International Journal of Research in Engineering, Science and Management
Volume-2, Issue-5, May-2019
www.ijresm.com | ISSN (Online): 2581-5792

Error Detection and Correction using Machine Learning Concepts

Asha Kutsa[1], Pauline Joseph[2], Garima Choudhary[3], Shrey Naik[4]
[1,2,3,4]*Student, Department of Computer Science Engineering, Ramaiah Institute of Technology, Bengaluru, India*

***Abstract*: The biggest problem that programmers face is errors. There are two types of errors: syntactical and logical. The identification of these errors is very time-consuming. This has led many researchers toward the topic of error detection and correction. This paper proposes a way to detect errors and suggest corrections to the user by comparing the user's code with the correct code that is stored in the system. Error detection and correction is incredibly useful in academic institutions where there is a constant inflow of students with no programming knowledge. The if-else block is one of the most simple and commonly used constructs in any programming language. However, a new programmer may be unaware of the importance of the ordering of the conditions within the block. In this paper, we try to identify the different rules which help in correct ordering within the if-else block. An error can be detected if a violation of these rules is found and can be prompted to the user. Machine learning models help in identifying latent errors in programs. Errors occur in the properties of the program. The machine learning concepts are tested on the properties of the program so that they are categorized first and then ranked depending upon the errors occurred. After analysing the properties, a subset of properties is selected the machine learning technique which may result in errors. A dynamic invariant detection is used by fault invariant detection to create program properties. Two algorithms are applied i.e decision tree and support vector machine on the properties for classification.**

***Keywords*: SVM, Machine learning, Data mining, Decision tree**

1. Introduction

At the beginning of a computer course, the students would still be unaware of the syntax and make many mistakes. The compiler generated error messages that are displayed after running incorrect code are difficult to understand. This makes it difficult to fix errors and wastes time and resources. This paper proposes an automatic error detection and correction system. The system focuses on the integration of data mining and machine learning concepts and also uses system programming to detect errors in the code. Data mining is used to store and organize the correct programs in the database. Using system programming, each correct program is given a unique identifier.

Machine learning concepts are used to analyze the code with errors, compare the incorrect code with the correct programs that are stored in the database, display the errors and also suggest corrections. This system helps save the programmers time and resources. These techniques can be used to model automated evaluation systems. Automated evaluation of programs can be done in two ways. Depending on the approach used in designing the system, they can either be static or dynamic.

Static assessment systems require the professor to provide a base model. The program provided by the student is compared to this model and it is not actually executed. Grades are awarded based on the amount of similarity between the two. Dynamic evaluation systems on the other hand, explicitly execute the students' programs and then check how correct the output is for different test cases. Marks are awarded based only on the success or failure of the student's program in the different test cases.

There are, however, many drawbacks to this type of automated system. Small syntax or logical errors may result in a failure of the program for a particular case and will result in the student losing all their marks, which seems a bit unfair for the student. Moreover, devious students may devise a way to rig the system, by ensuring that the required output is produced, even when it is not implemented properly. These disadvantages led to the need for an automated system that works differently.

There's no need for any test suite which is used for dividing success and fail runs therefore dynamic invariant detection is mostly used expensive programs. A subset of properties are used as input and outputs the subset which results in faults. A program analysis which is arbitrary generates program properties. Mostly the nature of the techniques in this paper are dynamic but these techniques are equally suitable for static analyses. The technique used has two steps: training and classification.

In training, machine learning concepts train the model. The properties which result in error are used and then fixed, this is done by machine learning models. Machine learning model created consists of fault revealing properties. In the second step which is the classification step, the user provides the trained model which is pre-computed and those properties which cause faults in large number are selected by the model. The main motive is to generate a model that suits the input set of points to those of points labels successfully.

2. Related works

This paper aims to propose an automated error detection and correction approach. Several people have previously worked on error detection and correction method in C programs using the

concepts of machine learning and data mining. Kunal Banerjee and K. K Sharma worked on the problem of the logical error caused due to incorrect precedence of the if-else statements which cannot be determined by a standard compiler. They rectified it by determining the precedence of if-else conditions. First, the conditions are ordered according to their precedence. Then these conditions are compared against the rules table. After this, the innermost conditions are executed first. After the conditions are normalized, the order of else-if construct is checked. At last, an analysis is done and time complexity is computed, which is later used to make it more efficient.

Michael D. Ernst and Yuriy Brun put forward a technique that uses the errors to generate machine learning models of program properties. It applies these models to the user's code and helps classify and rank properties that may lead to errors. Research on correcting logical errors in C programs was conducted by Prakash Murali, Abhay Ashok Patil and Atul Sandur. They used a combination of statistical control flow techniques and genetic algorithms for this. A subset of C language was taken in an attempt to probe the challenges that occur during error correction.

Xie and Engler demonstrated through their paper that errors which are correlated with redundancy are present in the source code where idempotent operations, redundant assignments on the files or dead code. Faults and errors are mostly caused due to redundant conditions. There are four steps which are important. First one, instead of using dynamic analysis they prefer metric which is statically computed.

In the second step, the relevance is increased by 45%-100% by an avg of 4860% in C programs which means it is increased by a factor of 4860%. In the third step, the experimental analysis is applied to the whole source file. Machine learning concepts are used by Dickinson et al. during program executions assuming that it is not as expensive for program execution but verifying of the correctness in every execution is very expensive comparatively. The goal is to find the runs with the most faults. Clustering is used to partition the test cases which is almost same as what partition testing is doing but clustering is done without any internal homogeneity guarantee.

3. Proposed work

A method for an automatic error detection system for c programs has been proposed. Each time code is executed by the user, it is compiled by the compiler. If the program is correctly executed, the system stores the program in the database with a unique identification. If the program does not compile or is not correctly executed, the incorrect code is compared with similar programs from the database. We take an example of an if-else construct. However, before we can compare the sample program with one already stored in the database, we need to establish certain rules with regard to the precedence of the ordering of conditions within the construct. The condition within the latter block should not be weaker than the condition within the former. If this condition is not met, then the subsequent block is similar to dead code and never gets executed.

This is because the first condition, in this case, would be a superset of the following conditions. Due to this, the conditions that appear in the later blocks are not checked because the conditions that appear in the first block are a superset of the latter blocks and have already been checked.

Let's take an example, suppose you have the code,

```
int x=50, i=0;
if (x>10)
   i=i+10;
else if(x>20)
   i=i+20;
else if(x>30)
   i=i+30;
```

In this case, the latter two statements are never executed because the set of values that corresponds to the first condition is a superset of the other two. This results in a logical error, which is not detected by the compiler. There are some rules which can help decide the precedence within the if-else block. The ordering of conditions within the student's programs can be tested against these rules. If there is a mismatch, feedback for correction can be provided automatically.

Let d1, d2,...,dn be the conditions.

Suppose we have two conditions c1 and c2 such that they have a relation c1=>c2. This means that if c1 is satisfied, c2 must also be satisfied ie, c1 is a stronger condition than c2.

For the following conditions, the ordering of conditions are as follows-

1) d1 ⇒ d2,

```
if (d1) {...}
else if (d2) {...}
```

2) d3 ⇒ d1 or d3 ⇒ d2

```
if (d3) {...}
else if (d1|| d2) {...}
```

3) d1 ⇒ d3 or d2 ⇒ d3

```
if (d1||d2) {...}
else if (d3) {...}
```

4) d1 ⇒ d3 and d2 ⇒ d4

```
if (d1 &&d2) {...}
else if (d3&& d4) {...}
```

5) d1 ⇒ d3 or d2 ⇒ d4

```
if (d1||d2) {...}
else if (d3||d4) {...}
```

The student's program is checked against these rules. If there is a violation of any of the rules then feedback can be given and the user. Therefore, these rules can detect violation in precedence. If the student program doesn't violate the precedence rules, it is added to the database and other programs can be compared to it. But if it does happen to violate the rules, it is compared against a similar program saved in the database.

When the two programs are compared, missing elements can be identified. Using some data mining concepts, these elements are organized. With the help of the machine learning device, the missing elements are analyzed and their function is learnt. Since the use of the invariants is known, we can either fill in the missing code or suggest the user do the same. For maximum efficiency, the code can be divided into functions or modules. This makes the detection easier as a small part of the programs are compared instead of the whole program. The correct solution can be embedded in the form of macros or new functions as per the user requirement.

4. Implementation

A. Machine learning algorithms

1) SVM

A supervised machine learning algorithm called SVM is used in both classification or regression challenges. Using the kernel functions, SVM tries to divide the labeled points. This function takes inputs as data and then transforms it into the required form. It also uses kernel functions to transform the point space and then chooses the best hyperplane which divides the labelled points successfully. We use linear kernel function of SVM which forms a hyperplane in the canonical point space. After training of the model, the points which are new can be classified depending upon where they belong to the model function.

2) Decision tree

In this algorithm, hyperplanes are used to separate the label points which is mostly perpendicular to one axis and also parallel to the other axes. The decision tree machine uses the greedy algorithm which can partition iteratively depending upon the entropy which is greater than a given threshold and then divides the partition to minimize entropy by adding a hyperplane across it. Fault Invariant Classifier implementation uses two experiments regarding automatic recognition of fault-revealing properties, plus a third experiment regarding whether fault-revealing properties help users find errors.

3) Measurements

Two quantities are measured in this experiment i.e. relevance and brevity. The importance and usefulness of the output is defined as relevance. Relevance is the ratio of the number of properties which are determined as correctly fault revealing to the total number of fault revealing properties. The brevity is defined for a set of properties is the opposite of the relevance. It the average number of properties a user can determine to be fault revealing.

5. Advantages and disadvantages

The advantages of this system are the time taken by programmers for debugging will be less as the errors will be detected and corrections will be suggested to them. Since the database stores all the correct code that it encounters, two similar programs can be compared based on space and time complexity and the most efficient program can be used. Another advantage of this system is that using cloud storage, multiple users from all over that world can access this database. Only programs that are compiled correctly will be stored in the database so that the error detection is more accurate. Every system also has disadvantages. Sometimes, if the algorithm or logic used for the code is not the same as anything stored in the database, the logical errors cannot be detected, making the system less efficient.

6. Conclusion

Since there are a large number of students enrolling in programming courses every year, it would be highly beneficial to have an automated system for error detection. There have been many proposals for such a system, but due to their shortcomings, there was a need for an improvised method to do the same. We have identified various cases wherein there is a proper ordering of conditions within the if-else block and we've also prescribed the rules. These rules can help detect violence in the precedence. If such a violation exists, feedback about the incorrect ordering of conditions within the if-else block can be provided to the programmer. The proposed methodology can help new programmers deal with various types of syntactical errors and can tell them how to deal with it. The scope of this paper extends past just the if-else block. It can be broadened to identify and correct logical errors written in various other languages like Java or C++.

In this paper, we propose that the experiment after evaluation shows that the Fault Invariant Classifier can easily determine if the properties are fault revealing. It is better to rank and select the top properties than selecting all the properties which are said to be fault revealing by the machine learning. In C programs the average of 45% of the top 80 properties can result in fault-revealing. For Java programs, about 59% of the top 80 properties may be fault revealing. It is not a compulsion that all properties which are fault revealing may cause an error but most of the preliminary studies did conclude that. Therefore, we conclude that on average the user only has to examine 3 of the properties to be determined an error.

We determine that decision trees substantially underperforms support vector machines because the decision tree algorithm is easier to read for the humans and permit a preliminary examination of what slots appear to be most important. This decision tree technique can be explained as "learning from fixes". the machine learner can be trained on pairs of programs where one consists an error and the other one is the fixed

version that solves the error.

References

[1] K. K. Sharma, Kunal Banerjee, Indra Vikas, Chittaranjan Mandal, "Automated Checking of the Violation of Precedence of Conditions in else-if Constructs in Student's Programs", IEEE International Conference on MOOC, Innovation and Technology in Education (MITE), 2014

[2] Yuriy Brun, Michael D. Ernst, "Finding latent code errors via machine learning over program executions", Proceedings of the 26th International Conference on Software Engineering (ICSE)., 2004.

[3] Tatiana Vert, Tatiana Krikun, Mikhail Glukhikh, "Detection of Incorrect Pointer Dereferences for C/C++ Programs using Static Code Analysis and Logical Inference", Tools & Methods of Program Analysis, 2013.

[4] T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang, "Ability-training-oriented automated assessment in introductory programming course," Computers & Education, vol. 56, no. 1, pp. 220 – 226, 2011.

[5] T. Wang, X. Su, Y. Wang, and P. Ma, "Semantic similarity-based grading of student programs," Info. and Software Technology, vol. 49, no. 2, pp. 99 – 107, 2007.

[6] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, High-Level Synthesis: Introduction to Chip and System Design. Kluwer Academic, 1992.

[7] K. K. Sharma, K. Banerjee, and C. Mandal, "A scheme for automated evaluation of programming assignments using FSMD based equivalence checking," in I-CARE, pp. 10:1–10:4, 2014.

[8] C. Karfa, C. Mandal, D. Sarkar, S. R. Pentakota, and C. Reade, "A formal verification method of scheduling in high-level synthesis," in ISQED, pp. 71–78, 2006.