Comparison of software defect prediction models based on machine learning

W H Zhang, R Y He, L J Wu, Y Jian and X Y Han
China Institute of Marine Technology and Economy, Beijing, China
E-mail:15600563989@163.com

Abstract. Software defect prediction has been widely used in software system development, among which the method based on machine learning has proved to be more effective. Firstly, the basic framework of the prediction model and the metric elements used in the prediction process are introduced in this paper. Secondly, the three main machine learning-based software defect prediction models (LR, SVM, and BPNN) are analyzed, and finally the prediction effects of the three models are compared and analyzed by using the experimental results of MDP.

1. Introduction
With the rapid development of computer technology and the increasing demand for intelligent technology products in the society, various types of software companies have continuously upgraded and evolved software products in order to improve their competitiveness in the industry, which makes the scale and complexity of the software system have grown rapidly, and potential risks have also been introduced. While the software brings convenience to people’s lives, various failures that occur during operation have also caused many serious social problems. For example, on July 28, 1962, a bug in the Mariner 1 aeronautical software caused the rocket to deviate from its intended orbit when launched, and the mission controller destroyed the entire rocket over the Atlantic Ocean; between 1985 and 1987, a failure of a radiotherapy equipment caused the therac-25 treatment equipment to emit fatal radiation during the treatment process, which directly resulted in the death of five patients, and the rest of the patients were seriously injured; the AT & T network paralysis in 1990, the Intel Pentium floating-point index division in 1993, and the first failure of the Ariane 5 launch vehicle. It can be seen that the potential defects in the software will not only affect the quality of the software, but also cause the collapse of the software system, and bring serious economic losses to individuals or enterprises.

The serious consequences caused by software defects have raised more and more software researchers and developers to pay close attention. In order to minimize the impact of software defects, methods such as software testing are applied to the development of software systems. Although this method has been proven to effectively improve the security and reliability of the system, according to Boehm statistics, software testing accounts for 30% to 50% of the development cost of the software system [1]. In the software development cycle, it needs to invest a lot of time and cost, so exhaustive testing of each software module is unrealistic. At the same time, software testing technology will also be limited by test resources.
In the 1970s, starting from the repositories, some researchers used statistics and machine learning methods to predict the number and types of defects in the software system based on the software metrics and the discovered defects, and obtained abundant research results. This method is the software defect prediction, which can be roughly divided into static defect prediction methods and dynamic defect prediction methods [2], among them, the static prediction methods are mainly based on the defect related metrics (relevant data information obtained by quantifying software code as static characteristics). Through statistical analysis of these data, a prediction model is constructed to predict the defect tendency, defect density or number of defects of the program module; while the dynamic defect prediction method is to predict the distribution of system defects over time by analyzing the time of defects or failures, and to discover the distribution of software defects with their life cycle or the time relationship of some stages. At present, researchers pay more attention to static prediction. Therefore, static prediction has also become the dominant method in software defect prediction [3-5].

Static software defect prediction methods mainly include machine learning, statistical methods, machine learning + statistical methods, and statistical methods + expert opinions [6]. Machine learning is one of the core research directions in the field of artificial intelligence, and has been widely used in the research of software defect prediction. Software defect prediction methods based on machine learning have also been proven to have efficient prediction performance in several practices. This paper mainly studies and discusses the above method.

2. Software defect prediction
2.1. Basic framework of prediction model
The software defect prediction model based on machine learning needs to use the marked defect historical data and abstract the metrics from these data, and select the optimal features according to a certain learning rule, so as to train the classifier to predict the unknown samples. The general process of software defect prediction model based on machine learning method is shown in Figure 1.

![Figure 1. Process of software defect prediction based on machine learning.](image-url)
The defect prediction process shown in the figure 1 is mainly divided into four parts: software defect data acquisition, metrics extraction, prediction model construction and performance evaluation [3, 7, 8].

(1) Use the version control system and defect tracking system to collect software defect data from the software warehouse that contains all the data of the software project development cycle.

(2) The data information obtained in (1) is represented by different software metrics, that is, the data is preprocessed, and then the program modules in the data are marked according to the actual requirements (the attribute measurement of the corresponding module is given defect category information); in order to avoid redundancy and overfitting caused by a large amount of data information, an appropriate number of feature data sets can be selected after labeling to reduce the computation and improve the performance of the classification model.

(3) Build a software defect prediction model in the adjusted data set by using appropriate learning algorithms and statistical techniques; for a new test module, it is necessary to extract its software metrics, then use the prediction model to predict, and finally output the final result of the prediction.

(4) After the prediction results are obtained, the prediction model is evaluated by using the evaluation performance related to confusion matrix. The evaluation results of the model are given through different performance evaluation indexes (accuracy, precision, recall, F-measure, etc.).

2.2. Software metrics

The updating of software products has stimulated the development of defect prediction technology, the most typical of which is that a large number of metrics are introduced into the software life cycle. IEEE gave a definition of software metrics in 1997: "Metric is a function, its input is software data, and its output is a single value, which can be used to explain the extent to which a given attribute of software has an impact on software quality. Software metrics are quantitative measurements of attributes that affect software quality" [9].

At present, Halstead metrics and McCabe metrics are the most basic of all metrics. In the 1990s, with the rapid development of object-oriented development methods, Chidamber and Kemerer [10] proposed CK metrics suitable for object-oriented programs. The following sections focus on these three metrics.

(1) Halstead metrics

Halstead studies software complexity from a statistical and psychological perspective, and regards programs as a sequence of symbols composed of executable lines of code. The complexity of the program is measured by calculating the number of operators and operations in the program. The main metrics involved include the length, capacity, difficulty, and workload of the program, among which the basic metrics are as follows:

- N1: Overall operator in the program
- N2: Overall operands in the program
- n1: Different operators in the program
- n2: Different operands in the program

It is known that H is the predicted length of the program, and N is the actual length of the program. The calculation formula of H and N given by Halstead is:

\[ H = n1 \log_2 n1 + n2 \log_2 n2 \]  \hspace{1cm} (1)
\[ N = N1 + N2 \]  \hspace{1cm} (2)

Halstead also gives the calculation formula of program capacity, program level and number of software defects, etc. Let V represent the program capacity, L represents the program level, and B represents the number of software defects. Where, L indicates the ratio of the program capacity in the most compact form of a program to the actual program capacity, reflecting the efficiency of the program. The calculation formula is as follows:
McCabe metrics
McCabe proposed the theory of software complexity measurement in 1976. He believes that the complexity of a program can be quantitatively measured by the complexity of the program control flow [11]. The McCabe metrics mainly include multiple metrics such as Cyclomatic complexity, Essential complexity, and Module Design complexity.

Cyclomatic complexity is used to measure the complexity of a module's decision structure. The quantity is expressed as the number of linearly independent paths in the control flow graph G, that is, the minimum number of paths required for reasonable error prevention.

Errors in a program may have a lot to do with high cyclomatic complexity. Its calculation method is very simple, the calculation formula is: \( V(G) = e - n + 2 \). Where, \( e \) represents the total number of edges in the control flow graph, and \( n \) represents the number of nodes in the control flow graph. More simply, Cyclomatic complexity can also be obtained from the number of decision nodes \( d \) in the control flow graph, that is, \( V(G) = d + 1 \).

Basic complexity is used to measure the unstructured degree of the program. By simplifying the structured part of the module control flow graph into nodes, the cyclomatic complexity of the simplified control flow graph after calculation is the basic complexity. The calculation formula is: \( EV(G) = V(G) - m \), where \( V(G) \) is the cyclomatic complexity and \( m \) is the number of structured subgraphs in the flowchart.

Module Design complexity is used to measure the calling relationship between modules. The higher the complexity, the higher the coupling between modules, and the harder it is to isolate, maintain, and reuse. It is the cyclomatic complexity obtained by removing the decision and loop structure that does not include the calling submodule from the module control flow graph. Module Design complexity is usually much smaller than the cyclomatic complexity.

(3) CK metrics
CK metrics mainly describe the degree to which the program conforms to object-oriented design, and includes 6 main metrics [12]:
- \( wmc \): Method weight in a class, which represents the sum of the complexity of all methods in a class. Here, CK metrics do not measure the complexity by cyclomatic complexity, but assigns the complexity of each method to 1, So \( wmc \) is equivalent to the number of methods in each class.
- \( dit \): Inheritance tree depth, which indicates how many classes a class inherits.
- \( noc \): The number of direct subclasses of a class.
- \( cbo \): Coupling between objects is a count of classes that are coupled to a class. When a method of a class calls a method of another class, or accesses a variable of this class, we think that the two classes are coupled. The \( cbo \) of class A is the size of the collection that refers to A or a class referenced by A
- \( rfc \): The size of the response set of the class, that is, the size of all the methods in this class and the collection of all the methods called by the methods in this class.
- \( lcom \): Lack of cohesion in methods. For each method pair in the class, if the pair of methods do not jointly refer to any instance variables, then \( lcom \) increases by one, otherwise decreases by one

Software defect prediction model based on machine learning
The realization of software defect prediction is mainly attributed to the use of machine learning algorithm and statistical methods. The current mainstream defect prediction models include Logistic
Regression model, Naive Bayes model, and Support Vector Machine model, Neural Network model, Decision Tree model, random forest model, etc.

3.1. Logistic Regression model (LR)

Logistic regression algorithm is actually a classification algorithm. Logistic regression is a non-linear regression method obtained by applying a logistic function on the basis of linear regression.

In machine learning, LR is mainly applied to binary classification, that is, to divide all data into two categories 0 and 1\(^{[13]}\), a given data set \(D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}\), where \(x_i \in \mathbb{R}^d, y_i \in \mathbb{R}(i = 1, 2, \ldots, n)\). Given the sample \(x_i\), the Logit linear regression model is used to predict the value of \(y_i\). If it is assumed that the probability of \(y_i\) being 1 is required, the specific calculation formula is as follows:

\[
P(y = 1|x; \theta) = \frac{1}{1 + e^{-\theta^T x}} \tag{6}
\]

Given that the hypothesis function of linear regression is:

\[
h_\theta(x) = \theta^T x \tag{7}
\]

Since the predicted value of the linear regression model is continuous, the range is \([0, 1]\). We can use the Sigmoid function to convert, let \(z = \theta^T x\):

\[
g(z) = \frac{1}{1 + e^{-z}} \tag{8}
\]

Generally speaking, when \(g(z) \geq 0.5\), \(z\) is classified as true, otherwise when \(g(z) < 0.5\), \(z\) is classified as false. In the LR model, we know that when it is assumed that the function \(h_\theta(x) \geq 0.5\), \(P(y = 1)\) is predicted to be a positive class; otherwise, it is predicted to be a negative class.

Assuming that there are \(m\) data in the training set, the loss function constructed by the maximum likelihood method in statistics is (the objective function of LR):

\[
J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i)) \tag{9}
\]

Next, we can use the gradient descent method to find the maximum value of the parameter \(\theta\) and classify the unknown samples according to the known samples.

3.2. Support Vector Machine model (SVM)

SVM is a classifier algorithm, and its core idea is the principle of structural risk minimization \(^{[14]}\). In the case of a small number of training samples, a better generalized learning performance can be obtained by establishing a classifier that does not depend on the dimension of the sample space \(^{[15]}\).

Given a training dataset on a feature space \(D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}\), where \(x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}, (i = 1, 2, \ldots, N)\), the basic idea of classification learning is to find a dividing hyperplane in the sample space based on the training set \(D\) to separate samples of different categories. The equation for dividing the hyperplane can be expressed by the following formula:

\[
\omega \cdot x + b = 0 \tag{10}
\]

The purpose of SVM is to select the hyperplane with the largest geometric interval, that is, to obtain the support vector by finding the optimal solution of the following formula:
\[
\min_{\omega, b} \quad \frac{1}{2} \| \omega \|^2 + C \sum_{i=1}^{N} \xi_i
\]

\[\text{s.t.} \quad y_i \cdot (\omega \cdot x_i + b) \geq 1 - \xi_i, \xi_i > 0, i = 1, 2, \ldots, N\]  

Where, the slack variable is \( \xi_i = \max \left(0, 1 - y_i \cdot (\omega \cdot x_i + b)\right) \). By constructing a Lagrangian function:

\[
L(w, b, \xi, \alpha, \mu) = \frac{1}{2} \| \omega \|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left( y_i (w \cdot x_i + b) - 1 + \xi_i \right) - \sum_{i=1}^{N} \mu_i \xi_i
\]

Substitute the derivative of Equation 12 into Equation 11, and transform it to convert Equation 12 to the following form:

\[
\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i, x_j) - \sum_{i=1}^{N} \alpha_i
\]

\[\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, N\]  

Find out the optimal solution \( \alpha^* = \left( \alpha_1^*, \alpha_2^*, \ldots, \alpha_N^* \right)^T \), \( w^* \) and \( b^* \), substitute into the hyperplane equation, and then get the classification decision function:

\[
f(x) = \text{sign} \left( w^* \cdot x^* + b^* \right)
\]

For nonlinear partition problems, we can introduce Gaussian radial kernel functions:

\[
K(x, z) = \exp \left( -\frac{x \cdot z^2}{2\sigma^2} \right)
\]

In this case, the classification decision function can be transformed into the following form:

\[
f(x) = \text{sign} \left( \sum_{i=1}^{N} \alpha_i^* y_i \exp \left( -\frac{x \cdot z^2}{2\sigma^2} \right) + b^* \right)
\]

When a new point \( x \) is predicted, it is only necessary to calculate its inner product of the point in the training sample to find the value corresponding to \( y \).

3.3. BP Neural Network model (BPNN)

The BP algorithm (Back Propagation) was provided by Rumelhart and Hinton of UCSD PDP team in 1986. The learning process of BP neural network includes the dissemination of positive information and error back-propagation. It is one of the most widely used and successful neural network models and one of the common methods of software defect prediction \[16\]. The topological structure of the entire network model is composed of the input layer, the hidden layer and the output layer. Its structure is shown in Figure 2.
BP neural network training is divided into two stages:

(1) The dissemination of positive information: input information is transmitted layer by layer and through the hidden layer from the input layer until the output of each cell is obtained;

(2) Error back-propagation: The output layer error starts from the output layer, layer by layer, and back propagates. The error of each unit of the hidden layer can be indirectly calculated, and the weight of the previous layer can be corrected with this error.

It is known that the actual output of the k-th neuron in the output layer is $y_k$, and the input is $net_k$, and the output of any neuron j in the hidden layer adjacent to the output layer is $y_j$.

$$net_k = \sum_j \omega_{jk} y_j$$  \hspace{1cm} (17)

$$y_k = f \left( net_k \right)$$  \hspace{1cm} (18)

where, $\omega_{jk}$ is the connection weight between neurons j and k.

According to Equation 11, the S-type output function is:

$$y_k = f \left( net_k \right) = 1/1 + e^{(net_k + a_k)/b_k}$$  \hspace{1cm} (19)

For the samples in the sample set $(X_p, Y_p)$, input $X_p$ into the network, then the actual output of the k-th neuron in the output layer is $y_{pk}$. Assuming that the expected output is $d_{pk}$, the error measure of the p-th sample in the BP network is:

$$E_p = \frac{1}{2} \sum_k \left( d_{pk} - y_{pk} \right)^2$$  \hspace{1cm} (20)

The error measure for the entire sample set in the BP network is:

$$E = \sum_p E_p$$  \hspace{1cm} (21)
Use the above formula to calculate the actual output and error measure based on the samples in the sample set, and adjust the connection weight, that is, 
\[ \omega_{jk} \leftarrow \omega_{jk} - \eta \frac{\partial E}{\partial \omega_{jk}} \], until \( \sum_{p} E_{p} < \varepsilon \).

4. Model comparison

4.1. Performance evaluation standard

According to section 2.1 of this paper, the last step in the software defect prediction process is the performance evaluation of the prediction model. The final prediction results are classified as defective or non-defective. Therefore, the prediction of software defects is a typical binary classification problem, and a basic confusion matrix can be obtained based on the prediction results, as shown in table 1.

| Actual Result | Forecast results |  |
|---------------|------------------|---|
|               | Defective        | Defect-free |  |
| Defective     | TP               | FN          |  |
| Defect-free   | FP               | TN          |  |

In the table, TP (True Positive) represents a true example, indicating the number of actual defective and correctly classified instances; TN (True Negative) represents a true negative example, indicating the actual number of defective and correctly classified instances; FP (False Positive) represents a false positive example, indicating the actual number of non-defective and misclassified instances, FN (False Negative) represents a false negative example, indicating the actual number of defective and misclassified instances.

Common performance evaluation criteria are as follows:

Precision (P): Precision, that is, the proportion of samples in the predicted positive class, which reflects the accuracy of the prediction model, also known as the Precision ratio. The calculation formula is as follows:

\[ P = \frac{TP}{TP + FP} \]  

Recall (R): Recall rate, which represents the proportion of correctly classified positive examples to actual positive examples, and reflects the probability that a defective module is correctly predicted. The formula is as follows:

\[ R = \frac{TP}{TP + FN} \]  

Accuracy (Acc): Represents the ratio of correctly predicted samples to the total number of test samples. The calculation formula is as follows:

\[ Acc = \frac{TP + TN}{TP + FN + FP + TN} \]  

F-Measure is an equilibrium measure between Recall (R) and Precision (P). Only when the values of R and P are large, F-Measure becomes correspondingly large. The calculation formula is as follows:

\[ F_i = \frac{2 \cdot P \cdot R}{P + R} \]
4.2. Experimental results and analysis

The data used in this experiment comes from the public data set of MDP (Metric Database Program) published by the official website of NASA. Five NASA data sets were selected as the experimental objects. See table 2 for details.

Table 2. Data sets.

| Dataset name | Characteristic number | Number of samples | Number of defective samples |
|--------------|-----------------------|-------------------|----------------------------|
| JM1          | 21                    | 7782              | 1672                       |
| MC1          | 38                    | 1988              | 46                         |
| CM1          | 37                    | 327               | 42                         |
| MW1          | 37                    | 253               | 27                         |
| PC1          | 37                    | 705               | 61                         |

In the training process, this paper uses 10-fold cross-validation to calculate the performance results. That is, the training data is divided into 10 parts, 9 parts are used for model training, and the remaining 1 part is used as the test model. In order to reduce the error caused by randomly dividing the data set in cross-validation, each group of experiments was repeated 10 times, and the average value was taken as the final performance prediction result.

For the five data sets in table 2, we mainly use the three machine learning methods (Logistic Regression model, Support Vector Machine model, BP Neural Network model) introduced in Chapter 3 to predict their defects and use the Precision (P), Recall (R) and F-Measure evaluate the performance of the three methods. The final prediction results are shown in table 3.

Table 3. Defect prediction results.

| Dataset name | BPNN | LR | SVM |
|--------------|------|----|-----|
|               | P    | R  | F   | P    | R  | F   | P    | R  | F   |
| JM1          | 0.825| 0.801| 0.813| 0.896| 0.872| 0.884| 0.905| 0.883| 0.894|
| MC1          | 0.794| 0.762| 0.778| 0.887| 0.852| 0.869| 0.885| 0.845| 0.865|
| CM1          | 0.772| 0.733| 0.751| 0.843| 0.801| 0.821| 0.826| 0.798| 0.812|
| MW1          | 0.775| 0.742| 0.758| 0.863| 0.822| 0.842| 0.875| 0.844| 0.859|
| PC1          | 0.735| 0.699| 0.716| 0.789| 0.742| 0.764| 0.801| 0.758| 0.779|
| Average      | 0.780| 0.747| 0.763| 0.856| 0.818| 0.836| 0.858| 0.826| 0.842|

It can be seen from the experimental results that the defect prediction effect of the BP Neural Network model on the five data sets is not ideal compared with that of the other two models. Due to the slower convergence rate in the modeling process, the generalization ability of this method is poor; The Logistic Regression model has better prediction results in the data sets JM1 and MC1 with higher sample size, but that in the data set PC1 with lower data size is significantly reduced. This model is easy to implement in defect prediction, but the prediction results are limited by the sample size; Combining the Precision, Recall and F-Measure value of the five data, the prediction accuracy of the Support Vector Machine model is high, but it is difficult to determine the optimal parameters used in the model, and the prediction ability is unstable.
5. Summary
This paper mainly introduces the software defect prediction model based on machine learning, and makes a simple analysis of the construction process of the prediction model and software metrics. Logistic Regression model (LR), Support Vector Machine model (SVM) and BP Neural Network model (BPNN) were selected from a large number of prediction models to perform defect prediction on five data sets in MDP data set, and the relevant experimental results are obtained by using Precision, Recall and F-measure. Through the performance evaluation and analysis, the three methods have their own advantages and disadvantages. Therefore, improving the prediction accuracy, reducing the error rate, enhancing the practicality of the model, and overcoming the shortcomings of each model will be the future research directions.

References
[1] Zheng RJ. 1992 Computer software testing technology. (Tsinghua University Press)
[2] Wang Q, Wu SJ and Li MS. 2008 Software defect prediction. Journal of Software 19(7) 1565–80 http://www.jos.org.cn/1000-9825/19/1565.htm
[3] Chen X, Gu Q, Liu WS, Liu SL and Ni C. 2016 Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software 27(1) 1,25 (in Chinese). http://www.jos.org.cn/1000-9825/4923.htm
[4] Menzies T, Greenwald J and Frank A. 2007 Data mining static code attributes to learn defect predictors IEEE Transactions on Software Engineering 33(1) 2-13
[5] Li L and Leung H. 2011 Mining static code metrics for a robust prediction of software defect-proneness Proceedings of the International Symposium on Empirical Software Engineering and Measurement. Los Alamitos, CA: IEEE Computer Society:207-14
[6] LUO Yunfeng and BEN Kerong. 2009 Survey of software fault static prediction methods. Journal of Frontiers of Computer Science and Technology 3(5) 449-59
[7] Li Y, Huang ZQ, Wang Y, Fang BW. Survey on Data Driven Software Defects Prediction. Acta Electronica Sinica,2017,45(4):982–988.
[8] Rathore SS and Sandeep K. 2017 A study on software fault prediction techniques. Artificial Intelligence Review 5 1–73.
[9] Schneidewind N. 1997 IEEE Standard For A Software Quality Metrics Methodology Revision And Reaffirmation Software Engineering Standards Symposium and Forum, 1997 Emerging International Standards ISESS 97, Third IEEE International 278
[10] Chidamber S R and Kemerer C F. 1994 A metrics suite for object oriented design IEEE Transactions on Software Engineering 20(6) 476-93
[11] McCabe T J. 1976 A complexity measure IEEE Transactions on software Engineering 4 308-20
[12] Chidamber, S. R., Kemerer and C. F. A metrics suite for object oriented design.
[13] Ruczinski I, Kooperberg C and Le Blanc M. 2003 Logic regression Journal of Computational and Graphical Statistics. 12(3) 475-511
[14] Gareth J, Daniela W, Trevor H and Robert T. 2015 An introduction to statistical learning with application in R. (China Machine Press) 234-58
[15] Cortes C and Vapnik V. 1995 Support-Vector Networks Machine Learning 20(3) 273-97
[16] Xiong J, Gao Y and Wang YY. 2016 Software Defect Prediction Model Based on Adaboost Algorithm. Computer Science 43(7) 186–90