Research on Software Multiple Fault Localization Method Based on Machine Learning

Meng Gao [1,2] , Pengyu Li [1,2] , Congcong Chen [2] and Yunsong Jiang [1,2]

[1]Beijing Institute of Control Engineering, No. 16, South Third Street, Zhongguancun, Haidian District, Beijing, China
[2] Beijing Sunwise Information Technology Ltd., No. 16, South Third Street, Zhongguancun, Haidian District, Beijing, China

Abstract. Fault localization is one of time-consuming and labor-intensive activity in the debugging process. Consequently, there is a strong demand for techniques that can guide software developers to the locations of faults in a program with high accuracy and minimal human intervention**.** Despite the research of neural network and decision tree has made some progress in software multiple fault localization, there is still a lack of systematic research on various algorithms of machine learning. Therefore, a novel machine-learning-based multiple faults localization is proposed in this paper. First, several concepts and connotation of software multiple fault localization are introduced, move on to the status and development trends of the research. Next, the principles of machine learning classification algorithm are explained. Then, a software multiple fault localization research framework based on machine learning is proposed. The process is taking the Mid function as an example, compares and analyzes the performance of 22 machine learning models in software multiple fault localization. Finally, the optimal machine learning method is verified in the multiple fault localization of the Siemens suite dataset. The experimental results show that the machine learning based on Random Forest algorithm has more accuracy and significant positioning efficiency. This paper effectively solved the problem of large amount of program spectrum data and multi-coupling fault location, which is very helpful for improving the efficiency of software multiple fault debugging.

1 Introduction

Software is fundamental to our lives today, and with its ever-increasing usage and adoption, its influence is practically ubiquitous. At present, software is critical to many security and safety-critical systems in industries such as medicine, aeronautics, and nuclear energy. Not surprisingly, this trend has been accompanied by a drastic increase in the scale and complexity of software. Unfortunately, this has also resulted in more software bugs, which often lead to execution failures with huge losses [1,2]. Furthermore, software faults in safety-critical systems have significant ramifications, including not only financial loss, but also potential loss of life, which is an alarming prospect [3].

Developing software programs are universally acknowledged as an error-prone task. The major bottleneck in software debugging is how to identify where the bugs are [4], this is known as fault localization problem. Nonetheless, faults in software are discovered due to erroneous behavior or some other manifestation of the faults, finding and fixing them is an entirely different matter. Fault localization, i.e., identifying the locations of faults, has historically been a manual task that has been recognized to be time consuming and tedious as well as prohibitively expensive [5], given the size and complexity of large-scale software systems today. Furthermore, manual fault localization relies heavily on the software developer's experience, judgment, and intuition to identify and prioritize code that is likely to be faulty. These limitations have led to a surge of interest in developing techniques that can partially or fully automate the localization of faults in software while reducing human input. Though some techniques are similar and some are very different, they each try to attack the problem of fault localization from a unique perspective, and typically offer both advantages and disadvantages relative to one another. With many techniques already in existence and others continually being proposed, as well as with advances being made both from a theoretical and practical perspective. There is a huge demand for technologies that can help programmers effectively locate errors and also stimulates the proposal of many fault localization techniques from a widespread perspective. As more and more researchers have devoted themselves to the area of software fault localization over the last 10 years, numbers of papers published on software fault localization from 1977 to 2018 are also increasing. This growth trend again supports the claim that software fault localization is not just an important but also a popular research topic as it has been discussed very heavily in top quality software engineering journals and conferences over the last 10 years [6].

The remainder of this article is organized in the following manner: we begin by describing software

[a] Corresponding author: ccchappiness@yeah.net

multiple fault localization techniques, research status and development trend, and machine learning classification model in Section 2. In Section 3, the specific process of software multiple fault localization method based on machine learning is summarized. Experimental results and analysis are described in Section 4. Finally, conclusions are presented in Sections 5.

2 Background

This section describes software multiple fault localization techniques, research status and development trend，machine learning classification model.

2.1 Software Multiple Fault Localization

Software multiple fault localization is to find the wrong instruction, procedure, or data definition implied in the source code of the program. The granularity of multiple fault localization can be program statements, basic blocks, branches, functions, or classes. Software multiple fault localization is mainly divided into static analysis method and dynamic testing method. The multiple fault localization based on static analysis method mainly uses program dependencies, constraint solving, theorem proving to analyse possible error locations in the program. The multiple fault localization based on dynamic testing method mainly uses test cases to collect program execution information and calculate possible error locations in the program. The process of software multiple failures caused by defects is shown in Figure 1 as follows.

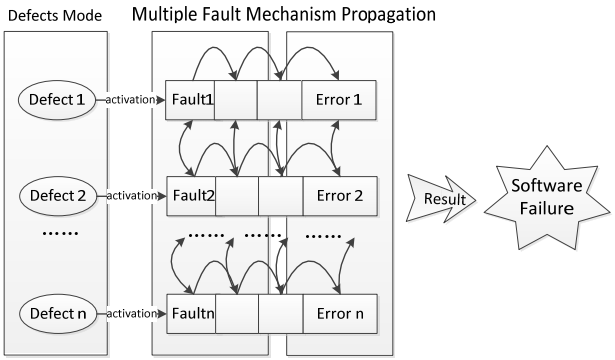

Figure 1. The process of software multiple failures caused by defects

First, during the operation of the system, defects may be activated, causing the system to malfunction; secondly, the fault will propagate as the system runs, and will continue to be converted into errors and passed between subsystems; finally, with the error continues to spread, and the error eventually reaches the user interface in the system, causing the system's incorrect behavior to be perceived by the user, which leads to system failure. It can be found that there is a multi-coupling fault action chain between the defect and the failure. Based on the multi-coupling fault action chain, the backtracking method can be used to find the defect occurrence position and then locate the defect. From the above failure mechanism, it can be found that software defects do not necessarily lead to software failure. Software defects will only be converted into runtime software failures when the specific conditions are met. Software errors will accumulate and effective propagation will occur. Eventually the software is invalidated.

According to the multiple fault mechanism of the defect described in Figure 1, the basic principle of the traditional fault localization is to re-run the defect program with the same input after setting the breakpoint, then check the corresponding program state and perform reverse reasoning, repeat the above process until find the defect. The basic techniques used for traditional fault localization are: breakpoints, single-step execution, output debug information, logging, event tracking, dump files, stack trace back, disassembly, observation and modification of data, control of debugged processes and threads.

2.2 Research Status and Development Trend of Fault Localization Techniques

In 2016, W.Eric Wong et al. [6] summarized a milestone development history of software fault localization technology in the paper "A Survey on Software Fault Localization", which from a publication repository that includes 331 paper published form 1977 to 2014. In his survey, the fault localization techniques were classified into eight categories, including slice-based techniques, program spectrum-based techniques, statistics-based techniques, program state-based techniques, machine learning-based techniques, data mining-based techniques, model-based techniques and miscellaneous techniques.

In 2015, Chen Xiang et al. [7] offers a systematic overview of existing research achievements of the domestic and foreign researchers in recent years in the paper “Review of Dynamic Fault Localization Approaches Based on Program Spectrum". The survey proposed a research framework based on program spectrum for dynamic defect localization and identified important influencing factors which can affect the effectiveness of fault localization. These factors include program spectrum construction, test suite maintenance and composition, number of faults, test case oracle, user feedback, and fault removal cost.

With the development of electronic technology, the software scale function is getting bigger and bigger, the internal logic relationship is more and more complicated, the number of defects in the system is also increasing, and the difficulty of software fault localization is increasing day by day. The traditional spectrum-based software fault localization and slice-based software fault localization solve the problem of software single fault location. Currently, there still exist the problems of low accuracy of software multiple fault localization and large amount of program spectrum data. Under the background of artificial intelligence and big data promotion, the development trend of the domestic and foreign researchers in recent years is as follows:

2.2.1 Software Multiple Fault Localization Techniques

Software fault localization techniques usually assume that only one defect is included in the error program, which is not the case. The presence of multiple faults in a program can inhibit the ability of fault localization techniques to locate the faults. This problem occurs for two reasons: first, when a program fails, the number of faults is generally unknown; second, certain faults may mask or obfuscate other faults. In recent years, researchers have studied how to locate error programs that contain multiple faults.

In 2007, Jones et al. [8] presented a parallel debugging approach to solving thus problem that leverages the well-known advantages of parallel work flows to reduce the time-to-release of a program, which consists of a technique that enables more effective debugging in the presence of multiple faults and a methodology that enables multiple developers to simultaneously debug multiple faults. Unlike Jones and others who use only program feature behavior, Abreu et al. [9] proposed a hybrid framework with logical reasoning in 2010. They uses both feature information from program execution and Bayesian inference to infer multiple instances. One of the characteristics of Bayesian inference that is questionable in the case of defects and their suspicious size is that it can well explain why multiple defects occur intermittently and cause program errors.

2.2.2 Machine Learning-Based Techniques

Machine learning is the study of computer algorithms that improve through experience. Machine learning techniques are adaptive and robust and can produce models based on data. In the context of fault localization, the problem can be identified as trying to learn or deduce the location of a fault based on input data such as statement coverage and the execution result of each test case.

Briand et al. [10] uses the C4.5 decision tree algorithm to construct rules that classify test cases into various partitions. The statement coverage of both the failed and successful test cases in each partition is used to rank the statements using a heuristic similar to Tarantula [11] to form a ranking. These individual rankings are then consolidated to form a final statement ranking which can be examined to locate the faults. This technique is more effective for bug locating, as only a relatively smaller amount of code needs to be examined to find bugs, compared to other state of the art contemporary techniques. Wong et al. [12] proposed a fault localization technique based on a back-propagation (BP) neural network, The coverage data of each test case and the corresponding execution results are collected, and to be used to train a BP neural network so that the network can learn the relationship between them. Then, the coverage of a suite of virtual test cases that each covers only one statement in the program is input to the trained BP network. The outputs can be regarded as the likelihood of each statement containing the bug.

In 2009, Ascari et al. [13] extended the BP-based technique [14] to object-oriented programs. As BP neural networks are known to suffer from issues such as paralysis and local minima. Subsequently, in 2012, Wong et al.[15] proposed an improved approach based on radial basis function (RBF) networks, which are less susceptible to these problems and have a faster learning rate [16], [17]. The RBF network is trained using an approach similar to the BP network. Once the training is completed, the output.

In 2013, He J.L et al. [18] proposed a novel neural-network-based multiple faults location model, which support degree of the input for each fault. The model learns the relationship between the faults and the candidate locations of faults using the constructed neural network. Constructing an ideal input as the input of learned neural network, the model can calculate the suspicious degree of each candidate location of fault, then obtain the sequence sorting by the suspicious degree, and complete the task of multiple fault localization.

The above research is only based on the research of neural network and decision tree. It has made some progress in software multiple fault localization. However, there is still a lack of systematic research on various algorithms of machine learning in software multiple fault localization. For the coupling, correlation and nonlinearity of software multiple fault, machine learning has strong generalization ability, adaptability and robustness. It can learn the inherent law implicit in the sample by learning the finite sample. In summary, there is an urgent need to use machine learning to train big data program spectrum to solve the problem of software multiple fault localization.

2.3 Machine learning classification model

Machine learning classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. The machine learning classification model attempts to construct a classifier by using a known-observed values, and predict the category of an unknown category object. Machine learning classification algorithms include Bayesian, neural networks, support vector machines, rules, decision trees, and integrated learning.

2.3.1 Bayesian Network

Bayesian network [19] is a probabilistic network, which is a graphical network based on probabilistic reasoning. The Bayesian formula is the basis of this probabilistic network. It is suitable for the expression and analysis of uncertain and probabilistic events. Reasoning can be made from incomplete, inaccurate or uncertain knowledge. The main goal of Bayesian inference is to estimate the value of a hidden node given the value of the observed node. Bayesian-based classification algorithms include Bayes Net, Naive Bayes, Naive Bayes Multinomial, Naive Bayes Multinomial Text, Naive Bayes Multinomial Updateable, Naive Bayes Updateable.

2.3.2 Neural Network

A is a mathematical model that mimics the structure and function of a biological neural network [20]. A neural network consists of a large number of nodes and nodes connected to each other. Each node represents a specific output function called an excitation function. The connection between every two node has a numerical weight value. The weight value can be adjusted experimentally, which enables the neural network to adapt to the input and be able to learn. The network connection, weight values, and stimulus functions determine the output of the network. The most common classification algorithm based on neural network is BP (Back Propagation) backpropagation neural network.

2.3.3 Support Vector Machine

SVM [21] is a supervised learning method widely used in statistical classification and regression analysis. The SVM is characterized by the ability to minimize both empirical errors and maximize geometric edges. Therefore, the support vector machine is also called the maximum edge classifier. Support vector machine technology has a solid theoretical foundation of statistics, and there are many successful cases in practice. SVM can be used well for high-dimensional data to avoid dimensional disasters. It has a unique feature that uses a subset of training instances to represent decision boundaries, which are called support vectors. The Sequential Minimal Optimization algorithm, the most common classification algorithm based on support vector machine.

2.3.4 Rule-based

Rule-based [22] classification is a technique for classifying records by using a set of judgment rules. In order to build a rule-based classifier, a set of rules need to be extracted to identify key relationships between dataset attributes and category labels. There are two ways to extract classification rules: the first one is the direct method, which extracts the classification rules directly from the data; the second is the indirect method, which extracts from other models such as decision trees, first finds the decision tree, and then from each The leaf node extracts a rule and merges the rules that can be merged through some optimization criteria. Rule-based classification algorithms include Decision Table, JRip, OneR, PART, and ZeroR.

2.3.5 Decision Trees

A decision tree is a mechanical way to make a decision by dividing the inputs into smaller decisions. The decision tree classification includes three parts: decision nodes, branches, and leaf nodes [23]. The decision node represents a test, usually representing an attribute of the sample to be classified, and different test results on the attribute represent a branch which represents a different value of a decision node. Each leaf node stores a category label indicating a possible classification result. The number in parentheses indicates the number of instances that arrive at the leaf node. Decision tree classification algorithms include Decision Stump, Hoeffding Tree, J48, LMT, Random Tree, REP Tree.

2.3.6 Ensemble Learning

Ensemble learning improves classification accuracy by aggregating prediction results from multiple classifiers. The ensemble method constructs a set of base classifiers from the training data and then classifies them by the predicted votes of each base classifier. The performance of the ensemble classifier is better than any single classifier because collective decision making is superior to individual decision making in terms of overall reliability and accuracy. There are two types of classification methods for building integrated classifiers. The first type processes the training data and subsamples to obtain a plurality of training sets according to a sampling distribution that determines the probability of sampling of a certain sample. Then use a specific learning algorithm to build a classifier for each training set. The second type processes the input features to obtain multiple training sets by selecting multiple subsets of the input features. This method is especially suitable for data with a large number of redundant features set. Ensemble learning algorithms include AdaBoostM1, Logit Boost, Random Forest [24]。

The logical view of ensemble learning is shown in Figure 2. The basic idea is to build multiple classifiers on the original data, then predict the categories of unknown samples separately, and finally gather the test results.

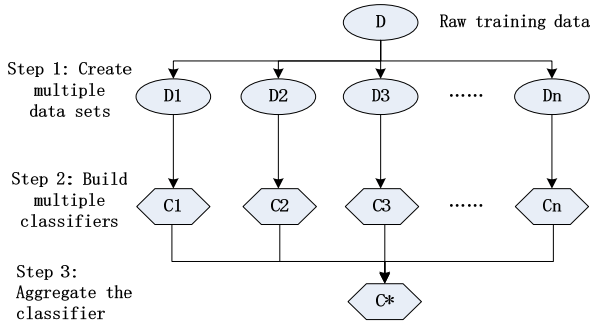

Figure 2. the Logical View of Ensemble Learning

The ensemble learning algorithm framework is as follows:

For *i* = 1 to k do
  Create training set Di by *D*
  Constructing a base classifier Ci by Di
End For
For Every test sample x∈**T do**
  C*(x) = **Vote**(C1(x),C2(x),…,Ck(x))
End For

Where, D represents the original training dataset, K represents the number of base classifiers, and T represents the test dataset.

3 Multiple Fault Localization Method

The specific process of software multiple fault localization method based on machine learning is summarized as follows. the block diagram is shown in figure 3.

(1) Input program spectrum dataset.

The program spectrum data consists of the statement coverage vector executed by the test case and the execution result of the test case.

(2) The program spectrum dataset is trained to obtain a optimal model.

The machine learning model (Bayesian, neural network, support vector machine, rules, decision tree, integrated learning) is used to train the program spectrum dataset, and optimize the training model by adjusting machine learning parameters.

(3) A statement suspiciousness ranking is obtained by testing optimal model through the virtual unit matrix.

Test the optimal model of machine learning by using the virtual unit matrix, obtain the suspiciousness value of each statement from the test results. Then sort them in order from high to low. The suspiciousness ranking can help facilitate the debugger to check the error line by line according to the suspiciousness of each statement, thereby improving the fault localization efficiency.

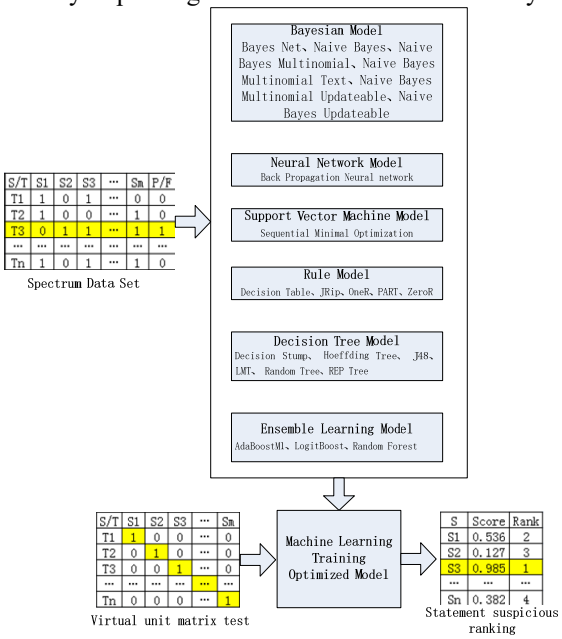

Figure 3. Block diagram of software multiple fault localization method based on Machine Learning

The mid function is a classic example of software fault localization demonstration. The mid function is a function to realize the intermediate value of three numbers. The mid-program is taken as an example to illustrate the software fault localization method of 22 machine learning models (RF is Random Forest, BP is Back Propagation Neural Network, LB is Logit Boost, AB is AdaBoostM1, NB is Naive Bayes, NBU is Naive Bayes Updateable, NBM is Naive Bayes Multinomial, NBMU is Naive Bayes Multinomial Updateable, DS is Decision Stump, SMO is, HT is Hoeffding Tree, BN is Bayes Net, RT is, NBMT is Naive Bayes Multinomial Text, DT is Decision Table, and REPT is REP Tree).

3.1 Software Single Fault Localization

For the single fault localization of line 4 of the mid function, 22 machine learning model training program spectrum data were used.

Function	Statement number	Test Cases Fault1 S4:m=x;					
		T1	T2	T3	T4	T5	T6
mid (x, y, z) {	input	1, 2, 3	3, 2, 1	5, 5, 5	5, 3, 4	4, 5, 3	2, 1, 3
m=z;	S1	1	1	1	1	1	1
if(y<z)	S2	1	1	1	1	1	1
if(x<y)	S3	1			1		1
m=y;	S4	1					
else if(x<z)	S5				1		1
m=x;	S6						1
else	S7		1	1		1	
if(x>y)	S8		1	1		1	
m=y;	S9		1				
else if(x>z)	S10					1	
m=x;	S11					1	
print(m);	S12	1	1	1	1	1	1
}	Pass/Fail	F	P	P	P	P	P

Figure 4. Single fault localization spectrum data

As can be seen from Figure 5, the best machine learning model for single fault localization is RF, BP, LB, AB, NB, NBU, NBM, NBMU, DS, SMO, HT, BN, which are ranked 1st in suspiciousness. The statement is the fault statement. The machine learning model RT, NBMT, DT, JRip, OneR, PART, ZeroR, J48, LMT, and REPT have poor resolution and no discrimination for the spectrum data of the mid function.

3.2 Software Double Fault Localization

For the double fault localization of lines 6 and 9 of the mid function, the machine learning model training program spectrum data is used.

Sn	RF	BP	LB	AB	NB	NBU	NBM	NBMU	DS	SMO	HT	BN	RT	NBMT	DT	JRip	OneR	PART	ZeroR	J48	LMT	REPT
S1	-0.80	-0.69	-1.00	-1.00	-1.00	-1.00	-0.58	-0.59	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S2	-0.80	-0.70	-1.00	-1.00	-1.00	-1.00	-0.58	-0.59	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S3	-0.64	-0.09	-1.00	-1.00	-0.92	-0.92	-0.30	-0.32	-1.00	-1.00	-1.00	-0.86	1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S4	-0.10	0.81	1.00	1.00	0.92	0.92	0.23	0.22	1.00	1.00	1.00	0.42	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S5	-0.98	-0.98	-1.00	-1.00	-1.00	-1.00	-0.58	-0.59	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S6	-0.80	-0.86	-1.00	-1.00	-1.00	-1.00	-0.43	-0.44	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S7	-0.90	-0.92	-1.00	-1.00	-1.00	-1.00	-0.67	-0.67	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S8	-0.88	-0.92	-1.00	-1.00	-1.00	-1.00	-0.67	-0.67	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S9	-0.80	-0.71	-1.00	-1.00	-1.00	-1.00	-0.43	-0.44	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S10	-0.84	-0.69	-1.00	-1.00	-1.00	-1.00	-0.43	-0.44	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S11	-0.80	-0.71	-1.00	-1.00	-1.00	-1.00	-0.43	-0.44	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67
S12	-0.80	-0.70	-1.00	-1.00	-1.00	-1.00	-0.58	-0.59	-1.00	-1.00	-1.00	-0.86	-1.00	-0.50	-0.71	-0.67	-1.00	-0.67	-0.50	-0.67	0.00	-0.67

Figure 5. Machine learning algorithms compare single fault suspicious calculation data

Function	Statement number	Test Cases Fault1 S6:m=y;Fault2 S9:m=z;									
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
mid (x,y,z) { int m;	input	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	5,4,2	2,1,3	5,2,6
m=z;	S1	1	1	1	1	1	1	1	1	1	1
if(y<z)	S2	1	1	1	1	1	1	1	1	1	1
if(x<y)	S3	1	1			1	1			1	1
m=y;	S4		1								
else if(x<z)	S5	1				1	1			1	1
m=x;	S6	1				1				1	1
else	S7			1	1			1	1		
if(x>y)	S8			1	1			1	1		
m=y;	S9			1				1	1		
else if(x>z)	S10				1						
m=x;	S11										
print(m);	S12	1	1	1	1	1	1	1	1	1	1
}	Pass/Fail	P	P	P	P	P	P	F	F	F	F

Figure 6. Double fault localization spectrum data

As can be seen from Figure 7, the machine learning model with the best double fault localization is RF, BP, LB, and AB. The statements with the first and second suspicious ranks are double fault statements. The machine learning model NB, NBU, NBM, DS, and RT suspiciousness ranks the first sentence for one of the fault statements. The machine learning model HT, BN, SMO double fault localization is not effective, no discrimination.

Sn	RF	BP	LB	AB	NB	NBU	NBM	NBMU	DS	RT	HT	BN	SMO
S1	-0.30	-0.82	-0.91	-0.87	-0.34	-0.34	-0.21	-0.21	-0.43	-1.00	-1.00	-0.18	-1.00
S2	-0.30	-0.83	-0.91	-0.87	-0.34	-0.34	-0.21	-0.21	-0.43	-1.00	-1.00	-0.18	-1.00
S3	-0.46	-0.93	-0.91	-0.87	-0.55	-0.55	-0.29	-0.30	-0.43	-1.00	-1.00	-0.18	-1.00
S4	-0.53	-0.97	-0.97	-0.92	-0.99	-0.99	-0.37	-0.38	-0.43	-1.00	1.00	-0.18	-1.00
S5	-0.34	-0.83	-0.91	-0.87	-0.34	-0.34	-0.19	-0.19	-0.43	-1.00	-1.00	-0.18	-1.00
S6	-0.06	0.17	0.00	0.00	-0.10	-0.10	-0.05	-0.05	-0.43	0.00	-1.00	-0.18	-1.00
S7	-0.18	-0.58	-0.91	-0.87	-0.10	-0.10	-0.05	-0.05	-0.43	-1.00	-1.00	-0.18	-1.00
S8	-0.10	-0.59	-0.91	-0.87	-0.10	-0.10	-0.05	-0.05	-0.43	0.33	-1.00	-0.18	-1.00
S9	0.27	0.04	0.28	0.12	0.34	0.34	0.16	0.15	0.33	-1.00	-1.00	-0.18	-1.00
S10	-0.56	-0.98	-0.97	-0.93	-0.99	-0.99	-0.37	-0.38	-0.43	-1.00	1.00	-0.18	-1.00
S11	-0.30	-0.84	-0.91	-0.87	-0.34	-0.34	-0.05	-0.05	-0.43	-1.00	-1.00	-0.18	-1.00
S12	-0.30	-0.83	-0.91	-0.87	-0.34	-0.34	-0.21	-0.21	-0.43	-1.00	-1.00	-0.18	-1.00

Figure 7. Machine learning algorithm compares double fault suspiciousness calculation data

3.3 Software Three Fault Localization

For the third fault localization of the 4th, 6th and 9th lines of the mid function, the machine learning model training program spectrum data is used.

Function	Statement number	Test Cases Fault1 S4:m=x;Fault2 S6:m=y;Fault3 S9:m=z;									
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
mid (x,y,z) { int m;	input	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	5,4,2	2,1,3	5,2,6
m=z;	S1	1	1	1	1	1	1	1	1	1	1
if(y<z)	S2	1	1	1	1	1	1	1	1	1	1
if(x<y)	S3	1	1			1	1			1	1
m=y;	S4		1								
else if(x<z)	S5	1				1	1			1	1
m=x;	S6	1				1				1	1
else	S7			1	1			1	1		
if(x>y)	S8			1	1			1	1		
m=y;	S9			1				1	1		
else if(x>z)	S10				1						
m=x;	S11										
print(m);	S12	1	1	1	1	1	1	1	1	1	1
}	Pass/Fail	P	F	P	P	P	P	F	F	F	F

Figure 8. Three fault localization spectrum data

As can be seen from Figure 9, the machine learning model with the best three-fault positioning effect is the three-fault statement in which the RF, BP, and LB are the first, second, and third sentences of the suspicious ranking. The machine learning model AB, NB, NBU, NBM, NBMU suspicious rankings 1 and 2 are defined as two fault statements. The machine learning model DS and the RT suspiciousness ranking 1st sentence is defined as one of the fault statements.

Sn	RF	BP	LB	AB	NB	NBU	NBM	NBMU	DS	RT
S1	0.01	-0.47	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S2	0.01	-0.47	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S3	-0.04	-0.34	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S4	0.35	0.85	0.94	0.89	0.99	0.99	0.33	0.33	1.00	1.00
S5	-0.31	-0.91	-0.69	-0.21	-0.29	-0.29	-0.14	-0.14	-0.11	-1.00
S6	0.16	0.79	0.32	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S7	0.07	-0.47	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S8	0.07	-0.51	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S9	0.42	0.60	0.22	0.21	0.42	0.42	0.20	0.20	-0.11	0.33
S10	-0.36	-0.92	-0.92	-0.92	-0.99	-0.99	-0.33	-0.33	-0.11	-1.00
S11	0.01	-0.48	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33
S12	0.01	-0.48	-0.48	-0.21	0.00	0.00	0.00	0.00	-0.11	0.33

Figure 9. Machine learning algorithm comparison three fault suspicious calculation data

3.4 Software Four Fault Localization

For the fourth fault localization of the 4th, 6th, 9th and 11th lines of the mid function, the machine learning model training program spectrum data is used.

Function	Statement number	Test Cases Fault1 S4:m=x;Fault2 S6:m=y; Fault3 S9:m=z;Fault4 S11:m=z;									
		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
mid (x,y,z) { int m;	input	3,3,5	1,2,3	3,2,2	5,5,5	4,4,3	5,3,4	3,2,1	5,4,2	2,1,3	5,2,6
m=z;	S1	1	1	1	1	1	1	1	1	1	1
if(y<z)	S2	1	1	1	1	1	1	1	1	1	1
if(x<y)	S3	1	1				1			1	1
m=y;	S4		1								
else if(x<z)	S5	1					1			1	1
m=x;	S6	1								1	1
else	S7			1	1	1		1	1		
if(x>y)	S8			1	1	1		1	1		
m=y;	S9			1				1	1		
else if(x>z)	S10				1	1					
m=x;	S11					1					
print(m);	S12	1	1	1	1	1	1	1	1	1	1
}	Pass/Fail	P	F	P	P	F	P	F	F	F	F

Figure 10. Four fault localization spectrum data

As can be seen from Figure 11, the machine learning model with the best four-fault positioning effect is the four-fault statement with the RF, BP, and LB being the suspicious rankings 1, 2, 3, and 4. The machine learning model AB suspicious degree rankings 1 and 2 are defined as two fault statements. The machine learning model of Random Forest and LogitBoost based on integrated learning and BP neural network excels in software multiple fault localization.

Sn	RF	BP	LB	AB
S1	0.04	-0.04	-0.14	0.01
S2	0.04	-0.02	-0.14	0.01
S3	0.01	0.07	-0.14	0.01
S4	0.37	0.89	0.88	0.68
S5	-0.20	-0.87	-0.51	0.01
S6	0.30	0.96	0.60	0.01
S7	0.09	-0.03	-0.14	0.01
S8	0.14	-0.05	-0.14	0.01
S9	0.41	0.40	0.17	0.01
S10	-0.20	-0.83	-0.69	0.01
S11	0.42	0.99	0.98	0.56
S12	0.04	-0.03	-0.14	0.01

Figure 11. Machine learning algorithm comparison four fault suspicious calculation data

4 Experimental Results and Analysis

This article uses the test dataset Siemens Suite [25] as the experimental object of the experiment, which contains seven programs, each with the correct version, several wrong versions and test cases. All of the above resources can be downloaded from the Software-artifact Infrastructure Repository (SIR) library at the University of Nebraska-Lincoln, as shown in the following table:

Table1. Summary of the Siemens of the Siemens Suite

Program	Description	Number of faulty versions	LOC	Number of test cases	Number of executable statements
Print_tokens	Lexical analyzer	7	565	4130	175
Print_tokens2	Lexical analyzer	10	510	4115	178
replace	Pattern replacement	32	563	5542	216
schedule	Priority scheduler	9	412	2650	121
schedule2	Priority scheduler	10	307	2710	112
tcas	Altitude separation	41	173	1608	55
tot_info	Information measure	23	406	1052	113

The entire system needs to go through： data collection, feature extraction, comparison of results, input data integration, machine learning model training and testing, virtual unit matrix, Comparison of multiple fault localization efficiency. The specific process is shown in the following figure:

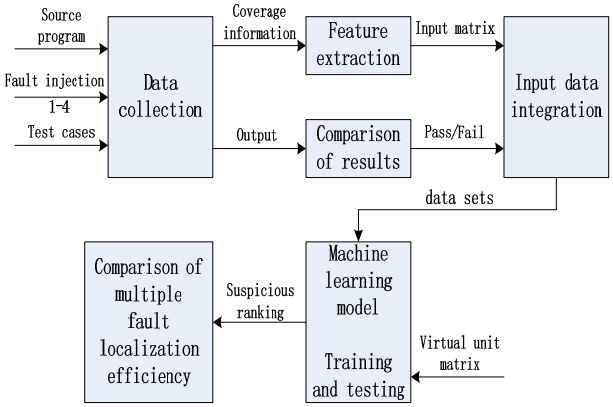

Figure 12. Siemens Suite basic process of multiple fault localization experiment

The main efficiency comparison criterion of the fault localization algorithm is to check as few statements as possible under the premise that the error has been located. Here, the accuracy [26] value is used to describe the positioning efficiency:

Accuracy = The suspicious ranking of the fault statement/

Total number of execution statements (1)

Among them: accuracy indicates the percentage of the statement that needs to be found before finding the real error statement. The smaller the value, the higher the efficiency of fault localization.

Among them: accuracy indicates the percentage of the statement that needs to be found before finding the real error statement. The smaller the value, the higher the efficiency of fault localization.

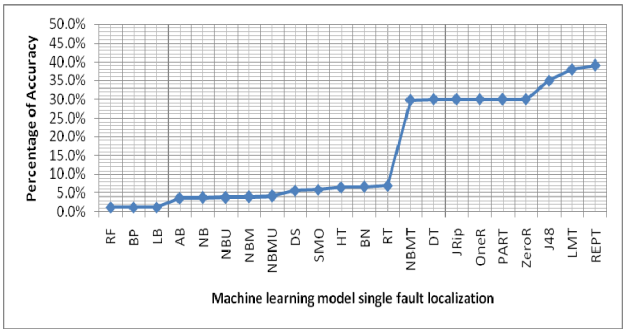

Figure 13. Machine learning model single fault localization accuracy

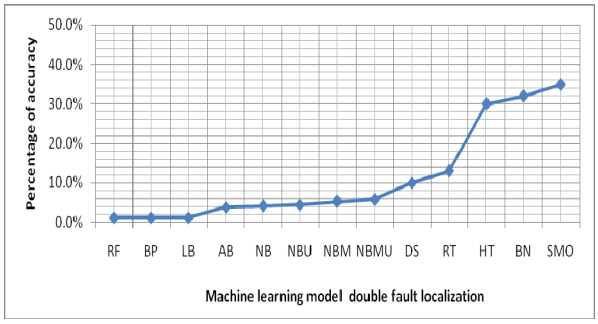

Figure 14. Machine learning model double fault localization accuracy

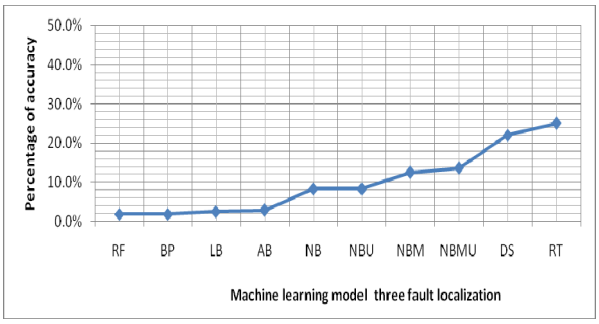

Figure 15. Machine learning model three fault localization accuracy

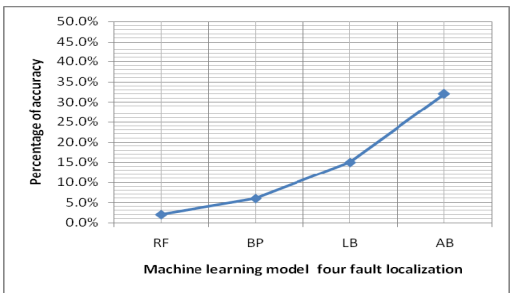

Figure 16. Machine learning model four fault localization accuracy

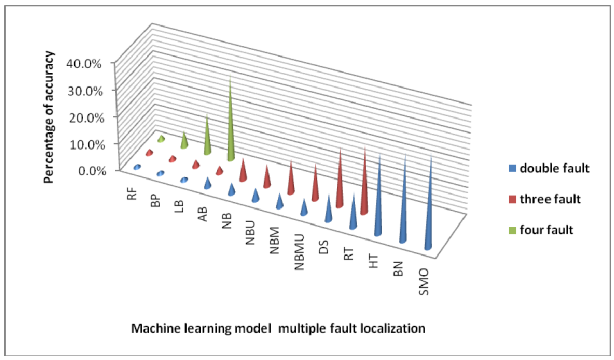

Figure 17. Machine learning model multiple fault localization accuracy

From Figure 13~17,Under the condition of single fault、double faults、three faults、four faults, the

accuracy of machine learning model of Random Forest and Logit Boost based on integrated learning and BP neural network perform well in software multiple fault localization.

This article introduces the standard Imp percentage [27], which is the percentage of the total number of lookup statements and the total number of statements in all error versions, in the case where all versions of the error for a single program are found. Obviously, the lower the Imp percentage, the less time it takes to represent the fault localization algorithm and the higher the efficiency.

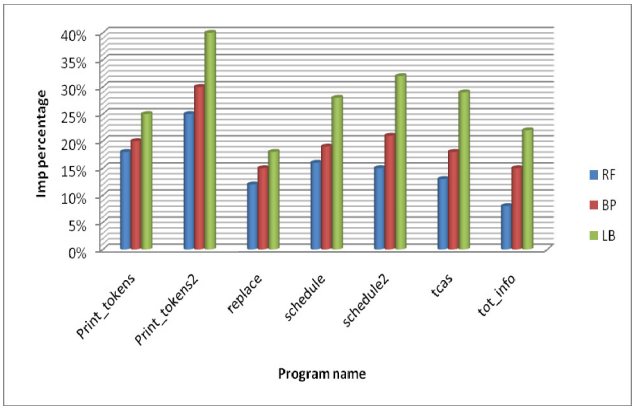

Figure 18. Comparison of Imp percentages of RF, BP, and LB programs

As can be seen from Figure 18, in the seven programs of the Siemens Suite, the Random Forest algorithm consumes the least search time and is more efficient than the LogitBoost and BP Neural Network algorithms.

5 Conclusion

A good software fault localization technology can save funds, save human resources, and accelerate the progress of software development. Therefore, how to solve these problems to improve the positioning with high accuracy and minimal or no human intervention has a great significant. Various techniques have been proposed to meet this requirement. However, the interactions between multiple faults which have not been fully considered in previous make the fault localization more complicated. In order to solve this problem, the paper proposed a multiple fault localization method, by comparing 22 kinds of machine learning model and using in Siemens suite dataset. The experimental results show that the Random Forest, BP neural network and Logit Boost machine learning model based on Ensemble Learning are excellent in software multiple fault localization. Among them, the Random Forest is suitable for solving the high-dimensional features of the program spectrum data, and the random forest model has strong generalization ability and high training speed. Therefore the machine learning model based on the random forest algorithm has more accurate multiple fault positioning effect and more significant positioning efficiency.

References

1. G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using Bayesian methods," *IEEE Trans. Softw. Eng.*, **33(10)**, pp. 675–686, (2007)
2. C. S. Wright and T. A. Zia, "A quantitative analysis into the economics of correcting software bugs," in *Proc. Int. Conf. Comput. Intell. Security Inf. Syst.*, Torremolinos, (2011)
3. W. E. Wong, V. Debroy, A. Surampudi, H. Kim, and M. F. Siok, "Recent catastrophic accidents: Investigating how software was responsible," in *Proc. 4th Int. Conf. Secure Softw. Integr. Rel.Improvement*, (2010)
4. I. Vessey, "Expertise in debugging computer programs: A process analysis," *International Journal of Man-Machine Studies*, **23(5)**:459–494,(1985).
5. T. Wang, "Post-mortem dynamic analysis for software debugging," *Ph.D dissertation*, FudanUniv. (2007)
6. W. E. Wong, R. GAO, Y. LI, R. ABREU, & F. WOTAWA, "A Survey on Software Fault Localization," *IEEE Trans. Softw. Eng.*,**42**, 707-740(2016)
7. X. Chen, X. Ju, W. Wen, Q. Gu, "Review of dynamic fault localization approaches based on program spectrum," *Journal of Software*, **26(2)** ,(2015)
8. J. A. Jones, J. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proc. ACM/SIGSOFT Int. Symp. Softw. Testing Anal.*, (2007)
9. R. Abreu, A. Gonz_alez, and A. J. Gemund, "Exploiting count spectra for Bayesian fault localization," *presented at the 6th Int.Conf. Predictive Models Softw. Eng.*, (2010)
10. L. C. Briand, Y. Labiche, and X. Liu, "Using machine learning to support debugging with tarantula," in *Proc. IEEE Int. Symp. Softw. Rel.*, (2007)
11. J. A. Jones, and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. Int. Conf. Autom. Softw. Eng.*, (2005)
12. W. E. Wong, T. Sugeta, Y. Qi, and J. C. Maldonado, "Smart debugging software architectural design in SDL," *J. Syst. Softw.*, **76(1),** pp. 15–28, (2005)
13. L. C. Ascari, L. Y. Araki, A. R. T. Pozo, and S. R. Vergilio, "Exploring machine learning techniques for fault localization," in *Proc. 10th Latin Am. Test Workshop*, (2009)
14. W. E. Wong and Y. Qi, "BP neural network-based effective fault localization," *Int. J. Softw. Eng. Knowl. Eng.*, **19(4),** pp. 573–597, (2009)
15. W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Trans. Rel.*, **61(1)**, pp. 149–169, (2012)

16. C. C. Lee, P. C. Chung, J. R. Tsai, and C. I. Chang, "Robust radial basis function neural networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, **29(6)**, pp. 674–685, (1999)
17. P. D. Wasserman, "Advanced Methods in Neural Computing," (1993)
18. J. L. He, H. Zhang, "Application of Artificial Neural Network in Software Multiple faults Location," *Jorunal of Computer Research and Development*, **50(3),**(2013)
19. T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining", *Inference and Prediction. Springer*,(2002)
20. T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud, "Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System," *IEEE Trans. Neural Networks*, **8(4)** pp. 902-909, (1997)
21. T. V. Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle, "Benchmarking Least Squares Support Vector Machine Classifiers," *Machine Learning*, **54(1)** pp. 5-32, (2004)
22. T. M. Khoshgoftaar and N. Seliya, "Analogy-Based Practical Classification Rules for Software Quality Estimation," *Empirical Software Eng.*, **8(4)**, pp. 325-350, (2003)
23. R. W. Selby and A. A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis, " *IEEE Trans. Software Eng.*, **14(12)** pp. 1743-1756, (1988)
24. L. Breiman, "Random Forests,"*Machine Learning*, **45(1),** (2001)
25. H. Do, S. Elbaum, G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*,**10(4)**,(2005)
26. Y. Lei, X. G. Mao, Z. Y. Dai, C. S. Wang, "Effective statistical fault localization using program slices," *2012 IEEE 36th Inter. Conf. Comp. Softw. and Appl*, (2012)
27. V. Debroy, W. E. Wong, X. Xu and B. Choi, "A Grouping-Based Strategy to Improve the Effectiveness of Fault Localization Techniques," *10th Inter. Conf. Qual. Softw.*, (2010)