Software Defect Identification Using Machine Learning Techniques

Evren Ceylan, F. Onur Kutlubay, Ayşe B. Bener
Boğaziçi University, Computer Engineering Department Istanbul, Turkey
evrenc@garanti.com.tr; kutlubay@cmpe.boun.edu.tr; bener@boun.edu.tr

Abstract

Software engineering is a tedious job that includes people, tight deadlines and limited budgets. Delivering what customer wants involves minimizing the defects in the programs. Hence, it is important to establish quality measures early on in the project life cycle. The main objective of this research is to analyze problems in software code and propose a model that will help catching those problems earlier in the project life cycle.

Our proposed model uses machine learning methods. Principal Component Analysis is used for dimensionality reduction, and Decision Tree, Multi Layer Perceptron and Radial Basis Functions are used for defect prediction. The experiments in this research are carried out with different software metric datasets that are obtained from real-life projects of three big software companies in Turkey. We can say that, the improved method that we proposed brings out satisfactory results in terms of defect prediction.

1. Introduction

The goal of a software engineer is to develop a software program that produces the desired results for its customers on time and within budget. In order to improve software quality, high-risk components within the software project should be caught as early as possible. Early detection of problems enables software project managers to make timely decisions and better planning of tight resources.

Current methods in software engineering present ‘defect prevention’ strategies rather than ‘defect detection’ [1] strategies. Defect prevention methods are conducted through the fault management and propagation processes. Reviews and inspections in these processes are the traditional ways of actively reducing defects originating from the development phase. These defect prevention methods are usually quite expensive and they cause dramatic increases in the project costs [1]. Alternatively, as it is proposed in our research, the implementation of machine learning techniques allows recognition of errors/ bugs during the development cycle. We believe that these techniques are well suited for defect identification and prediction in software projects.

2. Background

Managing software development is a challenging business. Most of the software projects considerably exceed their budget [2]. Cost of finding and fixing a defect exponentially increases as the project progresses from the requirements gathering phase to final field use. If the companies better manage their software development process, they will be able to increase customer satisfaction and profitability. Therefore it is important to introduce principles of software quality measurement as early as possible in the project life cycle.

Various researches had been carried out in order to measure the software quality for the last thirty years [3, 4, 5, 6]. Software measurement helps practitioners for identifying the software properties and quality attributes in a detailed manner. The area of software measurement involves the use of software metrics. Software metrics are the quantitative data, which are used to characterize properties of a source code, and they help to predict software resource requirements and software quality [7, 8]. Sequential measurements of quality attributes of processes can provide an effective foundation for initiating and managing process improvement activities [7]. An effective management of any software development process requires quantification and modeling of software metrics. The most important progress on *software metrics* was achieved by *McCabe* and *Halstead* [7]. McCabe’s cyclomatic complexity and Halstead’s program vocabulary are among the most important metrics that increased the attention and the number of researches in this area.

Testing is the most popular method for fault detection in most of the software projects. However, when the size of projects grow in terms of both lines of code and effort spent, the task of testing gets more difficult and computationally expensive with the use of sophisticated testing and evaluation procedures. If the relationship between the software metrics measured at a certain state and the properties of faults can be formulated together, it would be possible to predict similar faults in other parts of the code.

Metric analysis allows software engineers to assess software risks; and these metric data also provide information for defect prediction. Currently there are numerous researches on defect prediction models [11]. These models can be grouped according to the metrics used in the system. Most of the defect models use the previously mentioned metrics by *McCabe* and *Halstead* such as cyclomatic complexity and size of the software [11]. Apart form these, in order to estimate the defects in the programs; some testing metrics are also produced during the test phase of the project [12]. According to a research [13], some metrics depict common features on software risks. Instead of using all the metrics adopted, a basic one that will represent a cluster can be used. In that research, the most popular approach, "Principal Component Analysis", has to be applied in order to determine the clusters that include similar metrics. Many of the previous studies [14, 15] show that metrics in all the steps of a project such as design, implementation and testing should be utilized and connected with specific dependencies. Concentrating only a specific metric or process level is not enough for a satisfactory prediction model [9]. The main objective of all these prediction models is to estimate the reliability of the system, and analyze the efficiency of design and testing process over number of defects. According to another research [16], while finding and fixing bugs that are discovered in testing, it is necessary to assure reliability. The best way of assuring reliability is developing a robust, high quality product through all of the stages of the software lifecycle. That is, the reliability of the delivered code is related to the quality of all of the processes and products of software development (i.e. the requirements documentation, the code, test plans, and testing). According to that research, software reliability is not as well defined as hardware reliability. In that work, they are striving to identify and apply metrics to software products that promote and assess reliability.

The software metric data in our research, gives us the values for specific variables to measure a module/ function or the entire software. When combined with the weighted error/ defect data, this data set becomes an input for machine learning systems. To design a learning system, the data set in this research is divided into two parts: the training data set and the testing data set. The learning system is trained with the training data set using regression based algorithms and it is validated against the testing dataset.

Apart from the researches on software metrics, many studies are also carried out on machine learning algorithms for the purpose of defect prediction using the related software metrics. The machine learning methods are practical for complicated problem domains whose conditions change according to many different values and regularities. Since software problems can be formulated as learning processes and they are classified according to defect characteristics, regular machine learning algorithms are applicable to prepare a probability distribution and to estimate errors [15, 17].

According to a research [13], an enhanced technique for risk categorization is presented. This technique, is a combination of principal component analysis and artificial neural network methods called *PCA-ANN* and it provides an improved capability to identify high-risk software. The approach considers the combined strengths of pattern recognition, multivariate statistics and neural networks. Principal component analysis is utilized to provide a means of normalizing and orthogonalizing the input data, thus eliminating the ill effects of multicollinearity. A neural network is used for risk determination and classification in the same research. A significant feature of this approach is a procedure, named as termed cross-normalization. This procedure provides a technique with capability to eliminate data sets that include disproportionately large numbers of high-risk software modules. Another approach [18] is to use machine learning algorithms over the program execution and detect the number of faulty runs, which will lead us to find underlying defects. The executions in the system are clustered according to procedural and functional properties. Machine learning approach is also used to generate models of program properties that are known to cause errors.

According to another research [19], a software fault identification method is proposed via Dynamic Analysis and Machine Learning techniques. The goal of this research is to design a technique that assists users in finding errors in code by creating computer models of errors based on faulty code executions. In software projects, developers must manually detect and fix code errors. Testing can increase programmer confidence in the program. However, it is not possible to give guarantees that a program would not contain any errors that may result in faulty executions. It is desirable to obtain a tool to further increase the programmer's confidence in the program's correctness. Although there are an infinite number of possible code errors, these errors can be categorized. Therefore, errors in the same category are likely share same characteristics. Thus, it is hypothesized that, it is possible to learn the characteristics of error-exposing properties in programs and to recognize such properties in other programs. Such a technique would allow programmers to locate and remove errors in code and increase confidence in the program's correctness.

In the earlier versions of our research we used a dataset that was obtained from NASA repository of software metrics of projects IV and V. In this work [9], different machine learning algorithms are evaluated in terms of their capabilities in identifying and locating possible defects in a software project. Firstly, the dataset is normalized and cleaned against correlated and irrelevant values, and then, machine learning techniques are applied for error prediction. The learning process was guided by the error/ defect metrics data residing in the repository; and herein K-nearest neighborhood, k-means and ANN approaches are utilized to implement the learning architecture. Although the dataset was large in this system, the modules and attributes of the dataset were not well-identified. So, the researchers did not have an intensive knowledge about the input and output data values. In addition, the priority levels of defects per module were not stated in the dataset. Therefore, in the current version of our research, we aimed at establishing an improved defect estimation model using additional learning algorithms and well-identified datasets. In the implementation of our research, we used three different datasets such that we have an intense knowledge about the attributes and modules. These datasets belong to the real-life projects of different software companies in Turkey. In this research, we believe that our major contribution is estimating the defected modules earlier in a software development project.

3. Problem Statement

In defect prediction it is possible to conduct two types of research based on code metrics data. First is to predict if the code is defective or not. Second one is to predict the magnitude of the possible defect such as its severity, priority, etc. Our research has focused on the second type of predictions. By doing so, we aim to provide the software quality practitioner with an estimation about "which modules may contain more faults?" This information can be used to locate the scarce testing and validation resources to the modules that are predicted as "most defective".

Given a training data set, a learning system can be set up. This system would come out with a score point that indicates how much a test data and code segment is defective. After predicting this score point, the results can be evaluated with respect to popular performance functions.

In this research, we used Mean Squared Error (MSE) since the performance function for the results of the experiments aims second type of prediction as described above. The MSE is the most common function approximation technique for evaluating the performance of the regression experiments [10].

The unidentified dataset attributes in prior version of this research combined with inconsistent results drove the need for improved learning systems. Our initial efforts resulted in unsatisfactory system performance and inconsistent MSE results, causing underestimation problems. It is hypothesized that the primary source of this underestimation problem is due to the majority of the *minor-priority defects* in the datasets. Since these minor-priority defects slow down the learning performance, there happens to be a bias towards underestimating the defect values.

A solution to this underestimation problem is to abolish the minor-priority defects from the input space. This would help the learning process to make more precise estimations with MSEs.

4. Proposed Solution

We designed and proposed a learning system to predict the magnitude of a possible defect. The graphical representation of this system is shown in Figure 1. Input data represents the attributes (code metrics) of each module/ function in the related project. Since the input data in each dataset has many correlated values, principal component analysis (PCA) is carried out for transforming the correlated variables into uncorrelated ones. The main purpose of PCA is reducing the dimensionality of the datasets by eliminating the correlations among the attributes. After the implementation of PCA, a new dataset is obtained with fewer dimensions. The PCA results of the companies are given with respect to eigenvalue percentages. According to these percentages, it is sufficient to accept 5 input attributes for the dataset that belongs to Company-A. So, Company-A dataset will be represented with only 5 principal components whereas the total number of features in the original data set was 99. According to the Company-B dataset eigen value percentages, it is sufficient to accept 10 input attributes and this dataset will be represented with only 10 principal components whereas the total number of features in the original dataset was 50. At last, Company-C will be represented with 4 principal components according to the related PCA results.

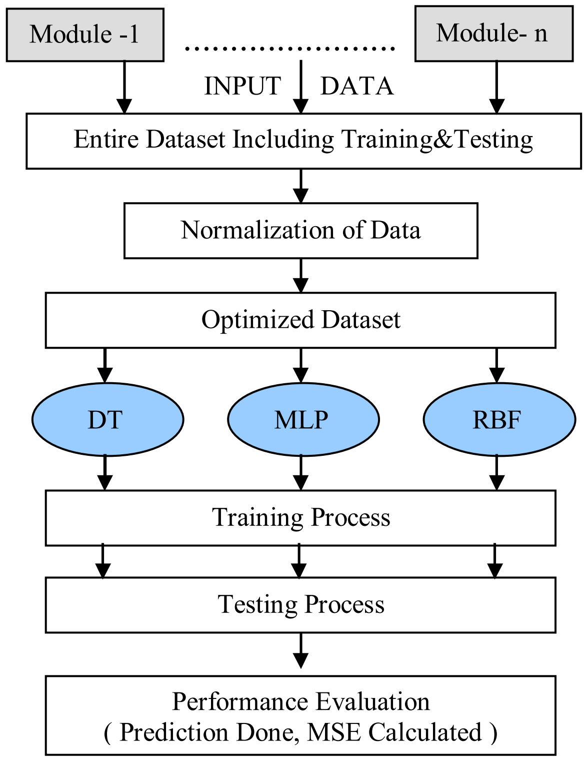

Figure 1. Learning System Architecture

Then the system is trained with this optimized dataset using the regression based algorithms such as decision tree (DT), multi layer perceptron (MLP) and radial basis functions (RBF). The outputs in the system state the number of total defects per module/ function. Then the system is tested by using the test data which is part of the entire dataset. After the testing process, the desired output is obtained. The output represents the total number of defects per each module/ function.

4.1. Evaluation Method of Experiments

When the training process in the system is terminated, the evaluation of the numerical values is required. The aim of the evaluation process is measuring the improvement rate of the designed system. As mentioned in the previous section, in our research, we used *Mean Square Error* (MSE) approximation.

When the target defect density is predicted with respect to the input data, the experimental results are evaluated with this MSE approximation function. In order to obtain the MSE values, the squares of the differences between the target and real value is calculated. Then the sum of these differences is divided by the total number of inputs. In other words, MSE is equal to the mean of the squares of the deviations from the target value. The formulation is given below:

$$\mathrm{MSE} = 1 / m * \sum_i (x_i - T_i)^2 \tag{4.1}$$

X_i and T represents the real output values and the corresponding target values respectively. Total number of inputs in the entire dataset is indicated by *m*.

In the performance evaluation section of our research, we also calculated the variance of the datasets in order to make a comparison with the mean square error values. The variances of the datasets are calculated according to the following formula:

$$VAR = 1 / (m-1) \,.\, \sum_i (x_i - mean_i)^2 \tag{4.2}$$

Again, X_i and m represent the real output values and the total number of inputs in the entire dataset. Also, mean_i represents the corresponding mean values.

In our research, firstly we calculate the x_i values according to the related classifier algorithm; MLP, RBF or decision tree. The x_i values are the prediction results whereas the T values are the desired values. Each x_i value has a corresponding target value in the given dataset. The difference between the predicted and the target value is calculated according to the above formula and this result gives us the *MSE* values which are used in the performance evaluation process of the proposed model.

4.2. General Properties of the Datasets

Many of the code metrics mentioned in the introduction section are used in this research as the inputs for the training systems. The general organization of these datasets represents similar characteristics such as their metrics and target values.

The largest dataset used in this research has 50 input dimensions (number of input columns) and approximately 300 different modules in the project (rows). The other two have smaller datasets with 99 input dimensions, and 118 modules, and with 8 inputs and 70 different modules, respectively. All datasets have one output indicating the number of defects per module.

5. Methodology

In this research we proposed a model based on regression techniques to detect and identify software defects. Designing a learning system for defect prediction requires experimentation of these techniques with the datasets collected from three companies. In order to achieve this goal, we designed two learning systems: the *initial learning system* and the *improved learning system*.

In the *initial learning* system, the entire (original) dataset is used whereas in the *improved learning system* a partial dataset that is obtained from the original one is used.

In a given dataset, there are many code metrics including the priority levels of defects in each module. The attribute called 'Priority Level' represents the criticality of each defect in the corresponding module. For example; assume that there are two different types of defects; one of them is a defect came out because of an error in the algorithm, while the other one is a defect in the GUI statement. Certainly, a *GUI-defect* is less critical than an A*lgorithm-defect* in the eyes of customers. Therefore, the criticality level of the GUI-defect is *Minor*, whereas it is *Major* for algorithm-defect. Depending on the information obtained from real-life projects, the datasets are likely to have much more *Minor-Priority* defects than the *Major* ones. Practically, when we generate a new dataset by abolishing the Minor-Priority defects, we get better experimental results.

In this research, all the initial experiments are done with 118, 300, and 70 different modules for Company-A, Company-B and Company-C respectively. After the minor-priority defect elimination, Company-A has 76 modules/ functions in the input space, whereas Company-B and Company-C have 182 and 55 modules/ functions respectively. The experiments for each machine learning algorithm are repeated 200 times (epochs) and the mean values of the results are taken into account in calculating the improvement rate. To assure the randomness of the data, all the datasets are regenerated from the original dataset by utilizing a shuffling algorithm based on random displacement of the rows in the data set.

5.1. Initial Learning System

The *initial learning system architecture* predicts the defect values of each module using the *entire* dataset. The system is trained with all the input values; and the generated output gives us the corresponding target value.

Since the performances of the algorithms (Decision Tree, MLP, and RBF) are not satisfactory enough in this learning system, we see that there is a bias towards underestimating the defect possibilities in the prediction process. In order to overcome this problem, we proposed a new learning architecture, which is the improved version of this learning system.

5.2. Improved Learning System

In this architecture, we trained the system with a partial dataset including only *major-priority* defects/ modules instead of using the entire dataset.

The main purpose of this improved learning architecture is to analyze the prediction performance of the system when the bias mentioned in the previous section is removed. In this experiment, we expect to improve the performance by carrying out more precise estimations.

6. Test Results

The experiments in this research are carried out by Matlab 6.5 tool. Some of the functions in standard Matlab library are used in our methods during the implementation process.

While designing a learning system for *neural network* experiments, multi layer perceptron (MLP) and radial basis function (RBF) methods are utilized according to their widely known algorithms. In neural network experiments, to obtain precise estimation values, we trained the system with different number of parameters. The learning rate in the system is set to 0.01 whereas the experiments are performed with 200 training cycles. Also, different number of hidden units (4, 8, 16, 64, and 128) is used in network generation process to see their corresponding performances and to select the best predictor.

6.1. Experimental Results Using the Entire Datasets

The first type of experiments is done with *entire* datasets including all the defect values, having *minor-priority*, *normal-priority* and *major-priority* levels. As mentioned earlier, these priorities represent the criticality level of each defect in the corresponding module and these levels are not equally weighted in the dataset. Therefore, the system is likely to have a bias towards underestimating the defect values because of these minor-priority defects. The results of these experiments are tabulated in Table 1, Table 2 and Table 3 below.

6.2. Experimental Results Using Datasets that do not Contain Minor-Priority Defects

The implementations in this type of experiments are done with input datasets that do not contain the minor-priority defect modules and their corresponding output. The results for *multi layer perceptron*, *radial basis functions* and *decision tree* methods represent more satisfactory behavior than in the first type of experiments. These experiments are done to see and verify the success of the second model proposed in Section 5.2. The following experiments show that the proposed system is able to make more precise estimations using the dataset containing only major-priority defects. By doing this, both the performance of the system and efficiency of the learning methods are advanced. The experimental results are given below.

Table 1. Results for Company A

A	Before PCA		Initial System		Improved System	
	MSE	Variance	MSE	Variance	MSE	Variance
MLP	240.55	254.21	154.23	286.16	51.97	216.29
RBF	235.42	239.82	150.09	294.48	50.74	241.83
Dec. Tree	237.17	287.41	151.42	298.16	52.06	214.64

Table 2. Results for Company B

B	Before PCA		Initial System		Improved System	
	MSE	Variance	MSE	Variance	MSE	Variance
MLP	378.76	379.65	286.18	410.81	122.44	579.86
RBF	372.49	378.03	281.43	404.28	119.26	581.24
Dec. Tree	376.10	380.18	285.21	402.26	127.12	559.41

Table 3 Results for Company C

C	Before PCA		Initial System		Improved System	
	MSE	Variance	MSE	Variance	MSE	Variance
MLP	94.43	96.23	47.18	90.81	32.48	128.36
RBF	89.21	90.84	44.68	89.97	30.55	134.66
Dec. Tree	98.28	102.57	49.16	92.16	33.26	124.46

According to the above tables, the *improved system*'s performance improved approximately 66.67% for Company A-Dataset whereas it improved approximately 57.59% for Company B-Dataset. The improved system performance for Company C-Dataset is approximately 32.61%. Since Company C-Dataset does not have many minor-priority defects, the system performance does not improve drastically when the implementations are carried out using the datasets with only Major-Priority defects.

7. Conclusion

In this research, an enhanced defect prediction system is proposed based on the machine learning and neural network methods. These techniques are used to identify potentially defective software and allow corrective action to be taken before software is released to the production environment.

The results of the 'initial system structure' show that the methods have many faulty defect predictions when the entire dataset is used. Conversely, the improved system performance showed better results approximately 32.61 per cent for Company-A and 60 per cent for the other two. As a result, our proposed model shows that the performance of the system is much more satisfactory when the system is trained using the dataset including only the *Major-Priority* defects.

When the results are considered in terms of algorithm performances, it is seen that all of the learning algorithms used in our research have similar prediction performances having similar mean square error values. However, the radial basis function (RBF) method has slightly smaller prediction errors than the other two methods. Since Radial Basis Functions have faster convergence, smaller extrapolation errors, and higher reliability, their mean square error values are smaller. Therefore, we can say that in our research the radial basis functions are more advantageous than the others.

As a future work, the learning rate, network type, and training parameters may be improved upon. Learning rates, other than those stated in this research, may further improve the system performance. Apart from this, if the project expenses (cost price of each module, budget of the project, cost of the resources and so on) are known in a software project, it may be possible to establish a *cost estimation model*.

Another area of future research could be combining the learning approaches that are used in our research. In order to achieve this, we may use the *'combining multiple learners'* method considering the boasting strategies such as training the next learner on the mistakes of the previous learner.

It is also possible to develop a *'software tool'* for companies. Such a tool would enable a software company to better manage its scarce resources in software development process.

Acknowledgements

We extend our gratitude to Logo, Innova and Garanti Teknoloji for making this research possible. We also thank Dr. Zehra Çataltepe for her useful comments and suggestions at different stages of this research.

References

[1] Futrell, R. T., D. F. Shafer and L. I. Shafer, *Quality Software Managemen*t, Prentice Hall PTR, 2002.

[2] Mainstay Partners, Making Technology Investments Count, *2004 Reports*, 2004.

[3] Zuse H., *A Framework of Software Measurement*, Walter de Grutger Publish, 1998.

[4] University of Texas, Software Quality Institute Report, May 2002.

[5] Rosenberg, L. and S. B., Sheppard, "Metrics in Software Process Assessment, Quality Assurance and Risk Assessment", *2nd International Symposium on Software Metrics*, London, October, 1994.

[6] Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, 1981.

[7] Florac, A., E. Park and D. Carleton, "Practical Software Measurement: Measuring for Process Management and Improvement", *Software Engineering Institute, Carnegie Mellon University*, April, 1997.

[8] Fenton, N., P. Krause and M. Neil, "Software Measurement: Uncertainty and Casual Modeling", *IEEE Software*, July/August 2002.

[9] Kutlubay O. and A. Bener, "A Machine Learning Based Model For Software Defect Prediction" *working paper, Boğaziçi University, Computer Engineering Department*, 2005.

[10] Mitchell, T. M., *Machine Learning*, McGraw-Hill and MIT Press, 1997.

[11] Henry, S. and Kafura, D., *The Evaluation of Software System's Structure Using Quantitative Software Metrics, Software—Practice and Experience*, vol. 14, no. 6, June 1984.

[12] Cusumano, M.A., *Japan's Software Factories*, Oxford University Press, 1991.

[13] Neumann, D.E., "An Enhanced Neural Network Technique for Software Risk Analysis", *IEEE Transactions on Software Engineering*, September, 2002.

[14] Boetticher, G.D., Srinivas, K. and Eichmann, D., "A Neural Net-Based Approach to Software Metrics", *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, June 1993.

[15] Fenton, N. and M. Neil, "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, Sept. 1999.

[16] Rosenberg L., T. Hammer and J. Shaw, "Software Metrics and Reliability", *Unisys/NASA GSFC.*

[17] Zhang, D., "Applying Machine Learning Algorithms in Software Development", *The Proceedings of 2000 Monterey Workshop on Modeling Software System Structures*, Italy, June 2000.

[18] Dickinson, W., D. Leon and A. Podgurski, "Finding Failures by Cluster Analysis of Execution Profiles", *ICSE*, May, 2001.

[19] Brun, Y. and D. E. Michael, "Finding Latent Code Errors via Machine Learning over Program Executions", *Proceedings of the 26th International Conference on Software Engineering*, (Edinburgh, Scotland), May, 2004.