International Journal of Computational Intelligence Systems, Vol. 7, No. 4 (August 2014), 715-723

Software Fault Estimation Framework based on aiNet

Qian Yin
Image Processing and Pattern Recognition Laboratory, Beijing Normal University,Beijing, China
yinqian@bnu.edu.cn

Ruiyi Luo
Image Processing and Pattern Recognition Laboratory, Beijing Normal University,Beijing, China
luoruiyi2008@163.com

Ping Guo
Image Processing and Pattern Recognition Laboratory, Beijing Normal University, Beijing, China
pguo@ieee.org

Received 13 March 2013

Accepted 29 March 2013

Abstract

Software fault prediction techniques are helpful in developing dependable software. In this paper, we proposed a novel framework that integrates testing and prediction process for unit testing prediction. Because high fault prone metrical data are much scattered and multi-centers can represent the whole dataset better, we used artificial immune network (aiNet) algorithm to extract and simplify data from the modules that have been tested, then generated multi-centers for each network by Hierarchical Clustering. The proposed framework acquires information along with the testing process timely and adjusts the network generated by aiNet algorithm dynamically. Experimental results show that higher accuracy can be obtained by using the proposed framework.

Keywords: software fault prediction, aiNet, testing, framework

1. Introduction

Software fault Prediction technology is very important for software testing because it can effectively guide the software testing and improve software quality. Lots of techniques have been proposed to identify fault-prone modules by classify the software modules[1,2,3]. The classical prediction method is using software metrics and fault data from a previous system or similar software project developed previously[4]. The problem arisen is that, the previous system or project has much difference with the developing system, such as the complexity, skill level of software engineers, management level and so on. Therefore, some researchers turn to use different methods to analyze the developing system instead of previous system. However, there are two problems for all researchers. One is how to obtain higher accuracy; the other is how to obtain prediction results earlier. In this paper, we propose a novel prediction method to deal with those two problems by introducing the artificial immune network (aiNet) algorithm[5] into the software fault estimation framework. The compressed representation was chosen from dataset to decrease computational complexity and eliminate data redundancy, because the aiNet algorithm does well in data expression. The prediction results can be obtained in the very early stage of software life cycle, because our framework can start prediction with little prior data and the aiNet algorithm can adjust its network dynamically with the data obtained increasing. The experiment results show that higher accuracy can be obtained by using the proposed framework.

The paper is organized as follows. In section 2, we generally introduce the software fault prediction method. In section 3, the framework using aiNet for software prediction is introduced. In section 4, two experiments are carried out on the dataset using this framework, and the results obtained are also shown in this section. Section 5 concludes this paper and gives some suggestion for the future work.

2. Software fault Prediction

The software fault prediction technology can be divided into static prediction and dynamic prediction. Static prediction mainly predicts the fault distribution or numbers based on software metrics data which relate to

Copyright: the authors

fault. Dynamic prediction methods are mainly based on the time to predict the fault distribution based on the fault or defect generated time[6,7,8,9]. Our method belongs to static prediction as we used software metrics for prediction in our software prediction framework this paper. Kitchenha, Basili, Khoshgoftaar have collected and analyzed software fault data and software product metrics, software technology (OO,Web) and other metrics data relate with software progress and executed process. They classify and using regression techniques on those data. The result shows that the faults distribution fits 2-8 principles. It needs classification and regression technique to recognize and predict those prediction prone modules. In learning problem, classification belongs to pattern recognition problem, which is to estimate indicator function (The indicator function have the value only 0 and 1). However, regression technique is used to solve regression function estimation problem, that is, estimate real function. Especially in software fault prediction, classify are mainly used to verify whether the module are belong to high-fault-prone or low-fault-prone. Regression techniques are mainly used to estimate the fault number. Those two technologies are used to guide the testing process, to save the expensive time and review cost. In this paper, like the classification method, we use software metrics data for prediction, and our method can also be improved to predict like regression technique.

The most common classify technologies that we call then qualitative classification are: linear discriminant analysis(LDA)[10], Boolean discriminant function(BDF)[11], classification and regression tree (CART)[12], clustering analysis(CA)[13], support vector machine (SVM) and so on. But unlike those classification methods, our method starts earlier and using multi-centers drew by aiNet to prediction.

3. The Estimation Framework based on aiNet

3.1 *Review of aiNet Algorithm*

Since the immune network theory has been proposed by Jerne in 1974[14], and the colonel selection and affinity maturation algorithms proposed by Burent, Ada & Nossal in 1987[15], artificial immune systems(AIS) have drawn much attention and inspired many interesting

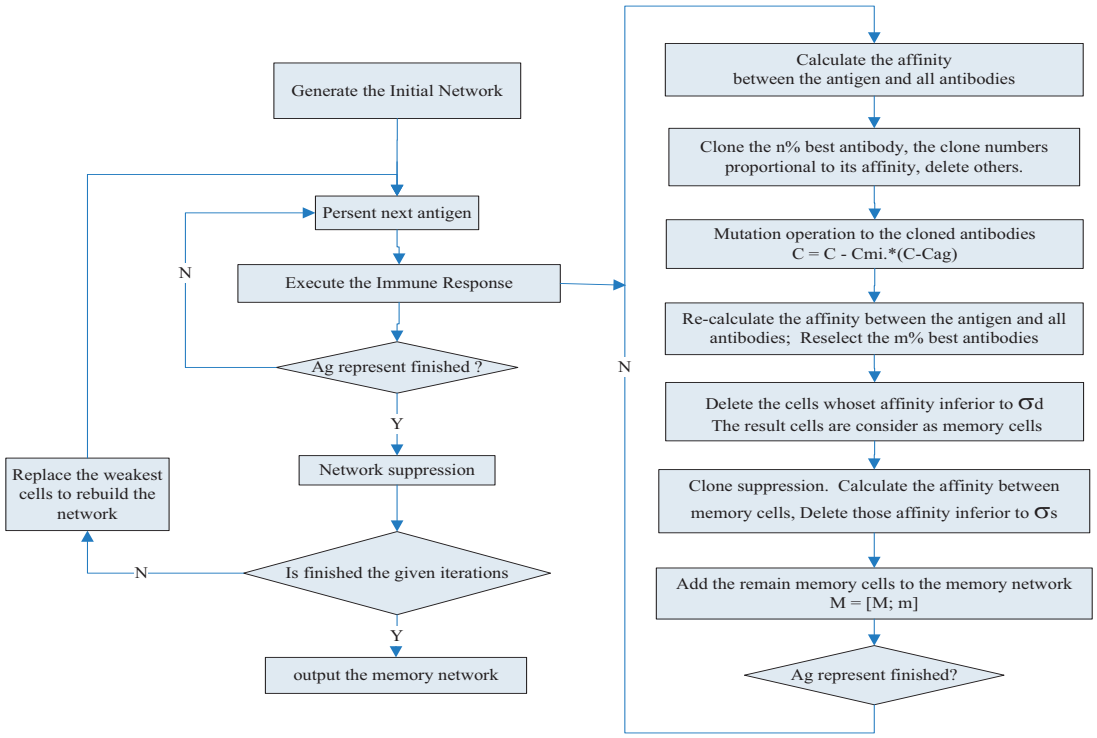


Fig. 1. Artificial Immune Network algorithm detail flowchart

Copyright: the authors

algorithmic solutions in fields like clustering, classification[5,16], data mining, data compression etc. aiNet algorithm, which is one branch of AIS, has the general purpose of finding a compressed representation of the input set, by eliminating redundancy of it. It forms the network by considering the amount and connectivity strength of antibodies and mirrors the internal structure of the immune system[17]. In our framework, the essence of the module metrics dataset is drawn by aiNet continuously with the dataset growing larger. Vectors that represent module metrics are considered as foreign antigens here, and be gradually present to stimulate the network.

During the stimulation process, some antibodies owning the better ability in recognizing and eliminating foreign antigens are transformed into long-life memory antibodies. Because the immune system continually produces novel antibodies and the set of memory cells are continuously growing, to avoid the memory cells increase without restrictions, non-stimulated and self-reactive cells were eliminated along with the two suppression process. Finally, certain worst individuals will be replaced by novel randomly generated ones. And more details about this algorithm have been described in Refs. 5. There are also some variants[18,19,20] of aiNet have drawn and gain some achievements.

As shown in the figure 2 cited from Refs. 5, the modules are considered as antigen, and each dim of the software metrics are consider as the epitopes. We use the modules to establish a memory cells' immune network. When a new module is generated, we judged it belongs to the network that can recognize it mostly. Also the real testing result will be considered as new antigen and cause the immune response to adjust the network. Epitopes, paratopes and idiotopes are related with the suppression mechanism we also used. The detail framework will be given in next section.

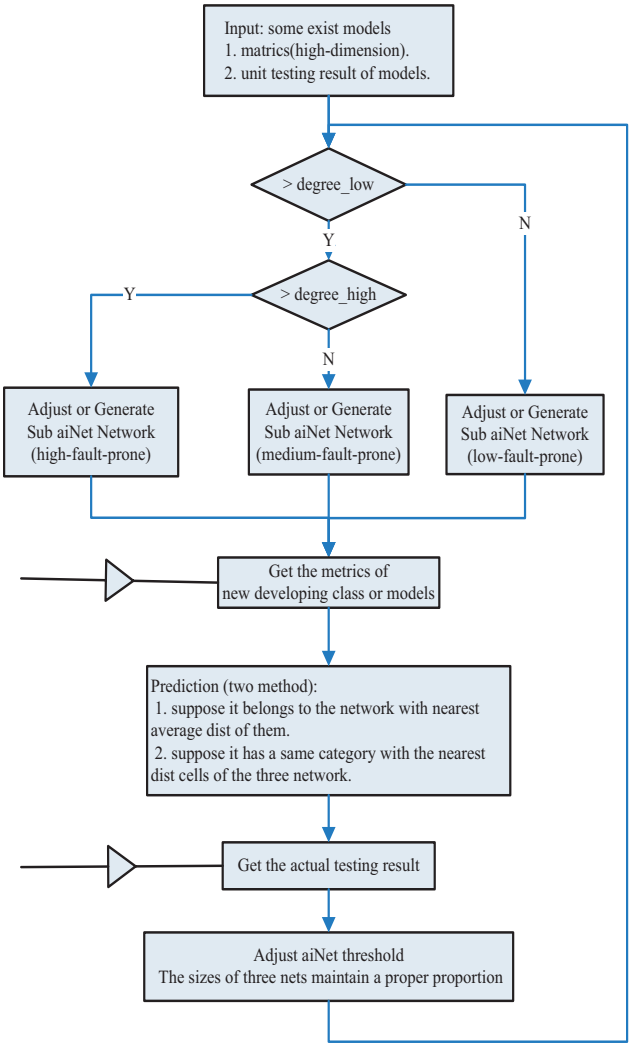


Fig. 3. Prediction Framework Flowchart based on aiNet.

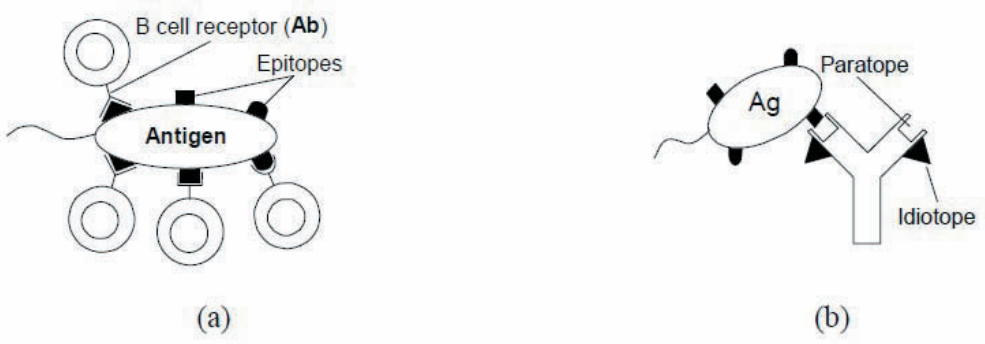


Fig. 2. B cell, antigen, antibody, epitopes, paratopes and idiotopes.

Copyright: the authors

3.2 *Prediction Framework based on aiNet*

Unlike the software reliability prediction with Elman network[21], the goal of software fault prediction contains a high prediction precision, adjustable error rate which predict low-risk module to high-risk module or on the contrary, and the ability to adapt different software in application. The framework we proposed tries to achieve those goals.

As the flowchart shown in the figure 3, our framework fits the real predicting and developing process. The degree we defined in the figure related with fault prone degree, the higher the degree, the more faults this module may contain. Complete prediction framework is as follows:

- Generate three aiNet networks at first.
- Initialize the network. Divide the first few modules which have been developed and tested into three classes based on testing result: low, medium, high fault prone. Then, use them as antigens to initialize the three networks respectively.
- Predict which class the newly developed module belongs to when a module is produced. The proper testing resources can be assigned.
- After obtaining the real testing result, feedback it to the corresponding network timely and adjust the network. Thus, it can be more accurate for the next prediction

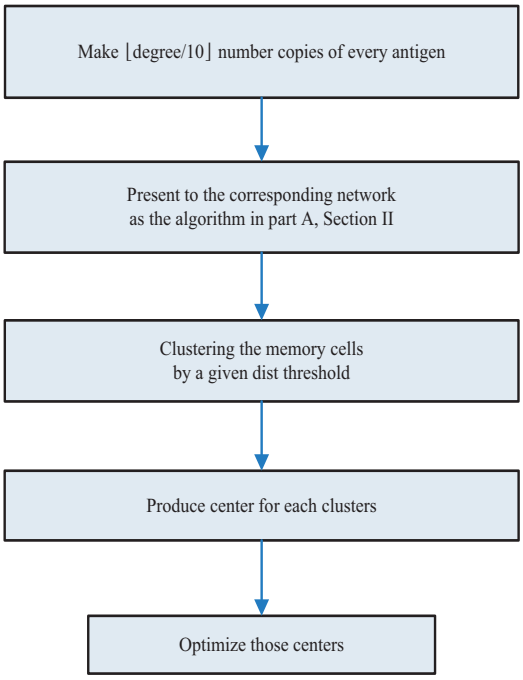


Fig. 4. Network adjust flowchart.

In the third step, there are two methods used to predict the newly developed modules. One is Menu Network Dist Prediction, which calculates the distance between the new module and the three networks respectively, like the Eq. (1). The new module belongs to nearest class. The other is Cell Center Dist Prediction. As the framework generate multi-centers for prediction though aiNet, we calculate the distance to select the nearest center. The new module has the same category with the nearest center. This process is the Eq. (2). Euclidean Distance is used in these two methods for measure.

$$aff_{net} = \sum_{i=1}^{netsize} \sqrt{\sum_{j=1}^{\dim} \left(s_j - ag_{ij}\right)^2} \;/\; netsize \qquad (1)$$

$$aff_{cent} = \sqrt{\sum_{j=1}^{\dim} \left(s_j - c_j\right)} \qquad (2)$$

The last step of the framework is a learning and feedback process which is along with the testing. Note that testing and developing progress can be executed parallel in software developing.

The second and fourth steps are used to adjust network and the details are shown in figure 4. Firstly, considering that higher degree modules are more dangerous to software, we enhance the influence of these modules by making copy_number(proportional to its degree) copies of each antigen before we present them to the networks. Then, generate some clusters for each network using hierarchical clustering with the given parameter δ_{dist}. And centers are formed using the mean value of the cluster. Finally, optimize those centers by deleting the outliers.

aiNet algorithm has some parameters. How to adjust them to obtain better prediction results will be discussed in next section in detail.

3.3 *Parameters control of aiNet for prediction*

Parameters are very important to the aiNet algorithm and greatly affect the result of our framework. Among those parameters the suppress threshold δ_s is the most important one. It influences the network structure and size.

The smaller δ_s, the more memory cells will be generated, and the size of network will be larger. According to the aiNet algorithm[5], antibodies in input

Copyright: the authors

set close (Euclidean distance) to antigens in the network will be remained, and the cells whose distance between each other are inferior to the threshold δ_s in memory cells set will be eliminated[15]. Therefore, the output set only consists of cells that are departed from each other with distance δ_s. The finally remained cells are memory cells and compose the memory networks. As the size of output sets should be kept proportional to the three raw testing result set, we suggest that δ_s be assigned a value proportional to the mean distance of the network.

Also the parameter σ_{dist} greatly affect the centers generated, and we used this to control the center numbers of each network proportion to the center number. Details about this parameter will be shown in section 4.2.1.

3.4 *An analysis of this framework*

The framework receives a set of software metrics (represented by the set S) as inputs, which are going to be presented to the network. The s_i is a multidimensional vector that represents the module developed during the software development. And the framework returns an immune network composed of a set of memory cells and connections between them (represented by the set M).

$$S = \{s_1, s_2, \dots, s_i, \dots s_n\} \quad (3)$$

$$M = \{m_1, m_2, \dots, m_i, \dots m_l\} \quad (4)$$

And S are divided into two or three subset S_1 and S_2 (or S_1, S_2 and S_3 if we want to predict three kinds: low, medium, high fault prone) according to their degree.

The first step is to create two (or three) initial sets of B-cells (represented by the set B, the subset is B_1, B_2). After this, an iterative process is performed by presenting the set of antigens to the network. For each antigen s_i, the stimulations between it and the whole B-cell set are calculated. S_1 is interacted with B_1 and S_2 is interacted with B_2. The procedure of S_1 is interacted with B_1 is represented by the following function, and others are the same with it:

$$M_i = f_{simulation}(s_i, B) \quad (4)$$

The function $f_{simulation}$ is a rather complex process. Its process is described in section 3.2. For each s_i, it generate a small memory set: M_i.

$$M = g_{supression}(M_1, M_2 \dots M_i, \dots M_l) \quad (5)$$

The function $g_{supression}$ collects the M_i generated by Eq. (5). And suppress the set to generate a memory set M.

The whole process will be repeated for some interactions when the M reaches a stable status. Finally there is an indictor function when the s_{i+1} is developed:

$$\begin{cases} s_{i+1} \in prone_{high} & \text{If closest one belong to High} \\ s_{i+1} \in prone_{low} & \text{If closest one belong to Low} \end{cases}$$

After the real testing result obtained. The s_{i+1} will be simulated with the whole corresponding B-cell set to generate its M_{i+1}. Also the suppression process will be executed to maintain the memory set M which is also the aiNet network.

4. Experiment

This part we will give out the experiment and the result of our frame. The dataset we used will be first introduced and two experiments especially about low and high fault prone prediction and low, medium and high fault prone prediction are designed. Note that all the data used for showing in this paper have been reduced the dimension by Principal Component Analysis (PCA). However, we use high dimension data during the progress because our model with aiNet can deal it well.

4.1 *Dataset Description*

The data used in this paper is from the Medical Imaging System (MIS) dataset[23]. This dataset represents the results of an investigation of software for MIS system. The system consists of approximately 4500 modules. Amounting to about 400000 lines of code written in Pascal, FORTRAN, assembler, and PL/M. What we used for analysis in this paper is randomly selected from the modules coding in Pascal or FORTRAN of the program and consists of approximately 40000 lines of codes which belongs to 390 modules. The dataset consists of change recorded which is an indicator of software development effort and 11 software complexity metrics for each of the programs. The metrics are shown in table 1.

Copyright: the authors

Table 1. MIS software metrics

Metrics	Description
LOC	Coding lines(including comments)
CL	Coding lines excluding comments
Tchar	The number of characters
Mchar	The number of character
Tcomm	The number of comment character
Dchar	The number of code characters
N	Halstead program length
N^	Halstead's estimated program length
NF	Jensen's estimator of program length
V(G)	McCabe's cyclomatic number
BW	Belady's bandwidth metric

4.2 Evaluation Criteria

In this paper, the classification error rate is used to estimate the experimental result. To classifying a module to be fault-prone or non-fault-prone, there are two types of prediction error can be made, Type I error and Type II error. Type I error means that a program is predicted with faults while in fact it is not. Type II error means that when we believe that a program is relatively fault-free but the fact is on the contrary. Type II error rate is more important than Type I error rate in considering the quality of a classification model. Whereas, we also calculate Type I error in our experiments. If Type I error rate is very high, software managers will waste much time on testing non-fault-prone modules. Type II error will cause lack of testing with the high fault modules, and much increase the workload and maintenance cost. The strategy to have both Type I and II as balanced as possible while keeping them as low as possible should also be considered[24,25]. Their calculate process please refer to table 2[26].

Here we used those measures to judge the result: Type I error means that a program is predicted with faults while in fact it is not. Type II error means that when we believe that a program is relatively fault-free but the fact is on the contrary. Generally, it is better to make a Type I error of misclassification than to make a Type II error. However, if the Type II error is too high, a lot of testing will be wasted. Some research has been done on how to balance the two error rate[22,23]. Besides those two measures, the Global Correct Classification Rate (GCCR) index is also used.

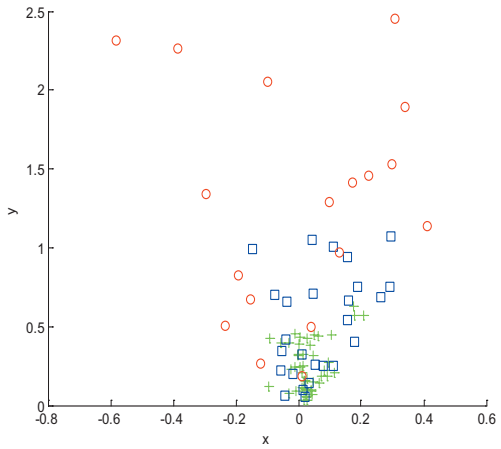
4.3 Experiment on high and low classification.

We use the dataset described in in section 3.2 to simulate the real developing and testing process. Figure 5(a) shows the distribution of the first 90 modules using to initialize the network. Blue points in the figure represent the modules containing changes less than 2, that is, its degree<2; similarly, the green ones, 2<=degree<10 and the red ones, degree>=10. Figure 5 (b) shows that the antigen with a higher degree are enhanced as described in section 3.2. As a higher recognition rate with the modules that contains more faults are wanted, we enhanced those modules by generating copy_number times of its copies. Such as, if a module's real fault degree is 23 after testing, 2 copies will be present and stimulate the network. In this experiment, we just consider the low fault prone modules and the high fault prone modules as in reference[19]. Figure 6(a) shows that the original metrics distribution of the first 90 modules. Figure 6(b) shows the centers generated by two networks after 90 modules. we could see that the centers generated by high fault prone modules are much more scattered. Those centers represent the modules well with a proper scale and distribution. We use those center to predict the new developed modules.

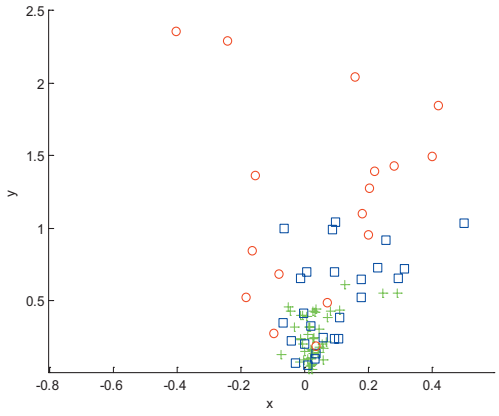
Table 2. Type I and Type II misclassification.

		Module actually has defects	
		No	Yes
Classifier predicts no defects	No	*a*	*b*
Classifier predicts some defects	Yes	*c*	*d*
			Accuracy=$(a+d)/(a+b+c+d)$
			Probability of detection=$d/(b+d)$
			Probability of false alarm=$c/(a+c)$
			Type I error of misclassification=$c/(a+c)$
			Type II error of misclassification=$b/(b+d)$

Copyright: the authors

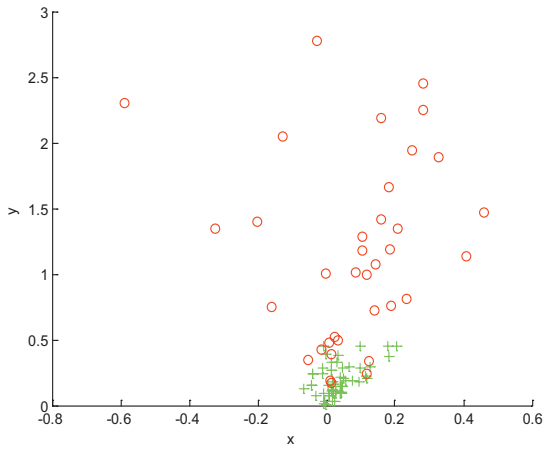


(a)

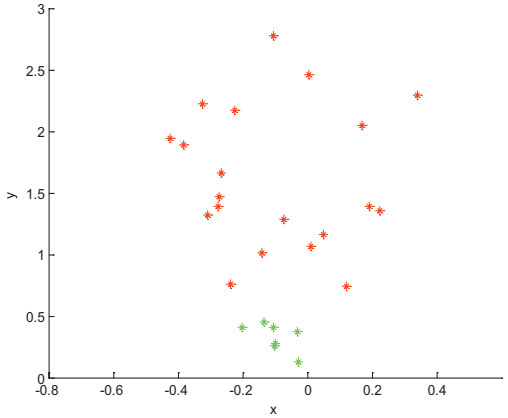


(b)

Fig. 5. Data distribution of first 90 modules



(a)



(b)

Fig. 6. Original metrics data and centers generated

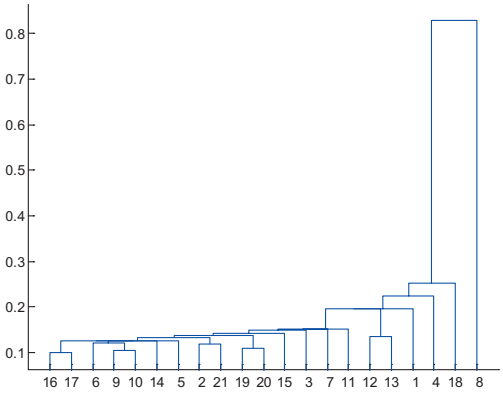


Fig. 7. Hierarchical clustering tree of the low-fault-prone network

The formation of those centers contains two processes have been described in figure 2: Firstly, cluster the network cells which can be explained as explained in the hierarchical cluster tree in figure 7. From the leaves to root, we cut the tree at a certain distances, which is σ_{dist}. Leaves that are still connected with each other will be grouped into a cluster. Secondly, produce the centers of each cluster using the mean value. And after centers generated, centers optimized process is executed. We eliminated the outlier centers in the low fault prone network especially and also eliminated the centers that are too much close with the centers in the other network based on the idea that the high default prone modules are scattered and the low ones are more centralized. Optimized step are important as the outliers effect the result greatly.

Copyright: the authors

From the table 3 we conclude that with the increasing number of modules, our method obtain better results. The type II error rate is also much lower than Ref. 19. In the process of prediction, the two methods given in section 3.2 are all used. However, the Cell Center Dist. Prediction method is about 20% more accurate than another. We consider it is because that: The Menu Network Dist. Prediction method generated only one center in essence. As the high fault prone modules is much scattered, more centers used during prediction process can obtain much better results.

Table 3. Accuracy rate of low, high fault prone prediction on MIS

Modules	Type I	Type II	GCCR
0-100	8%	18%	82%
100-200	7%	11%	86%
200-300	6%	10%	89%

4.4 Experiment on low, medium and high prediction.

In this experiment, the modules produced are classified into three classes: low default prone, high default prone or medium default prone. Figure 8 shows the centers generated of the three networks after 90 modules have been presented. The blue points represent the medium fault prone modules. And the other processes are the same as the last experiment.

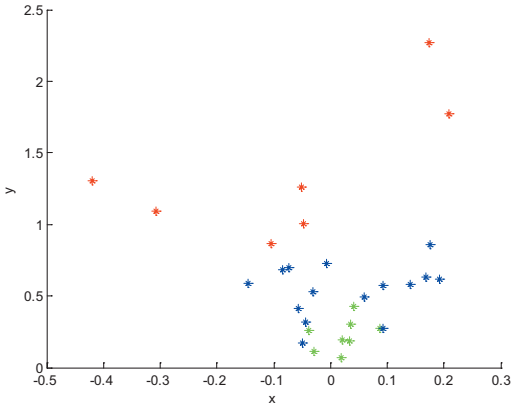


Fig. 8. Centers generated after 90 modules presented

As mentioned in figure 2, we can control the center numbers by adjusting δ_{dist} to influence the probability of the prediction result. It can be seen from table III that when the threshold δ_{dist} is assigned 0.3, there are 2 clusters generated of low-fault-prone network, and thus, 2 centers. We find that when δ_{dist} is the same, the higher fault prone network generated more centers than the low. The reason is that higher fault prone modules are more scarred. As when the threshold δ_{dist} is assigned 0.3, the lower fault prone network will generate 2 centers after the 90 modules represented. However, there will be 8 centers generated by high fault prone network, details please refer to the table 4.

Table 4. The center number of each network

δ_{dist}	Low	Medium	High
0.20	3	8	13
0.25	2	7	7
0.30	2	5	8

Table 5. Accuracy rate of low, medium and high fault prone prediction on MIS

Modules	H-F-P	M-F-P	L-F-P	GCCR
0~100	61%	42%	77%	57%
100~200	63%	30%	85%	60%
200~300	69%	40%	82%	62%

The H-F-P, M-F-P, L-F-P in the table 5 represent the accuracy rate of prediction results for high, medium and low fault prone modules respectively. We can see from the table, the prediction result is not as good as the previous experiment. The prediction accuracy of high fault prone modules is only 69%. The accuracy of low fault prone modules is 85%. This is due to that the medium modues always have not a distinct distribution with low and high prone moudues.

5. Conclusions and future work

In this paper, we proposed a novel software fault prediction framework mainly used for module or unit testing. Our prediction method can start at an early stage of software lifecycle and adjust itself along with the testing process. The framework using aiNet algorithm to simplify the metrics dataset and adjusts the network dynamically as the developing and testing process

Copyright: the authors

parallel goes on. Experiments show that the method in this paper we proposed has a better performance. Also, the framework has a better adjustable ability to emphasize on high-fault-prone module because the parameter of the hierarchical clustering process can be adjusted to generate proper number of centers. In the future work, we will try to simply the process and parameter control and the influence of different metrics should be studied. What's more, a integrated software fault prediction system can be established with an effective aiNet algorithm in future work.

6. Acknowledgements

The research work described in this paper was fully supported by the grants from the "863" Program (Project No.2009AA010314), the National Natural Science Foundation of China (Project No. 60675011, 90820010). Prof. Qian Yin and Ping Guo are the authors to whom all correspondences should be addressed.

7. References

1. M. D. Neil. Multivariate assessment of software products, Journal of Software Testing, Verification and Reliability, 1992, Vol.1,No.4, pp.17-37.
2. J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs, IEEE Transactions on Software Engineering, 1992, Vol.18, No.5, pp.423-433.
3. F. Xing, P. Guo, and M.R. Lyu. A novel method for early software quality prediction based on support vector machinem. In Proc. of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Chicago, Illinois, Nov. 2005, pp. 213-222.
4. Taghi M. Khoshgoftaar and Naeem Seliya, Analogy-based practical classi¯cation rules for software quality estimation," Empirical Software Engineering Journal, vol. 8, no. 4, pp. 325-350, December 2003.
5. De Castro L.N. and Von Zuben F.J.: An evolutionary immune network for data clustering. Proc. Sixth Brazilian Symp. Neural Networks, pp. 84-89 (2000)
6. Xie M. Software Reliability Modelling. Singapore: Word Scientific Publishing Co. Pte. Ltd., 1991.
7. Lyu M. Handbook of Software Reliability Engineering. Singapore: McGraw-Hill, 1996.
8. Trachtenberg M. Discovering how to ensure software reliability. RCA Engineer, 1982,27(1):53−57.
9. Jelinski Z, Moranda P. Software reliability research. In: Freiberger W, ed. Statistical Computer Performance Evaluation. New York: Academic Press, 1972. 465−484.
10. Munson JC, Khoshgoftaar TM. The detection of fault-prone programs. IEEE Trans. on Software Engineering, 1992,18(5):423−433.
11. Khoshgoftaar TM, Seliya N. Improving usefulness of software quality classification models based on boolean discriminant functions. In: Proc. of the 13th Int'l Symp. on Software Reliability Engineering. IEEE Computer Society Press, 2002. 221−230.
12. Khoshgoftaar TM, Yuan X, Allen EB. Balancing misclassification rates in classification-tree models of software quality. Empirical Software Engineering, 2000,5(4):313−330.
13. Zhong S, Khoshgoftaar TM, Seliya N. Analyzing software measurement data with clustering techniques. IEEE Intelligent Systems, 2004,19(2):20−27.
14. Jerne, N. K. (1974a), Towards a Network Theory of the Immune System, Ann. Immunol. (Inst.Pasteur) 125C, pp. 373-389.
15. Burnet, F. M. (1978), "Clonal Selection and After", In Theoretical Immunology, (Eds.) G. I. Bell, A.S.Perelson & G. H. Pimbley Jr., Marcel Dekker Inc., pp. 63-85.
16. Timmis J (2000) Artificial Immune Systems: A Novel Data Analysis Technique Inspired by the Immune Network Theory, Ph.D. Dissertation, Department of Computer Science, University of Wales.
17. Stibor T, Timmis J (2007) An investigation into the compression quality of ainet. In: Fogel D (ed) Proc of foundations of computational intelligence. pp 495-503
18. M. Neal. An Artificial Immune System for Continuous Analysis of Time-Varying Data. In J. Timmis and P. J. Bentley, editors, Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS), volume 1, pp.76 – 85.
19. M. Neal. Meta-Stable Memory in an Artificial Immune Network. In J. Timmis, P. Bentley, and E. Hart, editors, Proceedings of the Second International Conference ICARIS, pages 168 – 180.
20. J. Timmis and M. Neal. A Resource Limited Artificial Immune System for Data Analysis. Knowledge-Based Systems, 14:121 – 130, 2001.
21. Xuchao Cheng, Xinyu Chen, and Ping Guo. Software reliability prediction with an improved Elman network model, Journal of Communications, in press, 2011.
22. T. Khoshgoftaar, V. Joshi, and N. Seliya. Detecting Noisy Instances with the Ensemble Filter: A Study in Software Quality Estimation. International Journal of Software Engineering and Knowledge Engineering, Vol 16, No 1, 2006, pp. 1-24.
23. M.R. Lyu. Handbook of Software Reliability Engineering, IEEE Computer Society Press. McGraw Hill, 1996.
24. John C. Munson, Member, IEEE, and Taghi M. Khoshgoftaar, The Detection of Fault-Prone Programs IEEE Transactions on Software Engineering, 1992,vol. 18, no. 5, pp. 423-433
26. F.Xing, P. Guo, M.R.Lyu. Comparison of Methods of Controlling Two Types of Error Ratio Applied to Software Reliability Prediction. Journal of Nanjing University, 2005, Vol.41, pp.644-649.
27. http://mdp.ivv.nasa.gov/mdp_glossary.html

Copyright: the authors