Software Defect Prediction Model Based on Improved Deep Forest and AutoEncoder by Forest

Wenbo Zheng[†⋆✉], Shaocong Mo[§⋆✉], Xin Jin[¶], Yili Qu[‖], Zefeng Xie[††] and Jia Shuai[‖]
[†]School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049, China
[§]College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China
[¶]School of Management, Huazhong University of Science and Technology, Wuhan 430074, China
[‖]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[††]School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China

Abstract—**Software defect prediction is an important way to make full use of software test resources and improve software performance. To deal with the problem that of the shallow machine learning based software defect prediction model can not deeply mine the software tool data, we propose software defect prediction model based on improved deep forest and autoencoder by forest. Firstly, the original input features are transformed by the data augmentation method to enhance the ability of feature expression, and the autoencoder by forest performs the data of dimensionality reduction on the features. Then, we use the improved deep forest algorithm and autoencoder by forest to build software defect prediction model. The experimental results show that the proposed algorithm has higher performance than the original deep forest (gcForest) algorithm and other existing start-of-art algorithms, and has higher performance and efficiency than other deep learning algorithms.**

Keywords—**Software defect prediction, Deep forest, AutoEncoder by forest, Data augmentation.**

I. Introduction

As software systems continue playing a key role in all areas of our society, defects arisen from these software have had a major impact on businesses and the lives of people. However, due to the significant increase in the size and complexity of software code libraries, it has become increasingly difficult to identify defects in software code [1], [2]. The importance and challenges of defect prediction make it an active research area in software engineering [3], [4]. Extensive research has been used to develop predictive models and tools to help software engineers and testers quickly narrow down the most likely defective parts of the software code base [5], [6]. Early defect prediction helps prioritize and optimize the effort and cost of inspections and testing, especially when faced with cost and deadline pressures [7], [8].

Machine learning techniques have been widely used to build defect prediction models [9], [10]. Those techniques derive a number of features (i.e. predictors) from software code and feed them to common classifiers such as Naive Bayes [11], Support Vector Machine [12] and Random Forests [13]. But these methods are all shallow machine learning, and can not perfectly express the complex relationship between unstructured data. When the amount of data reaches a certain level, the learning ability of shallow algorithm is not as good as the deep learning algorithm.

Due to the huge computational hardware requirements of deep neural networks and the dependence of deep neural networks on a large number of hyper parameters, the software defect prediction methods based on deep neural networks are difficult to train to the optimal degree. Zhi-Hua Zhou [14] put forward a deep forest, make up the blank of the decision tree in the field of deep learning. Deep forests have much less parameters than deep neural network and the advantages of higher classification accuracy. Further, Zhi-Hua Zhou [15] put forward EncoderForest (eForest), a kind of auto-encoder, to do data reduction or feature extraction for training model. Therefore, *why not use deep forest and eForest to build sofeware defect prediction model?*

In this paper, we present software defect prediction model based on improved deep forest and autoencoder by forest. First of all, we propose an improved deep forest algorithm for the lack of multi-grained scanning in deep forests through data augmentation. Then, the improved deep forest algorithm and eForest are applied to software defect prediction problem. Finally, we use this model to experiment with the Eclipse bug dataset [16]. The experimental results show that the proposed algorithm is better than existing start-of-art algorithms and less time than the deep neural network algorithm. The contributions of our paper are as below.

(1) We apply the deep forest and eForest to software defect prediction problem and get better results.
(2) In our prediction system, the features are automatically learned through the forests model, thus eliminating the need for manual feature engineering which occupies most of the effort in traditional approaches.
(3) An extensive evaluation using real open source data provided by Eclipse repository demonstrates the empirical strengths of our model for defect prediction.

The outline of this paper is as follows. *Section* II reviews the works of defect prediction, deep forest and autoencoder by forest. *Section* III describes how our prediction model is

[⋆]Wenbo Zheng and Shaocong Mo contribute equally to this study. They are both the corresponding author.

[✉]E-mail: zwb2017@stu.xjtu.edu.cn, mosc@zju.edu.cn

DOI reference number: 10.18293/SEKE2019-008

built. We report our experiments to evaluate our approach in *Section* IV. In *Section* V, we conclude the paper and outline future work.

II. Related Work

A. Defect Prediction

In the field of software engineering, it is impossible to detect and eliminate all software defects by any means of detection and verification. It is impossible to develop a software system without any defects in an actual engineering project. Even if the developer is careful and refined, it cannot be ruled out that there are still some errors or unexpected defects in the software system. Software defect prediction is an important way to rationally use software testing resources and improve software performance. Software detect prediction technology can be used to predict more defects that may also exist as early as possible according to the metrics information and defects found in a software product, then testing and validation resources are allocated based on the result appropriately.

The machine learning based defect prediction technology can comprehensively and automatically learn the model to find defects in the software, which has become the main method of defect prediction. For constructing software detect prediction models, many algorithms such as KNN, neural networks [17], SVM [18], Nave Bayes [19], random forest [20] and ensemble learning method [21] can be used. Moreover, there are mainly three techniques are used for implementing the software detect prediction models, as classification, regression and clustering. However, none of these algorithms can perfectly express the complex relationship between unstructured data. When the amount of data reaches a certain level, the learning ability of shallow structure algorithms is not as good as that of deep structure algorithms.

B. Deep Forest

Zhi-Hua Zhou and Ji Feng [14] propose gcForest (multi-Grained Cascade Forest) shown in Fig. 1(a), a novel decision tree ensemble method. This method generates a deep forest(DF) ensemble, with a cascade structure which enables gcForest to do representation learning. Its representational learning ability can be further enhanced by multi-grained scanning when the inputs are with high dimensionality, potentially enabling gcForest to be contextual or structural aware. The number of cascade levels can be adaptively determined such that the model complexity can be automatically set, enabling gcForest to perform excellently even on small-scale data, which makes it possible to control training costs according to computational resource available. Moreover, contrast to DNNs, the gcForest has much fewer hyper-parameters and its performance is robust in different hyper-parameter settings. From experiments results, gcForest gets excellent performance by using the default setting, competitive to DNNs on a broad range of tasks, even across different data from different domains. Deep forest offers an alternative when deep neural networks are not superior, e.g., when DNNs are inferior to random forest and XGBoost.

C. AutoEncoder by Forest

After gcForest, Zhi-Hua Zhou and Ji Feng [15] propose autoencoder by forest called eforest, which is the first tree ensemble based auto-encoder. As we know, auto-encoding is a significant task; take convolutional neural networks (CNN) as an example, it is achieved by deep neural networks (DNNs) in auto-encoding way. This method presents a procedure for enabling forests to do backward reconstruction by utilizing the Maximal- Compatible Rule (MCR), as show in Fig. 1(b), the rule is defined by the traversing backward decision paths of the trees. Encoding and decoding are the two basic function for an auto-encoder. The eForest makes a tree ensemble to do forward encoding and backward decoding, so it is possible for forest to construct an auto-encoder. In addition, the eForest can be trained in both supervised or unsupervised way.

For encoding tasks, there is no difficulty for forest to do that. Here is the encoding procedure: Once the input data traverse down to the leaf nodes which belong to a trained tree ensemble model contained T trees, the procedure will give back a T dimensional vector.

For decoding task, here is the encoding procedure: Firstly, each path can be identified by the leaf node without uncertainty; Secondly, each path corresponds to a symbolic rule; Thirdly, the Maximal-Compatible Rule (MCR) can be calculated to reconstruct the original sample.

In all, auto-encoding task can be achieved by the eForest's forward encoding and backward decoding operations. Experimental results shows eforest has the following advantages: lower reconstruction error contrast to CNN or MLP based auto-encoder, faster training speed, damage-tolerable and high dataset adaption in same domain.

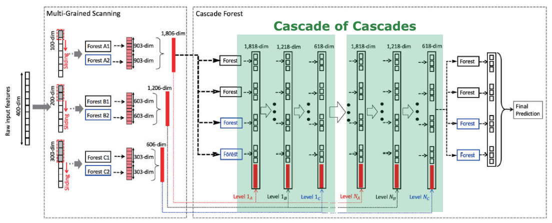
III. Software Defect Prediction Model Based on Forests

We propose our method in this section. Firstly, we propose an improved deep forest algorithm for the lack of multi-grained scanning in deep forests through data augmentation. Secondly, the improved deep forest algorithm and eForest are applied to software defect prediction problem, as shown in Fig. 2.

A. Improved Deep Forest Algorithm

Despite the superior performance of deep forests, there are still some shortcomings when using them to build the software defect prediction model.

- Software defect prediction is a binary classification problem, which is to analyze the quality of software modules. According to the classification rules, it is divided into fault-proneness or non-fault-proneness class. Because the main purpose of software defect prediction is to predict whether the modules in the software have fault-proneness modules, so we would set the fault-proneness module as a positive example, and the non-fault-proneness module as a negative example. Multi-grained scanning and cascade structures are used the process that generated class vectors to be aggregated into enhanced feature vectors.


(a) Multi-Grained Cascade Forest [14]

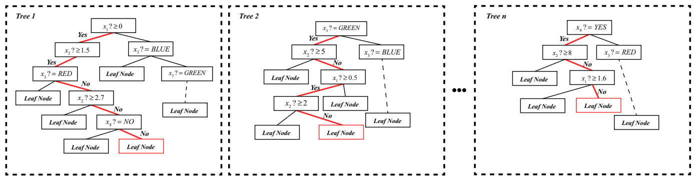

(b) Traversing backward along decision paths [15]

Fig. 1. Deep forest and autoencoder by forest

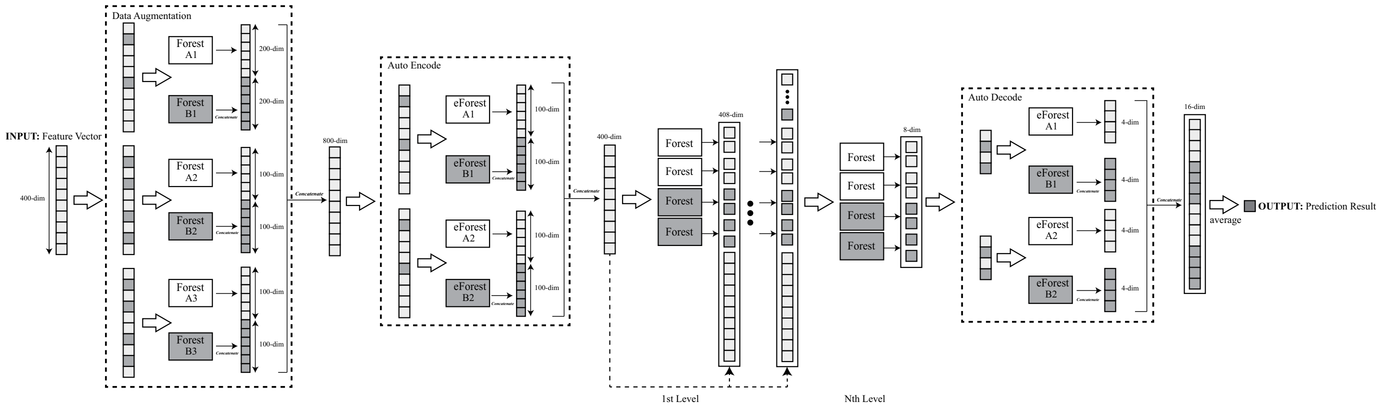


Fig. 2. Software defect prediction model using Forests

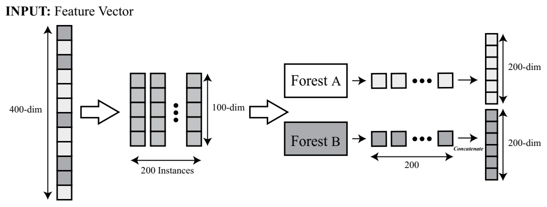


Fig. 3. The process of data augmentation as an example

But this process can cause redundancy of the feature space. For the software defect prediction problem, the sum of the probability of the module belonging to the positive class and the probability of the module belonging to the negative class is 1, that is, the two probabilities are linear correlation. If two probabilities are both used to fuse the feature vectors, which can cause feature space redundancy and increase the space complexity of the algorithm.

- Multi-grained scanning has significant effects on spatially related features, such as image matching and speech recognition, while features that are spatially uncorrelated (such as software defect prediction, text classification, etc.) may lose important information. The reason is that for spatially uncorrelated features, multi-grained scanning reduces the importance of the the first and the last feature. In the multi-grained scanning process, the first feature and the last feature are only scanning once, that is, these two features are used only once. If the first feature or the last feature is very important, multi-grained scanning can not effectively use this important feature.

To solve the problem that multi-grained scanning in deep forests may lose important information, data augmentation method is used to transform the original input features. Our data augmentation method is to randomly extract features from different original scales from the original input features from the defective/non-defective training samples. The vector by data augmentation is treated as a defective/non-defective instance. The instances which is the same size are combined to form a training entity. The all training entities would train a random forest and a complete random forest, respectively, to predict to generate class vectors. We transform class vectors to enhanced feature vectors.

To be specific, assume that the original input features are 400-dimension, we use randomly sampled methods 200 times, and each sample size is 100-dimensional feature. Each 100-dimensional sample feature is to form an instance. So a total of 200 instances are generated. The 200 instances is called an entity. The entity would be trained using a random The forest and a completely random tree forest, respectively, and then we get two class vectors which is 200-dimension. We transform two 200-dimensional vectors to one 400-dimensional vector, that is, the 400-dimensional original feature vector corresponds to the 400-dimensional enhanced feature vector. The process is shown in Fig. 3.

Similarly, we transform the two 200-dimensional features sampled from each original input feature to generate two 200-dimensional enhanced features vector. We do the concatenate

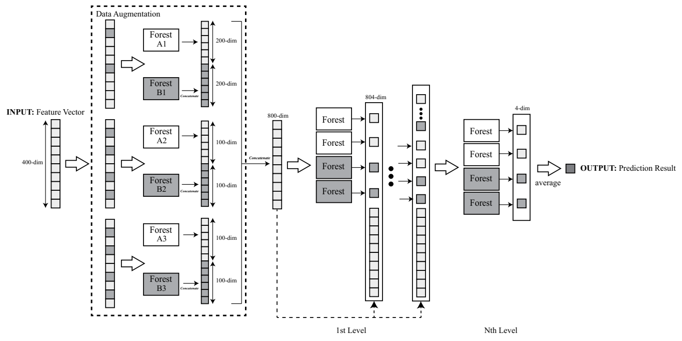


Fig. 4. The pipeline of improved deep forest based on data augmentation

operator to transform the three enhanced feature vectors to a 800-dimensional transformed feature vector. This is the re-representation of the original input features by data augmentation. In other words, each 400-dimensional original feature vector is re-represented by the 800-dimensional transformed feature vector.

The 800-dimensional transformed feature vector would be passed to the cascade forest structure. If each layer of cascade forest consists of 4 forests (two random forests and two completely random forests), the 804-dimensional feature vector would be obtained at the end of the first layer. Then, input the feature vector into the cascade forest structure of the next layer. Repeat the process until the verification performance indicates that the extension of the cascade forest structure should be terminated.

In the test phase, given a test example, we first get the corresponding 800-dimensional transformed feature vector through the data augmentation process, and then predict through the cascade forest structure until the last layer. The final prediction result is decided on the 4 probabilities of 4-dimensional feature vector in the last layer, and the average of the 4 probabilities is the final prediction result. If the average value is greater than or equal to 0.5, the test instance is predicted to be a positive class, otherwise it is a negative class. The process is shown in Fig. 4.

B. Software Defect Prediction Model using Forests

We use data augmentation algorithms to increase the amount of data, challenging the time and efficiency of our algorithm. Therefore, we use the eForest [15] to do data reduction. The process is shown in Fig. 4.

First, we transform original 400-dimensional features to 800-dimensional transformed vectors using data augmentation.

Then, the 800-dimensional transformed feature vector would be passed to the autoencoder using eForest. If autoencoder consists of 4 eForests, the 400-dimensional feature vector would be obtained. The 400-dimensional feature vector would be passed to the cascade forest structure. Because each layer of cascade forest consists of 4 forests (two random forests and two completely random forests) and we reset each forest to output 2-dimensional class vector, the 408-dimensional feature vector would be obtained at the end of the first layer. Then, input the feature vector into the cascade forest structure of the next layer. Repeat the process until the verification performance indicates that the extension of the cascade forest structure should be terminated. We get a 16-dimensional feature vector through cascade forest structure.

Finally, the 16-dimensional feature vector would be passed to the autodecoder which is the inverse structure of autoencoder. We get 8-dimensional feature vector. The final prediction result is decided on the 8 probabilities of 8-dimensional feature vector in the last layer, and the average of the 8 probabilities is the final prediction result. If the average value is greater than or equal to 0.5, the test instance is predicted to be a positive class, otherwise it is a negative class.

IV. Experiment Results and Analysis

Reporting the average of precision/recall across the two classes (defective and clean) is likely to overestimate the true performance, since our dataset is imbalance (i.e. the number of defective files are small). More importantly, predicting defective files is more of interest than predicting clean files. Hence, our evaluation is focus on the defective class.

A confusion matrix is used to store the correct and incorrect decisions made by a prediction model. For example, if a file is classified as defective when it is truly defective, the classification is a true positive (tp). If the file is classified as defective when it is actually clean, then the classification is a false positive (fp). If the file is classified as clean when it is in fact defective, then the classification is a false negative (fn). Finally, if the issue is classified as clean and it is in fact clean, then the classification is true negative (tn). The values stored in the confusion matrix are used to compute the widely-used Precision, Recall, and F-measure.

- Precision: The ratio of correctly predicted defective files over all the files predicted as being defective. It is calculated as:
$$pr = \frac{tp}{tp + fp} \tag{1}$$
- Recall: The ratio of correctly predicted defective files over all of the true defective files. It is calculated as:
$$re = \frac{tp}{tp + fn} \tag{2}$$
- F-measure: Measures the weighted harmonic mean of the precision and recall. It is calculated as:
$$F - measure = \frac{2 \times pr \times re}{pr + re} \tag{3}$$

All experiments were conducted using a 4-core PC with an Intel Core i7 6700HQ with the NVIDIA GTX 1080, 16GB of RAM, and Ubuntu Linux in practice.

To verify the advancement and robustness of our model, we designed two experiments:

(1) *Ablation Experiment*. In order to verify the advancement and rationality of our model, we experimented with each component.
(2) *Contrast Experiment*. In order to verify the robustness of our model, in the experiment separately to

TABLE I
THE RESULT OF ABLATION EXPERIMENT ON ECLIPSE BUG DATASET

Method		gcForest [14]				Improved Deep Forest				gcForest [14]+eForest [15]				Ours			
Data Augumention		×				✓				×				✓			
AutoEncoder/AutoDecoder		×				×				✓				✓			
Training Set	Test set	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure	Accuracy Rate	Precision	Recall	F-measure
file2.0	file2.0	0.8845	0.6708	0.4031	0.5036	0.8847	0.6712	0.4038	0.5042	0.8861	0.6522	0.4601	0.5399	**0.8865**	**0.6753**	**0.4603**	**0.5475**
	file2.1	0.8555	0.3208	0.2998	0.3099	0.8557	0.3217	0.3001	0.3106	0.8569	0.3361	0.3302	0.3331	**0.8572**	**0.3368**	**0.3307**	**0.3337**
	file3.0	0.8505	0.4912	0.2832	0.3592	0.8511	0.4912	0.2834	0.3594	0.8427	0.4486	0.273	0.3394	**0.8555**	**0.4492**	**0.2834**	**0.3475**
file2.1	file2.0	0.8501	0.4543	0.1733	0.2509	0.8505	0.4551	0.1735	0.2512	0.852	0.476	0.2133	0.2946	**0.8527**	**0.4763**	**0.2137**	**0.2950**
	file2.1	0.8978	0.5705	0.223	0.3206	0.8978	0.5715	0.2238	0.3216	0.8968	0.5446	0.2858	0.3748	**0.8988**	**0.5853**	**0.2860**	**0.3843**
	file3.0	0.8453	0.4397	0.1652	0.2401	0.8461	0.4407	0.1652	0.2404	0.8453	0.4518	0.2124	0.2889	**0.8660**	**0.4520**	**0.2125**	**0.2891**
file3.0	file2.0	0.8575	0.5155	0.2728	0.3568	0.8579	0.5158	0.2736	0.3575	0.8575	0.5124	0.3384	0.4077	**0.8583**	**0.5563**	**0.3390**	**0.4213**
	file2.1	0.8581	0.3221	0.281	0.3002	0.8584	0.3229	0.2811	0.3005	0.8556	0.3277	0.3173	0.3224	**0.8656**	**0.3281**	**0.3181**	**0.3230**
	file3.0	0.8662	0.5911	0.324	0.4186	0.8667	0.5912	0.3241	0.4186	0.8637	0.5554	0.3974	0.4633	**0.8674**	**0.5958**	**0.3980**	**0.4772**

realize the Naive-Bayes-Based method [11], Support-Vector-Machine-Based method [12], Random-Forests-Based method [13], Deep Tree-based method [22] and DNN-based method [23], Eclipse standard dataset on the experiment and comparing with the result of the experiment.

Note that in the experiment, the hyper-parameter settings we used were consistent with the reference gcForest [14] and eForest [15].

The data used in this experiment is derived from the Eclipse standard dataset which is one of the most widely used public datasets in software defect prediction research. Since this paper studies the software defect prediction of the classification task, and the Eclipse data set gives the number "post" of defects (refers to the number of defects after the software is released), it is necessary to convert the number "post" of defects into a defective class "hasDefects". The conversion method is:

$$hasDefect = \begin{cases} 0, & post = 0 \\ 1, & post \neq 0 \end{cases} \tag{4}$$

Since the structure of the "file" level data of the Eclipse data set is same, one of the versions of the data is used as the training data to learn the model, and the three versions of the data can be predicted separately, so that the "file" level data can be used for 9 predictions and verifications.

When the training set and the test set are from the same version, the ten-fold cross-validation is used, that is, the data set is equally divided into 10 parts, one of which is taken as the prediction set, and the 9 remaining data are used as the training set to construct the software defect prediction model and do classification prediction. The experiment was carried out for 10 rounds, and the average of 10 rounds of experiments was taken as the final result.

A. Ablation Experiment

In order to verify the advancement and rationality of our model, we experimented with each component. We use the accuracy rate, precision, recall rate and F-measure to evaluate the experimental results in the experiment. From Table.I, we know our method is better than others. This shows that the design of our algorithm is reasonable.

B. Contrast Experiment

In order to verify the robustness of our model, in the experiment separately to realize the Naive-Bayes-Based method

TABLE II
THE RESULT OF CONTRAST EXPERIMENT ON ECLIPSE BUG DATASET

Method	Training Set	Test set	Accuracy Rate	Precision	Recall	F-measure	Test Time/s
Naive-Bayes-Based	file2.0	file2.0	0.4049	0.2487	0.3818	0.3012	**939.10**
		file2.1	0.3950	0.1990	0.1361	0.1617	992.42
		file3.0	0.3621	0.1123	0.1366	0.1233	963.81
	file2.1	file2.0	0.7169	0.1784	0.1876	0.1829	965.69
		file2.1	0.5621	0.3514	0.2471	0.2902	981.40
		file3.0	0.4467	0.0499	0.1744	0.0776	**905.80**
	file3.0	file2.0	0.6412	0.4286	0.1163	0.1830	994.07
		file2.1	0.7742	0.2874	0.3415	0.3121	**963.66**
		file3.0	0.6147	0.3210	0.1127	0.1668	995.49
Support-Vector-Machine-Based	file2.0	file2.0	0.5266	0.4855	0.3372	0.3980	984.68
		file2.1	0.5802	0.4319	0.1014	0.1642	940.42
		file3.0	0.8175	0.2333	0.1178	0.1565	**936.28**
	file2.1	file2.0	0.8011	0.1011	0.1754	0.1283	971.73
		file2.1	0.8927	0.2631	0.1463	0.1880	991.80
		file3.0	0.4862	0.1785	0.1936	0.1857	984.54
	file3.0	file2.0	0.6165	0.4705	0.1964	0.2772	**910.49**
		file2.1	0.7281	0.2550	0.0325	0.0576	996.15
		file3.0	0.6047	0.5585	0.1147	0.1903	**944.24**
Random-Forests-Based	file2.0	file2.0	0.7175	0.5661	0.3932	0.4641	993.29
		file2.1	0.7424	0.1544	0.2673	0.1957	**926.15**
		file3.0	0.8227	0.3748	0.2418	0.2940	949.10
	file2.1	file2.0	0.8093	0.4630	0.1572	0.2347	**926.51**
		file2.1	0.7362	0.3983	0.2712	0.3227	**968.88**
		file3.0	0.7906	0.2622	0.1532	0.1934	949.81
	file3.0	file2.0	0.6972	0.4031	0.2695	0.3230	986.32
		file2.1	0.6683	0.1613	0.2884	0.2068	988.58
		file3.0	0.6860	0.4845	0.2445	0.3250	954.50
Deep Tree-based	file2.0	file2.0	0.7919	0.2475	0.3052	0.2734	2740.51
		file2.1	0.8236	0.4088	0.1980	0.2667	3462.27
		file3.0	0.8006	0.3925	0.1907	0.2567	2672.78
	file2.1	file2.0	0.8806	0.5172	0.2197	0.3084	2711.65
		file2.1	0.8445	0.4067	0.1393	0.2075	3313.87
		file3.0	0.8333	0.5195	0.2640	0.3501	3251.04
	file3.0	file2.0	0.8070	0.3062	0.2540	0.2777	3168.86
		file2.1	0.8413	0.5205	0.3410	0.4121	2537.39
		file3.0	0.4778	0.4877	0.1443	0.2228	2815.82
DNN-based	file2.0	file2.0	0.8856	0.5209	0.2136	0.3029	4070.51
		file2.1	0.8140	0.1531	0.3253	0.2082	4629.74
		file3.0	0.5592	0.2508	0.0559	0.0915	4449.19
	file2.1	file2.0	0.8438	0.4124	0.0675	0.1161	4643.71
		file2.1	0.8413	0.5248	0.1379	0.2184	4389.90
		file3.0	0.6942	0.4166	0.1111	0.1754	4447.18
	file3.0	file2.0	0.8260	0.5292	0.1614	0.2473	4767.79
		file2.1	0.6791	0.0655	0.1613	0.0932	4486.30
		file3.0	0.8282	0.4512	0.3592	0.4000	4566.45
Ours	file2.0	file2.0	**0.8865**	**0.6753**	**0.4603**	**0.5475**	**2552.15**
		file2.1	**0.8572**	**0.3368**	**0.3307**	**0.3337**	**2641.29**
		file3.0	**0.8555**	**0.4492**	**0.2834**	**0.3475**	**2974.45**
	file2.1	file2.0	**0.8527**	**0.4763**	**0.2137**	**0.2950**	**2666.39**
		file2.1	**0.8988**	**0.5853**	**0.2860**	**0.3843**	**2472.26**
		file3.0	**0.8660**	**0.4520**	**0.2125**	**0.2891**	**2560.44**
	file3.0	file2.0	**0.8583**	**0.5563**	**0.3390**	**0.4213**	**2699.37**
		file2.1	**0.8656**	**0.3281**	**0.3181**	**0.3230**	**2868.56**
		file3.0	**0.8674**	**0.5958**	**0.3980**	**0.4772**	**2604.83**

[11], Support-Vector-Machine-Based method [12], Random-Forests-Based method [13], Deep Tree-based method [22] and DNN-based method [23], Eclipse standard dataset on the experiment and comparing with the result of the experiment. We use the accuracy rate, precision, recall rate, F-measure and the time of test to evaluate the experimental results in the experiment.

From Table. II, we can get two points:

- In terms of performance, our algorithm is optimal in all cases. Therefore, the robustness of our algorithm is optimal among all methods.

- In terms of test time, our algorithm is not the least time in all cases, but our algorithm is the least time in all cases of deep learning. It shows that our algorithm has obvious advantages in efficiency compared to other deep learning algorithms.

From these two points, we know that our algorithm has good robustness and efficiency.

V. Conclusion and Future Work

In this paper, in order to solve the problem that the shallow machine learning algorithm can not deeply mine the software data features in the current software defect prediction, software defect prediction model based on improved deep forest and autoencoder by forest is proposed. Firstly, the original input features are transformed by the data augmentation method to enhance the ability of feature expression, and the autoencoder by forest performs the data of dimensionality reduction on the features. Then, we use the improved deep forest algorithm and autoencoder by forest to build software defect prediction model. The experimental results show that the proposed algorithm has higher performance than the original deep forest (gcForest) algorithm, and has higher performance and efficiency than other deep learning algorithms. However, the algorithm does not consider unbalanced data problem. In future, we will be to study how to further improve the prediction performance of the algorithm and consider a solution to the problem of data imbalance in the training set.

Acknowledgment

This paper was supported in part by the National Natural Science Foundation of China (Grant No. 61702386).

References

[1] F. Wu, X. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability*, vol. 67, no. 2, pp. 581–597, June 2018.

[2] S. Huda, S. Alyahya, M. M. Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, and J. Yearwood, "A framework for software defect prediction and metric selection," *IEEE Access*, vol. 6, pp. 2844–2858, 2018.

[3] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 534–550, June 2018.

[4] E. A. Felix and S. P. Lee, "Integrated approach to software defect prediction," *IEEE Access*, vol. 5, pp. 21 524–21 547, 2017.

[5] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, May 2018.

[6] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights modelfor cross-project software defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.

[7] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE Access*, vol. 6, pp. 24 184–24 195, 2018.

[8] T. Chen, S. W. Thomas, H. Hemmati, M. Nagappan, and A. E. Hassan, "An empirical study on the effect of testing on code quality using topic models: A case study on software development systems," *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 806–824, Sept 2017.

[9] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, Sept 2018.

[10] C. Hu, X. Xue, L. Huang, H. Lyu, H. Wang, X. Li, H. Liu, M. Sun, and W. Sun, "Decision-level defect prediction based on double focuses," *Chinese Journal of Electronics*, vol. 26, no. 2, pp. 256–262, 2017.

[11] T. Wang and W. Li, "Naive bayes software defect prediction model," in *2010 International Conference on Computational Intelligence and Software Engineering*, Dec 2010, pp. 1–4.

[12] H. Wei, C. Shan, C. Hu, H. Sun, and M. Lei, "Software defect distribution prediction model based on npe-svm," *China Communications*, vol. 15, no. 5, pp. 173–182, May 2018.

[13] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, Oct 2017, pp. 252–257.

[14] Z. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," *CoRR*, vol. abs/1702.08835, 2017. [Online]. Available: http://arxiv.org/abs/1702.08835

[15] J. Feng and Z. Zhou, "Autoencoder by forest," *CoRR*, vol. abs/1709.09018, 2017. [Online]. Available: http://arxiv.org/abs/1709.09018

[16] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, May 2007, pp. 9–9.

[17] R. Jindal, R. Malhotra, and A. Jain, "Software defect prediction using neural networks," in *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2014 3rd International Conference on.* IEEE, 2014, pp. 1–6.

[18] P. Selvaraj and D. P. Thangaraj, "Support vector machine for software defect prediction," *International Journal of Engineering & Technology Research*, vol. 1, no. 2, pp. 68–76, 2013.

[19] A. Okutan and O. T. Yıldız, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[20] S. G. Jacob *et al.*, "Improved random forest algorithm for software defect prediction through data mining techniques," *International Journal of Computer Applications*, vol. 117, no. 23, 2015.

[21] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering*, vol. 23, no. 4, pp. 569–590, 2016.

[22] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C. Kim, "A deep tree-based model for software defect prediction," *CoRR*, vol. abs/1802.00921, 2018. [Online]. Available: http://arxiv.org/abs/1802.00921

[23] A. V. Phan, M. L. Nguyen, and L. T. Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov 2017, pp. 45–52.