and accurately describe the overall situation of the code, the evaluation results of the quality model are more meaningful for the code. The quality situation is evaluated promptly; and because the model obtained from the QMOOD evaluation result as input is better than the model obtained from the CKJM result input, the accuracy rate is improved. Therefore, the paper can conclude that the quality attributes in the QMOOD model can better describe the code itself compared to the basic metrics. Therefore, the definition of the quality attributes in the model is reasonable and scientific, and at the same time, the quality evaluation of the code is more usable.

(2) In the in-depth experiment, according to the results of machine learning, with the increase of feature words selected by TF-IDF, namely the increase of dimension, the performance of the model is positively correlated with the dimension at the beginning, and some models begin to decline after reaching the extreme value. This is in line with the law of the algorithm itself, that is, as the feature words increase, the feature matrix is more complete, which can better cover the code situation, and then better evaluate the code; but after the dimensionality increases to a certain extent, the feature words only continue to increase. It will increase noise because the feature words extracted by TF-IDF are sorted according to their importance to the code, and the addition of more and more feature words will affect the overall judgment. In the same way, adding more and more representative quality attributes to the quality model and increasing the dimension can also improve the effectiveness of the quality model. This is also a common way to optimize the quality model. This research direction is Scientific and reasonable.

At the same time, after in-depth experiments and based on experimental data, this paper can also put forward some guiding opinions for the subsequent quality evaluation in different situations:

(1) At low dimensions, that is, below 50 dimensions, the introduction of the QMOOD quality model can better evaluate defects compared to conventional text feature extraction methods and can be better combined with machine learning models as an effective feature extraction method. In addition, because QMOOD performs better in the RandomForestClassifier and AdaBoostClassifier models. Compared with other models, the random forest algorithm itself improves the prediction accuracy and stability, and its accuracy rate is improved by the most, reaching over 6%. Therefore, this experiment can see that the QMOOD model can cooperate with better models in the quality assessment based on machine learning to achieve superimposed effects.

(2) Between 50 and 150 dimensions, this paper recommends using high-dimensional text feature

extraction forms when using the four machine learning models of SVC, MLPClassifier, RandomForestClassifier, and AdaBoostClassifier. Quality attributes as input are slightly inferior;

(3) In the 150 - 200 dimension, because AdaBoostClassifier has slipped from the extreme value more obviously, this paper recommends using QMOOD as the feature extraction effect in this case. In both sets of data sets, MLPClassifier has slipped from extreme values in high dimensions, and its performance is unstable. Therefore, it also recommends choosing QMOOD for feature extraction. Under the two models of SVC and RandomForestClassifier, the matching effect of selecting high-dimensional text feature extraction is better;

(4) The QuadraticDiscriminantAnalysis model is more compatible with the QMOOD quality model in any dimension, and it does not need to be considered separately in the application.

6 RELATED WORK

The status of software quality assessment technology in software development has gradually improved, and quality models and quality measurement tools have received more and more attention. Researchers have many commonly used models to measure software quality. Therefore, many times, after the quality evaluation model is proposed, it is necessary to determine whether the quality attribute definition of the existing model is reasonable and effective, whether the result of the quality evaluation is accurate and can fully explain the code situation, and whether it is helpful for the application of code quality evaluation, that is the effectiveness of the quality assessment model.

In the work of predecessors, the verification was mainly conducted among a few software versions, and only the relevance of the quality factors in the software of known quality to the actual quality situation was explored, and the quality evaluation model was not applied to the uncertain quality. In the evaluation of the situation, there is no large amount of data to support, there is lack of comparison with other evaluation methods, and it is impossible to make supporting suggestions for the actual work of quality evaluation in various situations. For example, Chawla MK and Chhabra I proposed an integrated cross-version quality measurement framework in 2016 [11], which is more prominent and novel in such studies in recent years. The exploration illustrates the quality model evaluation results and vulnerability information and changes, the relevance of data such as information. However, it has not directly verified the effectiveness of the proposed quality model in the actual prediction and quality evaluation process for unknown codes, nor has it passed a large number of experiments to compare the pros and cons of other evaluation methods in actual performance. It is based on the concept to verify the relevance of the proposed attributes in the individual software.

In addition, recent research has increasingly applied machine learning algorithms to related research on code evaluation including the code smell prediction technology that has attracted more attention in software quality in recent years. For example, M. Agnihotri and A. Chug in the study from 2020 introduced the use of random forest machine learning algorithms to perform object-oriented software measurement of code smell prediction [21]. Alshaaby A. and others also analyzed the machine learning technology for detecting code smell in 2020 and compared the performance of several algorithms. Among them, J48 and Random Forest algorithm performed better and pointed out that more research is needed to promote the machine. Application of learning algorithm in detecting code smell [22] including a data-driven method using machine learning for vulnerability detection proposed by J. A. Harer et al. in 2018, and a performance comparison [23] can show that the application of machine learning algorithms to the evaluation of code quality is reliable. Predecessors have also proved the effectiveness and feasibility of machine learning algorithms in the field of code quality evaluation, which is also a development trend in recent years.

Therefore, this paper proposes a method for validating the effectiveness of a quality model based on machine learning, which can fill the gaps in the previous research on quality models and can also quantitatively verify the effect of evaluation. Use the difference between quality attributes and other eigenvalues through machine learning, conduct a large number of experiments to explore the performance of attributes proposed by the quality model, and provide a new solution for verifying whether the quality model you want to use or the newly proposed quality model is a more effective plan. It also provides data-supported guidelines for subsequent practical applications based on codes. Appropriate methods can be selected for evaluation in different situations, and a verification and comparison method can also be provided when selecting a quality model.

7 CONCLUSION

In this paper, the focus is on quality models, measurement tools, feature extraction methods, and various machine learning models. In the experiment, comparative experiments are carried out by controlling variables and in-depth experiments are carried out. Finally, two conclusions can be drawn from the experimental results: under the same requirements, compared with the traditional text feature extraction method, the QMOOD quality evaluation model has improved the accuracy of defect prediction by more than 1% in both data sets. The quality attributes definition ratio of the QMOOD quality evaluation model is only based on statistical data. Basic metrics are more meaningful, reasonable, and effective; at the same time, it is verified that the current common way of optimizing quality models is to add more representative quality attributes, that is, to increase dimensions. This research direction is scientific and reasonable, researchers can also use the method proposed in this paper to verify after optimization. In addition, this paper is also a guide in different situations for subsequent practical applications.

Therefore, according to the research in this paper and the above experimental conclusions, the verification method proposed in this paper can well illustrate the usability and effectiveness of the quality assessment model. In subsequent research, by combining the quality model with better machine learning models, or even deep learning models, it can be better applied to the evaluation technology of code quality information such as defects. The machine learning-based quality evaluation attribute validity verification method proposed in this paper verifies the performance of the quality evaluation model in the actual application of quality evaluation. It can quantitatively evaluate the validity of the quality model and give the application of experimenters. The guidelines fill the gap here and provide directions and powerful tools for future research. This paper hopes to provide some verifiable methods for the subsequent research on quality model optimization, so that the quality assessment applied to more links in the software testing process can be more in line with expectations, to help existing research achieve better results.

Acknowledgment

This paper is supported by Guangxi Key Laboratory of Cryptography and Information Security(No.GCIS202103).

8 REFERENCES

[1] Yunzhan, G. (2019). Who will guarantee the software quality? *Innovation World Weekly*, *2019*(05), 8+24-25.
[2] Walia, M. (2010). Realizing efficiency and effectiveness in software testing through a comprehensive metrics model. *10th Annual International Software Testing Conference (STC 2010)*, Infosys, White Paper, Infosys Technologies Ltd.
[3] Australia S. (2001). Software engineering - Product quality-P. 1: Quality model. International Standard ISO/IEC 9126-1.
[4] McCall, J., Richards, P., & Walters, G. (1977). Factors in software quality. *NTIS* AD-A049-014, 015, 055, 1977.
[5] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., & Merritt, M. (1978). Characteristics of software quality.
[6] Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, *21*, 146-162. https://doi.org/10.1109/32.345830
[7] ISO (2005). Software engineering Software product Quality Requirements and Evaluation (SQuaRE) Guide to SQuaRE, ISO Geneva.
[8] Jagdish, B. & Carl, G. D. (2002). A hierarchical model for object-oriented design quality assessment, Software Engineering. *IEEE Transactions on*, *28*(1), 4-17. https://doi.org/10.1109/32.979986
[9] Spinellis, D., Gousios, G., Samoladas, I., et al. The SQO-OSS Quality Model: Measurement-Based Open Source Software Evaluation// 2008:2101-7.
[10] Kläs, M., Lochmann, K., & Heinemann, L. (2011). Evaluating a Quality Model for Software Product Assessment-A Case Study. *Proceedings of SQMB'11*.
[11] Chawla, M. K., & Chhabra, I. (2016). A quantitative framework for integrated software quality measurement in multi-versions systems. *International Conference on Internet of Things & Applications. IEEE*. https://doi.org/10.1109/IOTA.2016.7562743
[12] Jung, H. W. (2007). Validating the external quality sub characteristics of software products according to ISO/IEC 9126. *Computer Standards & Interfaces*, *29*(6), 653-661. https://doi.org/10.1016/j.csi.2007.03.004
[13] Correia, J. P., Kanellopoulos, Y., & Visser, J. (2009). A survey-based study of the mapping of system properties to ISO/IEC 9126 maintainability characteristics, Software Maintenance. *ICSM 2009. IEEE International Conference on, IEEE*, 61-70. https://doi.org/10.1109/ICSM.2009.5306346
[14] Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, *20*(2), 287-307. https://doi.org/10.1007/s11219-011-9144-9
[15] Barney, S., Petersen, K., Svahnberg, M., Aurum, A., & Barney, H. (2012). Software quality trade-offs: A systematic map. *Information and Software Technology*, *54*(7), 651-662. https://doi.org/10.1016/j.infsof.2012.01.008
[16] Spinellis, D. (2005). Tool Writing: A Forgotten Art? *IEEE Software*, *22*(4), 9-11. https://doi.org/10.1109/MS.2005.111
[17] Chawla, M. K. & Chhabra, I. (2014). Implementation of an object-oriented model to analyze relative progression of source code versions with respect to software quality. *International Journal of Computer Applications*, *107*(10). https://doi.org/10.5120/18790-0126
[18] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2012). Scikit-learn: Machine Learning in Python.
[19] Zhihua, Z. & Wang, J. (2009). Machine Learning and Its Application 2009. *Tsinghua University Press*.
[20] Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, *27*(2), 83-85. https://doi.org/10.1007/BF02985802
[21] Agnihotri, M. & Chug, A. (2020). Application of machine learning algorithms for code smell prediction using object-oriented software metrics. *Journal of Statistics and Management Systems*, *23*(7), 1159-1171. https://doi.org/10.1080/09720510.2020.1799576
[22] Alshaaby, A., Aljamaan, H., Alshayeb, M. (2020). Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review. *Arabian Journal for Science and Engineering. Section A, Sciences*, *45*(4), 2341-2369. https://doi.org/10.1007/s13369-019-04311-w
[23] Harer, J. A., Kim, L. Y., Russell, R. L., et al. (2018). Automated software vulnerability detection with machine learning.

Contact information:

Tianze GUO, postgraduate
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
No. 10, Xitucheng Road, Haidian District, Beijing
E-mail: 2019111420gtz@bupt.edu.cn

Hanli BAI, Senior Engineer
(Corresponding author)
Institute of Computational Aerodynamics, China Aerodynamics Research and Development Center,
No.6, South Section,Second Ring Road, Mianyang City, Sichuan Province,P.R. China
E-mail: starworks@cardc.cn

Yunzhan GONG, professor
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
No. 10, Xitucheng Road, Haidian District, Beijing
E-mail: gongyz@bupt.edu.cn

Yawen WANG, Associate Professor
1. State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
No. 10, Xitucheng Road, Haidian District, Beijing
2. Guangxi Key Laboratory of Cryptography and Information Security,
Guilin, Guangxi 541004, China
E-mail: wangyawen@bupt.edu.cn

Dahai JIN, Associate Professor
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
No. 10, Xitucheng Road, Haidian District, Beijing
E-mail: jindh@bupt.edu.cn