2018 IEEE International Conference on Software Quality, Reliability and Security Companion

Defect Prediction Based on The Characteristics of Multilayer Structure of Software Network

Yiwen Yang
School of Reliability and System Engineering
Beihang University
Beijing, China
yangyiwen78@buaa.edu.cn

Jun Ai
School of Reliability and System Engineering
Beihang University
Beijing, China
aijun@buaa.edu.cn

Fei Wang
School of Reliability and System Engineering
Beihang University
Beijing, China
wangfeiabn@buaa.edu.cn

***Abstract*—Software defect prediction can help us identify software defect modules and improve software quality. The existing defect prediction mainly analyzes the software code or the development process and uses the statistical feature data on the files or categories related to the software defects as the metrics. The method disregards the macroscopic integrity of software programs and the relevance of local defects to the surrounding program elements. For this reason, this study introduces a complex network technology into defect prediction, establishes a software network model, uses a complex network metric to design a set of metrics that can reflect the local and global features of defects, and proposes a dynamic prediction model optimization method based on the threshold filter algorithm. The effectiveness of the proposed metric and method is verified through comparison with the Predictive Model in Software Engineering dataset experiment and a practical engineering software data prediction experiment.**

Keywords—defect prediction, software metrics, complex network, machine learning

I. INTRODUCTION

Efficient and secure software systems highly depend on software quality. Software defects not only affect software quality, but also are the root cause of system errors, failures, and crashes [1]. As already defined in ISO 9000, a software defect means "a failure to meet the requirement that is related to the intended or specified use". The defect mentioned in this article is failure. How to find defects in software early and ensure software quality has become a crucial research direction.

Metrics that have strong correlations with software defects are the key to software defect prediction. However, from existing research, most of the metrics currently used are still using the statistical characteristics of documents or class granularity, such as the number, proportion, median, mean, variance, skewness, variation of a certain feature, and coefficient. The essence of this design is to discretize software program elements from the perspective of set theory and to predict defects from the perspective of local data value analysis and processing. This design disregards the macroscopical integrity of software program and destroys the mutual influence among the software program elements. Some flaw prediction methods work efficiently for National Aeronautics and Space Administration (NASA) or Predictive Model in Software Engineering (PROMISE) datasets but are ineffective in actual software projects.

To solve this problem, this study introduces a complex network technology into defect prediction, builds an object-oriented software network based on the relationship among software elements and their evolutionary relationships, and build many metrics to macroscopically describe the software program. The detailly information about software network is mentioned in background. A software network defect prediction model is established with seven different machine-learning algorithms to identify and predict the software defect modules. Each version of the actual software project comprises few defective nodes and many non-defective nodes. A predictive model optimization algorithm based on dynamic prediction threshold filtering algorithm is proposed in this study to build an efficient defect prediction model.

The remainder of this paper is organized as follows. Section II presents the background regarding software network, design of defect prediction metrics, and machine-learning algorithm. Section III indicates the metrics and algorithms. Section IV provides the experimental setup and result analysis. Section V concludes and states the future work.

II. BACKGROUND

A. Software Network

A software system comprises many entities and elements, such as classes, functions, and variables. Therefore, a software system can be transformed into a network system if we regard its elements as nodes and relationships as edges in the network system.

B. Design of Defect Prediction Metrics

The metrics for early defect prediction are focused on software scale, and the relationship between defects and basic attributes, such as software size and complexity, to predict the number of possible defects in the software. The core of the idea is to assume that the internal attributes of software (such as static code features) are related to its external performance (such as faults). The higher the complexity of software modules, the more defects they may contain [2]. Representative metrics include lines of code (LOCs), McCabe, and Halstead. With the popularity of object-oriented methods, several object-oriented program metrics have increased applications. D. Radjenović et al. summarized and analyzed the metric parameters used in the literature related to 106 defect predictions from 1991 to 2011. Forty-nine percent of the documents used object-oriented

978-1-5386-7839-8/18/$31.00 ©2018 IEEE
DOI 10.1109/QRS-C.2018.00019

metric meta-parameters, and 27% of the literature used traditional metric parameters [3] The most typical object-oriented models are Chidamber–Kemerer (CK) metrics and metrics for object-oriented design, followed by the model of Basili et al. [4] that is based on medium-sized information management systems to verify the correlation between CK metrics and defects in program modules. Man [5] found that class-scale metrics have potential mixed effects in analysis and exert impacts on the performance of defect prediction models. A linear regression method was proposed to remove the mixed effects.

Some research on software networks has proven that some software network characteristics have direct or indirect relationships with software quality. Vasa et al. [6] studied structural changes with the relationship between the number of software nodes and the number of edges to predict software size and cost. Ma et al. [7] defined network structure entropy to characterize and evaluate network complexity. Valverde et al. [8] and Ma et al. [9] studied characteristic subgraphs with high frequency in network and found that subgraphs without loops have high stability. Y. Yang, J. Ai et al. [10] used 12 characteristic parameters of a complex network to comprehensively measure the modularity, hierarchy, complexity, and vulnerability of software and discussed the correlation with software quality. Liu et al.

The topology features of many software networks are closely related to defects and can be used as metrics for software defect prediction. For example, Zimmermann et al. [11] abstracted a software system as a software dependency graph and built a defect prediction model around graph metrics. Bhattacharya et al. [12] established a software network metric set by using software network codes and related metrics in the software development process and tried to predict the severity of software failures, failure propensity, and other related information through some open-source software projects. S. Zhang, J. Ai et al. [13] studied the correlation between software network models and software bugs and indicated that defects are likely to appear in the nodes of feed-forward loop modules. Qu et al. [13] proposed a measure of the coupling relationship among software classes by classifying software networks and applied them to 10 large-scale Java open-source projects. The conclusions confirmed the measurement of the software community structure and its derived measures. The coupling relationship among software classes is significant. These metric parameters can be applied to software fault prediction, and their prediction is more effective than that of traditional metric parameters.

C. Machine-learning Algorithm

With the differences in the targets of defect prediction, the machine-learning methods used are different. If the program module is set to fine grain (for example, class level or file level), then the defect orientation of the predictive module is targeted, often using classification methods, including logistic regression (LR) [15], Bayesian networks [16], and decision-making tree [17]. If the module is set to coarse grain (for example, packet level or subsystem level), then the number of defects or defects in the prediction module is the target, often using a regression analysis method [18]. In addition to commonly used learning methods, active learning [19] and semi-supervised learning [20] are also gradually applied.

Elish et al. [21] systematically compared support vector machines (SVMs) with other eight machine-learning methods [LR, K nearest neighbor (KNN), multilayer perceptron, radial basis function, Bayesian belief network, naive Bayes (NB), random forest (RF), and decision tree (DT)] with NASA dataset and considered that the SVMs were generally better than the other eight methods. Catal and Diri [22] based on NASA dataset, used area under the curve (AUC) values as evaluation indicators, and deeply analyzed the effects of dataset size, metrics, and feature subset selection methods on the performance of defect prediction models. Results showed that the RF method exhibited the best performance on large-scale datasets, whereas NB had the best performance on small-scale datasets. Lessmann et al. [23] based on the NASA dataset, used AUC values as evaluation indicators, and systematically compared six categories (statistical methods, nearest neighbor methods, neural network methods, SVMs, DT methods, and integration methods). For different machine-learning methods, the performance difference among the optimal 17 machine-learning methods is insignificant. Ghotra et al. [24] found significant performance differences among different machine-learning methods after removing noise from NASA and PROMISE datasets. Shepperd et al. [25] used random-effect ANOVA to analyze the influencing factors that affected the performance of defect prediction models and found that the choice of machine-learning methods had no significant effect on performance, but significant differences existed among the different research groups. Panichella A [26] and others used genetic algorithms (GAs) to train predictive models. The research showed that the regression model trained by GAs was significantly better than traditional opponents. Similarly, another study [27] used GA to help generate test cases for Software Product Lines.

III. Model Construction Technology

In this part, we introduce the construction of software networks, the design of multilayer structure feature metrics, data-preprocessing and machine-learning algorithms, cross-validation methods, and prediction model optimization methods based on dynamic prediction threshold filtering algorithms.

A. Software Class Network Construction

In this study, class network is used for defect prediction research. The classes in a software source program are abstracted into nodes in the network. The number of functions inside the source program class and the connection relationship among the functions are abstracted into the internal attributes of the nodes in the network. The calling relations among program classes are abstracted into edges in the network $e_{ij} = < v_i, v_j > (i, j = 1, 2, ..., n)$. Therefore, a software version can be represented as a class network G in the form of $G = (V, E)$, where $V = (v_1, v_2, ..., v_n)$ represents the node set of class network, $E = (e_{ij} \mid i, j = 1, 2, ..., n)$ represents the set of directed edges, and n represents the total number of nodes in the network.

B. Design of Multilayer Structural Feature Metrics

In this study, the design of metrics mainly focuses on the internal, local, and global features of nodes in the network. The following is a detailed introduction.

1) Internal Feature Metrics

- *Funcount* for a node is the number of internal functions of the class node.
- *Indegree* for a node is the total number of connections it points to other nodes.
- *Outdegree* for a node is the total number of connections other nodes point to it.
- *Insidelinks* are the total number of connections within the internal functions of the node.
- *Bug* represents whether the *node is defective.*

Funcount, *Indegree*, *Outdegree*, and *Insidelinks* represent the internal structural parameters of each node in the software network. The larger the value is, the more important this node is in the network. This micro-level measurement parameter is useful in defect prediction.

2) Global Feature Metrics

- *Out_degree_centrality* for a node is the fraction of nodes its outgoing edges are connected to.
- *In_degree_centrality* for a node is the fraction of nodes its incoming edges are connected to.
- *Degree_centrality* for a node is the fraction of nodes it is connected to. The value can be calculated as

$$C_D(i) = \frac{\sum_{j=1}^{g} x_{ij} (i \neq j)}{g-1}, \quad (1)$$

Where g is the total number of nodes, $C_D(i)$ is the degree_centrality of node i, and x_{ij} is used to calculate the number of direct connections between node i and node j.

The above three parameters all belong to centrality, which is the most direct measure of node's centrality in the network analysis. The higher the centrality value of a node is, the more important the node is in the network.

- *Closeness_centrality* for a node v is the reciprocal of the sum of the shortest path distances from v to all other nodes. This measure reflects the degree of closeness between a node and other nodes in the network. The larger the value is, the closer the node is to the network center, and the faster it can reach other points. The value can be calculated as

$$C_v = \frac{|V|-1}{\sum_{i \neq v} d_{vi}}, \quad (2)$$

where V is the total sum of nodes in the network, and d_{vi} is the distance between nodes v and i.

- *Betweenness_centrality* for a node v is the sum of the fraction of all-pair shortest paths that pass through v. This measure emphasizes the ability to adjust between nodes and other nodes. If many shortest paths pass the node, then the node is important. The value can be calculated as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (3)$$

where σ_{st} is the number of shortest paths between nodes s and v, and $\sigma_{st}(v)$ is the number of shortest path through node v.

- *Eccentricity* of a node v is the maximum distance from v to all other nodes in G. The smaller the value is, the closer the node is to the network center.
- *Communicability_centrality* is a broader measure of connectivity, which assumes that information could flow along all possible paths between two nodes.
- *Katz_centrality* computes the relative influence of a node within a network by measuring the number of the immediate neighbors (first-degree nodes) and all other nodes in the network that connect to the node under consideration through these immediate neighbors.
- *Load_centrality* of a node v is the fraction of all shortest paths that pass through that node.
- *PageRank* computes a ranking of the nodes in the graph G based on the structure of the incoming links. The larger the value is, the more popular the node is, and the easier it is to connect.

3) Local Feature Metrics

- *Average_neighbor_degree* is the average of the neighborhood of each node. The value reflects the situation of the neighboring nodes for each node and can indicate the characteristics of the local network. The value can be calculated as

$$k_{nn,i} = \frac{1}{|N(i)|} \sum_{j \in N(i)} k_j, \quad (4)$$

where $N(i)$ are the neighbors of node i, and k_j is the degree of node j that belongs to $N(i)$.

- *Clustering* for a node is the fraction of possible triangles through that node that exists. This metric is a static statistical feature in a complex network. The larger the value is, the more influential the node is. The value can be calculated as

$$C_u = \frac{2T(u)}{\deg(u)(\deg(u)-1)}, \quad (5)$$

where $T(u)$ is the number of triangles through node u, and $\deg(u)$ is the degree of u.

- *Number_of_cliques* is the number of maximal cliques for each node.
- *Core_number* of a node is the largest value k of a k-core containing that node.

C. Dataset-preprocessing Method

The above metrics imply that the defect prediction dataset is nearly 20D, and many numerical differences exist among different metric categories. This study mainly adopts the method of maximum minimization and principal component analysis (PCA) dimension reduction to process data to establish an efficient prediction model.

1) Maximum Minimization Method

This study adopts the maximum minimization method to ensure that the size of each parameter is in the interval [0,1] to eliminate the influences of different data values of different categories on the defect prediction. The specific formula is as follows:

$$y = \frac{x - MinValue}{MaxValue - MinValue}, \quad (6)$$

where x and y are the values before and after parameter conversion, respectively. $MaxValue$ and $MinValue$ are the maximum and minimum values of the sample, respectively.

2) Principal Component Analysis

PCA is a method used to analyze data in multivariate statistical analysis. This method uses a small number of features to describe the sample to reduce the dimension of the feature space. The core idea of the PCA algorithm is to map n-dimensional features to k-dimensional (k<n), which is a linear combination of n-dimensional old features. These linear combinations maximize the sample variance and try to make the new k-dimensional features mutually uncorrelated. The sample data processed by the PCA algorithm not only reduces the data dimension and the training time of the predictive model, but also removes noise and finds patterns in the data.

We set the parameter n_components of the PCA algorithm to 0.95, so that the PCA algorithm determines the number of dimensions to be reduced according to the sample data feature variance and the variance and the minimum proportion threshold of the principal component feature are not lower than 0.95. We first use the training data to establish the PCA model and then use this model to perform the same processing on the training and test data to ensure the consistency of the training and test data feature parameters.

D. Machine-learning Algorithm

- Linear discriminant analysis (LDA) is a classification model that selects a projection hyperplane in the k-dimensional space, so that the distance among the projections of different classes on the hyperplane is as close as possible, whereas the distance among the projections of different classes is as far as possible. In LDA, each category of data is Gaussian and has the same covariance matrix
- The core idea of the KNN algorithm is a supervised classification algorithm that determines its own category based on the KNN classes. The idea is simple and easy to understand and implement. If the K value is set too small, then it will reduce the classification accuracy. If the setting is too large and the test sample belongs to a class with minimal data in the training set, then it will increase the noise and reduce the classification effect. After the experiment, the K value is set to 3 in this study. The best classification is achieved.
- The DT algorithm is a method of approaching discrete function values. This method processes data, uses an induction algorithm to generate readable rules and DTs, and uses the decision to analyze the new data. The internal implementation of the DT classifier algorithm used in this study adopts the classification and regression tree (CART) algorithm, which has been optimized. The model established by this algorithm is readable, and the classification speed is fast.
- The main idea of the LR algorithm is to learn fitting parameters from the sample set, establish a regression formula for the classification boundary line, fit the target value to [0,1], and discretize the target value to achieve classification. In this thesis, the penalty parameter is set to L2, and the C value is set to 1.0. This setting simplifies the model, prevents over-fitting, and gives high generalization ability.
- RF algorithm is an algorithm that integrates multiple trees through the idea of ensemble learning. Its basic unit is the DT, and its essence belongs to the ensemble learning method. In this dissertation, the parameter n_estimator is set to 10, and the parameter criterion is set to the Gini coefficient (gini). By using this method, the established model deviation will be small and the accuracy will be high.
- The SVM algorithm is a classifier defined by the classification hyperplane. For a given set of labeled training samples, the algorithm will output an optimal hyperplane to classify the test samples. The support vector clustering (SVC) algorithm is used in this thesis. The parameter C is set to 1.0. This setting will increase the penalty for the wrong sample and increase the accuracy of the training sample.
- Adaboost (AB) algorithm is an iterative algorithm with a core idea to train different weak classifiers for the same training set and then group these weak classifiers to form a strong classifier. This study uses the AB classifier algorithm. The base_estimator parameter is set to the CART classification tree, and the parameter algorithm is set to SAMME.R; therefore, the iteration speed will be fast, and the classification effect will be significant.

E. Cross-validation

Cross-validation is mainly used in modeling applications, such as principal component regression (PCR) and partial least square (PLS) regression modeling. The basic idea of cross-validation is to group datasets in a certain sense using one part as a training set and the other as a validation set or test set. The training set is first used. The classifier trains and then uses the validation set to test the trained model as a performance indicator for evaluating the classifier. The purpose of cross-validation is to obtain a reliable and stable model. An important factor in establishing a PCR or PLS model is the question of how many principal components to take. The cross-validation is used to verify the PRESS value of each principal component, and the number of principal components with a small PRESS value is selected or the number of principal components where the PRESS value no longer decreases.

Each software version in the PROMISE public dataset is small, and the sample size is insufficient. Thus, this study adopts a 10-fold cross-validation method to randomly divide the dataset into 10 parts, one of which is one at a time. As a test set, the remaining nine copies are used as a training set for training. This method not only completes 10-fold cross-validation, but also handles stratified sampling, thereby ensuring that the proportion of samples in each category of the training and test sets is the same as that of the original dataset.

F. Optimal Prediction Model Based on Dynamic Predictive Threshold Filtering Algorithm

In actual engineering projects, a new version of software is often released soon, and the time interval between the new version and the old version is short. With the file changes submitted at the time of the release of the new version, we determine the class file that is defective in the previous version. However, the changes between adjacent versions are usually insignificant. A class network composed of an already released software version may have hundreds or thousands of nodes, but only a few of them are defective nodes. In defect prediction, if all versions of the node set other than the predicted version are used as the training set, and all the sets of the predicted version are used as the prediction set, the training set not only has much difference between the defect and non-defect data, but also many of the data in the training set are redundant and uncertain. Such data cannot be directly used to construct a defect prediction model.

Therefore, aiming at the above problems, this study proposes a prediction model optimization method based on dynamic prediction threshold filtering algorithm. This method implements optimization of training datasets, threshold values, and prediction models. (1) Optimization of training set. The method will first consider all the defective nodes from the input training version network set to join the training sets, then will select the nodes that exist in all versions of the network with no defects to join the training set, and finally dynamically adjust the ratio of defective nodes and non-defective nodes in the training set. (2) Optimization of thresholds. This method will select n appropriate sets from the input training sets as the validation sets to determine the optimal thresholds for different machine-learning algorithms. (3) Optimization of the prediction model. According to the optimized training set and threshold, this method uses different machine-learning algorithms to establish a defect prediction model and selects the best defect prediction model suitable for the test set. The specific algorithm steps are as follows:

Algorithm: Optimal Prediction Model Based on Dynamic Predictive Threshold Filtering Algorithm

Input:

Training version network sets $L =< G_1, G_2, ..., G_n >$, Testing version network set G_t

Process:

Step 1: Traverse training set L, if G_i have defect nodes, then G_i is added to new empty set L_1; otherwise, G_i is added to new empty set L_2.

Step 2: L_1 is sorted from the smallest version to the largest. The last n number (the default value of n is 3) G is selected and integrated into the new verification version network set V. The remaining G and L_2 are integrated a new set H.

Step 3: Traverse training set H, if G_i have defect nodes, then add the nodes data into a new training set T, record the number of nodes in T set as *count*.

Step 4: From H, the largest version G' is selected. Selecting some nodes from G' join T which must meet the following two conditions: (1) the node ID must exist in each network G of set H. (2) the node number cannot exceed $m*count$ (the default value of m is 4).

Step 5: All node data of T are used as a training set, and every V_i of V is used as test set. Multiple machine-learning algorithms are adopted for prediction, and AUC is employed as a model evaluation index. Threshold is constantly changed to record threshold when n verification set AUC reaches the maximum value.

Step 6: The threshold of the minimum AUC of the verification set is selected as the threshold of the test set, and the data of the entire network node of the test set are used as the test set. All node data in the training set T are taken as the training set. Different machine-learning algorithms are utilized to predict.

Output:

Defective node id in the test set and the evaluation value of the best defection model

IV. Experiments and Discussion

A. Experimental Settings

This study uses the dataset published by PROMISE to conduct related experiments to verify the effectiveness of the designed multilayer structural feature metrics in defect prediction. We collect datasets, such as Poi, Velocity, Xalan, and Xerces from Tera-PROMISE (http://openscience.us/repo/index.html)data repository .

The specific information is shown in TABLE I. The above datasets contain 20 attributes, mainly including 6 basic CK metric parameters about the class (such as weighted method per class, depth of inheritance tree, and number of children) and other metric parameters for the class (such as LOC, measure of functional abstraction, cohesion among methods of class).

We collect 154 versions of the defect information from the v.1.1 version to the 2.9.30 version of software named BaseRecyclerViewAdapterHelper on Github to verify the validity of the optimal prediction model based on dynamic predictive threshold filtering algorithm.

We collect sufficient data and then begin to apply the data in a single version of the defect cross-validation, multilayer structural feature metrics, and CK metrics for defect effect comparison experiments of the optimal prediction model based on dynamic predictive threshold filtering algorithm. We evaluate the performance of all methods in terms of precision (P), recall (R), F-measure (F), and AUC, which are defined as follows:

$$P = \frac{tp}{tp + fp}, \tag{7}$$

$$R = \frac{tp}{tp + fn}, \tag{8}$$

$$F = \frac{2PR}{P + R}, \tag{9}$$

where tp, fp, and fn are the number of defective modules that are predicted as defective, the number of non-defective modules that are predicted as defective, and the number of

defective modules that are predicted as non-defective, respectively. AUC is the area under the ROC curve, which is a good combination of P, R, and different classification thresholds.

TABLE I. SUMMARY OF DATASETS

Datasets	Instances	Defective modules
Poi_1.5.0	212	130(61.30%)
Poi_2.0	287	36(12.50%)
Poi_2.5.1	348	232(66.66%)
Poi_3.0	389	269(69.10%)
Velocity_1.4	178	134(75.28%)
Velocity_1.5	191	127(66.49%)
Velocity_1.6.1	204	73(35.78%)
Xalan_2_4_0	398	66(16.58%)
Xalan_2_5_0	395	204(51.64%)
Xerces_1_2_0	299	51(17.05%)
Xerces_1_3_0	314	65(20.70%)
Xerces_1_4_4	173	134(77.45%)

B. Cross-validation within versions

We use a randomly sampled K-fold cross-validation method to conduct experiments to evaluate our established defect prediction model based on multilayer structural feature metrics. For each dataset, we set the sampling rate to 10% and averaged the results of 10 cross-validations as the final results.

In this experiment, we adopt seven widely used machine learning algorithms, namely, LDA, KNN, DT, LR, RF, SVC, and AB. We compare the performances of all classifier models in terms of precision, recall, and F-measure.

The average precision results are shown in TABLE II. The average recall results are shown in TABLE III. The average fscore results are shown in TABLE IV.

With the average of precision, recall, and fscore obtained from 10-fold cross-validation with 10 random samples, the prediction results obtained by using different machine-learning algorithms for different datasets have different prediction results. However, the prediction results are similar. In general, under different datasets, the precision of different algorithms is roughly stable between 0.7 and 0.8, the recall is roughly stable between 0.8 and 0.9, and the fscore value is stable between 0.7 and 0.8. These established defect prediction models are valid based on multilayer structural feature metrics.

C. Defect effect comparison experiments of multilayer structural feature and CK metrics

The performances of multilayer structural feature and CK metrics for defect prediction among releases are compared. We intercept the data of the two common classes to compare, so that the two are the same, except for different metric parameters, which effectively control the variables.

In this experiment, we adopt seven widely used machine-learning algorithms, namely, LDA, KNN, DT, LR, RF, SVC, and AB. We compare the performances of all classifier models in terms of precision. The details of the experimental results are shown in TABLE V. and TABLE VI.

The analysis of the experimental results in TABLE V. and TABLE VI. demonstrate that the model obtained by applying the multilayer metrics is totally better than that of the model established by the CK metric parameters on the Poi dataset. On other datasets, the model obtained by applying the multilayer metrics is partly better or worse than the model established by the CK metric parameters. In general, the model obtained using multilayer structural feature metrics is effective in defect prediction.

D. Experiment of the Optimal Prediction Model Based on Dynamic Predictive Threshold Filtering Algorithm

We collect the defect information of the software BaseRecyclerViewAdapterHelper from v.1.1 to 2.9.30, a total of 154 versions, and construct each version of the software network to verify the ability to predict the defect of the optimal prediction model based on dynamic predictive threshold filtering algorithm in an actual engineering project.

We use the last continuous versions to conduct experiments. The details of the experimental results are shown in TABLE VII.

When we use 2.9.10-2.9.29 as the training version and 2.9.30 as the prediction version, we can acquire the following experimental result. For the datasets, the best algorithm is SVC, the AUC value of the best defect model is 0.71, and the number of predicting defect nodes is 8. In contrast to the collection of defect information, eight nodes are indeed a defect node. In the version of 2.9.30, the total number of defect node is 1. Hence, the defect prediction model established by our method accurately predicts all defect nodes in this version.

Our approach does not always predict all the defect nodes in the version. When we use 2.9.7-2.9.26 as the training version and 2.9.30 as the prediction version to conduct experiment, the results show that we only predict correctly a defect node, but this version has two defect nodes. The method for optimal prediction model based on dynamic predictive threshold filtering algorithm is generally effective in defect prediction.

V. CONCLUSION

In this study, we use a complex network technology to design a set of multilayer structural feature metrics that not only reflects the local characteristics of software, but also shows the overall characteristics of the software. We use various algorithms, such as LR, DTs, and RFs, to construct a software defect prediction model. In the defect effect comparison experiments of a single version of the defect cross-validation and multilayer structural feature and CK metrics, we confirm the effectiveness of multilayer structural feature metrics in defect prediction. In addition, we propose the poptimal prediction model based on dynamic predictive threshold filtering algorithm for the problems encountered in actual engineering projects in defect prediction. We use an optimized training set, thresholds, and machine-learning algorithms to construct the best defect prediction model for the datasets. We prove the effectiveness of the method through experiments.

TABLE II. AVERAGE PRECISION OF THE CROSS-VALIDATION IN VERSION

	LDA	KNN	DT	LR	RF	SVC	AB
poi-1.5.0	0.66	0.71	0.67	0.7	0.72	0.69	0.7
poi-2.5.1	0.75	0.86	0.82	0.78	0.84	0.77	0.83
poi-3.0	0.73	0.82	0.82	0.8	0.82	0.81	0.81
velocity-1.4	0.8	0.85	0.83	0.82	0.86	0.81	0.86
velocity-1.5	0.67	0.75	0.79	0.68	0.78	0.81	0.76
xalan-2.5.0	0.67	0.75	0.8	0.68	0.8	0.81	0.76

TABLE III. AVERAGE RECALL OF THE CROSS-VALIDATION IN VERSION

	LDA	KNN	DT	LR	RF	SVC	AB
poi-1.5.0	0.93	0.77	0.68	0.85	0.7	0.86	0.76
poi-2.5.1	0.92	0.84	0.82	0.91	0.83	0.94	0.87
poi-3.0	0.95	0.85	0.81	0.9	0.81	0.9	0.85
velocity-1.4	0.96	0.93	0.78	0.96	0.92	1.0	0.91
velocity-1.5	0.91	0.8	0.81	0.89	0.74	0.82	0.74
xalan-2.5.0	0.91	0.8	0.8	0.89	0.8	0.82	0.74

TABLE IV. AVERAGE FSCORE OF THE CROSS-VALIDATION IN VERSION

	LDA	KNN	DT	LR	RF	SVC	AB
poi-1.5.0	0.77	0.74	0.66	0.76	0.7	0.76	0.73
poi-2.5.1	0.83	0.85	0.82	0.84	0.83	0.84	0.85
poi-3.0	0.82	0.83	0.82	0.85	0.82	0.85	0.83
velocity-1.4	0.87	0.89	0.8	0.88	0.88	0.9	0.88
velocity-1.5	0.77	0.77	0.8	0.77	0.76	0.81	0.74
xalan-2.5.0	0.77	0.77	0.8	0.77	0.79	0.81	0.74

TABLE V. AVERAGE PRECISION AMONG VERSIONS IN CK DATASETS

	LDA	KNN	DT	LR	RF	SVC	AB
poi	0.83	0.77	0.70	0.83	0.82	0.78	0.76
velocity	0.41	0.40	0.48	0.38	0.43	0.36	0.38
xalan	0.88	0.74	0.59	0.91	0.71	None	0.74
xerces	1.00	0.79	0.87	1.00	0.88	None	1.00

TABLE VI. AVERAGE PRECISION AMONG VERSIONS IN MULTILAYER STRUCTURAL FEATURE DATASETS

	LDA	KNN	DT	LR	RF	SVC	AB
poi	0.86	0.83	0.80	0.85	0.91	0.81	0.79
velocity	0.36	0.45	0.47	0.36	0.48	0.36	0.48
xalan	0.86	0.81	0.66	1.00	0.64	None	0.59
xerces	1.00	0.96	0.86	None	0.93	None	0.80

TABLE VII. DYNAMIC PREDICTIVE THRESHOLD FILTERING ALGORITHM EXPERIMENT RESULT

Train_version	Predict_version	NodeNum	BugNum	PredictNum	PredictRightNum	AUC_Model	Best_algorithms
2.9.10-2.9.29	2.9.30	80	1	8	1	0.71	SVC
2.9.9-2.9.28	2.9.29	80	1	7	1	0.71	SVC
2.9.8-2.9.27	2.9.28	80	1	17	1	0.62	RF
2.9.7-2.9.26	2.9.27	80	2	2	1	0.5	LDA
2.9.6-2.9.25	2.9.26	80	2	2	1	0.5	LDA

However, this method also has the disadvantage that the prediction accuracy is insufficiently high. In the future, we will continue to optimize the design of multilayer structural feature metrics and optimal prediction model based on dynamic predictive threshold filtering algorithm to improve the accuracy of defect prediction and achieve better prediction results.

REFERENCES

[1] W. E. Wong, X. Li, P. A. Laplante, “Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures,” Journal of Software and Systems, vol. 133, pp. 68-94, November 2017

[2] Zhang F, Mockus A, Keivanloo I, et al. Towards building a universal defect prediction model[C].Proceedings of the 11th Working Conference on Mining Software Repositories. ACM,2014: 182–191.

[3] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, “Software fault prediction metrics: A systematic literature review,” Inf. Softw. Technol., vol. 55, no. 8, pp. 1397–1418, 2013.

[4] Basili VR, Briand LC, Melo WL. A validation of object–oriented design metrics as quality indicators. IEEE Trans. on Software Engineering,1996,22(10):751−761.

[5] Zhou YM, Xu BW, Leung H. On the ability of complexity metrics to predict fault–prone classes in object–oriented systems. Journal of Systems and Software, 2010,83(4):660−674.

[6] Vasa R, Schneider J G, Woodward C, et al. Detecting structural changes in object oriented software systems.[J]. International Symposium Empirical Software Engineering. Noosa Heads: IEEE Computer Society Press, 2005. 479–486.

[7] Ma Y, He K, Du D. A Qualitative Method for Measuring the Structural Complexity of Software Systems Based on Complex Networks[C]. IEEE Computer Society, 2005:257–263.

[8] Valverde S, Solé R V. Network motifs in computational graphs: a case study in software architecture.[J]. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2005, 72(2Pt2):254-271.

[9] Ma Y, He K, Liu J. Network Motifs in Object-Oriented Software Systems [J]. Dynamics of Continuous, Discrete & Impulsive Systems (Series B: Applications & Algorithms), 2007, 14(S6): 166-172.

[10] Y. Yang, J. Ai, X. Li and W. E. Wong, "MHCP Model for Quality Evaluation for Software Structure Based on Software Complex Network," 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, 2016, pp. 298-308.

[11] Zimmermann T, Nagappan N. Predicting subsystem failures using dependency graph complexities. In: Proc. of the Int’l Symp. on Software Reliability. 2007. 227−236.

[12] Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-based analysis and prediction for software evolution[C].Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012: 419–429.

[13] S. Zhang, J. Ai and X. Li, "Correlation between the Distribution of Software Bugs and Network Motifs," 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), Vienna, 2016, pp. 202-213.

[14] Qu Y, Guan X, Zheng Q, et al. Exploring community structure of software Call Graph and its applications in class cohesion measurement[J]. Journal of Systems and Software, 2015, 108: 193-210.

[15] Y.Zhou and H. Leung. Empirical analysis of Object-briented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering,32(10):771-789,2006.

[16] N. Fenton et al., “Predicting software defects in varying development lifecycles using Bayesian nets,” Inf. Softw. Technol., vol. 49, no. 1, pp. 32–43, 2007.

[17] C. P. Chang, C. P. Chu, and Y. F. Yeh, “Integrating in-process software defect prediction with association mining to discover defect pattern,” Inf. Softw. Technol., vol. 51, no. 2, pp. 375– 384, 2009.

[18] Zimmermann T, Nagappan N. Predicting subsystem failures using dependency graph complexities. In: Proc. of the Int’l Symp. on Software Reliability. 2007. 227−236.

[19] Lu HH, Kocaguneli E, Cuki B. Defect prediction between software versions with active learning and dimensionality reduction. In: Proc. of the Int’l Symp. on Software Reliability Engineering. 2014. 312−322.

[20] Li M, Zhang HY, Wu RX, Zhou ZH. Sample-Based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 2012,19(2):201−230.

[21] Elish KO, Elish MO. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 2008,81(5):649−660.

[22] Catal C, Diri B. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Science, 2009,179(8):1040−1058.

[23] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. on Software Engineering, 2008,34(4):485−496.

[24] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proc. of the Int’l Conf. on Software Engineering. 2015. 789−800.

[25] Shepperd M, Bowes D, Hall T. Research Bias: The use of machine learning in software defect prediction. IEEE Trans. on Software Engineering, 2014,40(6):603−616.

[26] Panichella A, Alexandru C V, Panichella S, et al. A Search-based Training Algorithm for Cost-aware Defect Prediction[C]// Genetic and Evolutionary Computation Conference. ACM, 2016:1077-1084.

[27] X. Li, W. E. Wong, R. Gao, L. Hu, S. Hosono, “Genetic Algorithm-based Test Generation for Software Product Line with the Integration of Fault Localization Techniques,” Empirical Software Engineering, vol. 23, no.1,pp.1-51,February.2018