Hindawi
Complexity
Volume 2021, Article ID 6660830, 12 pages
https://doi.org/10.1155/2021/6660830

Research Article

Failure Analysis of Static Analysis Software Module Based on Big Data Tendency Prediction

Jian Zhu,[1] **Qian Li**,[2] **and Shi Ying**[1]

[1]*School of Computer Science, Wuhan University, Wuhan 430072, China*
[2]*School of Computer and Information Engineering, Guangxi Vocational Normal University, Nanning 530007, China*

Correspondence should be addressed to Shi Ying; 3106109784@qq.com

Received 3 December 2020; Revised 2 February 2021; Accepted 7 March 2021; Published 26 March 2021

Academic Editor: M. Irfan Uddin

With the continuous development of software, it is inevitable that there will be various unpredictable problems in computer software or programs that will damage the normal operation of the software. In the paper, static analysis software is taken as the research object, the errors or failures caused by the potential defects of the software modules are analyzed, and a software analysis method based on big data tendency prediction is proposed to use the software defects of the stacked noise reduction sparse analyzer to predict. This method can learn features from original defect data, directly and efficiently extract required features of all levels from software defect data by setting different number of hidden layers, sparse regularization parameters, and noise ratio, and then classify and predict the extracted features by combining with big data. Through experimental tests, the performance of the presented method is better than that of the comparison method in correct rate, accuracy rate, recall rate, F1-measurement, AUC value, and running time, which proves that the research results in this paper have more accurate failure prediction effect and can timely eliminate software failures.

1. Introduction

The application of software almost permeates every aspect of people's life. With the rapid development of the software industry, the increasing demand, the integration of functions, and the application of plug-ins make the scale of software larger and larger and also make the software more and more complex. Software failures may cause serious economic losses to enterprises and even threaten people's lives [1]. In this paper, combining with the encoder model in propensity prediction, aiming at the problems of slow convergence speed of loss cost function and sparsity regularization parameters and complexity and difficulty of parameter adjustment in traditional encoder, the loss cost function and sparsity constraint method of encoder are improved. At the same time, in order to reduce the influence of noise on data, the method can learn features from original defect data. By setting different number of hidden layers, sparse regularization parameters, and noise adding ratio, the required hierarchical features are directly and efficiently extracted from the software defect data, and then the extracted features are classified and predicted with logistic regression classifier [2–4]. Since the development of software defect prediction technology, it has been one of the research hotspots in the field of software engineering. It is also called two key technologies to improve software quality and reliability together with software defect detection technology in which the software defect detection technology is mainly to analyze the program modules that have failed to determine the specific location of the defect; the software defect prediction technology mainly measures the program modules that have not failed and predicts whether the module contains undiscovered defects through the constructed software defect prediction model.

In recent years, many domestic and foreign research scholars have devoted themselves to the research of software defect prediction technology and have achieved quite excellent research results. At present, according to different technologies, the existing software defect prediction technology can be simply divided into dynamic defect prediction technology and static defect prediction technology where dynamic defect prediction technology is to study the life cycle of the entire software system and predict the distribution of software defects over time based on the time when the software failure or system failure occurs; the static defect prediction technology is based on measurement data related to software defects such as the size and loop complexity of the software system to predict the defect tendency, defect density, or number of defects in software program modules.

The proposed software defect prediction method is used to predict the defects of the big data modeling platform system. Firstly, based on the documents and program codes in the development process of the big data modeling platform system, the program modules of the system are extracted, and the metric elements are designed and measured to obtain the characteristic data. Secondly, some program modules are randomly selected from all the program modules for manual testing and labeling. Then, the defect data set of labeled program modules is used as the training set to build a software defect prediction model to predict the defect tendency of unlabeled program modules [5, 6].

2. An Encoder Based on Propensity Prediction Mechanism

2.1. Loss Cost Function Based on Cross Entropy. In order to overcome the problem of low parameter update efficiency when using the cost function of square deviation in traditional encoders, it is hoped that the partial derivative of the loss cost function is independent of the derivative of the activation function [7], namely:

$$\begin{aligned}\frac{\varepsilon L}{\varepsilon w} &= (z_i - x_i)y_i,\\ \frac{\varepsilon L}{\varepsilon b} &= (z_i - x_i).\end{aligned} \tag{1}$$

In formula (1), ε is the sparse parameter, w is the weight matrix, b is the bias vector, l is the activation function, and x_i is the training sample set. For each sample i of the input layer, the feature y_i of the hidden layer of the sample can be obtained through coding operation and z_i is the reconstructed data of the sample [8, 9].

Taking $(\varepsilon L/\varepsilon b) = (z - x)$, for example,

$$\frac{\varepsilon L}{\varepsilon b} = \frac{\varepsilon L}{\varepsilon z}\cdot\frac{\varepsilon z}{\varepsilon y}\cdot\frac{\varepsilon L}{\varepsilon b} = \frac{\varepsilon L}{\varepsilon z}(1 - z). \tag{2}$$

Applying the cross-entropy function to the loss cost function of the encoder, the cross-entropy cost function can be expressed as follows:

$$\begin{aligned} l(x,z) &= -\frac{1}{m}\sqrt{\sum_{i=1}^{m}(x_i \ln z_i + (1-x_i)\ln(1-z_i))}\\ &= \frac{1}{m}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m}\left(x_i^j \ln z_i^j + \left(1-x_i^j\right)\ln\left(1-z_i^j\right)\right)}\end{aligned}. \tag{3}$$

Its partial derivative is expressed as follows:

$$\begin{aligned}\frac{\varepsilon L}{\varepsilon w} &= -\frac{1}{m}\sum_{i=1}^{m}\sqrt{\frac{x_i - z_i}{z_i(1-z_i)}\frac{\varepsilon f}{\varepsilon w}}\\ &= \frac{1}{m}\sum_{i=1}^{m}(z_i - x_i)y_i,\end{aligned} \tag{4}$$

$$\begin{aligned}\frac{\varepsilon L}{\varepsilon b} &= -\frac{1}{m}\sum_{i=1}^{m}\sqrt{\frac{x_i - z_i}{z_i(1-z_i)}\frac{\varepsilon f}{\varepsilon b}}\\ &= \frac{1}{m}\sum_{i=1}^{m}(z_i - x_i).\end{aligned} \tag{5}$$

In formula (4), f is the bias vector.

Compared with the square variance cost function, the cross-entropy cost function has obvious advantages; its partial derivative is independent of the derivative of the activation function, so it will not be affected by the saturation of sigmoid function. When the loss is large, the weight will be updated quickly; when the error is small, the weight will be updated slowly. Same as the cost function of square deviation, the cost function of cross entropy also has two properties:

(1) Nonnegative: within the scope of the definition domain, its value is nonnegative

(2) The smaller the difference between reconstructed data z_i and input data x_i is, the more its cost function approaches to 0 [10–12]

2.2. Sparse Constraint Method Based on L1 Rule. By penalizing nonzero activation of hidden neurons in the encoder, this paper proposes a sparse encoder (L1-SAE) based on the L1 rule for sparse constraint [13].

The sparse encoder based on L1 rule does not force all hidden neurons to share the same degree of sparsity but directly applies the average activation of hidden neurons to the sparsity constraint [14]. The expression of L1 rule is as follows:

$$\sum_{j=1}^{\delta}\rho_j = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{\delta}\sqrt{\alpha_j(x_i)^2}. \tag{6}$$

In formula (6), ρ is a sparsity parameter, which is a small value close to 0; $\alpha_j$represents the activation degree of hidden neuron j under the given input of x; and δ is the number of hidden layer neurons in the encoder [15, 16].

The overall cost function of the sparse encoder based on L1 rule is as follows:

$$\tau_{l1-\mathrm{SAE}}(\lambda,\lambda') = \tau_{\mathrm{AE}}(\lambda,\lambda') + \beta\sum_{j=1}^{\delta}\rho_j. \tag{7}$$

In formula (7), $\tau_{l1-\mathrm{SAE}}(\lambda,\lambda')$ is the loss cost of the encoder, $\tau_{\mathrm{AE}}(\lambda,\lambda')$ is the second term for the sparsity penalty term, and the sparsity regularization parameter β is the weight of the sparsity penalty item to control the relative importance of the sparsity penalty item.

In the sparse encoder based on KL divergence, two super parameters need to be set in advance: sparse regularization parameter β and sparsity parameter ρ, while in the sparse encoder based on L1 rule, only one super parameter needs to be set in advance: sparse regularization parameter β. Using fewer super parameters can significantly reduce the time needed to adjust model parameters [17–19].

The sparse encoder based on L1 rule has the following advantages:

(1) L1 rule is a convex quadratic optimization problem, which can be well realized and solved

(2) The sparse degree of hidden neurons in the sparse encoder can be learned

(3) Using fewer super parameters can significantly reduce the difficulty of parameter adjustment during training the model [20–22]

2.3. Improved Denoising Sparse Autoencoder (DSAE). An improved denoising sparse autoencoder (DSAE) is presented in this paper. First, noise processing is performed on the original data, and then sparse coding training is performed on the noise data after noise addition, which makes the encoder to learn to remove noise and obtain the original data without noise pollution, forcing the encoder to learn more robust representation of the original data and improving the generalization ability of the encoder [23–25].

The improved DSAE structure is shown in Figure 1.

In Figure 1, x_i is the original data, $\tilde{x}_i$ is the noise data after noise processing, y_i is the feature of hidden layer, and z_i is the reconstructed data. There are usually two ways to add noise as follows:

(1) Gaussian noise (GN): noise obeying Gaussian distribution is added to sample x_i;

(2) Masking noise (MN): some values in sample x_i are randomly selected and set to 0; the loss cost between reconstructed data and input data of the improved DSAE model is as follows:

$$\begin{aligned} l(x_i, z_i) &= l[x_i, f(w' y_i + b')] \\ &= l[x_i, f(w' f(\tilde{w}_i + b) + b']. \end{aligned} \quad (8)$$

Then, the loss cost function of the improved DSAE in the whole training sample set is as follows:

$$j(\lambda, \lambda') = \frac{1}{m}\sum_{i=1}^{m} l(x_i, z_i) + \frac{\theta}{2}\left(\|w\|^2 + \|w'\|^2\right) + \beta \sum_{j=1}^{\delta} \rho_j. \quad (9)$$

When the DSAE is trained, the back propagation algorithm and gradient descent method are used to iterate and update the network parameters so as to learn the encoder with optimal network parameters [26]. The specific algorithm of training encoder is shown in Algorithm 1.

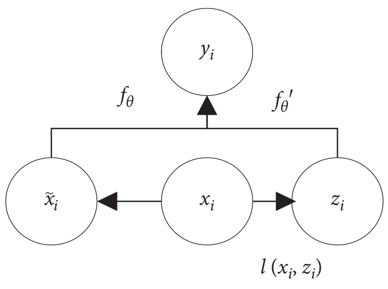


Figure 1: Structure of denoising sparse autoencoder (DSAE).

The algorithm flow of training encoder is shown in Figure 2.

The algorithm flow of training the analyzer is both left-saturated and right-saturated. Therefore, the sigmoid function is saturated. And if and only when its value approaches, the derivative is 0, so the sigmoid function is soft saturated.

The advantages of the sigmoid function are as follows:

(1) The input data are compressed to (0, 1), and the sigmoid function has monotonic continuity, optimization, and stability

(2) The derivative is easy to implement

However, the sigmoid function also has disadvantages that cannot be ignored:

(1) The amount of calculation is too large. When using the backpropagation algorithm to calculate the gradient, the derivation process involves division;

(2) When the input data are very large or very small, due to its soft saturation, the gradient disappears and the training of the deep network cannot be completed.

(3) Its output is not 0 as the mean value, which will allow the neurons in the next layer to get the nonzero mean value of the output of the previous layer as input, so that the trained network parameters tend to be all positive or all negative. As a result, the z-shaped drop occurs when the gradient descent method is used to optimize the network parameters.

3. Software Defect Prediction Based on Stacked Denoising Sparse Autoencoder (sDSAE)

3.1. sDSAE. Multiple improved denoising sparse encoders are stacked layer by layer to build a deep neural network model, and an improved stacked denoising sparse autoencoder (sDSAE) can be obtained, which can obtain deeper feature information of the input data. The feature information acquired by the deeper level has the stronger feature expression ability [27].

Figure 3 shows a three-layer sDSAE structure based on the stacked improved denoising sparse encoder.

In the training sDSAE, the characteristics of the first layer of the encoder are taken as the input of the second layer of the encoder, and the greedy training method of layer by layer is adopted to train each layer of the deep neural

```
Input: training data $x$, the number of input layer nodes inputSize, the number of hidden layer nodes hiddenSize, the weight attenuation
    coefficient $\theta$, the sparse regularization constant $\beta$, the maximum number of iterations of the optimization loss cost function maxIter
Output: optimal network parameters $w$, $b$
 (1) Initialize the encoder's network parameters: weight matrix $(w, w')$, bias vector $(b, b')$
 (2) Initialize iteration number$i = 1$, overall cost = 0;
 (3) Noise processing is carried out on training data $x$ to obtain noise data $\tilde{x}_t$;
 (4) for $i = 1$: max Iter optimizes the network parameters of the encoder by iterative methods
 (5) By coding the noise data $\tilde{x}_t$, the feature $y$ of the hidden layer is obtained.
 (6) In the decoding operation, the feature $y$ of the hidden layer is decoded to obtain the reconstructed data $z$;
 (7) Calculate the overall cost of the encoder;
 (8) Calculate the partial derivative$\Delta w \Delta b$ of the loss cost function
 (9) Calculate the gradient of network parameters $\forall w, \forall w', \forall b, \forall b'$;
(10) Update network parameters $w, w', b, b'$;
(11) End for loop
(12) Output the optimal network parameters $w$ and $b$.
```

Algorithm 1: Specific algorithm of training encoder.

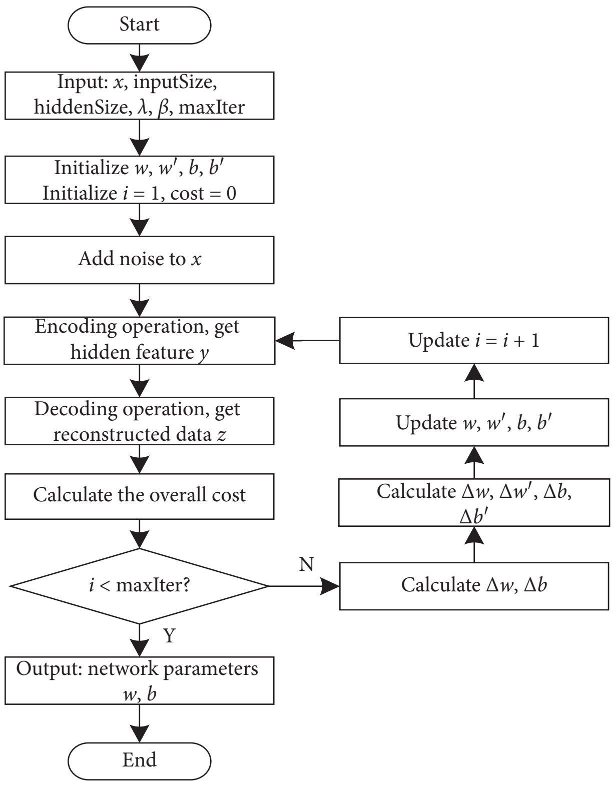

Start
Input: x, inputSize, hiddenSize, λ, β, maxIter
Initialize w, w', b, b' Initialize $i = 1$, cost = 0
Add noise to x
Encoding operation, get hidden feature y
Decoding operation, get reconstructed data z
Calculate the overall cost
$i <$ maxIter?
N
Y
Calculate Δw, Δb
Calculate Δw, $\Delta w'$, Δb, $\Delta b'$
Update w, w', b, b'
Update $i = i + 1$
Output: network parameters w, b
End

Figure 2: Algorithm flow of training encoder.

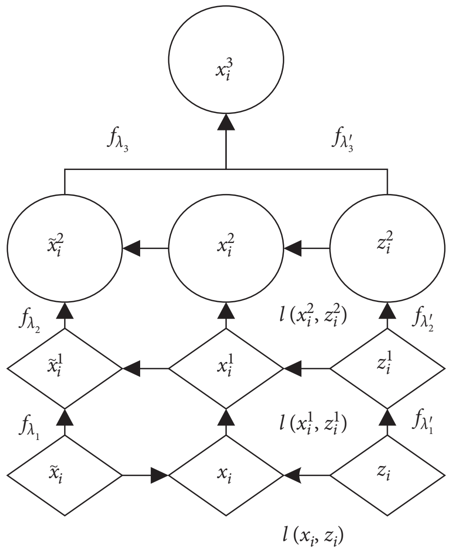


Figure 3: Three-layer sDSAE structure.

network successively, and then the entire deep neural network is trained to obtain network parameters. In other words, firstly, the original training data x_i are used as the input of the first-order analyzer to train the first-order feature x_i^1 of the original data, and then the first-order feature x_i^1 is used as the input of the second-order analyzer to train the second-order feature x_i^2 of the original data, and so on. Taking the $(n-1)$ feature x_i^{n-1} as the input of the nth level analyzer, the n-order feature x_i^n of the original data can be trained [28].

Assuming that a deep neural network model is composed of sDSAE and logistic regression classifier, the training of the whole model can be divided into two processes: unsupervised pretraining process and supervised fine-tuning process. Among them, the specific steps to realize the supervised fine-tuning process are as follows:

Step 1. A feedforward pass is performed, and a layer-by-layer greedy algorithm is used to train the *2L* layer and *3L* layer of the sDSAE until the output layer L_{ln} to obtain the network parameters of all layers.

Step 2. The loss cost is calculated between the classification result of the output layer logistic regression classifier and the corresponding label of the input data.

Step 3. The residual δ_{ni} is calculated for the output layer, and the calculation formula is as follows:

$$\delta_{ni} = -(\forall a_{ni} \cdot j) \cdot f'(z_{ni}). \tag{10}$$

The characteristic representation of the last layer of the sDSAE will be input to the output layer; that is, the output layer is the classifier, so the derivation process needs to be handled separately. In the logistic regression classifier, I is used to represent the label corresponding to the input data and Q is used to represent the conditional probability vector, and then $\forall j = \lambda^t (i - Q)$ [29] is used in the formula.

Step 4. The residuals for all hidden layers are calculated in the sDSAE; let $l = n, -1, \quad n, -2, \quad l, 2$, then

$$\delta_l = \left[(w')^t \delta_{l+1} \right] \cdot f'(z_i). \tag{11}$$

Step 5. Partial derivatives are calculated by using residuals as follows:

$$\begin{aligned} \forall_{w^t} j(\lambda, \lambda') &= \delta_{l+1} (\alpha_l)^t, \\ \forall_{b^t} j(\lambda, \lambda') &= \delta_{l+1}. \end{aligned} \tag{12}$$

Step 6. Network parameters are updated as follows:

$$\begin{aligned} \Delta w^t &= \Delta w^l + \forall_{w^l} j(\lambda, \lambda'), \\ \Delta b^t &= \Delta b^l + \forall_{b^l} j(\lambda, \lambda'). \end{aligned} \tag{13}$$

The above is an iterative step in the fine-tuning process. Through multiple iterations and updates, fine-tuning can obtain better network parameters and improve network performance.

The algorithm flow of training the sDSAE is shown in Figure 4.

The training stack noise reduction sparse analyzer has significant effects in the feature space related models (such as image matching and speech recognition), while the unrelated models in the feature space (such as software defect prediction and text classification) may loss important information. The reason is that for models that are unrelated in the feature space, multigranularity scanning reduces the importance of features at both ends of the feature space to a certain extent. In multigranularity scanning, both the first feature and the last feature are scanned only once; that is, both features are used only once. If the importance of the first feature or the last feature is very high, multigranularity scanning cannot effectively use this important feature.

3.2. Software Defect Prediction Method Based on sDSAE. In this paper, the sDSAE with four hidden layers is designed for feature extraction of software defect data, and a software defect prediction model is used to classify and predict extracted defect characteristics with logistic regression classifier. Its model structure is shown in Figure 5 [30].

The entire deep network including the logistic regression classifier is fine-tuned to obtain the optimal network parameters. The overall algorithm flow of software defect prediction based on sDSAE is as follows. (Algorithm 2).

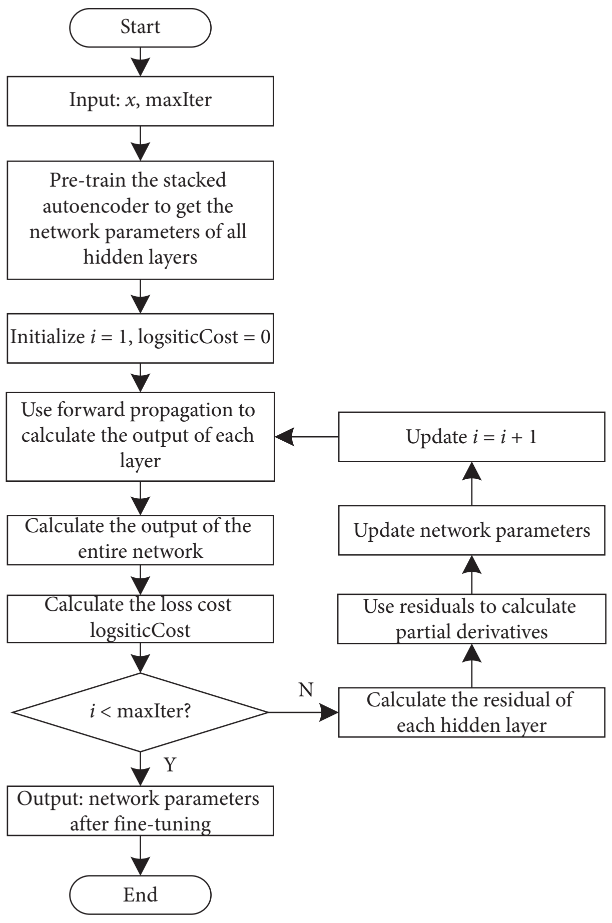

Start
Input: x, maxIter
Pre-train the stacked autoencoder to get the network parameters of all hidden layers
Initialize i = 1, logsiticCost = 0
Use forward propagation to calculate the output of each layer
Update i = i + 1
Calculate the output of the entire network
Update network parameters
Calculate the loss cost logsiticCost
Use residuals to calculate partial derivatives
N
i < maxIter?
Calculate the residual of each hidden layer
Y
Output: network parameters after fine-tuning
End

FIGURE 4: Algorithm flow of training sDSAE.

4. Experimental Setup and Result Analysis

4.1. Experimental Data Set. In this paper, the performance of the software defect prediction model is evaluated by using eclipse defect data set, which is one of the most widely studied public data sets in the field of software defect prediction and can be available from EclipseBugData.

There are six ARFF files in the Eclipse data set, corresponding to the defect records of the three versions (Eclipse 2.0, Eclipse 2.1, and Eclipse 3.0) of the Eclipse defect data set at two granularities (files, packages). The defect data records are divided into prerelease defects and postrelease defects; prerelease defects refer to defects found during the development process, and postrelease defects are defects found during the user's use phase. This experiment uses three versions of defect data records under the granularity of files and takes the defect tendency of the program modules after the software release as the prediction target. The class label hasDefects, which converts the defect number to the meaning of whether a software module has defects, is as follows:

$$\text{hasDefects} = \begin{cases} 0, & \text{post} = 0, \\ 1, & \text{otherwise.} \end{cases} \tag{14}$$

The statistical information of the Eclipse defect data set files granularity is shown in Table 1.

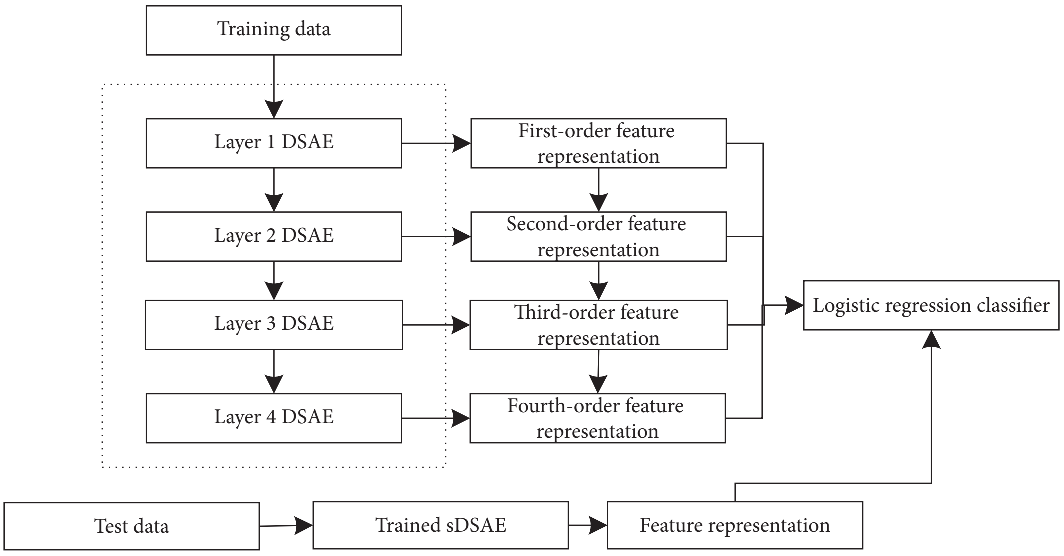

Training data
Layer 1 DSAE
Layer 2 DSAE
Layer 3 DSAE
Layer 4 DSAE
First-order feature representation
Second-order feature representation
Third-order feature representation
Fourth-order feature representation
Logistic regression classifier
Test data
Trained sDSAE
Feature representation

Figure 5: Software defect prediction model structure based on sDSAE.

```
Input: training data defectData, label defectLabel corresponding to training data, test data testData, number of input layer nodes inputSize of sDSAE, number of hidden layer nodes hiddenSizeL1, hiddenSizeL2, hiddenSizeL3, hiddenSizeL4, weight attenuation coefficient lambda, sparse regularization parameter beta, masking noise the masking rate noiseRatio, the maximum number of iterations maxIter to minimize the loss cost function, the weight attenuation coefficient of logistic regression classifier LogisticLambda, the maximum number of iterations LogisticMaxIter.
Output: predicted defect tendency label predLabel, predicted defect tendency probability value predScore.
(1) The software defect data defectData are preprocessed to obtain the processed training data trainData. The preprocessing process mainly includes removing invalid data and data standardization. The data standardization process refers to the process of making the training data conform to the standard normal distribution;
(2) Take the processed training data trainData as the input of the first layer of sDSAE and train to obtain the first-order feature sae1Features of the defect data;
(3) Take the first-order feature sae1Features of the defect data as the input of the second layer of sDSAE and train to obtain the second-order feature sae2Features of the defect data;
(4) Similar to Step 3, the third-order features sae3Features and fourth-order features sae4Features of the defect data can be obtained, respectively;
(5) The labels of each order feature and the software defect data obtained from Steps 2–4 are used as the logistic regression classification The input of the processor to construct a software defect prediction model
(6) "Fine-tuning" the constructed prediction model through the back propagation algorithm and gradient descent method to optimize the network parameters of each prediction model;
(7) The test data testData are preprocessed in the same way as the training data and then input to each trained prediction model to obtain the probability value predScore of the predicted defect tendency;
(8) If predScore ≥ 0.5, then predLabel = 1; otherwise, predLabel = 0.
```

Algorithm 2

4.2. Experimental Environment and Methods. The experimental environment is shown in Table 2.

In this experiment, because the feature number of defect data set is 200, the number of input layer nodes is set to 200; since there is no uniform rule for the depth selection of the model, which is usually determined by the experimental data and task requirements, the number of hidden layer nodes and hidden layer nodes are set according to the specific situation in the experiment; the weight attenuation coefficient lambda is set to $1e-3$. In order to make the symmetry of analyzer invalid and get better training effect, the weight matrix of analyzer is usually initialized randomly instead of all zero.

In the experiment, the network parameter optimization method is designated as L-BFGS, which uses the quasi-Newton method and limited memory BFGS algorithm to update the weights and limits the maximum number of iterations maxIter to 400. The weight attenuation coefficient LogisticLambda of the logistic regression classifier is set to 1e-4, and the loss cost function is also optimized by minFunc, the optimization method is L-BFGS, and the maximum number of iterations LogisticMaxIter is set to 100.

Table 1: Statistical information of Eclipse defect data set files granularity.

Data set	Feature number	Number of samples	Number of defects	Defect rate (%)
Eclipse2.0	200	6854	725	10.58
Eclipse2.1	200	5578	695	12.46
Eclipse3.0	200	6940	763	10.99

Table 2: Experimental environment configuration.

Configuration	Model
CPU	Intel i7-10700, 8 cores 16 threads 4.8 GHz
RAM	32G DDR4-3200
Operating system	Windows 7 enterprise edition
Application	MATLABR R2020a

The hyperparameters of the deep stacked forest algorithm are set according to the settings of the deep forest algorithm, as shown in Table 3.

The difference is that in the deep stacked forest, when sampling randomly, three scales and three times are used to sample the original features. The sampling scales are ($d/18$, $d/10$, $d/5$). The corresponding sampling times are 200, 100, and 50, respectively.

Since the file-level data of the Eclipse defect data set are the same in structure, one version of the defect data is taken as the training data set training model. First, the Eclipse file-level defect data sets are preprocessed; that is, the number of module defects post is converted to whether there are defects in the class label hasDefects. Then, the gcForest algorithm is used to construct a software defect prediction model and classify and predict the training set 9 times. Finally, the DSF algorithm is used to build a software defect prediction model and classify to predict. When the training data set and the test data set are from the same version, ten-fold cross-validation is used.

4.3. Experimental Results. An encoder with only one hidden layer is used to construct a software defect prediction model. The number of hidden layer nodes of the encoder is set to 100. The loss cost function uses the squared difference cost function and the cross-entropy cost function, respectively, and the prediction results have high correct rate, accuracy rate, recall rate, F1-measure, AUC value, and running time (average running time is taken during ten-fold cross-validation). The experimental data results are shown in Table 4, and the comparative test results are shown in Figure 6.

The encoder is used to extract the features of software defect data. There is no need to define the features in advance but only input the defect data into the network. The encoder will learn to obtain the feature representation of the defect data, and the obtained feature representation will be classified and predicted by logistic regression classifier, which can achieve good prediction effect.

It shows that the prediction model using the square difference cost function and the cross-entropy cost function is basically the same in the prediction accuracy rate and remains above 0.8, indicating that the prediction models using the two cost functions have high predictive capabilities; The prediction model using the cross-entropy cost function is better than that using the square error cost function, the prediction model of the function has advantages in prediction accuracy, recall, F1-measure, AUC value, and running time, and the running time of the prediction model using the cross-entropy cost function is only about 1/3 of the running time of the prediction model using the square deviation cost function.

In the experiments, the deep stacked forest algorithm mainly studies the influence of tree number and model depth on predictive model performance in the stacked forest structure and the effects of random sampling and stacked forests on the performance of prediction models in the deep stacked forest.

4.3.1. The Effect of the Depth of the Stacked Forest and the Number of Trees on the Performance of the Prediction Model in the Deep Stacked Forest Algorithm. In order to verify the influence of the depth of the stacked forest and the number of trees on the performance of the prediction model, a software defect prediction model based on the stacked forest is constructed where the depth of the stacked forest is one, two, three, and four layers and the number of trees is 500. The prediction results are compared on the correct rate, accuracy rate, recall rate, and running time (the average running time was taken during ten-fold cross-validation). The experimental results are shown in Figure 7.

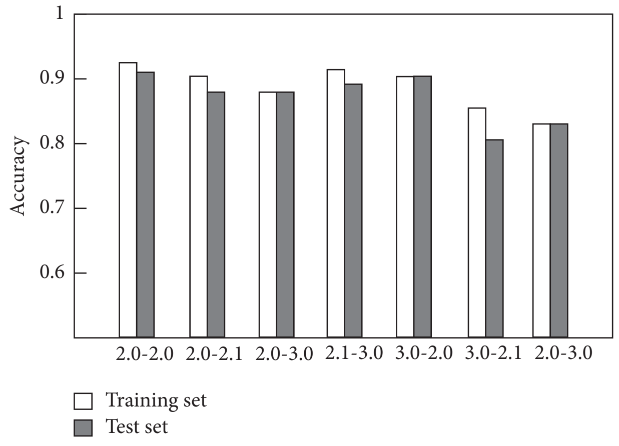
In Figure 7, for stacked forests of the same depth, when the number of trees in the forest is less than 200, the prediction accuracy and accuracy rates are basically increasing, and the prediction recall rate is basically decreasing; when the number of trees in the forest is greater than 200, its predictions of correct rate, accuracy rate, and recall rate are all stable. When the number of trees in the forest is less than 200, the learning ability of the random forest and the completely random tree forest in the stacked forest increases as the number of trees increases, so that the stacked forest can learn more detailed data information. Therefore, the performance of the prediction model increases as the number of trees in the forest increases. When the number of trees in the forest is greater than 200, the learning ability of the random forest and the completely random tree forest in the stacked forest has reached saturation, and increasing the number of trees will not increase the learning performance. Therefore, the performance of the prediction model is stable, but it can be seen from the comparison figure of the running time that

Table 3: Hyperparameter settings of gcForest and DSF.

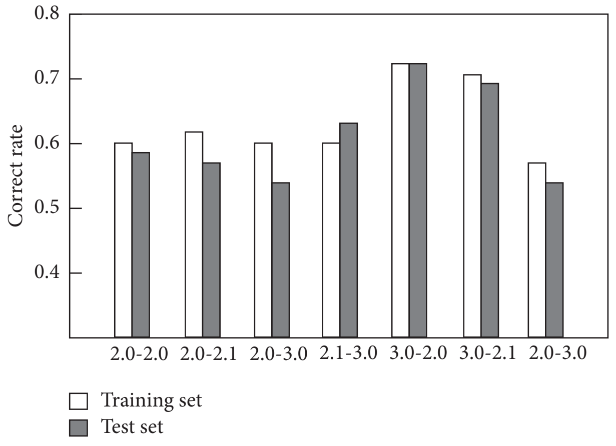
Deep Forest (gcForest)	Deep stacked forest (DSF)
Forest type:	Forest type:
Random forest, completely random tree forest	Random forest, completely random tree forest
Forest during multigranularity scanning:	Forest at random sampling:
Number of forests: 2	Number of forests: 2
Trees in the forest: 20	Trees in the forest: 20
The size of the sliding window: $\begin{cases} d/18 \\ d/10 \\ d/5 \end{cases}$	The size of the sliding window: $\begin{cases} d/18 \\ d/10 \\ d/5 \end{cases}$
Number of forests: 4	Random sampling times: 200, 100, 50 cascade forest:
Trees per tree in the forest: 200	Number of forests: 4, trees per tree in the forest: 200

Table 4: Experimental data results.

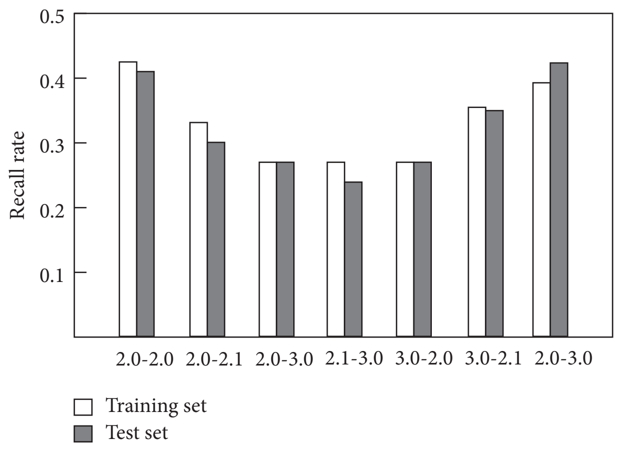
Index	The prediction model of the squared difference cost function		The prediction model of cross-entropy cost function	
	Prediction interval	Average	Prediction interval	Average
Correct rate	0.83–0.88	0.85	0.83–0.88	0.85
Accuracy	0.28–0.59	0.40	0.29–0.60	0.41
Recall rate	0.19–0.43	0.28	0.20–0.45	0.30
F1-measure	0.25–0.49	0.33	0.26–0.52	0.34
AUC value	0.66–0.83	0.72	0.68–0.84	0.73
Operation hours	41 s–103 s	64 s	11 s–31 s	19 s



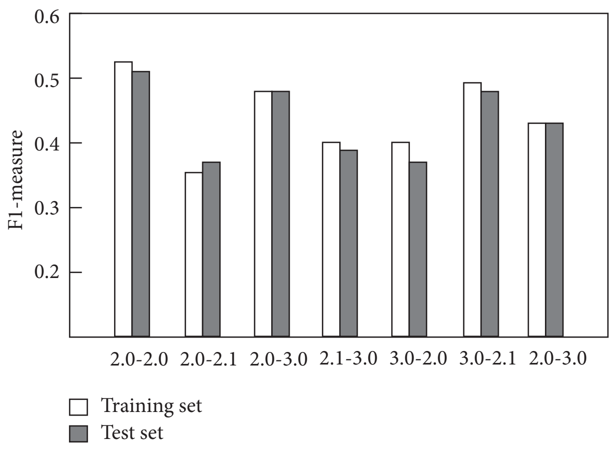
(a)



(b)

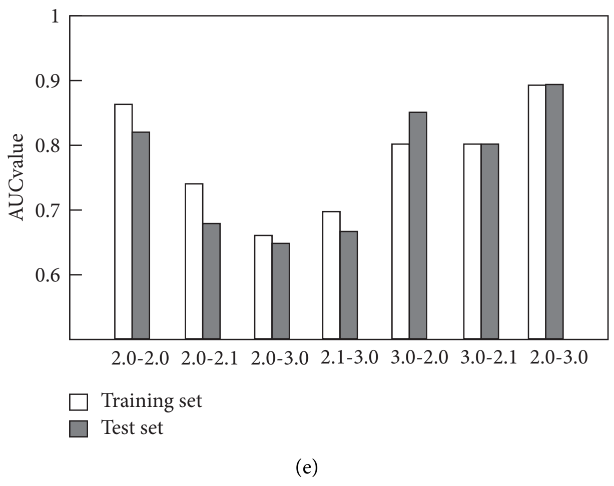


(c)



(d)

Figure 6: Continued.



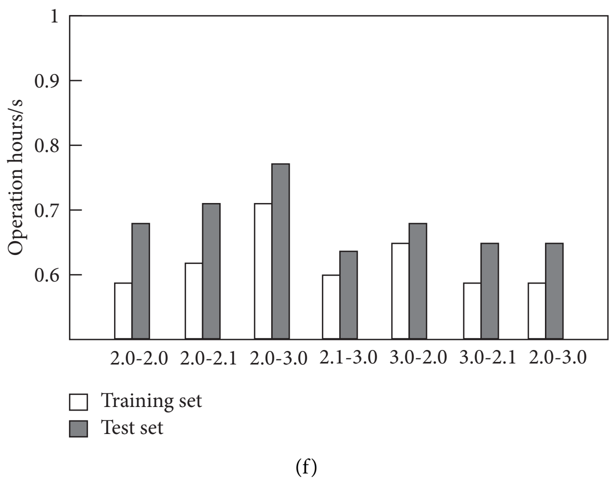
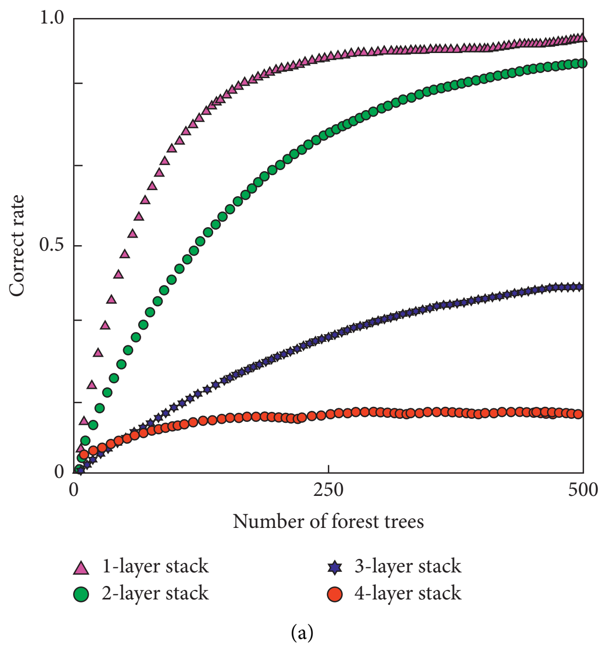


FIGURE 6: Effect of different cost functions on prediction model performance: (a) accuracy prediction comparison; (b) comparison of accuracy prediction; (c) recall prediction comparison; (d) forecast F1-measure comparison; (e) AUC value prediction comparison; (f). comparison of runtime forecasts



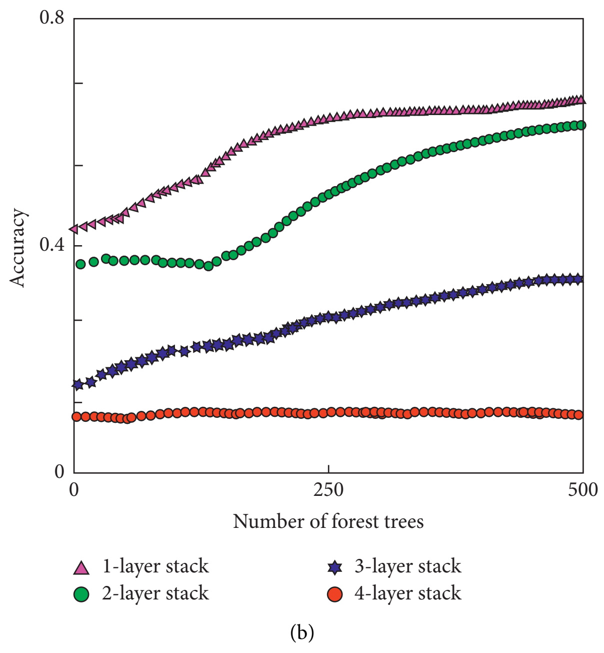
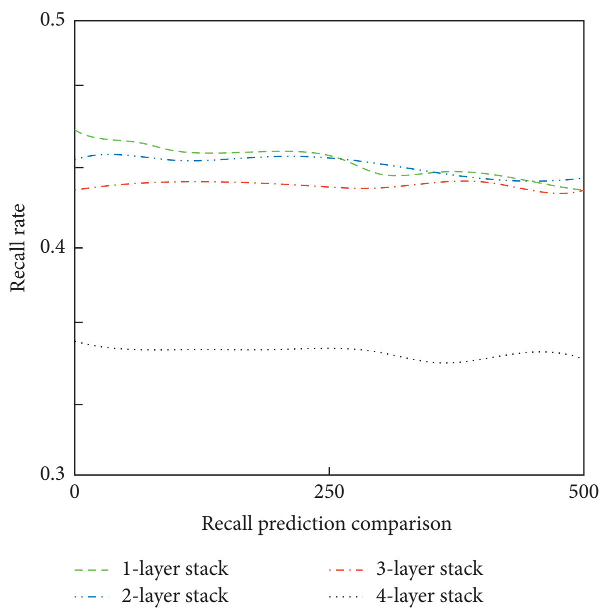
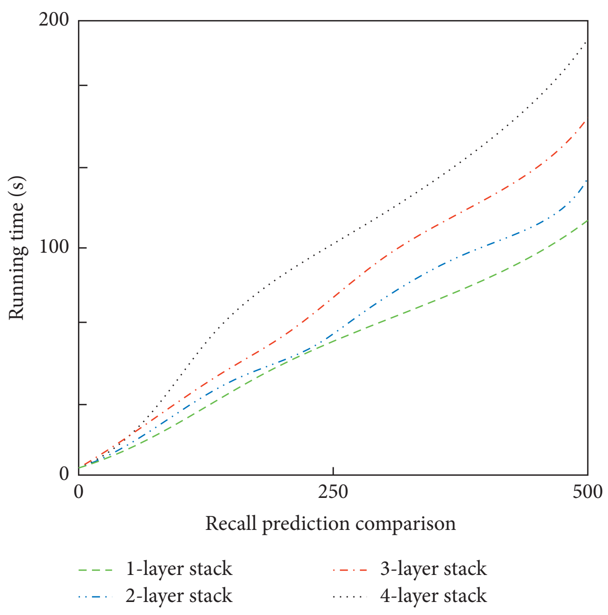


FIGURE 7: Continued.



(c)



(d)

Figure 7: The effect of the depth of the stacked forest and the number of trees on the performance of the prediction model: (a) comparison of accuracy prediction; (b) accuracy prediction comparison; (c) horizontal convergence of mine crossing; (d) comparison of runtime forecasts

the running time of the prediction model basically has a linear growth relationship with the number of trees in the forest. It is concluded that when the number of trees in the forest is 200, the performance of the stacked forest reaches the best. As for the number of identical trees in the stacked forest, the deeper the stacked forest is, the more accurate the prediction rate remains, and the prediction accuracy rate shows a downward trend, but the prediction recall rate shows an increasing trend. It is concluded that the three-layer depth of the stacked forest is the best choice.

4.3.2. The Effect of Cascaded Forest and Stacked Forest on the Performance of the Prediction Model. A software defect prediction model based on deep stacked forests does not perform feature transformation on random sampling and only uses stacked forests for layer-by-layer learning. It is compared with the software defect prediction model based on deep forest which does not use multigranularity scan and only uses cascade forest. The experimental results are compared with correct rate, accuracy rate, recall rate, F1-measure, AUC value, and running time (average running time is used for ten-fold cross-validation). The experimental results are shown in Figure 8.

In Figure 8, the result of using cascading forest is that the prediction correct rate is between 0.84 and 0.89 and the average is 0.86; the accuracy rate is between 0.30 and 0.67, and the average is 0.48; the recall rate is between 0.15 and 0.40, and the average is 0.25; the running time is between 402s and 688s, and the average value is 533s. The result of using stacked forest is that the prediction correct rate is between 0.84 and 0.89 and the average is 0.86; the accuracy rate is between 0.31 and 0.68, and the average is 0.49; the recall rate is between 0.14 and 0.41, and the average is 0.25; the running time is between 382s and 662s, and the average value is 510s.

The above data shows that the **employ** of stacked forests and cascaded forests has the same performance in prediction correct rate, accuracy rate, and recall rate. However, using stacked forests has an advantage in predicting running time than using cascading forests, which can indicate that the prediction models using stacked forests are better than those using cascaded forests.

Comparing the two prediction models based on random sampling and the two prediction models based on multigranularity scanning, it can be found that the prediction accuracy rate is equivalent, the prediction accuracy rate is slightly reduced, the prediction recall rate and runtime efficiency are different degrees of improvement, and especially the increase in running time is larger. It can be explained that the prediction model based on random sampling is superior to the prediction model based on multigranularity scanning. Comparing the two prediction models based on stacked forests and the two prediction models based on cascaded forests, it can be seen that the accuracy of prediction is equivalent, the accuracy and recall of prediction are improved to different degrees, and only a slight decrease in running time is not obvious. It can be explained that the prediction model based on stacked forest has better performance than the prediction model based on cascaded forest.

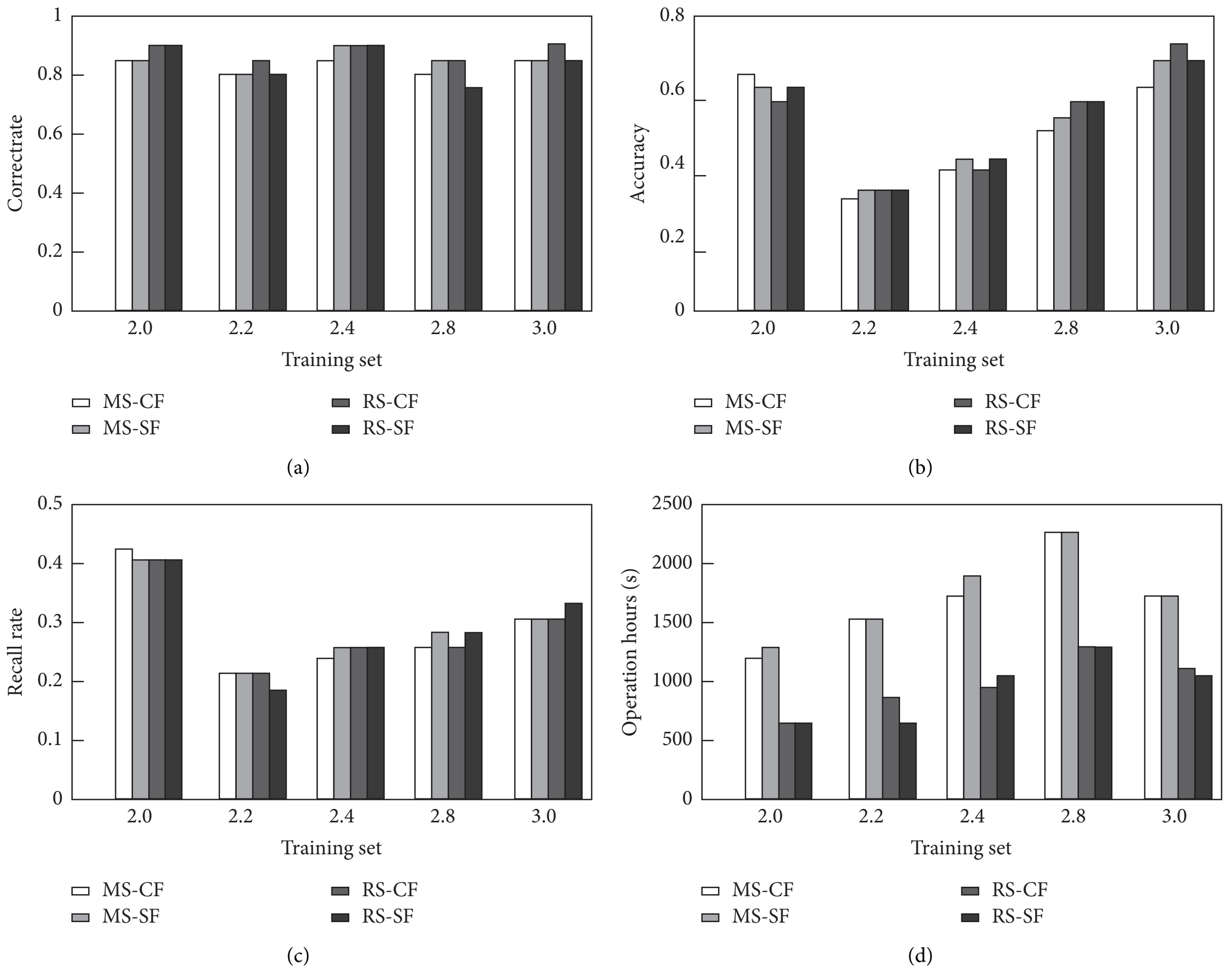


Figure 8: Comparison of cascaded forest and stacked forest: (a) comparison of accuracy prediction; (b) accuracy prediction comparison; (c) recall prediction comparison; (d) comparison of running time forecast

5. Conclusion

By improving the loss cost function and sparsity constraint method of the traditional encoder, the improved encoder uses cross-entropy loss cost function and L1 sparsity constraint rule and improves the robustness of the sparse encoder in order to eliminate the influence of noise on data. In the experiment, the prediction model of the squared difference cost function is used for comparative testing. The experimental data show that the prediction model based on the sDSAE for feature extraction is better than the prediction model without feature extraction or feature extraction based on PCA. Although the prediction accuracy and accuracy are slightly decreased, the prediction recall rate is greatly improved. More importantly, there is a significant improvement in the comprehensive evaluation index F1-measure, and the feature is based on the sDSAE. The extracted prediction model has a significant improvement in predicting AUC value than the prediction model based on PCA for feature extraction. It is proved that the feature extraction method proposed in this paper has a positive meaning for the performance improvement of the software defect prediction model.

Data Availability

The data used to support the findings of this study are included within the article

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by Online identification and diagnosis of software performance problem based on SaaS software running log analysis, and it was supported in part by the grants of National Natural Science Foundation of China (nos. 62072342 and 61672392).

References

[1] Y. Peng, "Research on static performance analysis and modeling methods of high-performance computing applications," *Computer Application Research*, vol. 38, no. 1, pp. 204–208+222, 2021.

[2] J. Li, "Discussion on the equivalent static analysis method in stress calculation," *Natural Gas Chemical Industry (C1 Chemistry and Chemical Engineering)*, vol. 44, no. 3, pp. 88–90, 2019.

[3] L. Liu and D. Guo, "Partial evaluation static input determination method for compiler testing," *Journal of Jilin University (Engineering and Technology Edition)*, vol. 50, no. 1, pp. 262–267, 2020.

[4] Z. Li, "Analysis of the overall protection mode of computer software," *Digital Communication Circle*, vol. 12, no. 3, pp. 86-87, 2019.

[5] Y. Zhang, Y. Tang, and K. Tan, "Design and implementation of a statistical tool for software quality static measurement information," *Software*, vol. 41, no. 8, pp. 94–96, 2020.

[6] H. Shao, H. Jiang, H. Zhao, and F. Wang, "A novel deep autoencoder feature learning method for rotating machinery failure diagnosis," *Mechanical Systems and Signal Processing*, vol. 95, pp. 187–204, 2017.

[7] M. R. Prusty, T. Jayanthi, and K. Velusamy Weighted-SMOTE, "A modification to SMOTE for event classification in sodium cooled fast reactors," *Progress in Nuclear Energy*, vol. 100, pp. 355–364, 2017.

[8] I. Nekooeimehr, K. Susana, and Lai-Yuen, "Adaptive semi-unsupervised weighted oversampling (A-SUWO) for imbalanced datasets," *Expert Systems With Applications*, vol. 46, pp. 405–416, 2016.

[9] Y. Zhang, Y. Zhang, F. Wen et al., "A fuzzy Petri net based approach for failure diagnosis in power systems considering temporal constraints," *International Journal of Electrical Power and Energy*, vol. 78, pp. 215–224, 2016.

[10] D. Tomar, S. Agarwal, and Y. Zhang, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, vol. 1, pp. 95–106, 2016.

[11] M. Mrugalski, M. Luzar, M. Pazera, M. Witczak, and C. Aubrun, "Neural network-based robust actuator failure diagnosis for a non-linear multi-tank system," *ISA Transactions*, vol. 61, pp. 318–328, 2016.

[12] T. Wang, J. Qi, H. Xu, Y. Wang, L. Liu, and D. Gao, "Failure diagnosis method based on FFT-RPCA-SVM for Cascaded-Multilevel Inverter," *ISA Transactions*, vol. 60, pp. 156–163, 2015.

[13] S. Kim, Y. Choi, and M. Lee, "Deep learning with support vector data description," *Neurocomputing*, vol. 165, pp. 111–117, 2015.

[14] F. Wang, "Analysis of the status quo of deep learning research," *Electronic Technology and Software Engineering*, no. 10, pp. 125–136, 2018.

[15] X. Zhu and J. Zhou, "Research on failure diagnosis of mechanical system based on colored Petri nets," *Computer Measurement and Control*, vol. 9, no. 8, pp. 759–764, 2017.

[16] J. Qiao, G. Wang, and X. Li, "Design and application of deep belief network based on adaptive learning rate," *Chinese Journal of Automation*, vol. 24, no. 8, pp. 546–559, 2017.

[17] M. Xie, Y. Wu, S. Huang, and M. Liu, "Power grid failure area identification based on non-ferrous automatic control Petri nets," *Power System Protection and Control*, vol. 5, no. 2, pp. 412–426, 2016.

[18] C. Xiang, Q. Gu, and W. Liu, "Research on static software defect prediction method," *Journal of Software*, vol. 10, no. 1, pp. 45–58, 2016.

[19] M. Gan, S. Wang, and M. Zhou, "Overview of reachable trees of unbounded Petri nets," *Acta Automatica Sinica*, vol. 18, no. 4, pp. 90–102, 2015.

[20] F. Zong, H. Huang, and Z. Ding, "Software failure location based on secondary location strategy," *Journal of Software*, vol. 36, no. 8, pp. 325–334, 2016.

[21] T. Zhang, Y. Li, and H. Hu, "Gender classification model based on cross-connected convolutional neural network," *Acta Automatica Sinica*, vol. 42, no. 6, pp. 858–865, 2016.

[22] L. Li and Y. Wang, "Pairwise combination test case set generation algorithm with weight parameters," *Computer Engineering*, vol. 13, no. 4, pp. 254–261, 2015.

[23] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," *International Conference on Software Engineering*, vol. 14, no. 5, pp. 297–308, 2016.

[24] Y. Liu, "Research on the application of computer technology in power system automation," *Communication World*, vol. 78, no. 8, pp. 859–871, 2019.

[25] X. Liu, W. Luo, and P. Huang, "Intelligent analysis of power information system monitoring and prediction under big data environment," *Microcomputer Application*, vol. 17, no. 7, pp. 484–492, 2019.

[26] Ye Kang, X. Leng, and F. Xiao, "Intelligent analysis method of power grid monitoring based on big data tag technology," *Electrical Measurement and Instrumentation*, vol. 18, no. 4, pp. 87–96, 2019.

[27] X. Leng, G. Chen, and Yu Jiang, "Data specification and data processing of big data analysis system for monitoring and operation of smart grid," *Automation of Electric Power Systems*, vol. 8, no. 19, pp. 674–686, 2018.

[28] X. Liu, Y. Huang, and Ji Yuan, "The application of big data analysis in the failure prediction of automation systems in the power industry," *Information and Computers*, vol. 11, no. 23, pp. 145–165, 2017.

[29] C. Xiang, L. Wang, and Q. Gu, "A review of cross-project software defect prediction methods," *Chinese Journal of Computers*, vol. 22, no. 1, pp. 287–298, 2018.

[30] W. Liu, C. Xiang, and Q. Gu, "Feature selection method based on cluster analysis in software defect prediction," *Science China: Information Science*, vol. 46, no. 9, pp. 1298–1320, 2016.