MRpredT: Using Text Mining for Metamorphic Relation Prediction

Karishma Rahman
karishma.rahman@student.montana.edu
Montana State University
Bozeman, MT

Indika Kahanda
indika.kahanda@montana.edu
Montana State University
Bozeman, MT

Upulee Kanewala
upulee.kanewala@montana.edu
Montana State University
Bozeman, MT

ABSTRACT

Metamorphic relations (MRs) are an essential component of metamorphic testing (MT) that highly affects its fault detection effectiveness. MRs are usually identified with the help of a domain expert, which is a labor-intensive task. In this work, we explore the feasibility of a text classification-based machine learning approach to predict MRs using their program documentation as the sole input. We compare our method to our previously developed graph kernel-based machine learning approach and demonstrate that textual features extracted from program documentation are highly effective for predicting metamorphic relations for matrix calculation programs.

KEYWORDS

Metamorphic relations, Metamorphic testing, Text classification

ACM Reference Format:
Karishma Rahman, Indika Kahanda, and Upulee Kanewala. 2020. MRpredT: Using Text Mining for Metamorphic Relation Prediction. In *Proceedings of May 23–29, 2020 (ICSEW'20).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387940.3392250

1 INTRODUCTION

Since its first introduction in 1998, Metamorphic testing (MT) has evolved as an effective testing technique for testing programs that face the *oracle problem.* Further, MT has led to revealing previously unknown faults in diverse applications such as compilers [14], search engines [17], and Google map navigation [16]. At the center of MT are metamorphic relations (MRs), which are necessary properties of the program under test (PUT) and specify relationships between multiple inputs and their corresponding outputs.

For example, consider a program that accepts a list of real numbers and sorts them in ascending order. Imagine that this program is provided with a list of 50,000 numbers. How do you determine the output produced by the sorting program is correct? Even though it is hard to determine whether the produced output is correct in this instance, we can develop relationships between the outputs of some related inputs. For example, from our knowledge about sorting, we know that if we permute the original input and supply it to the sorting program, it should produce the same ordering as before. This property of sorting can be used as an MR to test the sorting program. In MT, the original list of inputs is known as the *source test case*, and the list obtained by randomly permuting the original list is known as the *follow-up test case.* After executing the source and follow-up test cases on the sorting program, the produced outputs are checked against the MR; in this case, to verify the produced outputs have the same ordering. If they are not, there is a fault in the sorting program.

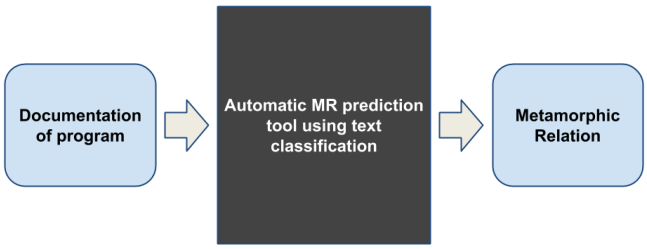

Figure 1: Overview of the approach.

Often, MR identification is performed manually and requires interaction with domain experts, especially when testing scientific programs. Therefore, this process can be a labor-intensive task that is often error prone [6]. Thus, developing automated methods for identifying MRs can improve the efficiency and effectiveness of MT. To this end, in this paper, we investigate utilizing text mining to extract information from various documentation sources associated with a program and use machine learning techniques to predict MRs for unseen programs automatically.

Text mining techniques are often used to explore and analyze a vast amount of unstructured text data and identify patterns informative for a given task [2]. This study presents the development of a method for automatically predicting MRs for a given program function through categorizing text data associated with those programs (see Figure 1). In particular, the text data used are collected from the Javadocs of the Java programs. Javadoc contains the application programming interface documentation from the Java source code [15]. We hypothesis that due to the close relationship of the Javadocs contents with the program functionality, textual features extracted from the documentation is highly informative for predicting MRs associated with those programs. Therefore, this study

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSEW'20, Seoul, Republic of Korea,
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3392250

aims to demonstrate the effectiveness of using text mined features generated from program documentation to predict MRs.

This paper is organized as follows: Section 2 provides the background knowledge of the methods and techniques used for the experiment. Section 3 explains the methodology of the Text Mining based machine learning approach to identify MRs for a function. Section 4 discusses the results obtained by using the approach introduced in the study. Section 5 concludes the paper by pointing out future works.

2 RELATED WORK

Several automated methods have been developed for MR prediction in previous work. Kanewala et al. [6] introduced *MRpred*, a method that uses a graph kernel-based machine learning approach to predict metamorphic relations for programs that perform numerical calculations. The initial step of this approach is to transform a function into its graph representation modeling the control flow and the data dependency information of the program [6]. Then they use a graph kernel function to compute a similarity score between two programs represented in the graph representation mentioned above. The computed graph kernel values are then provided to a support vector machine (SVM) classification algorithm to create the predictive model, which is used for binary classification [6].

They used a code corpus containing 100 functions that take numerical inputs and produce numerical outputs, to evaluate the effectiveness of their proposed methods [6]. Six MRs are identified; Permutative, Additive, Multiplicative, Invertive, Inclusive, and Exclusive. Their results show that graph kernels improve the prediction accuracy of MRs when compared with explicitly extracted features. Their results also show that control flow information of a program is more effective than data dependency information for predicting MRs, but sometimes, both of them can contribute to increasing the accuracy.

In one of our previous studies [11], we applied MRpred for predicting three high-level categories of MRs (i. e. , Permutative, Additive, and Multiplicative) for matrix-based programs. Our results show that the random walk kernel can effectively predict these MRs [11]. This study motivated us to use matrix-based programs as the subject programs for the experiments described in this paper.

Further, in several past studies, researches have used different text analysis techniques. They have investigated many ways to improve the software testing process using text analysis techniques. In [13], the authors have conducted a mapping study where they listed the activities of software testing, which are improved by using text analysis techniques. They are static black-box test-case prioritization, robustness testing, test case generation, and test case prioritization [13].

Our work described in this paper also focuses on the automated identification of MRs for programs but using a different source of data, which is the various documentation sources associated with the program. In particular, we use text mining techniques to classify the Javadoc associated with a program and use machine learning to predict MRs for previously unseen programs. To the best of our knowledge, this is the first such study.

3 METHODOLOGY

In this work, we model the task of predicting MRs for a given program as a supervised classification problem. In particular, we treat each MR independently and apply a separate binary classifier for each MR. Binary class labels correspond to the existence of a specific MR, and the input feature representing each program is generated solely using its documentation.

The following subsections describe the text corpus and the MRs used in this study. The overview of the text classification approach for predicting MRs of the dataset is also discussed here. Moreover, the details of the experimental setup are mentioned, as well.

3.1 Data

3.1.1 Text Corpus. A total of 93 program's Javadocs, which handle matrix operations, are used for this study. They are collected from Apache Commons Math Library3[1], la4j (Linear Algebra for Java)[2], and JAMA (Java Matrix package)[3], which are open-source projects. These programs perform a variety of operations on matrices such as addition, multiplication, subtraction, and searching. Figure 2 shows the raw data example of the Javadocs of a program.

3.1.2 Metamorphic Relations. For this study, we identified ten MRs manually that are generally applicable to matrix calculations. These MRs are shown in Table 1 with the change made to the input and their expected output change. These MRs are used as class labels for the supervised classification model.

3.2 Models

Javadoc contains the information of a program's operation, inputs (parameters) and outputs (returns) that are directly related to the MRs satisfied by a given program. Our method solely uses Javadoc information to predict MRs for Java programs. Figure 3 shows an overview of our method. We implemented our models using the scikit-learn[4] python library.

The first step of this method is to extract the Javadoc documentation from the source code using Java Parser[5] and pre-processes them using the lemmatization [10] technique. Then, text feature extraction methods are applied to those pre-processed Javadocs to obtain the feature vectors. Bag of words (BoW) model [10], is used as the feature representation. To make the text learnable by machines, they must be converted to numerical vectors, and the BoW model is a standard technique to obtain such a structure [10]. In this representation, the features (f_i, see Figure 3) are the tokens extracted from the source and feature-values are the frequency of their occurrence within each program documentation. These feature vectors of the programs are then supplied into the machine learning classification algorithm with their associated MR labels. Here, MR labels are identified manually for all the programs, where the label is '1' if an MR is satisfied by the program, and '0' otherwise.

We used two popular machine learning algorithms, Naive Bayes [7] and Support Vector Machines (SVMs) [3], as the underlying classification algorithms. In many other domains, these two algorithms

[1] http://commons.apache.org/proper/commons-math/javadocs/api-3.6/
[2] http://la4j.org/apidocs/
[3] https://math.nist.gov/javanumerics/jama/doc/
[4] https://scikit-learn.org/stable/
[5] https://javaparser.org/

```
/**
 * Adds given {@code value} (v) to every element of this matrix (A).
 *
 * @param value the right hand value for addition
 *
 * @return A + v
 */
public Matrix add(double value) {
    MatrixIterator it = iterator();
    Matrix result = blank();

    while (it.hasNext()) {
        double x = it.next();
        int i = it.rowIndex();
        int j = it.columnIndex();
        result.set(i, j, x + value);
    }

    return result;
}
```

Figure 2: Raw data of a program which adds a value to the matrix elements.

Table 1: The Metamorphic Relations used in the study

Metamorphic Relation	Change made to the input	Expected change in the output
MR1: Permutation of row	Change the row order of the a matrix	Output size will remain same
MR2: Permutation of column	Change the column order of the matrix	Output size will remain same
MR3: Permutation of elements	Change the element position of the matrix	Output size will remain same
MR4: Matrix addition	Adding another matrix to the input matrix	Elements value will increase or remain same.
MR5: Scalar addition	Adding a value to the matrix	Element values will increase or remain same
MR6: Addition with the Identity matrix	Adding Identity matrix to the input matrix	Only diagonal element value will change or output will increase
MR7: Matrix multiplication	Multiplying another matrix to the input matrix	Elements value will increase.
MR8: Scalar multiplication	Multiplying a value to the matrix	Elements value will increase or remain same.
MR9: Element by element multiplication with the Identity matrix	Multiplying Identity matrix element by element to the input matrix	Diagonal element will be same or output will remain same
MR10: Transpose	Transpose the input matrix	Diagonal element will be same or output will remain same

are historically found to be very effective for text classification tasks [1, 5, 9, 12]. We implement both using the default parameters available in scikit-learn. The *trained* predictive model is then used to predict the labels (i.e. MR) for the unseen program by supplying the corresponding feature vector generated solely from its Javadoc documentation text, as shown in Figure 3.

3.3 Experimental Setup

In this experiment, 10-times stratified 10-fold cross-validation is used to evaluate the effectiveness of the models. It is a cross-validation technique where each fold contains roughly the same percentage of data belonging to each class compared to the full dataset [8]. Also, in the case of prediction problems, the mean response value is maintained relatively equal in all the folds [8]. This process is repeated ten times to negate any selection-bias. This approach is not only useful for the fair assessment of the models but also helps to alleviate over-fitting [8]. We compare our models to MRpred [6].

The evaluation measure used in this study is *Area Under the receiver operating characteristic Curve (AUC)*. AUC measures the probability that a randomly chosen positive example (i.e. a program labeled with a certain MR) will be ranked higher by the predictive model than a randomly chosen negative example [4]. AUC takes values ranged in [0, 1] where higher values indicate better performance, but a value of 0.5 is equivalent to a random classifier.

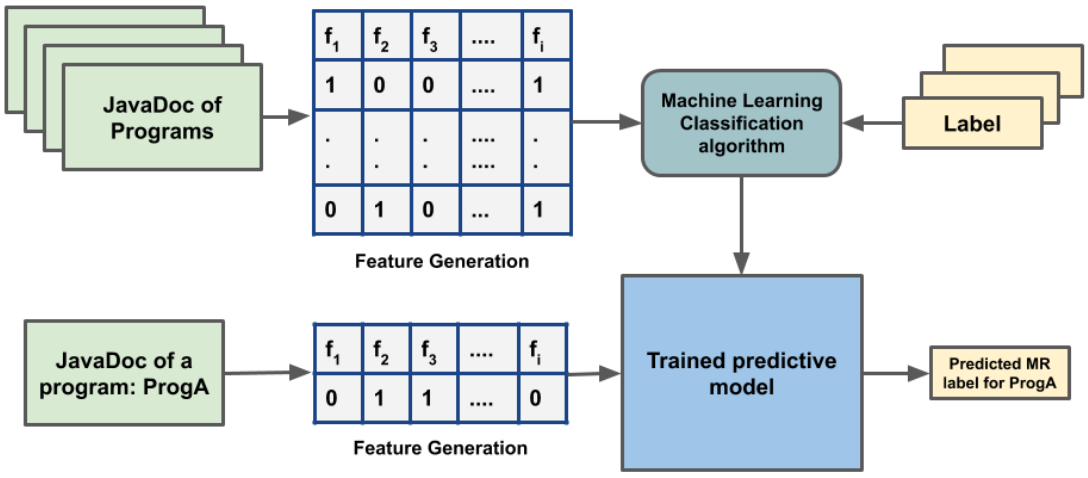

Figure 3: Text classification approach for predicting MRs for Java programs.

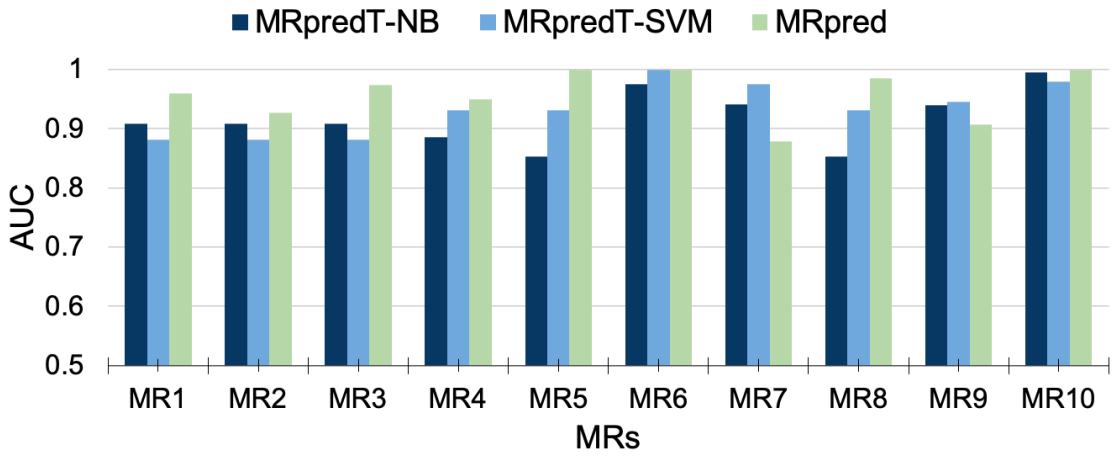

Figure 4: AUC scores of Naive Bayes and SVM models using text classification (MRpredT-NB and MRpredT-SVM), and for the SVM model using Random Walk Graph Kernel (MRpred). AUC: Area Under the Receiver Operatng Characteristic Curve. MR: Metamorphic Relations.

AUC is used as the evaluation metric in this experiment as it does not depend on the discrimination threshold of the classifier and is considered a better measure for analyzing learning algorithms (compared to metrics such as accuracy) [4].

4 RESULTS AND DISCUSSION

In Figure 4, the AUC scores of our metamorphic relation prediction models (MRpredT-NB: Naive Bayes, and MRpredT-SVM: SVM classification model) are compared to MRpred (i.e., classification model using random walk graph kernel with SVMs). For 4/10 MRs (i.e., MR1, MR2, MR3, and MR10), MRpredT-NB outperforms its SVM counterpart (i.e. MRpredT-SVM).

For 6/10 MRs, MRpredT-SVM model performs better than MRpredT-NB. Among the above 6 MRs, the highest possible AUC score (1.0) could be observed when predicting the addition with Identity matrix (MR6) MR. The other MRs also reported AUC values higher than 0.87, indicating that our text-based approach generated effective predictive models when using SVM for all the MRs. However,

the prediction scores achieved by the NB is also promising. The highest score is 0.99 when predicting transpose (MR10) MR.

Furthermore, for 2/10 MRs (MR7 and MR9), one of the two MRpredT models outperform MRpred providing ample evidence for the effectiveness of using text data for MR prediction. Interestingly, for MR7 and MR9, both MRpredT models perform better than MRpred, warranting further investigation. One the other hand, MRpred is still the clear winner for MR1, MR3, MR5, and MR8.

5 CONCLUSION AND FUTURE WORK

The metamorphic testing technique is beneficial to test programs that do not have a test oracle. The effectiveness of this technique highly depends on the set of MRs used for testing. But the identification process of MRs is mostly performed manually. This study proposes to use a text classification method to predict MRs for functions that perform matrix calculations using their documentation. The results show that for these types of programs, the text classification-based machine learning approach can be effective in predicting MRs, especially when using the SVM model. However, there are many avenues for future investigation, as described below.

First, we would like to investigate the effectiveness of heterogeneous features by combining MRpredT's text features with the program features used in MRpred. In addition, text features extracted from the source code itself could be a fruitful Addition. Another aspect worth investigating is exploring the text features (i.e., tokens) identified as the most effective for each MR by our MRpredT models. This may provide valuable information for going beyond the standard BoW model and developing domain-specif features.

With the recent popularity of deep learning, we intend to utilize recurrent neural networks as the underlying machine learning model, which will also negate the need for hand-engineering features. This will also provide the opportunity for exploring the effectiveness of more sophisticated features based on semantic similarity (i.e., word and BERT embeddings). However, this will require obtaining much larger datasets as these models are known to be data-hungry. Considering the highly resource-consuming nature of the manual labeling process, a semi-supervised learning technique that allows the use of unlabeled data may be a viable alternative.

REFERENCES

[1] Saleh Alsaleem. 2011. Automated Arabic Text Categorization Using SVM and NB. *Int. Arab J. e-Technol.* 2 (01 2011), 124–128.

[2] Michael W. Berry. 2003. *Survey of Text Mining.* Springer-Verlag, Berlin, Heidelberg.

[3] Theodoros Evgeniou and Massimiliano Pontil. 2001. Support Vector Machines: Theory and Applications, Vol. 2049. 249–257. https://doi.org/10.1007/3-540-44673-7_12

[4] David J Hand and Christoforos Anagnostopoulos. 2013. When is the area under the receiver operating characteristic curve an appropriate measure of classifier performance? *Pattern Recognition Letters* 34, 5 (2013), 492–495.

[5] S. Hassan, M. Rafi, and M. S. Shaikh. 2011. Comparing SVM and naïve Bayes classifiers for text categorization with Wikitology as knowledge enrichment. In *2011 IEEE 14th International Multitopic Conference*. 31–34. https://doi.org/10.1109/INMIC.2011.6151495

[6] Upulee Kanewala, James M. Bieman, and Asa Ben-Hur. 2016. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software Testing, Verification and Reliability* 26, 3 (2016), 245–269. https://doi.org/10.1002/stvr.1594 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1594

[7] Pouria Kaviani and Sunita Dhotre. 2017. Short Survey on Naive Bayes Algorithm. *International Journal of Advance Research in Computer Science and Management* 04 (11 2017).

[8] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence-Volume 2*. 1137–1143.

[9] Xiaoli Li and Bing Liu. 2003. Learning to Classify Texts Using Positive and Unlabeled Data. *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03): 2003; Acapulco, Mexico*, 587–594.

[10] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge university press.

[11] Karishma Rahman and Upulee Kanewala. 2018. Predicting Metamorphic Relations for Matrix Calculation Programs. In *Proceedings of the 3rd International Workshop on Metamorphic Testing (MET '18)*. ACM, New York, NY, USA, 10–13. https://doi.org/10.1145/3193977.3193983

[12] Monica Rogati and Yiming Yang. 2002. High-Performing Feature Selection for Text Classification. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM '02)*. Association for Computing Machinery, New York, NY, USA, 659–661. https://doi.org/10.1145/584792.584911

[13] Faiz Ali Shah and Dietmar Pfahl. 2016. Evaluating and Improving Software Quality Using Text Analysis Techniques - A Mapping Study. (2016).

[14] Qiuming Tao, Wei Wu, Chen Zhao, and Wuwei Shen. 2010. An automatic testing approach for compiler based on metamorphic testing technique. In *2010 Asia Pacific Software Engineering Conference*. IEEE, 270–279.

[15] Wikipedia contributors. 2019. Javadoc — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Javadoc&oldid=914403859 [Online; accessed 10-September-2019].

[16] Zhi Quan Zhou, Liqun Sun, Tsong Yueh Chen, and Dave Towey. 2018. Metamorphic relations for enhancing system understanding and use. *IEEE Transactions on Software Engineering* (2018).

[17] Zhi Quan Zhou, Shaowen Xiang, and Tsong Yueh Chen. 2015. Metamorphic testing for software quality assessment: A study of search engines. *IEEE Transactions on Software Engineering* 42, 3 (2015), 264–284.