International Conference on Multidisciplinary approaches in Engineering, Management & Science
Sri S.Ramasamy Naidu Memorial College, Sattur, Tamil Nadu, India (MAEMS-18)
14th December 2018 www.conferenceworld.in ISBN:978-93-87793-62-0

MACHINE LEARNING BASED METAMORPHIC TESTING FOR SOFTWARE QUALITY ASSESSMENT

A.Josephine Prapulla[1], Dr.L.Manjunatha Rao[2]

[1]*Research scholar, Bharathiar University. Coimbatore.*

[2]*Professor and Director Dr. B. R. Ambedkar institute of technology. Bengaluru.*

ABSTRACT:

A software quality assessment is a disciplined examination of the software processes used by an organization, based on a process model. Metamorphic testing is used to verify the functional correctness of software in the absence of an ideal oracle. The ability to automatically detect failures and anomalies using MRs becomes difficult task in the day to day developing area. In this paper, the machine learning algorithm is introduced to automatically detect failures and anomalies using MRs can also provide hints for the construction of run-time self-correction mechanisms. The Naive bayes classification algorithm is used to improve the detection failures and anomalies to improve the self correction in the search engine results. The proposed technique acquires improved precision and recall when compared to conventional methods.

Keywords*: MT, MR, Search engine, Naive bayes, Supervised machine learning*

I. INTRODUCTION:

In software testing, a test oracle is needed to determine whether the program under test exhibits an acceptable behavior. Typically, it is very costly for testers to obtain a suitable test oracle for the program under test [1]. For some programs (e.g., scientific programs), which Weyuker [2] refers to as \non-testable" programs, obtaining test oracles may be extremely difficult or even impossible. In such a circumstance, testers may be unable to decide whether the program outputs are correct for most given inputs.

Metamorphic testing has been shown to be a simple yet effective technique in addressing the quality assurance of applications that do not have test oracles, i.e., for which it is difficult or impossible to know what the correct output should be for arbitrary input. In metamorphic testing, existing test case input is modified to produce new test cases in such a manner that, when given the new input, the application should produce an output that can be easily be computed based on the original output.

The introduction of metamorphic testing can be traced back to a technical report by Chen et al. [5] published in 1998. However, the use of identity relations to check program outputs can be found in earlier articles on testing

of numerical programs [6], [7] and fault tolerance [8]. Since its introduction, the literature on metamorphic testing has flourished with numerous techniques, applications and assessment studies that have not been fully reviewed until now. Although some papers present overviews of metamorphic testing, they are usually the result of the authors' own experience [9], [10], [11], [12], [13], review of selected articles [14], [15], [16] or surveys on related testing topics [3]. At the time of writing this article, the only known survey on metamorphic testing is written in Chinese and was published in 20091 [17]. As a result, publications on metamorphic testing remain scattered in the literature, and this hinders the analysis of the state of the art and the identification of new research directions.

In this paper, the naive bayes classification algorithm is introduced to automatically detect failures and anomalies using MRs can also provide hints for the construction of run-time self-correction mechanisms. The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 briefly discussed about the proposed system. The experimental results are discussed in section 4 and section 5 concludes the paper.

II. RELATED WORKS

Metamorphic testing was introduced by Chen *et al.* [4] as a technique to alleviate the oracle problem. The basic idea is illustrated in Figure 1. For the software under test (SUT), the executions of a source test case (TC) and a follow-up TC are compared against some metamorphic relation (MR). In Figure 1, even if no oracle is available to verify each individual output, sin(*x*1) or sin(*x*2), the sine function can still be tested by comparing the pair of outputs against the given MR: sin(*x*1)=sin(*x*2).

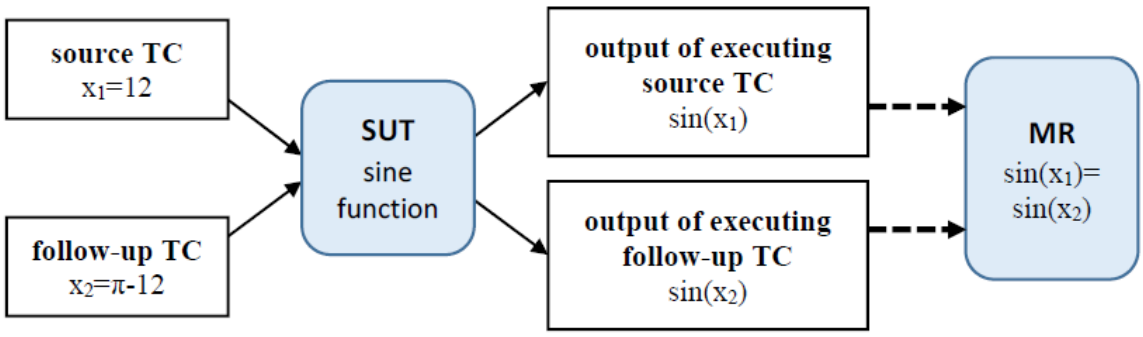

Fig. 1. Illustration of metamorphic testing

MRs can be of various forms, such as equalities, inequalities, periodicity properties, convergence constraints, subsumption relationships, etc. They resemble the concept of program invariants [15] but important distinctions must be made. While an invariant has to hold for every possible program execution, an MR is a relation between some executions (e.g., a pair of source TC and follow-up TC executions in Figure 1). As a result, MRs are

necessary properties of the intended program's functionality: If an MR violation is detected, then metamorphic testing reveals the presence of defects in the SUT.

One of the earliest case studies of applying metamorphic testing to scientifc software was performed by Chen *et al.* [7]. Their SUT was a numerical program implementing the alternating direction implicit method to solve the partial diferential equation (Laplace equation) with Dirichlet boundary conditions. Because one cannot fnd exact solutions to such numerical problems, an MR was developed to alleviate the oracle problem. The case study showed the efectiveness of the MR by detecting the subtle errors in the SUT that special TCs (e.g., symmetric boundary conditions in a square plate) failed to reveal. Building on the work by Chen *et al.* [7], metamorphic testing has been applied to a variety of scientifc software systems, including casting simulation [9], ad-hoc network protocol simulators [8], open queuing network modeling [9], and Monte Carlo modeling for the simulation of photon propagation [10].

In summary, metamorphic testing has been applied to verify and validate scientifc software that produces complex output like complicated numerical simulations. Most approaches like [11] develop the MRs in a one-shot manner, organize them in the same hierarchy, and analyze their executions separately. The study presented next motivates our novel way of hierarchically creating the MRs.

III. PROPOSED METHODOLOGY

Our approach is based on the concept of metamorphic testing, summarized below. To facilitate that approach, we first identify the metamorphic relations that a classification algorithm is expected to exhibit between sets of inputs and sets of outputs. We then utilize these relations to conduct our testing of the implementations of the algorithms under investigation.

In this work, however, our approach focuses on the machine learning classifiers. According to the general anticipated behaviors of these algorithms, we define our MRs formally as follows.

MR-0: Consistence with affine transformation

The result should be the same if we apply the same arbitrary affine transformation function, $f(x) = kx + b$, $(k \neq 0)$ to the values of any subset of attributes for each sample in the training data set S and the test case t_s.

MR-1.1: Permutation of class labels

Assume that we have a class-label permutation function *Perm*() to perform one-to-one mapping between a class label in the set of labels L to another label in L. If the source case result is l_i, applying the permutation function to the set of corresponding class labels C for the follow-up case, the result of the follow-up case should be $Perm(l_i)$.

MR-1.2: Permutation of the attribute

If we permute the *m* attributes of all the samples and the test data, the output should remain unchanged.

MR-2.1: Addition of uninformative attributes

An uninformative attribute is one that is equally associated with each class label. For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we add an uninformative attribute to each sample in *S* and respectively a new attribute in t_s. The choice of the actual value to be added here is not important as this attribute is equally associated with the class labels. The output of the follow-up test case should still be l_i.

MR-2.2: Addition of informative attributes

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we add an informative attribute to each sample in *S* and t_s such that this attribute is strongly associated with class l_i and equally associated with all other classes. The output of the follow-up test case should still be l_i.

MR-3.1: Consistence with re-prediction

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we can append t_s and c_t to the end of *S* and *C* respectively. We call the new training dataset *S'* and *C'*. We take *S'*, *C'* and t_s as the input of the follow-up case, and the output should still be l_i.

MR-3.2: Additional training sample

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we duplicate all samples in *S* with label l_i, as well as their associated labels in *C*. The output of the follow-up test case should still be l_i.

MR-4.1: Addition of classes by duplicating samples

For the source input, suppose we get the result $c_t = l_i$for the test case t_s. In the follow-up input, we duplicate all samples in *S* and *C* that do not have label l_i and concatenate an arbitrary symbol "*" to the class labels of the duplicated samples. That is, if the original training sample set *S* is associated with class labels <*A, B, C*> and l_i is *A*, the set of classes in *S* in the follow-up input could be <*A, B, C, B**, C**>. The output of the follow-up test case should still be l_i. Another derivative of this metamorphic relation is that duplicating all samples from any number of classes which do not have label l_i should not change the result of the output of the follow- up test case.

MR-4.2: Addition of classes by re-labeling samples

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we re-label some of the samples in S with labels other than l_i, through concatenating an arbitrary symbol "*" to their associated class labels in C. That is, if the original training set S is associated with class labels <*A*, *B*, *B*, *B*, *C*, *C*, *C*> and c_0 is *A*, the set of classes in S in the follow-up input may become <*A*, *B*, *B*, *B**, *C*, *C**, *C**>. The output of the follow-up test case should still be l_i.

MR-5.1: Removal of classes

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we remove one entire class of samples in S of which the label is not l_i. That is, if the original training sample set S is associated with class labels <*A*, *A*, *B*, *B*, *C*, *C*> and l_i is *A*, the set of classes in S in the follow-up input may become <*A*, *A*, *B*, *B*>. The output of the follow-up test case should still be l_i.

MR-5.2: Removal of samples

For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we remove part of some of the samples in S and C of which the label is not l_i. That is, if the original training set S is associated with class labels <*A*, *A*, *B*, *B*, *C*, *C*> and l_i is *A*, the set of classes in S in the follow-up input may become <*A*, *A*, *B*, *C*>. The output of the follow-up test case should still be l_i.

Supervised machine learning classifiers consist of two phases. The first phase (called the training phase) analyzes the training data; the result of this analysis is a model that attempts to make generalizations about how the attributes relate to the label. In the second phase (called the testing phase), the model is applied to another, previously unseen data set (the testing data) where the labels are unknown.

In the Naïve Bayes Classifier, for a training sample set S, suppose each sample has m attributes, <att_0, att_1, ..., att_{m-1}>, and there are n classes in S, {l_0, l_1, ..., l_{n-1}}. The value of the test case t_s is <a_0, a_1, ..., a_{m-1}>. The label of t_s is called l_{ts}, and is to be predicted by NBC.

NBC computes the probability of l_{ts} to be of class l_k, when each attribute value of t_s is <a_0, a_1, ..., a_{m-1}>. NBC assumes that attributes are conditionally independent with one another given the class label, therefore we have the equation:

$$P(l_{ts} = l_k \mid a_0 a_1 \ldots a_{m-1}) = \frac{P(l_k) \prod_j P(a_j \mid l_{ts} = l_k)}{\sum_i P(l_i) \prod_j P(a_j \mid l_{ts} = l_i)}$$

After computing the probability for each $l_i \in \{l_0, l_1, ..., l_{n-1}\}$, NBC assigns the label l_k with the highest probability, as the label of test case t_s.

Generally NBC uses a normal distribution to compute $P(a_j \mid l_{ts} = l_k)$. Thus NBC trains the training sample set to establish a distribution function for each element att_j of vector $<att_0, att_1, ..., att_{m-1}>$ in each $l_i \in \{l_0, l_1, ..., l_{n-1}\}$, that is, for all samples with label $l_i \in \{l_0, l_1, ..., l_{n-1}\}$, it calculates the mean value μ and mean square deviation σ of att_j in all samples with l_i. Then a probability density function is constructed for a normal distribution with μ and σ.

For test case t_s with m attribute values $<a_0, a_1, ..., a_{m-1}>$, NBC computes the probability of $P(a_j \mid l_{ts} = l_k)$using a small interval δ to calculate the integral area. With the above formulae NBC can then compute the probability of l_{ts} belonging to each l_i and choose the label with the highest probability as the classification of t_s.

IV. EXPERIMENTAL RESULTS

At first, 500 English words were randomly selected from an English dictionary. These 500 words were regarded as the original queries, while the follow-up queries were the original query to which was added the domain names of the first five results of the original query. The experiment was done in google search engine.

The precision and recall are the performance metrics used to evaluate our proposed method. The performance is evaluated using the keyword "Issack Newton".

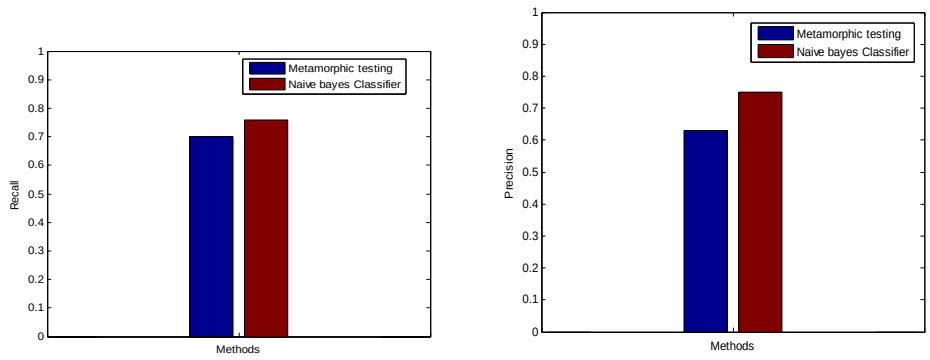

Figure 2: Precision and recall comparison

Figure 2 shows the precision and recall comparison of metamorphic testing and modified metamorphic testing with supervised machine learning algorithm and proves that, the modification done the reasonable improvement in the precision and the recall values.

V. CONCLUSION

In this paper, the machine learning algorithm is proposed to automatically detect failures and anomalies using MRs can also provide hints for the construction of run-time self-correction mechanisms. Our contribution is a set of metamorphic relations for classification algorithms, as well as a technique that uses these relations to enable scientists to easily test and validate the machine learning components of their software; this technique is also applicable to problem-specific domains as well.

REFERENCES

[1] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. A comprehensive survey of trends in oracles for software testing. Technical Report CS-13-01, University of Sheeld, 2013.

[2] E. J. Weyuker. On testing non-testable programs. The Computer Journal, 25(4):465{470, 1982.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," Software Engineering, IEEE Transactions on, vol. 41, no. 5, pp. 507–525, May 2015.

[4] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: a new approach for generating next test cases. Technical report, The Hong Kong University of Science and Technology, Hong Kong, China, 1998.

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.

[6] W. J. Cody, Software Manual for the Elementary Functions. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1980.

[7] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," Journal of Computer and System Sciences, vol. 47, no. 3, pp. 549 – 595, 1993. [Online]. Available: http://www.sciencedirect.com/science/ article/pii/002200009390044W

[8] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," IEEE Transactions on Computers, vol. 37, no. 4, pp. 418–425, Apr. 1988. [Online]. Available:http://dx.doi.org/10.1109/12.2185

[9] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in Eleventh Annual International Workshop on Software Technology and Engineering Practice, 2003., Sept 2003, pp. 94–100.

[10] T. H. Tse, "Research directions on model-based metamorphic testing and verification," in 29th Annual International Computer Software and Applications Conference, 2005. COMPSAC 2005., vol. 1,July 2005, pp. 332 Vol. 2–.

[11] T. Y. Chen, "Metamorphic testing: A simple approach to alleviate the oracle problem," in Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE), 2010, June 2010, pp. 1–2.

[12] W. K. Chan and T. H. Tse, "Oracles are hardly attain'd, and hardly understood: Confessions of software testing researchers," in 13th International Conference on Quality Software (QSIC), 2013, July 2013, pp. 245–252.

[13] T. Y. Chen, "Metamorphic testing: A simple method for alleviating the test oracle problem," in Proceedings of the 10th International Workshop on Automation of Software Test, ser. AST '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 53–54. [Online].Available: http://dl.acm.org/citation.cfm?id=2819261.2819278

[14] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "Metamorphic testing: Applications and integration with other methods: Tutorial synopsis," in 12th International Conference on Quality Software (QSIC), 2012, Aug 2012, pp. 285–288.

[15] Z. Hui and S. Huang, "Achievements and challenges of metamorphic testing," in ourth World Congress on Software Engineering (WCSE), 2013, Dec 2013, pp. 73–77.

[16] U. Kanewala and J. M. Bieman, "Techniques for testing scientific programs without an oracle," in 5th International Workshop on Software Engineering for Computational Science and Engineering (SECSE), 2013, May 2013, pp. 48–57.

[17] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang, "Survey of metamorphic testing," Journal of Frontiers of Computer Science and Technology, vol. 3, no. 2, pp. 130–143, 2009.