Bài báo Predictive Mutation Testing của ﻿Jie Zhang , Lingming Zhang, Mark Harman, Dan Hao, Yue Jia, and Lu Zhang

Bài này làm trên ngôn ngữ java, dự báo các mutation bậc 1, sử dụng model Random Forest Classifier để dự đoán dựa trên 14 thuộc tính được trích xuất bằng các công cụ. Nó được chạy trên 163 dự án trong thế giới thực theo hai kịch bản ứng dụng (phiên bản chéo và dự án chéo). Các kết quả thử nghiệm chứng minh rằng PMT cải thiện hiệu quả của thử nghiệm đột biến lên tới 151,4 lần trong khi chỉ làm giảm độ chính xác nhỏ. Nó đạt được giá trị AUC trên 0,80 cho phần lớn các dự án, cho thấy sự cân bằng tốt giữa hiệu quả và hiệu quả của thử nghiệm đột biến dự đoán.

Feature

 ﻿1) numExecuteCovered, which refers to how many times the mutated statement is executed by the whole test

suite.

2) numTestCovered, which refers to how many tests from the test suite cover the mutated statement.

﻿3) typeStatement, which refers to the type of the mutated statement, e.g., assignment statement, conditional statement, return statement, or method invocation.

Note that for Java they can be easily categorized by analyzing bytecode instruction types. To illustrate, INVOKEVIRTUAL, INVOKESTATIC, and INVOKESPECIAL denote method invocation while IRETURN, ARETURN, DRETURN, and FRETURN denote return statements.

4) typeOperator, which refers to the type of the mutation operator. Note that mutation testing tools usually record the mutation operators used to generate each mutant, and it is easy to fetch such information.

﻿5) infoComplexity, which refers to the McCabe Complexity [41] of the mutated method.

6) depInheritance, which refers to the maximum length of a path from the mutated class to a root class in the inheritance structure.

7) depNestblock, which refers to the depth of nested blocks of the mutated method.

﻿8) numChildren, which refers to the total number of direct subclasses of the mutated class.

9) LOC, which refers to the number of lines of code in the mutated method.

10) Ca, which refers to the number of classes outside the mutated package that depend on classes inside the package.

11) Ce, which refers to the number of classes inside the mutated package that depend on classes outside the package.

12) instability, which is an indicator of the package’s resilience to change, and is computed based on the Ca and Ce values, i.e., .

﻿13) typeReturn, which refers to the return type of the mutated method.

14) numMutantAssertion, which refers to the total number of assertions in the test methods that cover each mutant.

15) numClassAssertion, which refers to the total number of test assertions inside the mutated class’s corresponding test class.

Model

﻿Random Forest, which is a generalization of tree-based classification, as our default classification technique, because Random Forest greatly improves the robustness of classification models by maintaining the power, flexibility, and interpretability of tree-based classifiers [47], and has been shown to be more effective in dealing with imbalanced data [34].

Data

﻿We use 9 projects that have been widely used in previous software testing research [50], [55], [56] as the cross-project scenario subjects to evaluate PMT. In particular, we use the latest versions (i.e., the HEAD commit) of these projects as the base subjects. To facilitate the evaluation of PMT in the cross-version scenario, we prepare multiple versions for each base subject as in existing work [50]. More specifically, we select each version by counting backwards 30 commits at a time from the latest version commit of each project and generate up to 10 versions per project. Note that projects may have fewer than 10 versions due to the limited version history or execution problems for our mutation testing or feature extraction tools.

To further extensively evaluate PMT, we collect another 154 projects. We started with the first 1000 most popular Java projects. 10 from Github in June 2015; 388 of these projects were saved because they are single-packaged, and were successfully built with Maven11 and passed all their Junit tests; then, 234 projects were further removed because they cannot be handled by the PIT tool or the other supporting tools used in our study.

The sizes of the final 154 projects range from 172 to 92,176 lines of code. The 154 projects have various distribution on the proportion of killed mutants. 24

projects have notably imbalanced data that the proportion of killed mutants is below 0.2 or above 0.8. More details of all these subjects (e.g., the subject name, version number and the statistics) are available at our project homepage.

Mutpy

MutPy is a mutation testing tool for Python 3.3+ source code. MutPy supports standard unittest module, generates YAML/HTML reports and has colorful output. It applies mutation on AST level. You could boost your mutation testing process with high order mutations (HOM) and code coverage analysis.