Contents lists available at SciVerse ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Linking software testing results with a machine learning approach

Alexandre Rafael Lenz, Aurora Pozo, Silvia Regina Vergilio*

Computer Science Department, Federal University of Paraná (UFPR), Brazil. CP 19:081, CEP: 81531-970, Curitiba, Brazil

ARTICLE INFO

Article history:
Received 15 March 2012
Received in revised form
10 January 2013
Accepted 15 January 2013
Available online 6 April 2013

Keywords:
Machine learning
Software testing
Test coverage criteria

ABSTRACT

Software testing techniques and criteria are considered complementary since they can reveal different kinds of faults and test distinct aspects of the program. The functional criteria, such as Category Partition, are difficult to be automated and are usually manually applied. Structural and fault-based criteria generally provide measures to evaluate test sets. The existing supporting tools produce a lot of information including: input and produced output, structural coverage, mutation score, faults revealed, etc. However, such information is not linked to functional aspects of the software. In this work, we present an approach based on machine learning techniques to link test results from the application of different testing techniques. The approach groups test data into similar functional clusters. After this, according to the tester's goals, it generates classifiers (rules) that have different uses, including selection and prioritization of test cases. The paper also presents results from experimental evaluations and illustrates such uses.

1. Introduction

The main goal of the software engineering is to produce high quality software. In this sense, software testing is considered a fundamental activity of the software development. Its main goal is to reveal faults, through the execution of "good test cases". A good test case is one that has a high probability of finding an unrevealed fault. Testing techniques and criteria have been proposed to achieve the testing goals with minimal effort and costs. Testing criteria are predicates to be satisfied and are usually derived by applying one of the following techniques: functional, structural (control and data-flow based criteria) and fault-based (mutation based testing). These criteria consider distinct aspects to derive test data and can reveal different kind of faults. Hence, a testing strategy should apply the criteria in a complementary way, and the use of supporting tools is very important to reduce costs.

The existing tools usually implement structural and fault based criteria, and generally produce different results. Results from structural criteria include: required elements, executed paths and structural coverage. Supporting tools for fault-based techniques are usually based on mutation testing, and generate: number of mutants created by each operator, mutation score, dead mutants, etc. In this kind of test each mutant, as well as the corresponding mutation operator, describe a specific fault. One of the most known and used functional criterion, Category Partition (Ostrand and Balcer, 1988), divides the input domain into equivalence classes considering the functionality of the program, and/or its input and output. After this, it selects at least one test data from each class. But this division is commonly very subjective and influenced by the tester. Due to this, the functional criteria are difficult to be automated and are generally manually applied.

In the literature, we find works on automatic generation or improvement of functional test data. Works that investigate the use of machine learning (ML) techniques present promising results. These techniques are capable of acquiring knowledge from data and performing very well on subjective tasks (Mitchell, 1997). Some works use the specification, and inputs and outputs for reducing test sets (Last and Kandel, 2003; Saraph et al., 2003). Other ones use faults (Briand et al., 2008) or the structural coverage to learn the program specification or behavior of the program (Bowring et al., 2004). We can see that works on the automatic generation of functional test data generally consider only a kind of test information. Furthermore, they do not integrate information resulting of different testing techniques that should be applied in a test strategy. They do not have the goal of establishing relationships among them.

To overcome this limitation, in a previous work (Lenz et al., 2011) we introduced a ML approach with such goal. The approach has as entry the information produced by different testing tools and criteria, and uses clustering techniques to group test cases into clusters. The clusters can be used as functional equivalence classes. In this way, the approach contributes to automate the functional criterion Category Partition, by using information resulting from the application of complementary testing techniques. Now, the present paper extends previous work. The approach is refined with additional steps, where the test results and clusters feed ML classifier algorithms, which produce sets of if–then rules to classify test cases. The obtained rules are very helpful during the regression testing and can be used in different ways to reduce testing costs and effort. The rules can be used to produce a strategy for reduction and prioritization of test data according to tester's goals. Moreover, the paper presents more complete experimental results

* Corresponding author. Tel.: +55 11 33613681; fax: +55 11 33613205.
E-mail addresses: arlenz@gmail.com (A. Rafael Lenz), aurora@inf.ufpr.br (A. Pozo), silvia@inf.ufpr.br (S. Regina Vergilio).

including three clustering algorithms and different sets of attributes related to the available testing information. The paper also illustrates possible uses of the generated clusters and classifiers for reduction of regression test data sets in comparison with other common strategies.

The paper is organized as follows. Section 2 introduces some background on software testing. Section 3 reviews the ML field. Section 4 introduces the approach. Section 5 describes how the approach was evaluated. Section 6 presents evaluation results. Section 7 illustrates possible uses of the introduced approach in testing tasks such as: reduction, selection and prioritization of test cases. Section 8 contains related work. Section 9 presents conclusions and future works.

2. Software testing

There are in the literature different testing techniques and criteria. The functional criteria (or black box testing) derive test data only based in the specification or functionalities of the program. The most used functional criterion is Category Partition (Ostrand and Balcer, 1988) that divides the input domain into equivalence classes and selects at least one element in each class. The division is based on the functionalities of the program and generally considers the input and outputs produced. The main disadvantage of this criterion is that the partition is commonly subjective and difficult to automate. Therefore, it is often manually applied.

Structural criteria consider the structure of the implementation to derive the tests. They generally require the execution of complete paths of the program to cover certain required elements such as nodes, edges, paths or associations between variables and their consequent uses. Poketool (***Po**tential-Uses **Cri**teria **Tool** for program testing*) (Maldonado et al., 1992) is a tool that implements the following structural criteria: (1) control-flow based criteria: all-nodes (AN), all-edges (AE); and (2) data-flow based criteria: all-potential-uses (PU), all potential-du-paths (PDU) and all-potential-uses/du (PUDU).

Fault-based criteria derive test data based on specific faults that can be present in the code due to common programmer's mistakes. The most known fault-based criterion is Mutation Analysis, which generates mutant versions of the program P being tested through the application of mutations operators. Each mutant describes a fault that can be present in P. Test data are generated to distinguish the output produced by P and its mutants. *Proteum* (***Pro**gram **Te**sting **U**sing **M**utants*) (Delamaro and Maldonado, 1996) is a tool that supports mutation testing of C programs.

After the application of the testing criteria, a lot of information is produced and available. In many cases, some test cases need to be re-executed during regression testing. Let P be a program, let P' be a modified version of P and let T be a test suite for P. Regression testing is concerned with validating P'. It is fundamental to get confidence that the changes made in the program are correct and to ensure that unchanged parts of the program were not affected. This may include the reuse of T, and the creation of new test cases.

Rothermel et al. (2004) consider four main methodologies that are used in the context of reusing an existent T:

- retest-all (Leung and White, 1989): reuses all test cases of T. To rerun all tests conducted before is desired but also is an expensive and effort-consuming task;
- regression test selection (Rothermel and Harrold, 1996): this problem can be described as: given P', a new version of program P, and T an existing test set, the problem is how to select $T' \subset T$ to execute on P'. Different test selection techniques were proposed. Some of them are based on the program specification, but most of them consider the information about the code of the program. They also have distinct goals (Rothermel and Harrold, 1996): to locate modified elements or components and to select tests that exercise those components (coverage based techniques); to select minimal sets of test (minimization techniques); to select tests that can expose one or more faults (safe techniques);
- test suite reduction (Chen and Lau, 1996; Harrold et al., 1993; Jones and Harrold, 2001; Offutt et al., 1995): has the goal of removing redundant test cases from T by reducing the test-suite size and cost. But this can also reduce the fault detection capability of the test suites; and
- test case prioritization (Elbaum et al., 2001a,b; Jones and Harrold, 2001; Srivastava and Thiagarajan, 2002; Wong et al., 1997): schedules test cases so that those with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. For example, testers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or increases the likelihood of detecting faults early in testing. Many different prioritization techniques have been proposed, but the techniques most prevalent in literature and practice involve those that utilize simple code coverage information, and those that supplement coverage information with details on where code has been modified.

The results produced by our approach can be used for selection, reduction and prioritization of test cases according to aspects that the tester wants to emphasize. These uses are illustrated in Section 7, which also provides a comparison with some traditional techniques that can be used to perform these tasks.

3. Machine learning

Machine learning techniques implement mechanisms for automatically inducing knowledge from examples (Mitchell, 1997). Each example is represented by a vector V of attributes values. According to the available information they are classified into: supervised and unsupervised learning.

Supervised learning usually formulates the problem as a classification problem. The training data consist of pairs of inputs (vectors) and desired outputs. The classification task produces a model based on the data, which is used to classify unseen data item according to its attributes. For example, in a classification problem, a hospital may want to classify medical patients into those who have high, medium or low risk to acquiring a certain illness. In this paper, to illustrate the use of our approach we use the algorithm C4.5 (Quinlan, 1993; Mitchell, 1997) based on Decision Trees. The C4.5 algorithm uses the information gain and entropy measures to decide on the importance of the attributes. C4.5 recursively creates branches corresponding to the values of the selected attributes, until a class is assigned as a terminal node. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the sub-tree rooted at the new node. Each branch of the tree can be seen as a rule, whose conditions are formed by their attributes and respective tests.

Unsupervised learning detects relationships among examples, e.g., the determination of similar groups of examples. It is distinguished from supervised learning in that the learner is given only unlabeled examples. Clustering can be considered the most important unsupervised learning task. Clustering techniques explore similarities between patterns, grouping the similar ones into categories or groups. For example, in a medical application we