ATLANTIS PRESS

Advances in Intelligent Systems Research, volume 136

4th International Conference on Sensors, Mechatronics and Automation (ICSMA 2016)

Software Defect Prediction based on Adaboost algorithm under Imbalance Distribution

Yan Gao[1,2, a], Chunhui Yang[1,3,b]

[1] China Electronic Products Reliability and Environmental Testing Research Institute

[2] South China University of Technology

[3]Key Laboratory for Performance and Reliability Testing of Foundational Software & Hardware, MIIT

[a] thegoldfishwang@163.com, [b] yangch@ceprei.com

Keywords: Software defect prediction, Adaboost, Neural Network, Imbalance distribution.

Abstract. Software defects will lead to software running error and system crashes. Many methods were proposed to solve this problem. However, the imbalance distribution of software defects leads to the major bias and accuracy loss for most software defect prediction methods. In this paper, we propose an application which combine Adaptive Boosting(AdaBoost) and Back-propagation Neural Network(BPNN) algorithm to train software defect prediction model. BPNN was utilized as a weak leaner in AdaBoost and tweaked in favor of instances misclassified. The experiments show that the proposed method in the paper significantly improves the performance than the previous models, which is effective to deal with the imbalance software defect data.

Introduction

Software is artifact created with a set of algorithms, practices and running modules. It contains some flaws which will affect project and product performance inevitably[1]. The software will be run error or the system will be crashed in the worse case. Therefore, software defect prediction plays an important role in producing high quality software production. The traditional software detect technology is testing the software after software development. However, this method can not detect flaw opportunely. It ignores the flaw which affects the subsequent process greatly and increases the error rate during software development. As a result, it requires more debugging after software defect detected which increase the workload of programmers.

A prediction of software defect in the programming process is proposed to test software flaw during software development. In most cases, the software failure rate is much lower than the software success rate. Therefore, the software attributes are imbalance which will cause the forecasting error. If we used traditional classification method to predict software defect, this kind of imbalance data will lead to training imbalance. That means the failure model will be ignored when model training and the failure model will be hard to detect after simulation. More workload will just be needed to recognize if the real success software was judged as the failure software. But if the failure software was judged as success, inestimable economic losses will be happened. •Modeling the prediction. It is the most important step in software defect architecture. The classify prediction performance is depended on the classify algorithm. Each classification algorithm is sensitive to their target data. And their preferences will also have influence on the final prediction result. Therefore, choosing a suitable classification algorithm and effectively parameters are the key in modeling the prediction.

AdaBoost, one of the most popular classification method based on cascade classifier model [2], is applied to predict software defect. Different to the traditional detect method, e.g. linear Regression, nonlinear regression [3,4] and cluster analysis[5], data classification is provided with self-learning, which have adaptive to imbalance software attributes. Generally, AdaBoost is usually applied to deal with pattern recognition, such as face recognition [6,7]. It contains a lot of weak learner and train the weight of each attributes automatically to regulate the output accuracy slightly in each iteration. With this property, AdaBoost is a good predictor for imbalance data. In the meanwhile, it avoid the overfitting and have a nice generalization for small sample data. It is obviously that AdaBoost is an adaptive method for software defect prediction.

We utilized BPNN as the weak leaner for AdaBoost(BP-AdaBoost) to build a prediction model. BPNN is a mature method in dealing with software quality problem. Until now several software quality research was done with BPNN [8]. It is a mapping of nonlinear function which is good at deal with non-linear compute problem [9,10]. In the meanwhile, it has high recognition rate for big sample data but low recognition rate for small sample data. Plenty of papers address this issue and improve BPNN[11]. Until now, it is still a study hotspot for scholars. In our test, BP-AdaBoost can overcome the imbalance data problem and suitable for software defect prediction.

A. Software Defect Architecture

The construction of software defect-proneness prediction architecture is divided into four parts:

- Data collection. Translating the code of a software into a sample set which can be statistical analysis is an important step of software quality data collecting.
- Data pre-processing. Choosing the suitable attributes, data normalization and principal component analysis are the usual preprocessing step to remove redundant attributes and enhance the software quality.
- Building and training of prediction model. It is the most important step in software defect architecture. Different classification method will lead to very different result. And each classification method will sensitive to its specific data. Therefore, choosing a suitable classification method is the key to build the favourable architecture.
- Assessing the prediction model. Several evaluators were applied to test and evaluate the robustness of our model, such as precision and recall.

B. Software Metrics

Generally, metrics of software modules are the main input of defect prediction model. Moreover, it is the quantitative study of software attributes, which including architecture-oriented and object-oriented. Aimed at the connection between function and the internal structure complexity of functions, architecture-oriented metrics, the traditional metrics methods are focus on the module length, cohesion, module coupling and data coupling. It can not deal with the object-oriented software attributes and poor to evaluate the the quality of object-oriented system. while the object-oriented metrics emphasize the object relationship e.g. inheritance, abstraction, encapsulation, and polymorphism which mainly aim at object-oriented software development. It is suitable to object-oriented system.

With the effort of programmer, the valid data modules usually much more than the failure data modules, which result in the imbalance distribution. However, imbalance data distribution will lead to biased modeling. With greater likelihood on the valid data judgement but less on the failure judgement, the prediction result based on that biased models is inaccurate. Improving the classification performance of the minority class is the challenge in software defect prediction. And building the software defect prediction model to find out the failure module of the software during software development is the main idea of our prediction model.

Plenty of software prediction methods are developed for imbalance data analysis. However, because of their performance evaluation criteria and inductive bias, these methods focus on the overall maximum classification accuracy and their prediction tend to support the large sample data. As a result, they ignore the accuracy of small sample data and resulting in the model bias. It is necessary to build a novel algorithm which adaptive to the imbalance data analysis.

Method

A. BP-AdaBoost

Based on the classify of strong leaner and weak leaner, applying a possibility to train the weak leaner into strong leaner was first proposed by Valiant and Kearns and succeed by Schapire. It is Adaboost. Adaboost is a machine learning meta-algorithm. Aimed to each training set and adjust each weight for each weak leaner based on the accuracy in previous iterative is the main idea of

AdaBoost[12]. It combine such weak leaner into a stronger leaner. The classification model is shown inFig.1.

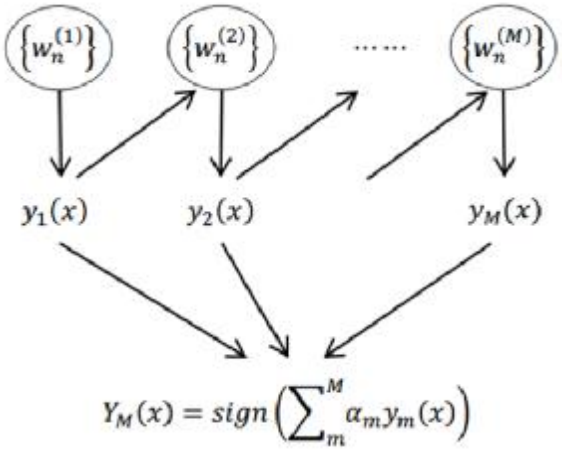

Fig. 1. Adaboost cascade classification model. $y_1(x), y_2(x) \ldots y_m(x)$ represent attributes and x is the input signal. $w_n^{(1)}, w_n^{(2)}, \ldots, w_n^{(m)}$ represent weight for each attribute. The output result of the strong learner in the m times iterative is calculated by function: $Y_m(x) = \mathrm{sign}(\sum_m^M \alpha_m^{(m)} y_m(x))$. α represent a series weights of attributes. Adjusting weight in each iterative to improve the result slightly is the characteristic of AdaBoost.

Adaboost only apply a frame for weak learners. Therefore, it adapts to every classifier. Moreover, it adapt to each attribute and avoiding the attributes' dimension reducing. Last but not least, it reduces the risk of overfitting effectively.

BPNN is utilized as the weak learner in this paper. In order to prevent over-fitting, it is important to choose an appropriate iterative number, hidden layers number and nodes' number for BPNN. The implement BP-AdaBoost process is shown in the BP-AdaBoost algorithm.

TABLE 1 BP-AdaBoost algorithm

Algorithm: BP-AdaBoost algorithm

1: Procedure Input(X,Y)

X represents the sample series and Y represents the label set of X.

The training sample set is $S = (x_i, y_i | i = 1,2,\dots,m)$, $x_i \in X, y_i \in Y = (-1,+1)$

2: end Procedure

3: Procedure Initialization $D_1(i) = \frac{1}{m}$, $w_{ii} = random$, $\theta_i = random$

$D_t(i)$, w_{ii}, and θ_i represent the sample weight, network weight and neurons threshold respectively in the t_{th} iterative.

4: end Procedure

5: Procedure Iteration $t_{th}(t = 1,\dots,T)$

6: **for all** t=1,…,T **do**

7: $y_i(i) \leftarrow f|\sum w_{i:}x_{:} - \theta_i|$ Forward propagation

8: $w_{ii}(t+1) \leftarrow w_{ii}(t) + \varphi * e_j * x_i$ Back propagation

9: **end for**

10: $\epsilon_t \leftarrow P_{w_i \sim D_t}[h_t(X_i) \neq y_i]$ Weak hypothesis

11: $\alpha \leftarrow \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$, α represents the sequence weights

12: **Upadte function**

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & if \ h_t(X_i) = y_i \\ e^{-\alpha_t} & if \ h_t(X_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$ Z represents a normaliazation factor

13: end Procedure

14: Procedure Output (final hypothesis)

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

15: end Procedure

B. Implementation

BP-AdaBoost was applied to classify several sets of experimental data of the U.S. space agency to verify the application in software defect prediction[13]. Data sets includes KC1、PC1、PC5 and MC1. Attributes of each software modules were translated into digitization. A traditional practice of such database is selecting attributes at the very beginning of application, which will improve the simulation result effectively. However, as introduction above, BP-AdaBoost adapt to deal with relevant properties because of its self-adjusting weights. Therefore, in order to evaluate the performance BP-AdaBoost comprehensively, we retain all attributes and evaluate the their prediction model with the defect predict output.

To compare the performance between BP-AdaBoost and other strong classifier, we constructed a weak BPNN classifier for AdaBoost and a strong BPNN for comparison. The parameters of these two classifiers are set as below:

- Weak BPNN classifier (The performance of this classifier should be higher than or equal to 50%.):
  The input dimension(d): 30
  Number of hidden layer(m): 1
  Number of nodes in hidden layer(n): n=d/5
- Strong BPNN classifier (The performance of this classifier should be good.):
  The input dimension(d):30
  Number of hidden layer(m): 5
  Number of nodes in hidden layer(n): n=2d+1

Experiments

A. Evaluation Criteria

There are several data analysis evaluate criteria e.g. accuracy, precision, recall, specificity, F-Measure and G-Mean. All of them are derived from dichotomous dataset
e.g. positive sample and the negative sample, which is shown in TABLE.2.

TABLE 2 Confusion matrix

	Predicted Positive Samples	Predicted Negative Samples
Positive samples	TP	FN
Negative samples	FP	TN

Accuracy respects the percentage between total number of correct predict sample and the whole sample. Similarly, ***precision*** respects the percentage between total number of correct prediction of positive sample and the whole positive sample. ***Recall*** respects the percentage between the number of correct prediction of positive sample and the total number of prediction of positive sample. While *F-Measure* and *G-Mean* respects the relationship between ***precision*** and *recall*. The calculated function of evaluate criteria is shown in TABLE.3.

TABLE 3 Evaluation Criteria

Evaluator	Function
Accuracy	$(TP+TN)/(TP+TN+FP+FN)$
Precision	$TP/(TP+FP)$
Recall	$TP/(TP+FN)$
Specificity	$TN/(FP+TN)$
F-Measure	$2*precision*recall/(precision+recall)$
G-Mean	$Sqrt(recall*specif)$

B. Dataset

We analysis our test result with evaluation criteria. The detail of the dataset is shown in TABLE.4. The positive sample number is much bigger than negative sample number. The number of negative sample and positive sample is unbalance. Moreover, the sample number in this data set is unbalance too. There are 4 subsets CM1, KC3, PC4, PC5 are used in the experiments.

TABLE 4 The detail of data set

data	examples	dimensionality
CM1	327	37
KC3	194	39
PC4	1287	37
PC5	1711	38

C. Experiment results

We training such unbalance dataset with BPNN and BP-AdaBoost to study the performance of BP-AdaBoost in such adverse conditions. 90 percent from each data set was taken as training data and the rest 10 percent was utilized as test data in each simulation. The BPNN prediction model is built in MATLAB with the newff.m function. We integrate the test result in 5 repeat times and summarizing them in TABLE. 5.

Table 5. The detail of experiment results

Dataset	Algorithm	Accuracy	Precision	Recall	Gmean	Specificity	F-measure
CM1	BP	0.867	0.905	0.947	0.292	0.926	0.526
	BP-Adaboost	0.771	0.895	0.805	0.498	0.848	0.663
KC3	BP	0.789	0.822	0.946	0.095	0.880	0.300
	BP-Adaboost	0.769	0.889	0.878	0.291	0.883	0.505
PC4	BP	0.894	0.898	0.989	0.286	0.941	0.532
	BP-Adaboost	0.913	0.941	0.945	0.621	0.943	0.766
PC5	BP	0.754	0.758	0.972	0.181	0.852	0.419
	BP-Adaboost	0.770	0.784	0.944	0.312	0.856	0.543

BP-AdaBoost's results of the evaluation parameters e.g. accuracy, precision, recall, specificity, F-Measure and G-Mean are much better than BPNN. The average BPNN specificity of four data sets is 12.8 percent, while the average BP-AdaBoost specificity is 22.2 percent, which is 9.4 percent higher than BPNN's. And the average G-Mean value of BPNN is 34.03 percent, which is 10.65 percent lower than BP-AdaBoost. In the meanwhile, the total accuracy of both two algorithms are similar. Experimental results shows that BP-AdaBoost improve the accuracy of the minority class under a high total accuracy. Furthermore, it also performs better than BPNN in terms of the comprehensive classification performance.

The further research is shown in Fig. 2. We choose series with different sample size to demonstrate the comprehensive performance of BP-AdaBoost in software defect prediction. We test four data sets e.g. CM1, KC3, PC4 and PC5. The sample number of both CM1and KC3 is small. The total sample number of them are 327 and 194 respectively, while the sample number of PC4 and PC5 are nearly five times of them. We plot the curve of the evaluation criteria of these four data sets with a fixed step length 10, which is shown in Fig. 2. The ordinate represent the accuracy percentage result of four data sets.

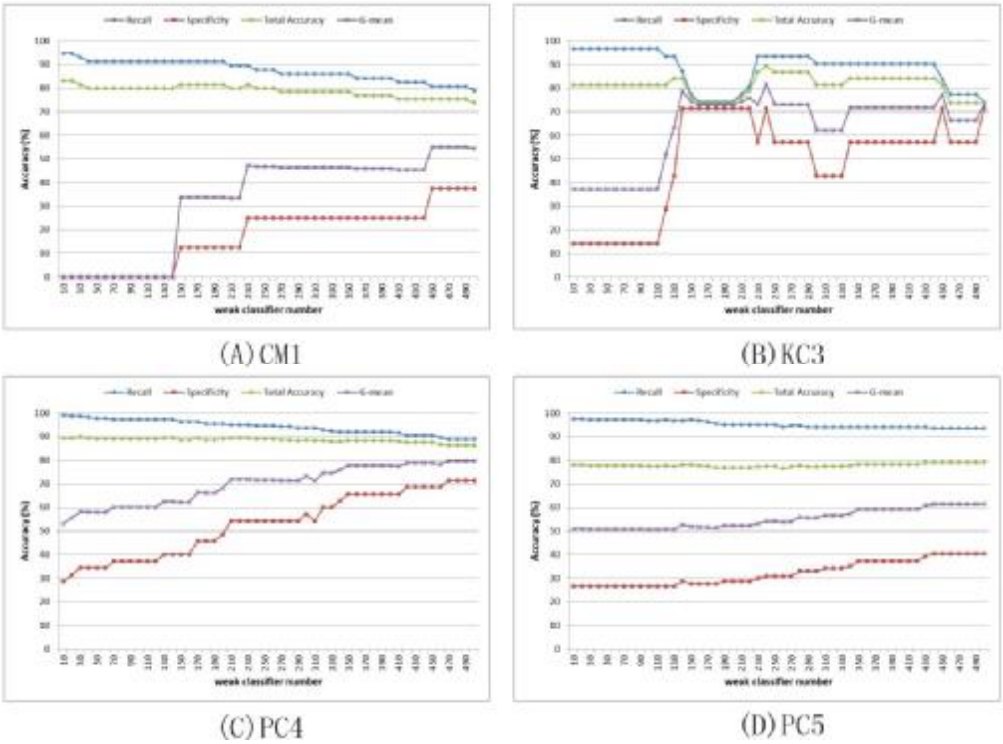

Fig 2. Evaluation Criteria of BP-AdaBoost.

Although the total accuracy rate and recall curve decrease slightly, the specificity and G-Mean curve increase significantly by contrast. This phenomenon can be observed from (A), (B), (C) and (D). The recall curve is very depend on the total accuracy. Therefore, the recall curve is not an independent and meaningful

parameter to support the evaluation. Different to recall, specificity and G-Mean are independent evaluation criteria. Their curves are meaningful and can reflect the performance of classifier. Specificity and G-Mean results of (C) and (D) are increase smoothly, while results of (A) stepped rise in the figure. Moreover, curves in figure (B) are messy because of its sample number. Data (C) and (D) have much more data sample than data (A) and (B). If the sample number is not big enough, a high accuracy rate prediction model is hard to be developed. A bigger sample number series will attribute to a better prediction result.

The curve of fig. 2 shows the good adaptability of BP-AdaBoost in software defect prediction. However, it still poor to predict the minority class data. Although its poor prediction of minority class data, it can handle the unbalance data set and feedback a high accuracy rate result. We believe that it is a suitable classifier for software defect prediction. Moreover, it adapt to combine other weak classifier but not only combine to BPNN, which bring the possibility to deal with a variety of classification problems. On the other hand, the Cascading multiple weak classifiers is the essence of adaboost schemes. To build a suitable weak classifier is the key for AdaBoost application.

Conclusion

In this paper, we applied BP-AdaBoost in software defect prediction. In the meanwhile, we evaluate the prediction result of BP-AdaBoost and BPNN to compare their performances. During our research result, we find out that the BP-AdaBoost performance is much better than BPNN in software defect prediction, especially in dealing with the unbalance class. Moreover, BP-AdaBoost have a stronger generalization then BPNN.

We innovative implement BP-AdaBoost in software defect prediction and compare its performance with traditional classify method. We will study the deep analysis and comprehensive assessment base on the application of AdaBoost in software defect prediction in the future. Furthermore, we will study the combination of AdaBoost and other classification method such as decision tree, support vector machine and neural networks.

Acknowledgment

The work was supported by Postdoctoral Science Foundation of China (No. 2014M552171), and GDSTP (2014A040401015).

References

[1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A general software defect-proneness prediction framework,” Software Engineering, IEEE Transactions on, vol. 37, no. 3, pp. 356–370, 2011.

[2] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” Journal of computer and system sciences, vol. 55, no. 1, pp. 119–139, 1997.

[3] T. M. Khoshgoftaar and N. Seliya, “Tree-based software quality estimation models for fault prediction,” in Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on. IEEE, 2002, pp. 203–214.

[4] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, “Accuracy of software quality models over multiple releases,” Annals of Software Engineering, vol. 9, no. 1-2, pp. 103–116, 2000.

[5] T. M. Khoshgoftaar, Y. Liu, and N. Seliya, “A multi-objective module order model for software quality enhancement,” Evolutionary Computation, IEEE Transactions on, vol. 8, no. 6, pp. 593–608, 2004.

[6] S. Ma and T. Du, “Improved adaboost face detection,” in Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on, vol. 2. IEEE, 2010, pp. 434–437.

[7] H. Zhang, Y. Xie, and C. Xu, "A classifier training method for face detection based on adaboost," in Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on. IEEE, 2011, pp. 731–734.
[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Cognitive modeling, vol. 5, 1988.
[9] N. Gupta and M. P. Singh, "Estimation of software reliability with execution time model using the pattern mapping technique of artificial neural network," Computers & operations research, vol. 32, no. 1, pp. 187–199, 2005.
[10] T. M. Khoshgoftaar, A. S. Pandya, and H. B. More, "A neural network approach for predicting software development faults," in Software Reliability Engineering, 1992. Proceedings., Third International Symposiumon. IEEE, 1992, pp. 83–89.
[11] M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," Journal of systems and software, vol. 76, no. 2, pp. 147–156, 2005.
[12] Z. Zhang and X. Xie, "Research on adaboost. m1 with random forest," in Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, vol. 1. IEEE, 2010, pp. V1–647.
[13] M. Chapman, P. Callis, and W. Jackson, "Metrics data program," NASA IV and V Facility, http://mdp. ivv. nasa. gov, 2004