Software Defect Prediction Using Non-Negative Matrix Factorization

Ruihua Chang
Xi’an Research Inst. of Hi-Tech, Xi’an, China
Email: sxwcrh@163.com

Xiaodong Mu and Li Zhang
Xi’an Research Inst. of Hi-Tech, Xi’an, China
Email: mu_msn@msn.com and zhangli_522@126.com

***Abstract*—Quality is considered as an important issue in the fields of software engineering. However, building quality software is very expensive, in order to raise the effectiveness and efficiency of quality assurance and testing, software defect prediction is used to identify defect-prone modules in an upcoming version of a software system and help to allow the effort on those modules. Although many models have been proposed, this problem has not resolved thoroughly. For overcoming these limits, recent results show that researcher should pay more attention to improve the quality of the data. Aimed at this purpose, in this paper, we propose a novel approach to resolve the problem of software defect prediction. The method is classification using Non-Negative Matrix Factorization (NMF). In this paper, NMF algorithm is not only used for extracting external features but also as a powerful way for classification of software defect data. Experiments demonstrating the efficiency of the proposed approach are performed for software defect data classification. And the results show that it outperforms the state of the art techniques tested for this experiment. Finally, we suggest that it can be a useful and practical way addition to the framework of software quality prediction.**

***Index Terms*—software defect, prediction, Non-negative Matrix Factorization, software metrics, F-Measure**

I. Introduction

Quality can be used to assess and estimate final software product quality. Over the past decades, researchers have addressed the importance of integrating quantitative validation in the software development process, in order to meet different requirement. Traditional software development focus on software correctness, introducing performance issues later in the development process. This style of developing has been often referred as a “fix-it-later” approach [1]. However, building high quality software is very expensive; the effort for applying software quality assurance measures is very limited. At the same time, many of them face a tradeoff between quality and cost. To raise the effectiveness and efficiency of quality assurance and testing, defect prediction is used to identify defect-prone modules in an upcoming version of a software system and help to allow the effort on those modules. In this research fields, there has been a growing interest in the subject and several approaches to early software performance predictive analysis have been proposed. Although many models have been proposed, this problem has not resolved thoroughly. In a recent study, Khoshgoftaar and Seliy[2] have empirically demonstrated that while using a very large number of diverse classification techniques for building software quality classification models, classification accuracy does not show a dramatic improvement. Instead of searching for a classification technique that performs well for a given software measurement dataset, they concluded that the software process should focus on improving the quality of the data. As pointed out by Menzies et al. [3] all data miners hit a performance ceiling effect when they cannot find additional information that better relates software metrics with defect occurrence. What we observe from recent results is that current research paradigm, which relied on relatively straightforward application of machine learning tools, has reached its limits. Some researchers start to resolve this problem. For example, Burak [4] used project data from multiple companies to resolve it. However, these features from different sources come at a considerable collection cost. Another way to avoid these limits is to mining as more knowledge as possible. Considering these observations, we pay more attention to increasing the information content in metric data and improve the quality of the data.

NMF is a popular technique and has been recently developed for decomposing a data matrix into non-negative factors and it has been used for object and pattern recognition, data analysis, and dimensionality reduction. However, to the author's knowledge, there are a few references to research software defect prediction based on NMF algorithms. NMF has been already applied for more and more fields with its advantage of straightforward implementation. After researching, in this paper, we propose a novel approach to resolve above problem about software defect prediction. We show that

Manuscript received August 10, 2010; revised and accepted September 10, 2010.
Corresponding author: Ruihua Chang;sxwcrh@163.com.

NMF is a powerful technique for successful extraction of features in software defect prediction. And in our method, software defect data set is represented as a nonnegative matrix, as negative data is meaningless. Its output is easy to interpret for researchers and testers. So in this paper, NMF algorithm is not only used for extracting external features but also as a powerful way for classification of software defect data. Finally, the application for software defect data classification is investigated by comparing with the state of the art classifiers.

The rest of this paper is organized as follows: Firstly we introduce related work about software defect prediction and NMF algorithms. Secondly the data and measure of evaluation used in the experiments are described. Thirdly, we present the results of applying the NMF algorithm for classification of software defect data. We simultaneously analyze and compared the results. Finally, we give our conclusion and works in the future.

II. Related work

In this section, we will briefly present overview of software defect prediction and review the major results of non-negative matrix factorization.

A. Software defect prediction

Before explaining defect prediction, we should first define what we are trying to predict: 'defect'. Unfortunately, the perception of what a defect is varies in different contexts. Based on contextual classification of software systems, in defect prediction perspective, a defect is defined by the context of the software system, considering what practitioners want to predict [4]. Accordingly a defect predictor is a tool or method that guides testing activities. Defect predictors are used to make an ordering of modules to be inspected by verification and validation teams. Software defect prediction is handled as a regression problem or a classification problem. For both types, the granularity level of predictions may vary depending on the availability of data. In this paper, we employ the second application type and view it as a supervised binary classification problem. Software modules are represented with software metrics, and are labeled as either defective or non-defective.

Until now, a wide range of statistical and machine learning models have been developed and applied to predict defects in software such as linear regression, discriminate analysis, decision trees, neural networks and naive bayes and so on. Munson and Khoshgoftaar [5] investigate linear regression models and discriminate analysis to conclude the performance of the latter is better. Bullard et al. [6] employ a rule based classification model in a telecommunication system and reported that their model produces lower false positives, which are considered as high cost classification errors. A cascading classifiers approach is also performed by Tosun et al., where they report decreased testing efforts on embedded software. Specialized prediction models for embedded systems are also investigated by Khosghoftaar et al. [7], where they built a classification and regression tree for predicting high risk software modules in telecommunications system software. They also investigate genetic programming approaches to optimize multiple objectives for minimizing the false positives while maximizing the number of detected defects. They presented the applicability of their model on real life industrial software. Nagappan et al. [8] also used linear regression analysis with the STREW metric suite. This suite of metrics was extracted from the testing process and is used to estimate the post-release defects. They validate their approach on industrial, open source and student projects and find strong correlations between the proposed metric suite and post-release defects. On open source software, Denaro and Pezze [9] analyzed Apache using logistic regression with static code features and their 80% prediction performance pointed 50% of the modules to be inspected. Nevertheless, In January 2007, Menzies et al. published a study [10] that defined a repeatable experiment in learning defect predictors. The intent of that work was to offer a benchmark in defect prediction that other researchers could repeat/ improve/ refute. Surprisingly, very simple bayes classifiers (with a simple logarithm pre-processor for the numeric) outperformed the other studied methods. They have later tried to find better data mining algorithms for defect prediction. The experiments that have found no additional statistically significant improvement from the application of the further data mining methods include: logistic regression, average one-dependence estimators, under- or over-sampling, random forests, RIPPER, J48, OneR, Bagging and Boosting. Lessmann et al. also investigated this issue and in a very recent paper in IEEE TSE, he reported no statistical difference between the results of 19 learners, including naive bayes, on the same datasets.

In brief, those works present promising results. However, until now, the ML-based works show two main disadvantages: most prediction models are not easily interpreted by the programmers and testers; and most approaches require a pre-process step in order to obtain a balanced dataset.

As the importance of data sets, we introduce the data sets used usually in the fields. The data sets contain three folds: original, open source, and public domain. First, as for original data sets are usually used in empirical studies in industries. Especially, Ref. [11] used principal component analysis on the code metrics and built regression models to predict the likelihood of post-release defect fro five Microsoft software systems which are Internet Explorer 6, IIS W3 Server core, Process Messaging Component, DirectX and NetMeeting. Next, as for the open source software data, studies such as Ref.[12] collected and used for the evaluation of their software defect prediction approaches. Finally, the public domain data set, two of the most famous public domain data set is the NASA and Promise's Metrics Data Program (MDP) [13-14]. For example, studies such as [10, 15] used the NASA's MDP. By using such public domain data sets, a new approach can be easily comparable with other approaches.

B. Non-negative matrix factorization

Nonnegative Matrix Factorization (NMF) [16] is a recently developed technique for nonlinearly finding purely additive, parts-based, linear, and low-dimension representations of nonnegative multivariate data to consequently reveal the latent structure, feature or pattern in the data. Given a non-negative data matrix V, NMF finds an approximate factorization V into non-negative factors W and H. The non-negativity constraints make the representation purely additive (allowing no subtractions), in contrast to many other linear representations such as principal component analysis [17] (PCA) and independent component analysis (ICA).

Also many extended NMF algorithms have been proposed. Local Non-negative Matrix Factorization (LNMF) has been developed by Li et al. in order to increase the basis images sparseness [18]. Both NMF and LNMF consider the database as a whole and treat each image in the same way. There is no class information integrated into the cost function. An extension of LNMF algorithm called Discriminant Non-negative Matrix Factorization (DNMF) which takes into account class information has been proposed in Ref. [19].

In this paper, we extend the application of NMF to software defect prediction by making use of NMF algorithms advantage. NMF coupled with a classifier is applied for software defect data recognition.

II. NON-NEGATIVE MATRIX FACTORIZATION ALGORITHMS

The Non-negative Matrix Factorization problem can be stated as follows [16]:

Given a non-negative matrix V □$R_{m\times n}$, non-negative matrices W □$R_{m\times r}$ and H □$R_{r\times n}$, respectively, we aim at such factorization that V ≈ WH.

$$\begin{pmatrix} V_{11} & \cdots & V_{1n} \\ \vdots & \ddots & \vdots \\ V_{m1} & \cdots & V_{mn} \end{pmatrix} \approx \begin{pmatrix} W_{11} & \cdots & W_{1r} \\ \vdots & \ddots & \vdots \\ W_{m1} & \cdots & W_{mr} \end{pmatrix} \begin{pmatrix} H_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ H_{r1} & \cdots & a_{rn} \end{pmatrix}$$

The value of r is selected according to the rule $r<nm/(n+m)$ in order to obtain dimensionality reduction. Each column of W is a basis vector while each column of H is a reduced representation of the corresponding column of V. In other words, W can be seen as a basis that is optimized for linear approximation of the data in V.

During decomposition, the cost function is either $C_1(V,WH) = \|V - WH\|_F^2$ (where $\|\bullet\|_F$ is the Frobenius norm) or the generalized Kullback-Leibler (K-L) divergence $C_2(V,WH) = \sum_{i,j}(V_{ij}\log V_{ij}/(WH)_{ij} - V_{ij} + (WH)_{ij})$ When cost function C_1 is chosen, the formulae for updating of H and W are:

$$W_{ia} \leftarrow W_{ia}\frac{(VH^T)_{ia}}{(WHH^T)_{ia}} \quad (1)$$

$$H_{bj} \leftarrow H_{bj}\frac{(W^TV)_{bj}}{(W^TWH)_{bj}} \quad (2)$$

Where if cost function C_2 is used, the updating formulae for H and W are:

$$W_{ia} \leftarrow W_{ia}\sum_j \frac{V_{i\mu}}{(WH)_{ij}}H_{bj} \quad (3)$$

$$W_{ia} \leftarrow \frac{W_{ia}}{\sum_j W_{ja}} \quad (4)$$

$$H_{bj} \leftarrow H_{bj}\sum_i W_{ia}\frac{V_{ij}}{(WH)_{ij}} \quad (5)$$

The matrices W and H are initialized with positive random values. NMF provides the above simple learning rule guaranteeing monotonic convergence to a maximum without the need for setting any adjustable parameters.

III. CLASSIFICATION EXPERIMENTS BASED ON NMF

A. Description of the Datasets

As in any machine learning problem, software defect prediction models require a set of features (i.e. independent variables) to characterize the problem and to give estimation on the defect proneness of the system (i.e. dependent variable). In software quality, these attributes are referred to as software metrics. Metrics are the attributes that represent software; they are the raw data for software domain. An effective management of any software development process requires monitoring and analysis of software metrics.

Considering the software defect prediction problem, defect predictors have been successfully learned from product and process metrics. While product metrics are derived from the software product itself, process metrics are derived from the processes that yield the product. Although we only use product metrics in this dissertation, we will provide brief information about process metrics for the sake of completeness.

The software metrics and dataset used in this study are five mission critical NASA software projects [13], which are all high assurance and complex real-time system. NASA makes extensive use of contractors from many other industries including government and commercial organizations. It is practical to leverage the useful information in order to predict the quality of an ongoing similar project.

TABLE I. CHARACTERISTIC OF DATASETS

Data	Lang.	#Mod.	Feature	Description
KC3	JAVA	458	40	processing and delivery of satellite metadata
CM1	C	498	22	NASA spacecraft instrument
MC2	C++	161	40	Video guidance system
PC3	C	1563	38	Flight software for earth orbiting satellite
PC4	C	1458	38	Flight software for earth orbiting satellite

Table I summarizes the characteristic of 5 datasets used in this study. And Table II presents the part of metrics used in the 5 datasets considering the length of paper.

TABLE II. METRICS IN THE DATASETS

Metrics	Type
V(g)	McCabe
EV(g)	McCabe
IV(g)	McCabe
LOC	McCabe
UniqOp	Basic Halstead
UniqOpnd	Basic Halstead
TotalOp	Basic Halstead
TotalOpnd	Basic Halstead
UniqOp	Basic Halstead
N	Derived Halstead
V	Derived Halstead
L	Derived Halstead
D	Derived Halstead
I	Derived Halstead
E	Derived Halstead
B	Derived Halstead
T	Derived Halstead
LOCcode	Line Count
LOCComment	Line Count
LOCBlank	Line Count
LOCCodeAndComment	Line Count
LOCcode	Line Count
LOCComment	Line Count
LOCBlank	Line Count
……	……

B. Prediction Performance Measures

Evaluation measures [20] play a crucial role in both assessing the classification performance and guiding the classifier modeling.

After a classification process, data samples can be categorized into four groups as denoted in the confusion matrix presented in Table III.

TABLE III. CONFUSION MATRIX

		Predicted	
		Defective	No Defective
Actually	Defective	True Positive (TP)	False Negative (FN)
	No Defective	False Positive (FP)	True Negative (TN)

And several measures can be derived from the confusion matrix:

True Positive Rate: $TPR = TP / (TP + FN)$

True Negative Rate: $TNR = TN / (TN + FP)$

False Positive Rate: $FPR = FP / (TN + FP)$

False Negative Rate: $FNR = FN / (TP + FN)$

Positive Predictive Value: $PPV = TP / (TP + FP)$

Negative Predictive Value: $NPV = TN / (TN + FN)$

Clearly neither of these measures is adequate by themselves. So some different evaluation criteria are devised and they are presented in Table IV.

TABLE IV. SEVERAL MEASURES

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$Recall = \frac{TP}{TP + FN} \times 100\%$$

$$Precision = \frac{TP}{TP + FP} \times 100\%$$

$$F - Measure = \frac{2 \times Recall \times Precision}{Recall + Precison} \times 100\%$$

$$pd = Recall = \frac{TP}{TP + FN} \times 100\%$$

$$pf = \frac{FP}{FP + TN} \times 100\%$$

Traditionally, accuracy is the most commonly used measure for these purposes. For classification with the class imbalance problem, accuracy is no longer a proper measure since the rare class has very little impact on accuracy as compared to the prevalent class [18]. F-Measure represents a harmonic mean between recall and precision, a high F-Measure value ensures that both recall and precision are reasonable high. According to the results of above, we choose the F-Measure with confusion matrix as our performance measure on the test data.

C. Unsupervised NMF classification for software defect data

Unsupervised NMF classification is a technique in which the algorithm uses only the predictor attribute values. There are no target attribute values and the learning task is to gain some understanding of relevant structure patterns in the data. Each row in a data set represents a point in n-dimensional space and NMF classification algorithms investigate the relationship between these various points in n-dimensional space.

In NMF classification, using data from the training set, the data matrix V is created (each column v_j contains a feature vector computed from software defect data sets). The training procedure is performed by applying an NMF algorithm to the data matrix yielding the basis matrix W and the encoding matrix H.

In the test phase, for each test data recording, represented by a feature vector v_{test}, a new test encoding vector is obtained by:

$$H_{test} = W_+ * V_{test} \tag{6}$$

where W_+ is defined as the Moore-Penrose generalized inverse matrix of W. Having formed during training N classes of encoding vectors hl, 1 = 1, 2, . . . , N (by applying an NMF algorithm on V, yields matrices W and H as in (1), a nearest neighbor classifier is employed to classify the new test sample by using the cosine similarity measure (CSM). The class label l' of the test data is:

$$l' = \arg \max_{l=1,2,\ldots,N} \left\{ \frac{h_{test}^T * h_l}{\| h_{test} \| * \| h_l \|} \right\} \tag{7}$$

thus maximizing the cosine of the angle between h_{test} and h_l.

Fig. 1 presents the flow of classification method based on NMF algorithm.

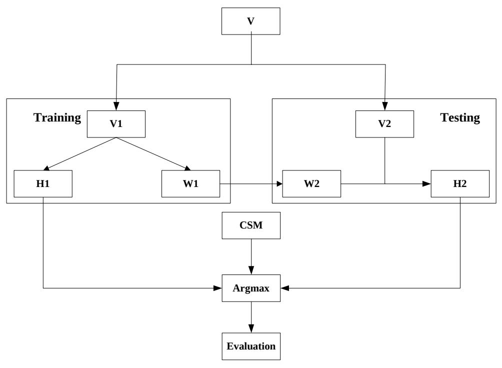

Figure 1. Classification method based on NMF

Step1: Split software defect data set V into V_1 and V_2, respectively, as training sets and testing sets.

Step2: Initialization of Matrices W_1 and H1 during training sets. Set parameter r=min (nm / (n+m)) and choose C1 cost function.

Step3: Perform NMF for training sets decomposition: $V_1=W_1*H_1$.

Step4: Take $W_2=W_1$;

Step5: Calculate W_1^+ as the Moore-Penrose generalized inverse matrix of W1, $H_2=W_1^+ * V_2$.

Step6: Classifier Design: a nearest neighbor classifier is employed to classify the new test sample by using the cosine similarity measure (CSM).

Step7: Evaluation of Classifier.

IV. Comparisons and Analysis of Experimental Results

In this section, we investigate the results of employing the Non-Negative Matrix Factorization algorithm (NMF) for feature extraction and classification. Our experimental environment was Pentium (R) 3.2G CPU, 1G DDR memory, Windows XP operating system and so on. Classifier based NMF algorithm was development and implemented using MatLab 7.0a.

In this experiment, we split the data set into training data sets and testing data sets, respectively, 80% and 20%, firstly. In order to avoid bias, we run the experiment 100 times and calculated its average.

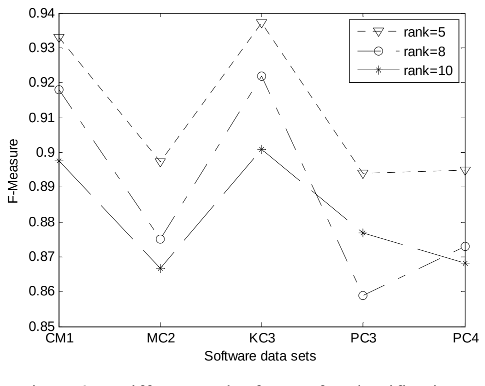

Figure 2. Different rank of NMF for classification

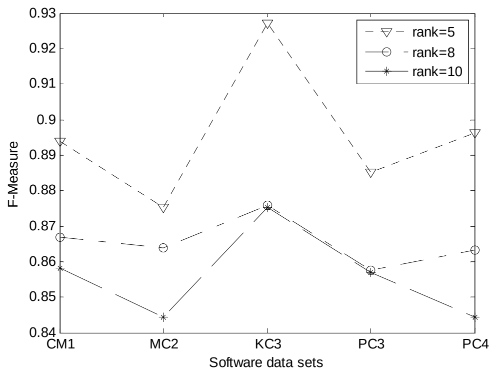

Figure 3. Different rank of NMF for classification

Fig. 2 and Fig. 3 present the results of iterative 200 times and 500 times. Meantime, we recorded the time that was taken by them. From Fig. 2, it is observed that when the r sets smaller, its performance of classification is better during four software defect data sets. The result of Fig. 3 is in the same with the results of Fig. 2.

In order to investigate the efficiencies of NMF classification, we recorded the cost time of it. From Table V and Table VI, we can find that the algorithms of NMF classification are effective as it takes time less than 0.3 second almost for all software defect data sets.

TABLE V. COST TIME ITERATIVE 200 TIMES(S)

Rank	CM1	MC2	KC3	PC3	PC4
5	0.2599	0.2498	0.2517	0.2506	0.2539
8	0.2545	0.2478	0.2547	0.2494	0.2559
10	0.2528	0.2548	0.2623	0.2542	0.2608

TABLE VI. COST TIME ITERATIVE 500 TIMES(S)

Rank	CM1	MC2	KC3	PC3	PC4
5	0.2534	0.2498	0.2522	0.2526	0.2571
8	0.2533	0.2478	0.2553	0.2494	0.2570
10	0.2530	0.2548	0.2509	0.2539	0.2616

And in order to compare its applicability in software defect prediction, we also chose three classifiers trained on five software datasets and compared with NMF classification. Ten-fold cross validation method is used to validate their performance. The classifiers are Naïve Bayes, RIPPER and C4.5.

Naïve Bayes (NB) classifiers use statistical combinations of features to predict for class value. Such classifiers are called ‘naive’ since they assume all the features are statistically independent. Nevertheless, a repeated empirical result is that, on average, seemingly Naïve Bayes classifiers perform as well as other seemingly more sophisticated schemes.

Rule learners like RIPPER (RR) generate lists of rules. When classifying a new code module, we take feature extracted from that module and iterate over the rule list. The output classification is the first rule in the list whose condition is satisfied. To noisy dataset, RIPPER is more search-efficient.

Decision tree learners like C4.5 build one single-parent tree whose internal nodes test for feature values and whose leaves refer to class ranges. The algorithm is known to be one of the most robust induction learning algorithms available [21].

In Fig. 4, 5nmf, 8nmf and 10nmf means NMF with its rank equals 5, 8 and 10, separately. From this Figure, we can find that NMF outperforms the three classifiers, especially for MC2 dataset. To PC4 dataset, all above classifiers receive similar F-Measure, but we find that C4.5, NB and RIPPER take much more time than classification based NMF algorithm during experiments. From Fig. 4, it is obvious that to different software defect data, the performance of NMF classification is better almost, and difference among software data sets is trivial.

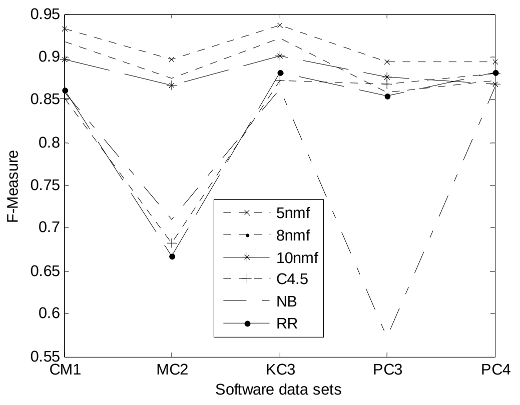

Figure 4. Comparison between NMF and 3 different classifiers

V. Conclusions

Non-negative matrix factorization (NMF) is a recent method for matrix decomposition. Although NMF has been successfully applied to several research fields, it is at the beginning of software defect classification using NMF algorithm. In this paper, we introduce an algorithm named NMF to extract external features and propose a new method of classifying software defect data. Classification using NMF algorithms provides simple learning rule guaranteeing monotonic convergence to a local maximum without the need for setting any adjustable parameters. Also it is easily interpreted by the programmers and testers. The results indicate that the standard NMF algorithms can perform classification with high F-Measure even compared with the state of art classifiers. In the future, extended NMF can be applied to the problem of software defect prediction. And a supervised NMF classification scheme could be developed, considering information of software defects.

Acknowledgment

The authors would like to thank the anonymous reviewers and the editor for their constructive evaluation of this paper. They would also like to thank the various members of author's Laboratory, Xi'an Research Inst. of Hi-Tech, for their helpful comments.

References

[1] S.Balsamo, A.D.Marco, P.Inverardi and M.Simeoni, "Model-Based Performance Prediction in Software DevelopmentL: A Survey,"IEEE Tranctions on Software Engineering, vol 30, 2004.
[2] T.M.Khoshgoftaar, P.Rebous, N. Seliya, "Software Quality analysis by combining multiple projects and learners," Software Quality Journal vol. 17, pp. 25-49, 2009.
[3] T.Menzies, B.Turhan, A.Bener, G.Gay, B.Cukic, and Y.Jiang, "Implications of ceiling effects in defect predictors," in: Proc. of PROMISE 2008 Workshop(ICSE),2008.
[4] Burak, "Improving the Performance of Software of Defect Predictors with Internal and External Information Sources," Bogazici University, 2008.
[5] J.C. Munson, and T.M. Khoshgoftaar, "The Detection of Fault-Prone Programs," IEEE Transactions on Software Engineering, vol. 18, pp. 423-433, 1992.
[6] L. Bullard, T.M. Khoshgoftaar and K. Gao, "An application of a rule-based model in software quality classification," Machine Learning and Applications, 2007. ICMLA 2007, Sixth International Conference on, pp. 204 - 210, 2007.
[7] T.M. Khoshgoftaar and E. Allen, "Predicting Fault-Prone Software Modules in Embedded Systems with Classification Trees"", HASE, 1999,
[8] N.Nagappan, "Toward a software testing and reliability early warning metric suite," Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on, pp. 60 - 62, 2004.
[9] G. Denaro and M., Pezze, "An Empirical Evaluation of Fault-Proneness Models," Proceedings of International Conference on Software Engineering, pp. 241-251,2002.
[10] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Transactions on Software Engineering, vol.33, pp.2-13, 2007.
[11] N. Nagappan, T.Ball, Andreas Zeller, "Mining Metrics to Predict Component Failures," in: ICSE'06 20-28, 2006
[12] G. Denaro and M. Pezze, "An empirical evaluation of fault-proneness models," The 24th International Conference on Software Engineering (ICSE'02), 2002.
[13] http://mdp.ivv.nasa.gov/
[14] G.Boetticher, T.Menzies, and T.Ostrand, "PROMISE Repository of Empirical Software Engineering Data," West Virginia University, Department of Computer Science, http://promisedata.org/repository, 2007.
[15] I. Gondra, "Applying machine learning to software fault-proneness prediction," Journal of Systems and Software, vol.81, pp.186-195, 2008.
[16] D.D.Lee, H.S.Seung, "Learning the parts of objects by non-negative matrix factorization," Nature, vol.1401, pp.788-791, 1999.
[17] Q.S.Chen, X.W.Chen and Y.Wu, "Optimization Algorithm with Kernel PCA to Support Vector Machines for Time Series Prediction," Journal of Computers,vol.5,pp.380-387,2010
[18] S. Z. Li, X. W. Hou and H. J. Zhang, "Learning spatially localized, parts-based representation,"in: Conf. Computer Vision and Pattern Recognition, pp.207–212,2001
[19] I. Buciu and I. Pitas, "A new sparse image representation algorithm applied to facial expression recognition" in: Proc. IEEE Workshop on Machine Learning for Signal Processing, pp.539–548, 2004.
[20] I. H. Witten and E. Frank, Data Mining Practical Machine Learning Tools and Techniques Second Edition. Bei Jing: China Machine Press, p.315-322, 2007.

[21] T.M.Khoshgoftaar, X.Yuan and E.B.Allen, "Balancing Misclassification Rates in Classification-Tree Models of Software Quality," Empirical Software Engineering, vol. 5, pp.313-330, 2000
[22] K.O. Elish and M.O. Elish, "Predicting defect-prone software modules using support vector machines," Journal of Systems and Software, vol.81, pp.649-660, 2008.
[23] B. C. Andréde, P. Aurora and R.V. Silvia, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," The Journal of Systems and Software, vol. 83, pp. 868-882, 2010.

Ruihua Chang is female and is born in Taiyuan, Shanxi, China, in 1982.

She received her B.S. degree from Shanxi normal university in 2005, Linfen, China. In the same year, she jointed Xi'an, Research Inst. of Hi-Tech and received M.S. degree in 2008. Currently, she is pursuing her Ph.D. degree in Xi'an Research Inst. of Hi-Tech. Her main research interests currently focus on software testing, data mining, software metrics and measurement and software defect prediction. She has published several papers in journal and international conference.

Xiaodong Mu is male and born in the city of QiXia in Shandong Province, China, in 1965.

He received Ph.D. degree in computer application from Xi'an Research Inst. of Hi-Tech Since 2001.

He jointed the Department of Computer Science and Engineering at Xi'an Research Inst. of Hi-Tech.

Currently he is as a full professor and a director of the Department of Computer Science and Engineering, Xi'an Research Inst. of Hi-Tech, where he is also the director of the Laboratory. His research interests include performance and reliability modeling and analysis of computer and communication systems, pattern reorganization, machine learning and virtual simulation.

In these fields, Prof. Mu has authored or coauthored over 50 scientific journal papers or conference proceedings. And he has published several book chapters and edition of special issues of journal and international conference.

Li Zhang is female and born xi'an, Shanxi, China, in 1966.

She received B.S. degree from Information Engineering University of the People's Liberation Army in 1992. She received her M.S. degree in computer science at Xi'an Research Inst. of Hi-Tech, Xi'an, China in 1996. And she received her Ph.D. degree in computer theory at University of West north in 2005.

Since 1992, she has been working in the Computer Science Department at Xi'an Research Inst. of Hi-Tech, Xi'an, China where she is currently an assistant professor. Her main research interests include reliability modeling and analysis of computer and communication systems, pattern reorganization, fault diagnose and distributed simulation. She has published numerous peer-reviewed research papers in various conferences and journal. Also she has published several book chapters and edition of special issues of journal and international conference.