Received June 9, 2019, accepted June 30, 2019, date of publication July 9, 2019, date of current version September 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2927489

Predicting Software Defects Using Self-Organizing Data Mining

JUN-HUA REN AND FENG LIU
School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Jun-Hua Ren (renjunhua.net@163.com)

This work was supported in part by the National Key Research and Development Plan under Grant 2016YFF0204002, and in part by the Research Project under Grant 2016ZNZZ01-01.

ABSTRACT The study predicts the software defect of ranking and classification by utilizing the self-organizing data mining method. The causal relation between software metrics and defects in software modules is established. In the analysis, software metric parameters are considered as the influencing factors and independent variables; defect label values of software modules are considered as dependent variables. When ranking is predicted during the model training process, the bugs of the defect-free modules are replaced with a negative value and those of the defective modules remain unchanged. During classification predictions, the false values of the defect-free modules are replaced with a negative value, whereas the true values of the defective modules are replaced with a positive value ≥ 1.5. Then, case studies and comparison based on data sets of NASA, SoftLab and Promise are conducted by imposing different algorithms. The results show that in the ranking tests, the self-organizing data mining method achieves the smallest errors. In the classification tests, the F-measure values obtained in self-organizing data mining method are the most optimal among the tested algorithms. The self-organizing data mining method is high efficiency and feasible for predicting the software defects.

INDEX TERMS Label function, software defect prediction, software metrics, self-organizing data mining.

I. INTRODUCTION

Software defects are the primary factors affecting software quality [1]. Software defect prediction is an active research topic in computer science [2], [3]. Software defect prediction includes both dynamic and static predictions. Dynamic software defect prediction is mainly based on empirical or statistical means for estimating the distribution of the software defects over the software's life cycle. Static software defect prediction establishes a model for predicting the number and distribution of defects in unknown modules based on the metrics related to software defects. Even though several software defect prediction methods have been proposed, the technology is still considered to be inaccurate.

Software defects may be caused by several factors. The factors are mainly determined based on the characteristics of the software itself and the development process, including the size and the complexity of the software, the developers' understanding of the customer requirements, the algorithms and grammar used during the software development process, and the level of cooperation among the development team. Coding errors are the main cause of software defects. The metrics of software system describe the features of software system to some extent. Some of these features are visible, whereas some are invisible, although all features are included in the metric dataset. Thus, a logical mapping relation exists between the software defects and metrics. Software defect prediction technology is implemented to determine software defects using these implicit characteristics and to further discover whether the software module is defective via the implicit logical relationship in the metric dataset. Software metrics provide a measure of the software quality and indexes and parameters to describe the characteristics of software products. The existing software metrics include the McCabe, Halstead, and Childamber Kemerer or the complexity metrics, and all of these are extensively used in software defect prediction [4], [5]. As the software development has become increasingly complex, a large number of metrics have been introduced and their validity and range of applications are considered to be active research topics [6].

The objectives of the current software defect prediction technologies based on metrics can be approximately divided

The associate editor coordinating the review of this article and approving it for publication was Haider Abbas.

into the following two categories: ranking and classification. The former intends to predict the number of defects in software modules, whereas the latter intends to simply predict whether the module contains defects. Both the aforementioned approaches help the developers to efficiently deploy resources and have been researched for several decades. As early as 1971, Akiyama formulated the relation between the number of defects and lines of code (LOC), which can be commonly known as the Akiyama model [7]. Subsequently, Arthur, Ottensteln, and Lipow proposed a relation between the number of defects and the complexity metrics [8]–[11]. These models contained only one variable, such as LOC. With the increasing diversity of metrics, various regression techniques are used for describing the relation between metrics and the number of defects, including multiple linear regression [12], negative binomial regression (NBR) [13], support vector machine (SVM) [14], [15], and random forest algorithms [16]. Simultaneously, several classification algorithms have also been used to construct software defect prediction models for performing a classification task. The most common models include logistic regression (LR) [17], Bayesian belief network (BBN) [18], decision tree [19], and naive Bayes (NB) [20], [21]. Recently, several methods with neural networks, genetic algorithms, and combinations thereof have been used to predict the software defects for performing the ranking and classification tasks with the increasing complexity of computer and machine learning technology [22]–[30].

The causal relation between the software defects and the software development process is often uncertain. The existing methods for software defect prediction target various factors that affect the software reliability based on relevant standards and practical knowledge. Further, a subset of metrics that affects the software reliability is typically selected for constructing the software defect prediction models. Because of the large number of metrics related to software defects, the research community has not identified a comprehensive method for judging the relation between metrics and the number of software defects. The traditional software defect prediction models are based on the relations between the software defects and metrics such as correlation, dependency, consistency, and causality. Further, the functional relations between the input and output variables are generally constructed using statistical analysis for obtaining predictive models. Among these technologies, software defect prediction based on causality analysis is the most extensively used technology. Zhang et al. used the Eclipse JDT and Eclipse PDE datasets to develop the self-organizing data-mining models [31]. In their study, however, only 8 among the 48 possible metrics were considered for model construction, and the specific mathematical model was not provided. Jing X Y, Wu F, Dong x, et al. focus on solving the imbalance of classification by using SDA and dictionary learning methods, so that the performance of software defect prediction model in the project can be improved and better than other algorithms [32], [33]. Furthermore, they expanded the application to other cross-project and obtained better results [34]–[37]. There are not many studies on causal factors between Bug and metric elements at present. To fill this gap in literature, this study proposes a general method for performing software defect prediction using an extensive range of metrics. An extensive selection of databases is used for model construction and testing.

In the present study, a general overview of the self-organizing data mining method (SODM) is introduced in section II. Then, the methods that are used to evaluate the prediction models are presented in Section III. The related case studies are described in sections of IV, V and VI. In Section IV, the method of establishing the prediction model and the selection of optimization model is discussed in detail. Section V offers general steps and methods for establishing defect prediction model with self-organized data- mining so that illustrate the usefulness of self-organized data mining for obtaining predictive models of software defects. Furthermore, case studies and comparisons with other methods are made in section VI. Conclusions are obtained from the results of the case study and are presented along with the future work in section VII.

II. SELF-ORGANIZING DATA-MINING THEORY AND METHODS

In software defect predictions based on metrics, the metrics are equivalent to independent variables and the number or tendency of the software defects are dependent variables. Any functional relation between the independent and dependent variables can be expressed using a Kolmogorov–Gavbor polynomial as follows:

$$y=a_0+\sum_{i=1}^{m}a_ix_i+\sum_{i=1}^{m}\sum_{j=1}^{m}a_{ij}x_ix_j+\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}a_{ijk}x_ix_jx_k+\cdots \tag{1}$$

With sufficient data volume, the coefficients in Eq. (1) can be fitted to derive an expression for the function trajectory. However, the parameter values for a_0, a_i, … cannot be completely determined. Further, the number of terms drastically increases as the time and number of variables increase, increasing the difficulty of computation. The self-organizing data-mining method proposed by Ivakhnenko can solve this computational problem using a multi-layer self-organizing structure [38], [39]. Thus, a series of mathematical models is constructed and iteratively evaluated using the biologically inspired algorithms. A series of active neurons is generated by cross-combining each input unit of the system, and each neuron performs an optimal transfer. Further, the neurons that are closest to the target variables are selected from the generated neurons. These selected neurons are strongly combined to produce new neurons. This competitive survival and evolution process is repeated until the new neural network does not outperform the previous generation.

Starting from the set of m input variables $x_1, x_2, \ldots x_m$ (Such as x_1-wmc, x_2-dit, x_3-noc, x_4-cbo, x_5-cbo,), the regression Eq. (2) is calculated for each pair of inputs x_i,

x_i and output y, which is the number or the tendency of defects:

$$y = a + bx_i + cx_j + dx_i^2 + ex_j^2 + fx_ix_j \tag{2}$$

The m (m−1)/2 higher order variables are generated to replace the original m variables $x_1, x_2, \ldots x_m$ for estimating y (the output variable). Further, each equation is evaluated based on a criterion after finding these regression equations using a set of input and output observations, and the optimal regression equations are selected and preserved. Subsequently, a set of the optimal y estimates of the quadratic equation is produced (each estimate depends on only two independent variables). The observed values of the second-generation input variables are generated by each of these newly obtained regression equations and are used to replace the original observation values $x_1, x_2, \ldots x_m$.

Using the same method as above, the y quadratic regression equation is calculated for these new input variables, and the regression equations with $m_1(m_1 - 1)/2$ new variables will be obtained for estimating y. Further, the optimal variables are selected, the third-generation input variables are selected to replace the second-generation input variables, and the quadratic regression equation is constructed by combining the third-generation input variables. This process is continued until the accuracy of the estimations decline than the previous equation. The optimal estimation in the last generation is selected from the quadratic polynomials after the iteration of the regression equations is terminated. Further, the complex Ivakhnenko polynomial Eq. (1) is obtained after reverse algebraic substitution.

The core technology of self-organizing data mining can be referred to as the group method of data handling (GMDH) [40], [41], which classifies the data into training and test sets. In the training set, the interior criterion is used for estimating the parameters, and the exterior criterion is used for selecting the interior candidate model. This process is repeated until the exterior criterion value is no longer improved. This termination rule guarantees the data fitting accuracy and presents likely predictions at a certain noise level, yielding a complexity model that achieves optimal balance. Further, the process of optimal model generation is depicted in Fig. 1.

The GMDH algorithm is implemented using the following steps:

Step 1. The sample data (N data) are divided into a training set A and a testing set B ($N_\omega = N_A + N_B, \omega = A \cup B$);

Step 2. A general relation is established between the output and the inputs. Further, the K–G polynomial is commonly used as the reference function. For a system containing three inputs and a single output, a quadratic K–G polynomial can be used for the following reference function:

$$\begin{aligned} f(x_1, x_2, x_3) = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1^2 \\ + a_5x_2^2 + a_6x_3^2 + a_7x_1x_2 + a_8x_1x_3 + a_9x_2x_3 \end{aligned}$$

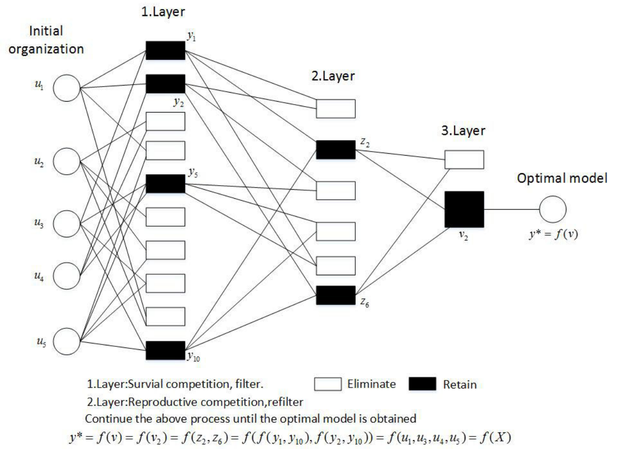

1.Layer
Initial organization
2.Layer
3.Layer
Optimal model
u_1
u_2
u_3
u_4
u_5
y_1
y_2
y_5
y_{10}
z_2
z_6
v_2
$y^* = f(v)$
1.Layer:Survial competition, filter.
2.Layer:Reproductive competition,refilter
Continue the above process until the optimal model is obtained
□ Eliminate
■ Retain
$y^* = f(v) = f(v_2) = f(z_2, z_6) = f(f(y_1, y_{10}), f(y_2, y_{10})) = f(u_1, u_3, u_4, u_5) = f(X)$

FIGURE 1. Generation of the optimal complexity model.

Further, m terms are used as the initial models in the network structure.

$$v_1 = a_0, \quad v_2 = a_1x_1, \quad v_3 = a_2x_2, \cdots, v_{10} = a_9x_3x_2,$$

where m=10;

Step 3. Select one (or several) of the criteria with complementary nature as the objective function or select some external criteria;

Step 4. The $y_k = f_k(v_i, v_j)(k = 1, 2, \ldots, 10)$ transfer functions represent the first-layer intermediate models, which are adaptively generated using the self-organizing processes, and the number of variables and structure of functions are observed to differ. Meanwhile, the y_k parameter ($k = 1, 2, \ldots, 10$) are estimated based on the training set A;

Step 5. The first-layer intermediate models are screened on the detection set B according to the external criteria. The selected intermediate model ω_k ($k = 2, 3, 6, 7, 9$) will be used as the input variable of the second layer in the network;

Step 6. By repeating steps 4 and 5, the intermediate models of layers 2 and 3 are generated. Finally, an explicit optimal complexity model is developed for performing analysis (Fig. 1). By considering the state after layer 3 as an example, the model contains only four variables v_i, and the variable v_2 in five initial variables is automatically eliminated during the screening.

From the above steps, we can observe that the GMDH method requires no specific prior knowledge and assumptions about premises. Because of the recursive nature of the GMDH algorithm, GMDH modeling can effectively resist the noise interference and avoid overfitting. Further, the predictive ability of the model is good, and the final model can be easily interpreted.

III. PERFORMANCE EVALUATION OF THE PREDICTION MODELS

The software defect prediction models that are used for ranking task are often derived from the regression algorithms. Their performances can be evaluated based on the degree of regression fitting (R^2) [42], the average absolute error

(***AAE***) [43], and the average relative error (***ARE***) [12]. These indices can be defined as follows:

$$\begin{cases} R^2 = 1 - \dfrac{SS_{err}}{SS_{tot}} = 1 - \dfrac{\sum\limits_{i=1}^{n} (y_i - f_i)^2}{\sum\limits_{i=1}^{n} (y_i - \bar{y}_i)^2} \\ AAE = \dfrac{1}{n} \sum\limits_{i=1}^{n} |y_i - f_i| \\ ARE = \dfrac{1}{n} \sum\limits_{i=1}^{n} \dfrac{|y_i - f_i|}{y_i + 1} \end{cases} \tag{3}$$

In Eq. (3), n represents the number of analyzed software modules, y_i denotes the number of actual bugs in the ith software module, f_i denotes the predicted bugs in the ith software module, and $\bar{y}_i$ denotes the mean value of y_i.

The performance evaluations of the software defect prediction models while performing classification task are based on the confusion matrix, which includes the measures of accuracy, precision, recall, F-measure, and so on [3]. These indices can be defined as follows:

$$\begin{cases} accuracy = \dfrac{TP + TN}{TP + FP + FN + TN} \\ \text{Precision} = \dfrac{TP}{TP + FN} \\ recall = \dfrac{TP}{TP + FP} \\ f - measure = \dfrac{2 \times \text{Recall} \times \ precision}{\text{Recall} + \text{Precision}} \end{cases} \tag{4}$$

In Eq.(4), TP denotes the number of accurate positive predictions, i.e., the number of modules that are actually defective and are correctly predicted to be defective; FP denotes the number of inaccurate positive predictions, i.e., the number of modules that are actually defective but that are predicted to be defect-free; FN denotes the number of inaccurate negative predictions, that is, the number of modules that are actually defect-free but that are predicted to be defective; TN represents the number of correct negative predictions, that is, the number of modules that are actually defect-free and that are predicted to be defect-free.

In addition, the fault-percentile average is used to evaluate the defect prediction models for ranking [14]. However, this study focuses only on R^2, ***AAE***, and ***ARE*** to ensure simplicity.

IV. MODELING EXPERIMENTS USING SELF-ORGANIZING DATA-MINING

A. ESTABLISHMENT OF A SELF-ORGANIZING DATA-MINING MODEL OF SOFTWARE DEFECT PREDICTION WITH A RANKING TASK

Software defect prediction models for ranking task are intended to predict the number of defects in software modules; these defects can be used to order the software modules based on their defect numbers and therefore can test software modules in that order (modules with more defects are tested first) [23]. Herein, we use the wspornaganiepi dataset in the PROMISE database [44] as an example to demonstrate the effectiveness of the self-organizing data-mining prediction method. c

First, a prediction model using self-organizing data-mining was trained using the wspornaganiepi dataset in the PROMISE database. The dataset comprises 18 software modules, and each module is measured using 20 metrics and different metric parameters that correspond to bugs. The defective modules contain 29 bugs. Further, the metric parameters for each module are considered to be independent variables ($x_1 \sim x_{30}$), and the bugs corresponding to the modules are considered to constitute the dependent variable x_{21}. The software defect prediction model is established based on self-organizing data-mining as follows:

$$\begin{cases} x_{21} = 1.727z_{52} + 0.04595z_{51}^2 + 1.611 \\ z_{51} = 4.145x_{10} - 2.795 \\ z_{52} = 0.08045z_{41} + 1.001z_{42} \\ z_{41} = 0.5495x_{19} - 1.252 \\ z_{42} = -0.1128z_{31} + 1.016z_{32} \\ z_{31} = 0.5906x_9 - 1.608 \\ z_{32} = 1.064z_{22} + 0.5043z_{21}z_{22} + 0.08342z_{22}^2 \\ z_{21} = 6.715x_{15} - 3.485 \\ z_{22} = 0.9954z_{12} - 0.4849z_{11}z_{12} \\ z_{11} = 0.5906x_9 - 1.608 \\ z_{12} = 1.873x_{13} - 0.03264x_5x_{13} + 0.2054x_{13}^2 - 0.5265 \end{cases} \tag{5}$$

The fitting results between the bugs predicted by Eq. (5) and the actual bugs are depicted in Fig. 2.

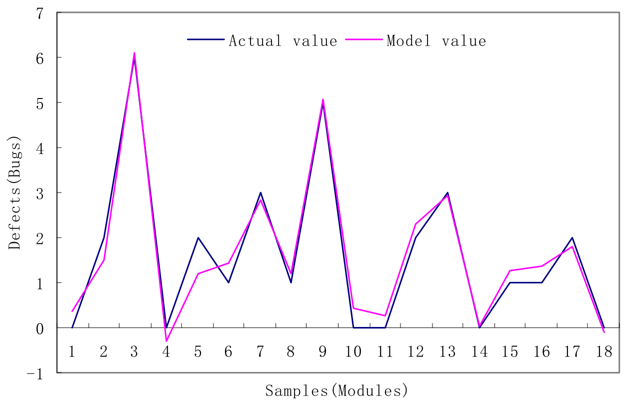


FIGURE 2. Comparisons between the bugs calculated by model (5) and actual bugs.

Polynomial Eq. (1) is obtained by iterating Eq. (5) from backward to forward as follows:

$$\begin{aligned} x_{21} &= 0.0195\, x_{13}^4 - 0.0064\, x_{19}x_{13}^4 + 0.0005\, x_{19}^2x_{13}^4 \\ &\quad - 0.1145\, x_{19}x_{13}^3 - 0.0064\, x_5x_{13}^3 + 0.0021\, x_{19}x_5x_{13}^3 \\ &\quad + 0.0092\, x_{19}^2x_{13}^3 - 0.0004\, x_{19}^2x_5x_{13}^3 + 0.3554\, x_{13}^3 \\ &\quad + 1.0754\, x_{13}^2 - 0.419\, x_{19}x_{13}^2 + 2.1126\, x_{15}x_{13}^2 \\ &\quad - 0.3497\, x_{15}x_{12}x_{13}^2 + 0.0183\, x_{10}x_{12}x_{13}^2 + 0.0005\, x_{13}^2x_{13}^2 \end{aligned}$$

$$
\begin{aligned}
&-0.0001\mathrm{x}_{19}x_{13}^2x_{13}^2+0.028_{13}^2x_{13}^2-0.0014x_{19}^2x_1x_{13}^2\\
&+0.000014x_{19}^2x_5^2x_{13}^2+0.0003x_{15}^2x_{13}-0.0117x_{13}^2x_{13}\\
&+0.0033x_{19}^2+0.897x_{15}x_{15}+0.0558x_{15}x_{19}x_5x_{13}\\
&-3.1903x_{15}x_{19}x_{13}-5.5589x_{15}-0.3449x_{15}x_{13}\\
&+19.7752x_{15}x_{13}+0.1418x_{19}x_5x_{13}+0.8003x_{19}x_{13}\\
&+0.0864x_5x_{13}-4.9603x_{13}-0.0389x_9+0.7895x_{10}^2\\
&-0.225x_{19}-1.0647x_{10}+3.3763 \qquad (6)
\end{aligned}
$$

The metrics included in wspornaganiepi corresponding to each variable in x_{21} are x_5—rfc, x_9—npm, x_{10}—lcom3, x_{13}—moa, x_{15}—cam, and x_{19}—max_cc.z_{ij} is the active neuron observed during the iterative process of the model-formation algorithm.

From the Eq.(5) and (6), we can observe that the software defect prediction model uses only six of the 20 metrics that are included in the dataset because the remaining metrics were eliminated during the iterative self-organizing process. The polynomial for x_{21} is the defect prediction model that is formed with self-organizing data-mining based on the wspornaganiepi dataset and that can be used to predict the original bugs. Further, the calculated result of x_{21} is not an integer; however, the prediction results must be integers[]. Therefore, the results are rounded up using the following operation:

$$bugs = INT(x_{21} + 0.5) \qquad (7)$$

Here, INT represents an integer operation.

The calculated x_{21}, the rounded number of bugs calculated using Eq. (7), and the actual number of bugs are compared in Table 1.

TABLE 1. Comparison between the predicted and actual bugs (Bugs labelled=INT(x_{21} + 0.5).

Values Calculated by model(5)	Bugs labeled	Actual bugs*	Values Calculated by model(5)	Bugs labeled	Actual bugs*
0.37	0	0	0.43	0	0
1.48	*1*	*2*	0.27	0	0
6.10	6	6	2.29	2	2
-0.31	0	0	2.92	3	3
1.21	*1*	*2*	0.04	0	0
1.42	1	1	1.26	1	1
2.84	3	3	1.35	1	1
1.19	1	1	1.81	2	2
5.08	5	5	-0.10	0	0

* Data from wspornaganiepi dataset [44]

The data presented in Table 1 denote that the number of predicted bugs in all the modules is consistent with the actual number of bugs, except that one less bug is predicted in the second and fifth modules. The correlation coefficient R^2 is 0.983, the ***AAE*** is 0.111, and the ***ARE*** is 0.037 between the actual and predicted bugs.

The x_{21} value of in each module is not calculated by polynomial (6) while iteratively fitting the model parameters but is iterated from the last to the first term according to model (5).

B. ESTABLISHMENT OF THE SELF-ORGANIZING DATA-MINING MODEL FOR SOFTWARE DEFECT PREDICTION WITH A CLASSIFICATION TASK

Software defect prediction with a classification task is used to predict whether software modules have defects, which can help developers decide whether software modules should be tested. Prediction modules for the classification task require both high detection rates of defect-prone modules and little wastage of tasting resources (caused by erroneous predictions of defect-free software modules) [23].

The software defect prediction model for classification task was trained using the Ar3 dataset in the SOFTLAB database [45]. This dataset contained 63 modules, and each module contained 29 metric parameters and a defect label, which is false or true. The modules corresponding to false are defect-free, whereas the modules corresponding to true are defective. First, false labels were replaced with −0.5 and true labels were replaced with 1.5 in this demonstration. A total of 29 metrics parameters are obtained as independent variables; the corresponding defect label values of −0.5 and 1.5 are considered to be dependent variables, and the following model was formed using the self-organizing data-mining method:

$$
\begin{cases}
x_{30} = -0.09397z_{41} + 0.7469z_{42} - 0.246\\
z_{41} = 0.06518x_{14} - 1.101\\
z_{42} = 1.093z_{32} - 0.136z_{31}z_{32}\\
z_{31} = 0.05602x_{26} - 0.5398\\
z_{32} = 0.9541z_{22} + 0.7441z_{21}z_{22} - 0.532z_{21}^2\\
z_{21} = 0.0000142x_{15} - 0.4949\\
z_{22} = 0.7891z_{12} + 0.2363z_{11}^2\\
z_{11} = 0.0001822x_2x_{10} - 0.371\\
z_{12} = 0.003897x_{14}x_{22} - 0.00344x_{22}^2 - 0.3172
\end{cases} \qquad (8)
$$

The corresponding metrics to the variables presented in model (8) are x_2 -blank_loc, x_{10} -halstead_vocabulary, x_{14}-halstead_difficulty, x_{15}-halstead_effort, x_{22}-multiple_condition_count, x_{26}-design_complexity, x_{30}-defects, z_{ij} is the active neuron during the iterative process of the model-formation algorithm. Only six of the 29 metrics were selected in the final model. The fitting results between defects calculated by Eq. (8) and the actual defects are depicted in Fig. 3.

For the sake of convenience, Eq. (8) was not iterated for obtaining the corresponding Eq. (1); however, the prediction values of each module were directly computed according to each item of Eq. (8) in each iteration from backward to forward. Let $\widehat{E}(x)$ indicate the tendentiousness of the defects in the module based on the value predicted by model (8).

$$
\widehat{E}(x) = \begin{cases} 0, & when \quad x_{30} < 0\\ 1, & when \quad x_{30} \geq 0 \end{cases} \qquad (9)
$$

The classification results predicted based on Eq. (8) and Eq. (9) are compared with the actual defects presented for Ar3 dataset in Table 2.

TABLE 2. Comparisonof the model calculation, model label, and actual values (Defects labelled = 0, when $x_{30} < 0$, Defects labelled = 1, when $x_{30} \geq 0$).

No	Values calculated by model(8)	Values labeled	Actual values	No	Values calculated by model(8)	Values labeled by (9)	Actual values	No	Values calculated by model(8)	Values labeled	Actual values
1	1.469	1	1	22	−0.222	0	0	43	−0.386	0	0
2	−0.382	0	0	23	−0.113	0	0	44	−0.384	0	0
3	−0.392	0	0	24	−0.364	0	0	45	−0.384	0	0
4	−0.408	0	0	25	−0.433	0	0	46	−0.657	0	0
5	−0.361	0	0	26	−0.355	0	0	47	−0.368	0	0
6	−0.298	0	0	27	−0.375	0	0	48	1.863	1	1
7	−0.208	0	0	28	−0.379	0	0	49	−0.337	*0*	*1*
8	−0.361	0	0	29	−0.392	0	0	50	0.191	1	1
9	−0.287	0	0	30	−0.379	0	0	51	−0.144	0	0
10	−0.374	0	0	31	−0.403	0	0	52	−0.409	0	0
11	−0.391	0	0	32	−0.401	0	0	53	−0.224	0	0
12	−0.394	0	0	33	−0.376	0	0	54	1.511	1	1
13	−0.374	0	0	34	−0.382	0	0	55	1.522	1	1
14	−0.739	0	0	35	−0.385	*0*	*1*	56	−0.386	0	0
15	−0.427	0	0	36	−0.438	0	0	57	−0.047	0	0
16	−0.393	0	0	37	−0.413	0	0	58	−0.123	0	0
17	−0.393	0	0	38	−0.397	0	0	59	−0.375	0	0
18	−0.433	0	0	39	−0.394	0	0	60	1.22	1	1
19	−0.379	0	0	40	−0.57	0	0	61	−0.372	0	0
20	−0.379	0	0	41	−0.264	0	0	62	−0.418	0	0
21	−0.352	0	0	42	−0.393	0	0	63	−0.437	0	0

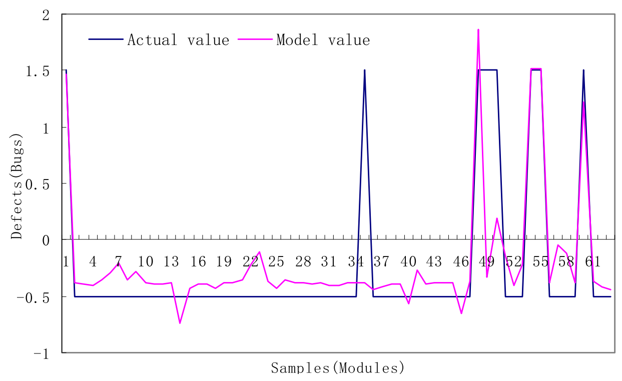


FIGURE 3. Comparisons between the defects by model (8) and the actual defects.

Table 2 shows that according to the self-organizing data-mining Eq. (8), 55 defect-free modules of the 63 modules are accurately predicted; 6 of 8 defect modules are accurately predicted, and 2 actually defective modules are predicted as being defect-free. Calculating the evaluation index of Eq. (8), we find that the accuracy, precision, recall, and F-measure of the model are 0.968, 1.0, 0.75, and 0.857, respectively.

C. DISCUSSION

In the above defect predictions for classification, the false label is replaced with −0.5 to indicate that the module is defect-free, and the true label is replaced with 1.5 to indicate that the module is defective. Further, we verified the effectiveness of defect classification while replacing false and true labels with different negative and positive scalar values, respectively. We verified the false labels by replacing them with 0, −0.5, and −1.0 and the true labels by replacing them with 1, 2, and 4 for establishing the prediction model with self-organizing data-mining. Because the order of the modules in the dataset does not affect the model formation, we ordered the modules in ascending order based on the number of the defects so that the defect-free and defective modules would be clustered respectively together in the resulting graphical representations to ensure clarity. Fig. 4 depicts the plots exhibiting different false and true values for Ar3 classification.

In Fig. 4, the plots of defect-free and defective modules are observed to be entirely above the horizontal coordinate axis if false is equal to 0 and true is equal to 1. Therefore, it is difficult to observe the difference between defective and defect-free modules. However, if the binary values are replaced with negative and positive scalars, the horizontal axis distinguishes the defective and defect-free modules. These plots depict that the whole curve is translated upward if the true value is increased and the false value is fixed, and vice versa. For performing software testing, the choice of scalar values to represent false and true should be considered from a comprehensive point of view. In general, predicting a defective module to be defect-free is considerably expensive than predicting a defect-free module to be defective [46]. Therefore, the label values for the defect-free and defective modules should be selected using repeated tests to determine an appropriately conservative model. Defect-free modules can be generally represented with a negative number less than or equal to −0.1, and defective modules can be generally represented with a positive number greater than or equal to 1.5 because the number of bugs in a defective module is at least 1.

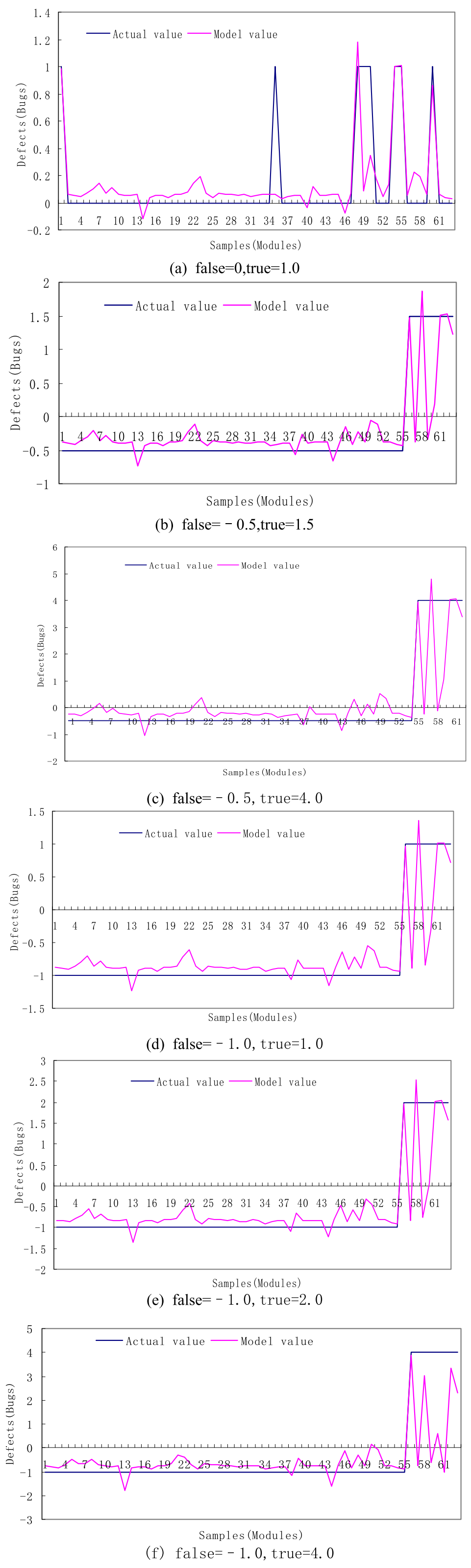


FIGURE 4. Classification changes when defects are labelled with different values.

In previous quantitative studies, the false label of defect-free module is usually represented by 0 and the true label of defective module is represented by 1. However, when an automated data-mining system is used to build a predictive model, this binary representation is considered to be inadequate [58]. A defect-free module is completely opposite to a defective module; therefore, some ambiguity will remain if 0 is used to classify the modules as defect-free because the distinction between a positive value and 0 is not as clear as the distinction between positive and negative. Because positive numbers represent a defect, a negative number should be used to represent the absence of defects. Furthermore, the difference between 0 and 1 is not quantitatively large enough to distinguish two objects with completely different properties when the magnitude of the difference is used to train a model using several iterations. Moreover, in terms of the number of bugs, there is at least one bug in the defective module and the average number of bugs is definitely>1 in all defective modules. Therefore, negative values are used herein to indicate defect-free modules and the value greater than or equal to 1.5 is used to indicate defective modules in the defect prediction of classification task B.

Because of the difficulties associated with accurately counting software defects using a predictive model, majority of studies have focused on defect prediction for classification task [23]. According to the above defect prediction for the classification task, we can further improve the self-organized data-mining technique to develop models that can be used to ranking predictions based on the number of defects. Because the numbers of bugs corresponding to each module in the ranking prediction is 0 or a positive integer, the label 0 of bugs in the defect-free modules can be replaced with a negative value, while the number of bugs in the defective module is left unchanged. Further, the self-organizing data-mining model is trained by considering all the software metric parameters as independent variables and the corresponding outputs of the number of bugs as dependent variables. $\widehat{E}(x)$ provides the label function of software defects based on the established model, and the number of bugs is the model (10) output.

$$\widehat{E}(x) = \begin{cases} 0, & \textit{when bug} \leq 0 \\ INT(bug + k), & \textit{when bug} > 0,\ k \textit{ is a positive} \end{cases} \tag{10}$$

Here, INT represents an integer operation.

The Eclipse PDE dataset in the SEIP database [47] was also used in a case study to denote the effectiveness of expressing defective and defect-free modules with different false value during training. Details of this data can be seen in the TABLE 7. 576 modules are included in the dataset, and each module is associated with 48 metrics parameters and some known number of bugs. The datasets include 466 defect-free modules and 110 defective modules. The total number of bugs in the defective modules is 242. First, each module of the dataset is sorted according to the known number of bugs; further, the label 0 of bugs in the defect-free modules is replaced by -0.3 so that the defective and defect-free modules can be distinguished by the horizontal axis in a graphical display. The final model returned by self-organizing data-mining using this set of intermediate outputs is given in model (11) as follows:

$$\begin{cases} x_{49} = 1.212z_{82} - 0.1291z_{81}z_{82} + 0.04893z_{82}^2 + 0.178 \\ z_{81} = 1.737x_{45} - 0.6062 \\ z_{82} = 1.076z_{72} + 0.1631z_{71}z_{72} \\ z_{71} = 0.03764x_{22} - 4.945 \\ z_{72} = -0.7929z_{61} + 1.735z_{62} \\ z_{61} = 0.7773z_{22} + 0.05725z_{21}z_{22} \\ z_{21} = 0.9019z_{12} + 0.08939z_{11}z_{12} \\ z_{11} = 0.1439x_{19} - 0.9916 \\ z_{12} = 0.003072x_2 - 0.05708x_9 + 0.001641x_2x_9 - 0.3778 \\ z_{22} = -0.04946z_{11} + 1.15z_{12} - 0.06492z_{11}z_{12} \\ z_{11} = 0.4201x_{36} - 0.6082 \\ z_{12} = 0.003072x_2 - 0.05708x_9 + 0.001641x_2x_9 - 0.3778 \\ z_{62} = 0.5069z_{51} + 0.5196z_{52} \\ z_{51} = 1.088z_{42} + 0.3919z_{41}z_{42} \\ z_{41} = 2.005x_{43} - 0.2611 \\ z_{42} = 0.9622z_{32} - 0.221z_{31}z_{32} + 0.1441z_{32}^2 \\ z_{31} = 0.002816x_1 + 0.0001271x_1x_{46} - 0.4549 \\ z_{32} = 0.7712z_{22} + 0.1186z_{21}z_{22} + 0.05782z_{22}^2 \\ z_{21} = 343.2x_{38} - 0.0417 \\ z_{22} = -0.04946z_{11} + 1.15z_{12} - 0.06492z_{11}z_{12} \\ z_{11} = 0.4201x_{36} - 0.6082 \\ z_{12} = 0.003072x_2 - 0.05708x_9 + 0.001641x_2x_9 - 0.3778 \\ z_{52} = 0.3663z_{41} + 0.4985z_{42} + 0.0544z_{42}^2 \\ z_{41} = 0.004447x_2 + 0.1135x_6 - 0.0008241x_6^2 - 0.4806 \\ z_{42} = 0.9329z_{32} - 0.08852z_{31}^2 + 0.0607z_{32}^2 \\ z_{31} = 0.00306x_2 + 0.0001789x_2x_{46} - 0.3972 \\ z_{32} = -0.09529z_{21} + 0.9156z_{22} + 0.1521z_{21}z_{22} \\ z_{21} = 1.077x_{13} - 1.799 \\ z_{22} = 1.128z_{12} - 0.07237z_{11}z_{12} \\ z_{11} = 0.4201x_{36} - 0.6082 \\ z_{12} = 0.004285x_3 - 0.06183x_9 + 0.002187x_3x_9 - 0.3817 \end{cases} \tag{11}$$

In model (11), x_1-LOC, x_2-SLOC_P, x_3-SLOC_L, x_6-C_SLOC, x_9-HCLOC, x_{13}-AVCC, x_{19}-PACK, x_{22}-MI, x_{36}-NQU, x_{38}-SIX, x_{43}-MPC, x_{45}-INTR, x_{46}-CCOM, x_{49}-bug_cnt, and z_{ij} is the active neuron during the iterative process of the model-formation algorithm.

The prediction results of model (11) were labelled according to model (10) with k=0.5 to count the predicted number of defects in each module. 31 modules from among the 466 defect-free modules were inaccurately predicted to be defective; the total number of defects was 33 because 2 modules were predicted to have 2 bugs, 1 module was predicted to have 3 bugs, and each of the remaining 28 modules had 1 bug. 133 bugs were predicted to be present among the 110 defective modules. The evaluation indices R^2, ***AAE***, and ***ARE*** of the model (11) are 0.86, 0.26, and 0.12, respectively.

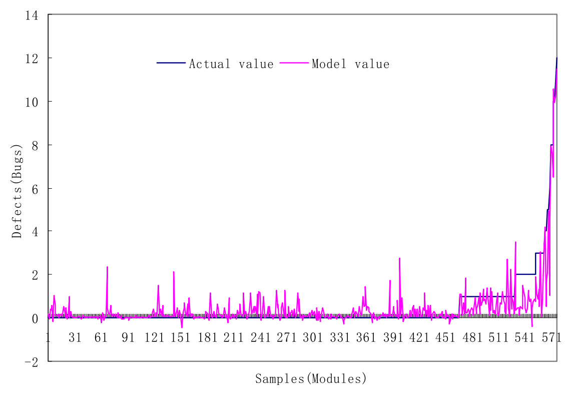
To observe the influence of different label values on the model's prediction results, the label 0 of the defect-free modules in Eclipse PDE dataset was replaced with −0.5. The self-organizing data-mining model predicted that 26 bugs were present among 466 defect-free modules, and 119 bugs were predicted among 110 defective modules. The evaluation indices R^2, ***AAE***, and ***ARE*** of the model were 0.85, 0.28, and 0.13, respectively. If the defect-free modules were labelled as 0 during model training, the self-organizing data-mining model predicted 57 bugs among 466 defect-free modules and 149 bugs among 110 defective modules. The evaluation indices R^2, ***AAE***, and ***ARE*** of the model results were 0.85, 0.28, and 0.16, respectively. If the defect-free modules were labelled with 0 but the prediction output of the model were labelled with $x_{49} < 0, \widehat{E}(x) = 0$ and when $x_{49} > 0$, $\hat{E}$(x)=INT(x_{49}), the resulting model predicted 22 bugs among 466 defect-free modules and 108 bugs among 110 defective modules. The evaluation indices R^2, ***AAE***, and ***ARE*** of the model were 0.85, 0.28, and 0.13, respectively. The graphical representations of the model results trained with different values of false in the Eclipse PDE dataset are depicted in Fig. 5. The predictions of the model as calculated with different labels are presented in Table 3.

According to the evaluation indexes in Table 3, R^2 is the largest and ***AEE*** and ***ARE*** are the smallest when false= −0.3. Thus, when bugs 0 in defect-free modules are replaced by −0.3,the evaluation indexes of the model are better than those of other models.

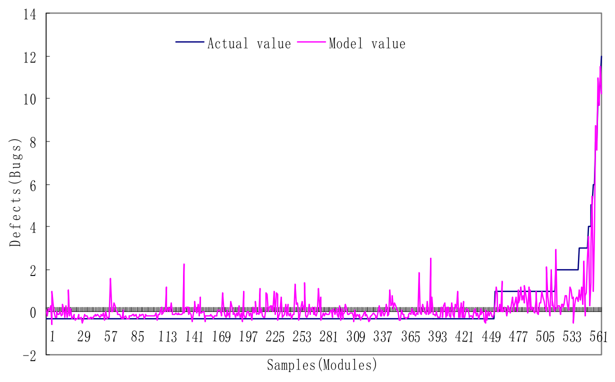
For comparing between false = 0, $x_{49} < 0$, $\widehat{E}(x) = 0$, $x_{49} > 0$, $\widehat{E}(x) = INT(x_{49} + 0.5)$ and false= −0.3, $x_{49} < 0$, $\widehat{E}(x) = 0, x_{49} > 0, \widehat{E}(x) = INT(x_{49}+0.5)$, the former prediction increased 16 bugs (149 bugs minus 133 bugs) in defective modules and 24 bugs(57 bugs minus 33 bugs) in defect-free modules than the latter. The false-positives for 24 bugs denote the cost of accurately detecting 16 more bugs in defective modules. In other words, more defect-free modules will be inevitably detected as defective models when predicting more bugs in defective modules. In contrast, the label function with $x_{49} < 0, \widehat{E}(x) = 0$, and $x_{49} > 0$, $\widehat{E}(x) = INT(x_{49} + 0.5)$ was more accurate than $x_{49} < 0, \widehat{E}(x) = 0$ and $x_{49} > 0$, $\widehat{E}(x) = INT(x_{49})$ if defect-free was represented with 0. The latter function predicted few bugs in defective modules, which increased the risk of software failure. Furthermore, in this example, if false is equal to −0.3 and the label function is $x_{49} < 0, \widehat{E}(x) = 0$, and $x_{49} > 0\widehat{E}(x) = INT(x_{49} + 1.5)$, the model predicts 183 bugs in 466 defect-free modules and 233 bugs in 110 defective modules. The bugs predicted in the defective modules accounted for 96.3% of the actual bugs; however, the evaluation indices R^2, ***AAE***, and ***ARE*** of the model were 0.84, 0.47, and 0.37, respectively, which were worse than the values mentioned previously. In addition, it can be seen from Fig. 5 (d) that more defective modules may be predicted as defect-free modules when bugs 0 in defect-free modules are replaced by −1.0, which increases the risk of software failure obviously.

TABLE 3. Comparison of the predictions of false different label in Eclipse PDE.

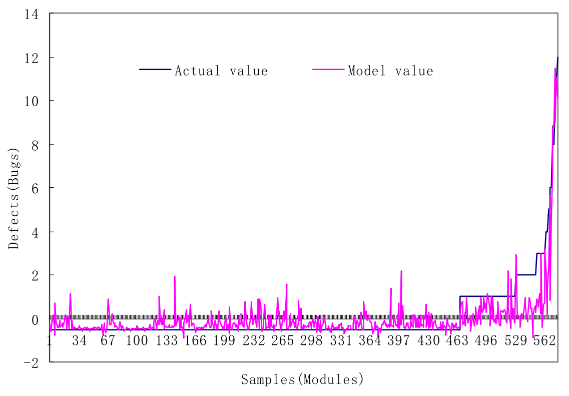
False values	Label functions	Bugs predicted in defect-free modules	Bugs Predicted in Defected modules	Indexes		
				R^2	AEE	ARE
False=-0.3	$x_{49} < 0,\ \widehat{E}(x)=0$; $x_{49} > 0,\ \widehat{E}(x)=INT(x_{49}+0.5)$	33	133	0.86	0.26	0.12
False=-0.5	$x_{49} < 0,\ \widehat{E}(x)=0$; $x_{49} > 0,\ \widehat{E}(x)=INT(x_{49}+0.5)$	26	119	0.85	0.28	0.13
False=0	$x_{49} < 0,\ \widehat{E}(x)=0$; $x_{49} > 0,\ \widehat{E}(x)=INT(x_{49}+0.5)$	57	149	0.85	0.28	0.16
	$x_{49} < 0,\ \widehat{E}(x)=0$; $x_{49} > 0,\ \widehat{E}(x)=INT(x_{49})$	22	108	0.85	0.28	0.13



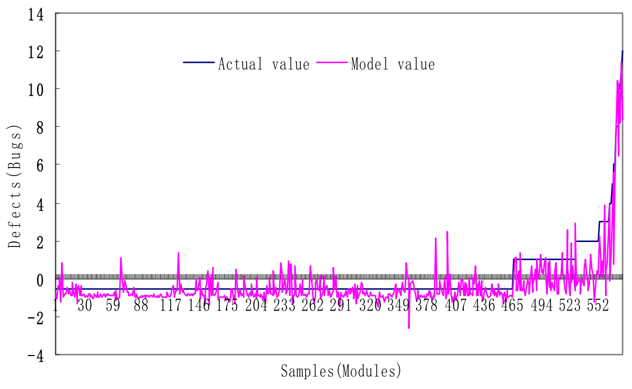
(a) The bugs 0 of the defect-free modules remain unchanged



(b) The bugs 0 of the defect-free modules were replaced by －0. 3



(c) The bugs 0 of the defect-free modules were replaced by －0. 5



(d) The bugs 0 of the defect-free modules were replaced by －1. 0

FIGURE 5. Graphical model with different false values of Eclipse PDE.

V. GENERAL STEPS AND METHODS FOR ESTABLISHING DEFECT PREDICTION MODEL WITH SELF-ORGANIZED DATA- MINING

The experiment in Section IV suggests that replacing the false label with a negative scalar allows the defect-free modules to be accurately predicted, whereas the label function $\widehat{E}(x) =$ INT $(x_{49} + k)$ (with k as the parameter) can improve the accuracy of the label of defective modules. Therefore, we propose the following general steps and methods to predict software defects using self-organizing data-mining techniques.

A. GENERAL STEPS AND METHODS FOR THE RANKING PREDICTION

For ranking prediction based on the bugs in software modules, the method described in IV.A can be used to establish a predictive model. The following steps can be used for predicting the bugs in datasets that are difficult to choose metrics using self-organizing data-mining:

Step 1. The defect prediction models are established based on a known software dataset. First, the bug values of 0 corresponding the defect-free modules are alternately replaced with a value less than or equal to -0.1, and the bugs in the defective modules remain unchanged. Further, using the self-organizing data-mining software tool [47], [48], some predictive models are established by considering each module metric parameters as an independent variable and the number of bugs as dependent variables. The graphics of each model are observed; the model corresponding to the bug value of the defect-free module when maximum defect-free modules are gathered under the horizontal coordinate axis and maximum defective modules above the horizontal coordinate axis is chosen as the prediction model. To improve the visibility without affecting the validity of the model, the modules in known dataset can be arranged in the ascending order based on the number of bugs before modeling.

Step 2. The prediction values of the model are calculated, and the label function is constructed. For the self-organizing data-mining model selected, the appropriate label function is formulated according to the prediction value of the metric parameters of each module in the known software dataset to translate the model output into the number of bugs. If x represents prediction value of the model, $\widehat{E}(x)$ represents the label function. While $x < 0$, $\widehat{E}(x) = 0$; while $x > 0$, $\widehat{E}(x) = \text{INT}\ (x+\text{k})$ (in general, k=0.5). The selection of k scalar is considered to achieve the optimal balance among the model errors, the cost of software testing, and the importance of the software.

Step 3. The self-organizing data-mining model is evaluated. Further, the evaluation indices R^2, $\boldsymbol{AAE}$, and $\boldsymbol{ARE}$ of the model are computed. If each evaluation index is reasonable, the prediction model based on self-organizing data-mining is considered to be valid and can be used to predict the bug number of modules in new software dataset.

Step 4. The number of bugs in new software modules is predicted using the established self-organizing data-mining model and the selected label function. Each metric parameters of the new software module is provided as input into the established prediction model; further, the bug values of each module are calculated, and the final numbers of bugs in each module are determined using the established label function.

B. GENERAL STEPS AND METHODS FOR THE CLASSIFYING PREDICTION

For classifying prediction based on the defect tendency in software modules, the method described in IV.B can be used to establish a predictive model. The following steps can be used for predicting the defect tendency in datasets that are difficult to choose metrics using self-organizing data-mining:

Step 1. The defect prediction models are established based on a known software dataset. First, the metric parameters of each module are considered as independent variables and the corresponding label false and true values are considered as dependent variables. In addition, all the false values corresponding to the defect-free modules are replaced with a numbers less than or equal to -0.1 alternately, and all the true values corresponding to the defective modules are replaced with a number ≥ 1.5 in the known software dataset. The self-organizing data-mining prediction models are established using a self-organizing data-mining software tool [47], [48]. First, the figures of the model output are observed and the prediction model is selected so that the figures of the defect-free modules are gathered as much as possible under the horizontal coordinate axis and the figures of the defective modules are gathered as much as possible above the horizontal coordinate axis. To improve the visibility without affecting the validity of the model, the modules in known software datasets can be arranged in ascending order based on the false and true label values prior to modeling.

Step 2. The prediction model for self-organizing data mining is evaluated. The prediction values for the self-organizing data-mining model selected corresponding to the suitable replacement value in the first step are calculated. According to the prediction values of the model and the actual bugs, the determined label function is used to identify the bugs in the module. x represents the prediction value of each bug in a module based on the self-organizing data-mining model, and $\widehat{E}(x)$ is the label function. In general, $x < 0, \widehat{E}(x) = 0$ and $x > 0, \widehat{E}(x) = 1$ are chosen as the label functions. The validity of the model is evaluated according to the label and the actual results.

Step 3. The defects of the new software module are predicted by the selected model. The new software module metric parameters are input into the established model, and the prediction values are computed. Then, the new software modules are classified according to a predicted value of ≤ 0 or > 0. The modules with predicted value of ≤ 0 are labelled as 0, indicating that the software module is defect-free. The modules with predicted values >0 are labelled as 1, indicating that the software module is defective.

VI. THE CASE STUDIES AND COMPARISON WITH OTHER METHODS

A. CASE STUDIES AND THE COMPARISON WITH OTHER METHODS FOR THE RANKING PREDICTION

Using the steps and methods of software defect prediction for ranking prediction mentioned in V.A, we selected several datasets from PROMISE software defect database for modeling and prediction. Due to computer memory limitation, we only select 12 datasets within 1000 modules (instances) for modeling and prediction. Details of this data can be seen in the TABLE 4. The defect number 0 in each dataset is replaced by a different negative number, and then each module (instance) is identified by model (10). The evaluation indexes of the result are shown in Table 5.

The results of this method with six other methods used in the literatures [49]–[51], namely the traditional Linear Regression, Multilayer perceptron, Negative Binomial Regression, Zero-Inflated Poisson Regression, Decision Tree Regression, and Genetic Programming, are compared in the Table 6.

TABLE 4. Data set information for ranking experiments.

Group	DataSet	Number of total Modules	Buggy(%)	DataSet	Number of total Modules	Buggy(%)	Number of Metrics
	Lucene2.4	340	203(59.7%)	Xerces1.3	453	69(15.23%)	
	Poi3.0	442	281(63.57%)	Xerces1.4	588	437(74.32%)	
	sklebagd	20	12(60%)	Camel1.2	608	216(35.53%)	
	pdftranslator	33	15(45.45%)	Camel1.4	872	145(16.63%)	
PROMISE	Jedit4.2	367	48(13.08%)	Camel1.6	965	188(19.48%)	20
	Ivy2.0	352	40(11.36%)	Ant1.7	745	166(22.3%)	
	Synapse1.2	256	86(33.59%)	Xalan2.4	723	110(15.21%)	
	Velocity1.6	229	78(34.06%)	Xalan2.5	803	387(48.2%)	
	Wspornaganiepi	18	12(66.7%)	Xalan2.6	885	411(46.44%)	
	Ivy2.0	352	40(11.36%)	Xalan2.7	909	898(98.79%)	

TABLE 5. Evaluation for Predictive Results of Self-organizing Data Mining Model for 12 Data Sets (Bugs labelled in SODM=INT (Bugs calculated +0.5).

Data set	Bugs **0** replaced*	Indexes			Data set	Bugs 0 replaced*	Indexes		
		R^2	AAE	ARE			R^2	AAE	ARE
Ant-1.7	-0.5	0.728	0.327	0.140	Sklebagd	-0.5	0.903	0.010	0.033
Camel-1.6	-0.5	0.751	0.445	0.184	Pdftranslator	-0.5	0.935	0.121	0.055
Xerces-1.3	-0.5	0.926	0.251	0.114	Jedit-4.2	-0.5	0.855	0.163	0.067
Xalan-2.7	-0.5	0.69	0.216	0.079	Ivy-2.0	-0.2	0.654	0.110	0.049
Lucene-2.4	-1.0	0.849	1.061	0.346	Synapse-1.2	-0.5	0.645	0.402	0.167
Poi-3.0	-1.0	0.731	0.608	0.259	Velocity-1.6	-1.0	0.779	0.567	0.216

* Negative number of 0 bug replaced in modeling

TABLE 6. Comparison of Defect Prediction Results for Four Different Data Sets by Different Methods (Bugs labelled in SODM=INT (bugs calculated by model +0.5)).

Data set	Evaluating indexes	METHODS						
		LR	MLP	DTR	GP	NBR	ZIP	SODM*
Xerces 1.3	AAE	0.56	0.74	0.44	0.46	1.24	0.96	0.251
	ARE	0.39	0.5	0.2	0.33	1.16	0.77	**0.114**
Xerces 1.4	AAE	1.73	2.24	1.68	2.58	1.07	0.98	1.605
	ARE	0.57	0.68	0.47	1.03	0.5	0.42	0.486
Camel1.2	AAE	1.07	0.98	1.01	1.02	0.94	0.81	0.896
	ARE	0.61	0.51	0.56	0.6	0.6	0.57	**0.42**
Camel1.4	AAE	0.48	0.63	0.43	0.62	0.84	0.66	**0.325**
	ARE	0.27	0.38	0.24	0.41	0.73	0.5	**0.138**
Camel1.6	AAE	0.67	0.92	0.63	0.7	0.93	0.84	**0.445**
	ARE	0.4	0.64	0.35	0.41	0.78	0.68	**0.184**
Ant1.7	AAE	0.38	0.53	0.39	0.61	0.97	1.37	**0.327**
	ARE	0.2	0.27	0.2	0.4	0.88	1.1	**0.14**
Xalan2.4	AAE	0.23	0.33	0.25	0.23	1.18	0.92	**0.171**
	ARE	0.12	0.22	0.14	0.16	1.08	0.81	**0.078**
Xalan2.5	AAE	0.54	0.75	0.56	0.49	0.79	0.78	0.565
	ARE	0.39	0.51	0.4	0.34	0.55	0.56	**0.263**
Xalan2.6	AAE	0.53	0.63	0.52	0.68	1.01	1.06	**0.483**
	ARE	0.33	0.36	0.3	0.45	0.71	0.6	**0.213**
Average value	AAE	0.688	0.861	0.657	0.821	0.997	0.931	**0.563**
	ARE	0.364	0.452	0.318	0.459	0.777	0.668	**0.226**

*SODM represents self-organizing data mining method

It can be seen from Table 5 that the ***AAE*** and ***ARE*** of SODM are the minimum compared with other tested methods on Xerces 1.3, Camel 1.4, Camel 1.6, Ant 1.7, Xalan 2.4 and Xalan 2.6. Although the ***AAE*** of SODM method is not the smallest on Camel 1.2 and Xalan 2.5 datasets, its ***ARE*** is the smallest. The ***AAE*** and ***ARE*** average values of SODM are the smallest among the tested methods. The results demonstrate that the SODM obtains the best performance of error rate.

TABLE 7. Data set information for classifying experiments.

Group	DataSet	Number of total Modules	Buggy(%)	Number of Metrics	Group	DataSet	Number of total Modules	Buggy(%)	Number of Metrics
NASA	CM1	344	42(12.21%)	38	NASA	KC2	522	107(20.5%)	21
	MC2	127	44(34.65%)	40	SOFT-LAB	Ar1	121	9(7.44%)	29
	MW1	264	27(10.23%)	20		Ar3	63	8(12.7%)	
	KC3	200	36(18.00%)	40		Ar4	107	20(18.69%)	
	PC1	759	61(8.04%)	20		Ar5	36	8(22.22%)	
	PC2	745	16(2.15%)	36		Ar6	101	15(14.85%)	
	PC3	1125	140(12.44%)	20	SEIP	PDE_R2_0	576	111(19.27%)	48

TABLE 8. Evaluation of Prediction Results Using Self-organizing Data Mining (Defect labelled=0 when defect calculated<0; Defect labelled=1 when defect calculated≥0 in SODM).

Data set	Evaluating indexes				Data set	Evaluating indexes			
	accuracy	Pr*ecission*	Re*call*	$F-measure$		accuracy	Pr*ecission*	Re*call*	$F-measure$
Ar1	0.975	0.214	0.67	0.324	MC2	0.688	0.634	0.943	0.758
Ar4	0.84	0.548	0.85	0.667	MW1	0.88	0.744	0.537	0.624
Ar5	0.914	0.778	0.875	0.824	KC3	0.822	0.697	0.736	0.716
Ar6	0.842	0.478	0.733	0.579	PC1	0.768	0.697	0.943	0.801
KC2	0.828	0.57	0.645	0.605	PC2	0.899	0.126	0.625	0.21
CM1	0.819	0.571	0.667	0.615	PC3	0.728	0.63	0.414	0.5

TABLE 9. Comparisons between self-organizing data mining and other prediction methods (Defect labelled=0 when defect calculated<0, Defect labelled=1 when defect calculated≥0 in SODM).

Data set	Evaluating indexes	Methods						
		SVM	CC4.5	NB	CEL	CBNN	CDDL	SODM
MC2	Pd	0.51	0.64	0.35	0.56	0.79	0.83	**0.94**
	Pf	0.24	0.49	0.09	0.38	0.54	0.29	0.59
	F-measure	0.52	0.48	0.45	0.49	0.56	0.63	**0.76**
KC3	Pd	0.33	0.41	0.46	0.29	0.51	0.71	**0.74**
	Pf	0.08	0.16	0.21	0.12	0.25	0.34	0.14
	F-measure	0.38	0.38	0.38	0.33	0.38	0.44	**0.72**
MW1	Pd	0.21	0.29	0.49	0.25	0.61	0.79	0.54
	Pf	0.04	0.09	0.19	0.11	0.25	0.25	**0.04**
	F-measure	0.27	0.27	0.31	0.27	0.33	0.38	**0.62**
PC3	Pd	0.64	0.34	0.28	0.41	0.65	0.77	0.41
	Pf	0.41	0.08	0.09	0.13	0.25	0.28	0.12
	F-measure	0.28	0.29	0.29	0.36	0.38	0.42	**0.50**
CM1	Pd	0.15	0.26	0.44	0.43	0.59	0.74	0.67
	Pf	0.04	0.11	0.18	0.15	0.29	0.37	0.14
	F-measure	0.20	0.25	0.32	0.27	0.33	0.38	**0.62**
PC1	Pd	0.66	0.38	0.36	0.46	0.54	0.86	**0.94**
	Pf	0.19	0.09	0.11	0.13	0.17	0.29	0.40
	F-measure	0.35	0.32	0.28	0.32	0.32	0.41	**0.80**
Average value	Pd	0.417	0.387	0.397	0.400	0.615	0.783	0.707
	Pf	0.167	0.170	0.145	0.170	0.292	0.303	0.238
	F-measure	0.333	0.332	0.338	0.340	0.383	0.443	**0.670**

B. CASE STUDIES AND COMPARISON WITH OTHER METHODS FOR THE CLASSIFYING PREDICTION

Using the steps and methods of software defect prediction for classification prediction mentioned in V.B, we choose SOFTLAB [45] and NASA defect database to build models for prediction. Because of computer memory limitation, we only select 12 data sets within 1000 modules (instances) for modeling. Details of this data can be seen in the TABLE 7.

Firstly, the false in each dataset is replaced by a negative number less than or equal to -0.1, and the ture is replaced by a positive number greater than or equal 1.5, and then the self-organizing data mining model is established. Then, prediction results by the model is identified by Eq.(10). The evaluation index of the result is shown in Table 8.

The predicted results of SODM with those of support vector machine (SVM) [54], Compressed C4.5 decision tree (CC4.5) [55], weighted Nave Bayes (NB) [56], coding based ensemble learning (CEL) [57], cost-sensitive boosting neural network (CBNN) [58] and Dictionary learning (CDDL) [33] used in relevant literature are shown in table 9 The pd in Table 7 is the mentioned Recall in formula (4), and the pf is the error prediction rate of the flawless module, i.e. pf = FP/(FP+TN). It can be seen from the pd value in table 9 that although the CDDL obtains the best pd value (0.783) among all the methods, the SODM adopted method in this paper also obtains the 0.707 equivalent value, and the pd value of the SODM on MC2, KC3, PC1 datasets is higher than that of the other five methods. The NB obtained the best pf value (0.145), but its pd value (0.397) was lower. Although the pf value (0.238)of SODM is not best, it was better than pf values(0.292 and 0.303, respectively)of the CBNN and CDDL. In addition, although the values of pd and PF of SODM are not all good in Table 9, Compared with the average value of the comprehensive index F-measure, the index of SODM is superior to other methods.

Based on the situation, the upper integral function |**x**| (the smallest integer greater than or equal to **x**) or the lower integral function |x| (the largest integer less than or equal to x) can be used as the label function.

C. THE COMPLEXITIES OF THE SELF-ORGANIZING DATA MINING WITH TRADITIONAL APPROACH

The self-organizing data mining method based on GMDH is an algorithm that simulates the brain to identify complex nonlinear systems. This algorithm has obvious advantages in grouping data. However, when dealing with large data, the SODM algorithm is relatively weak although the fitting condition is terminated appropriately. Therefore, Wang [59] studied how to improve the computational efficiency of SODM algorithm without changing the accuracy of SODM algorithm. An algorithm that combines the inverse matrix method with the fast selection method was proposed to solve the fitting equation, and the fitting end conditions were reasonably selected and the best model was quickly selected by using extrapolation criterion. The algorithm average complexity was $O(n)$, this value is comparable to the traditional Naïve Bayes method given in [56]. The worst complexity of SODM algorithm was $O(n^2)$, compared with the complexity $O(mn \log n) + O(n(\log n)^2$ of traditional decision tree induction method given in Ref [60], the improved fast selection SODM algorithm has certain advantages.

VII. CONCLUSION

The self-organizing data-mining methods were comprehensively tested for ranking and classifying the software defective prediction. All the metric parameters of the software modules were directly provided as input into the algorithm as influencing factors to automatically establish a screening. The self-organizing modeling method was demonstrated using tests with open source datasets. Further, generalizable steps and methods for software defect prediction were proposed. According to TABLE 6 and TABLE 9, as compared with other traditional methods, the average ***AAE*** and the average ***ARE*** of the SODM approach reduces 9.4% and 9.2% respectively. The average F-measure value of SODM approach is improved by 22.7% compared with other methods. Therefore, case studies and comparisons with other methods show that self-organizing data mining is effective in predicting software defect. Self-organizing data mining provides clear mathematical explanations and graphical representations compared with other methods. Because of the causal relation between common software metrics and defects, the self-organizing data-mining methods can be used as models of complex systems of software. This method can be used to analyze the causal relation between the software defects and multiple software metrics and to extract relevant features based on the complete set of software metrics.

Similar to any other method, self-organizing data mining cannot handle all the tasks of software defect prediction. In some cases, the evaluation indices that were obtained by software defect prediction models for ranking prediction were observed to be less than ideal. The methods for selecting values with which defect-free modules can be represented during model training, for selecting the label function, and for evaluating the models require further study. Also, Self-organizing data mining methods for cross-project software defect prediction, including homogeneous cross-project ranking, homogeneous cross-project classification and heterogeneous cross-project software defect predictions, need further research.

REFERENCES

[1] L. Bettini, D. D. Ruscio, L. Iovino, and A. Pierantonio, "Quality-driven detection and resolution of metamodel smells," *IEEE Access*, vol. 7, pp. 16364–16376, Aug. 2019.

[2] S. Riaz, A. Arshad, and L. Jiao, "Rough noise-filtered easy ensemble for software fault prediction," *IEEE Access*, vol. 6, pp. 46886–46899, 2018.

[3] R. H. Chang and P. Jia, "Review of static software defect prediction based on metrics," *Fire Control Command Control*, vol. 40, no. 2, pp. 1–4, Feb. 2015.

[4] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Softw. Eng.*, vol. 13, no. 5, pp. 561–595, 2008.

[5] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.

[6] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Softw.*, vol. 12, no. 3, pp. 161–175, Jun. 2018.

[7] D. S. Wang and Y. Z. Gong, "On software defects," *Appl. Res. Comput.*, vol. 25, no. 12, pp. 3531–3533, Dec. 2008.

[8] E. Arthur, Ferdinand, "A theory of system complexity," *Int. J. Gen. Syst.*, vol. 1, no. 1, pp. 19–33, Jan. 1974.

[9] L. M. Ottenstein, "Quantitative estimates of debugging requirements," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 5, pp. 504–514, Sep. 1979.

[10] M. Lipow, “Number of faults per line of code,” *IEEE Trans. Softw. Eng.*, vol. SE-8, no. 4, pp. 437–439, Jul. 1982.

[11] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Where the bugs are,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 24, pp. 86–96, 2004.

[12] T. M. Khoshgoftaar and E. B. Allen, “Ordering fault-prone software modules,” *Softw. Qual. J.*, vol. 11, no. 1, pp. 19–37, Jan. 2003.

[13] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Predicting the location and number of faults in large software systems,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, Apr. 2005.

[14] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas, “Regression via Classification applied on software defect estimation,” *Expert Syst. Appl.*, vol. 34, no. 3, pp. 2091–2101, Apr. 2008.

[15] Z. Yan, X. Y. Chen, and P. Guo, “Software defect prediction using fuzzy support vector regression,” *Proc. 7th Int. Conf. Adv. Neural Netw.*, Shanghai, China, 2010, pp. 17–34.

[16] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, “Comparing the effectiveness of several modeling methods for fault prediction,” *Empirical Softw. Eng.*, vol. 15, no. 3, pp. 277–295, Jun. 2010.

[17] V. R. Basili, L. C. Briand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996.

[18] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 675–689, Sep. 1999.

[19] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[20] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic review of fault prediction performance in software engineering,” *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Oct. 2011.

[21] T. C. Liang, “A software defect prediction model based on Naive Bayesian method of public immunity,” M.S. thesis, Dept. Comput., Nanjing Univ. Posts Telecommun., Nanjing, China, 2015.

[22] L. Guo, Y. Ma, B. Cukic, and H. Singh, “Robust prediction of fault-proneness by random forests,” in *Proc. IEEE 15th Int. Conf. Softw Rel. Eng.*, Nov. 2004, pp. 417–428.

[23] X. X. Yang, “Metrics-based software defect prediction,” Ph.D. dissertation, Dept. Inf. Sci. Technol., Univ. Sci. Technol. China, Hefei, China, 2013.

[24] Y. Ran, “Research on software defect prediction model based on SAPSO-BP,” M.S. thesis, Dept. Comput. Inf. Sci., Southwest Univ., Chongqing, China, 2014.

[25] C. X. Chen, “Research on software defect prediction based on classifiers fusion technology,” M.S. thesis, Dept. Comput. Commun., China Univ. Petroleum, Beijing, China, 2015.

[26] N. S. Wang, “The research of software defect prediction model based on support vector machine optimized by genetic algorithm,” M.S. thesis, Dept. Comput. Sci. Technol., Beijing Inst. Technol., Beijing, China, 2015.

[27] X. T. Rong, “Research on static multi-objective software defect prediction strategy,” M.S. thesis, Dept. Comput. Sci. Technol., Taiyuan Univ. Sci. Technol., Taiyuan, China, 2017.

[28] L. N. Qin, “Research on static prediction of software defect,” M.S. thesis, Dept. Comput. Sci., Central China Normal Univ., Wuhan, China, 2011.

[29] H. Qiao, “Research on software defect prediction techniques,” M.S. thesis, Dept. Cryptogr. Eng., PLA Inf. Eng. Univ., Zhengzhou, China, 2013.

[30] X. S. Jiang, “Fish optimization of multi-kernel SVM in the application of the software defect prediction,” M.S. thesis, Dept. Comput. Inf. Sci., Chongqing Normal Univ., Chongqing, China, 2016.

[31] D. P. Zhang, G. Q. Liu, and K. Zhang, “Software defect prediction model based on GMDH causal relationship,” *Comput. Sci.*, vol. 43, no. 7, pp. 171–175, Jul. 2016.

[32] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, “An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 321–339, Apr. 2017.

[33] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, “Dictionary learning based software defect prediction,” in *Proc. IEEE Int. Conf. Softw. Eng.*, Jun. 2014, pp. 414–423.

[34] Z. Q. Li, X. Y. Jing, F. Wu, X. K. Zhu, B. W. Xu, and S. Ying, “Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction,” *Autom. Softw. Eng.*, vol. 25, no. 2, pp. 201–245, Aug. 2017.

[35] Z. Li, X.-Y. Jing, and X. Zhu, “Heterogeneous fault prediction with cost-sensitive domain adaptation,” *Softw. Test. Verification Rel.*, vol. 28, no. 2, p. e1658, Jan. 2018.

[36] W. Fei, X.-Y. Jing, S. Ying, S. Jing, and Y. Sun, “Cross-project and within-project semisupervised software defect prediction: A unified approach,” *IEEE Trans. Rel.*, vol. 67, no. 2, pp. 581–597, Jun. 2018.

[37] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, “On the multiple sources and privacy preservation issues for heterogeneous defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 391–411, Apr. 2019.

[38] C. Z. He, *Self-Organizing Data Mining Algorithms of Self-Organization Data Mining and Economic Forecast*, 1st ed. Beijing, China: Science in China Press, 2005, pp. 40–49.

[39] J. A. Mueller and F. Lemke, “Self-organising data mining: An intelligent approach to extracting knowledge from data,” *Syst. Anal. Model. Simul.*, vol. 43, no. 2, pp. 231–240, Feb. 2003.

[40] A. G. Ivankhnenko and J. V. Koppa, “The group method of data handling for the solution of the various interpolation problems of cybernetics,” in *Proc. IFAC Symp. Identificat. Parameter Estimation*, 1970, pp. 66–75.

[41] P. A. Karnazes and R. D. Bonnell, “System identification techniques using the group method of data handling,” *IFAC Proc. Volumes*, vol. 15, no. 4, pp. 622–627, 1982.

[42] G. Denaro and M. Pezze, “An empirical evaluation of fault-proneness models,” in *Proc. 24th Int. Conf. Softw. Eng.*, Orlando, FL, USA, 2002, pp. 241–251.

[43] K. Gao and T. M. Khoshgoftaar, “A comprehensive empirical study of count models for software fault prediction,” *IEEE Trans. Rel.*, vol. 56, no. 2, pp. 223–236, Feb. 2007.

[44] J. Marian and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, Timioara, Romania, Sep. 2010, Art. no. 9.

[45] B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, “The PROMISE repository of empirical software engineering data,” Ph.D. dissertation, Dept. Comput. Sci., West Virginia Univ., Morgantown, WA, USA, 2012.

[46] J. Y. He, “Search based semi-supervised ensemble learning research for cross-project defect prediction,” M.S. thesis, Dept. Softw., Tianjin Univ., Tianjin, China, 2016.

[47] M. Goran, G. G. Tihana, and D. B. Bojana, “A systematic data collection procedure for software defect prediction,” *Comput. Sci. Inform. Syst.*, vol. 13, p. 61, Jan. 2016.

[48] F. Lemke, “Selforganize!—A software tool for modelling and prediction of complex systems,” *Syst. Anal. Model. Simulat.*, vol. 20, nos. 1–2, pp. 17–27, Jan. 1995.

[49] *Knowledge Miner*. Accessed: 2016. [Online]. Available: http://www.knowledgeminer.net

[50] S. S. Rathore and S. Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction,” *Soft. Comput.*, vol. 21, no. 24, pp. 7417–7434, Jul. 2016.

[51] S. S. Rathore and S. Kumar, “A decision tree regression based approach for the number of software faults prediction,” *Softw. Eng. Notes*, vol. 41, no. 1, pp. 1–6, Feb. 2016.

[52] S. S. Rathore and S. Kumar, “Predicting number of faults in software system using genetic programming,” in *Proc. Int. Conf. Soft Comput. Softw. Eng.*, vol. 62, 2015, pp. 303–311.

[53] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

[54] K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support vector machines,” *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.

[55] J. Wang, B. J. Shen, and Y. T. Chen, “Compressed C4.5 models for software defect prediction,” in *Proc. IEEE Int. Conf. Qual. Softw.*, Xi’an, China, Aug. 2012, pp. 13–16.

[56] T. Wang and W. H. Li, “Naïve Bayes software defect prediction model,” *Proc. IEEE Int. Conf. Comput. Intell. Softw. Eng.*, Wuhan, China, Dec. 2010, pp. 1–4.

[57] Z. B. Sun, Q. B. Song, and X. Y. Zhu, “Using coding-based ensemble learning to improve software defect prediction,” *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1806–1817, Nov. 2012.

[58] J. Zheng, “Cost-sensitive boosting neural networks for software defect prediction,” *Expert Syst. Appl.*, vol. 37, no. 6, pp. 4537–4543, Jun. 2010.

[59] A. H. Wang, "Research on the method of rapid processing data packets," *J. Chongqing Normal Univ.*, vol. 32, no. 4, pp. 113–117, Jul. 2015.

[60] L. Witten, E. Frank, and M. Hall, *Data Mining Practical Machine Learning Tools and Techniques*, 3rd ed. Amsterdam, The Netherlands: Elsevier, 2010, pp. 132–133.

JUN-HUA REN received the B.S. degree in computer science and technology from the Century College of Beijing Posts and Telecommunications University, Beijing, China, in 2010, and the M.S. degree in software engineering from the Chinese Academy of Sciences, Beijing, in 2013. She is currently pursuing the Ph.D. degree with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing. She has published over eight journals and conference papers. Her current research interest includes software testing.

FENG LIU received the B.S. degree in computer software from Beijing Jiaotong University, Beijing, in 1983, and the Ph.D. degree in economics from the Renmin University of China, Beijing, in 1997. He is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University, and the Director of the Engineering Research Center of Network Management. He is also a Visiting Professor with the Computer Science Institute, Chinese Science Academy. He is also a member of the Editorial Board of the *Journal of Computer Research and Development*. He has authored or coauthored seven books and has published over 72 articles in journals and at conferences. His research interests primarily include computer technology, software engineering, electronics, and information.