A Machine Learning Based Approach for Software Test Case Selection

Victor Cheruiyot* and Baidya Nath Saha
Department of Mathematical and Physical Sciences, Concordia University of Edmonton, Alberta, T5B 4E4, Canada

* Corresponding author

doi:https://doi.org/10.21467/proceedings.115.25

ABSTRACT

Testing is conducted after developing each software to detect the defects which are then removed. However, it is very difficult task to test a non-trivial software completely. Hence, it's important to test the software with important test cases. In this research, we developed a machine learning based software test case selection strategy for regression testing. To develop the method, we first clean and preprocess the data. Then we convet the categorical data to its numerical value. The we implement a natural language processing to calculate bag of features for text feature such as testcase title. We evaluate different machine learning models for test case selection. Experimental results demonstrate that machine learning based models can aovid manual labour of the domain experts for test case selection.

Keywords: Software testing, Regression testing, Machine learning algorithms

I. Introduction

Software testing is a quality control activity which concentrates on detecting defects and then they are removed. After completion of coding software products are subjected to testing with the help of different test cases. These tests are essential and necessary to assess the effectiveness of the software. However, it is impossible to completely test any nontrivial module or system becuase it suffers from both theoretical and practical perspectives. Theoretically it suffers from halting problem: it's impossible to write a program that tests whether every program halts in a finite amount of time. Practically, executing all test cases involves enormous time and cost. Hence, it's crucial to test a subset of test cases which are important from user's perspective based on the frequency of usage, criticality, and probability of failure. This research work is limited to develop a machine learning based test case selection strateg for regression testing. Regression testing retests software that has been changed or extended by new features during software development.

In this research we carried out the following activities to develop a machine learning based test case selection strategy. First, we clean the data by dropping irrelevant features. Then we preprocess the data by encoding categorical features into its numerical values. After that we compute the bag of words for text data such as test case title, which identify the important functionality for test case selection. All these features are fed into machine learning based classifiers, such as logistic regression, Gaussian and multinomial naïve Bayes. Experimental results demonstrate that machine learning based algorithm can successfully detect important test cases and relieve the domain expert from extensive manual labour for test case selection.

II. Literature Review

Xu *et al.* [1] developed a fuzzy expert systems which have the ability to emulate fuzzy human reasoning and judgment processes and their system could detect critical test cases for system test by correlating knowledge represented by customer profile, analysis of past test case results, system failure rate, and change in system architecture. They developed this fuzzy expert system for a large telecommunications system and obtained satisfactory results.

Proceedings DOI: 10.21467/proceedings.115; Series: AIJR Proceedings; ISSN: 2582-3922; ISBN: 978-81-947843-5-7

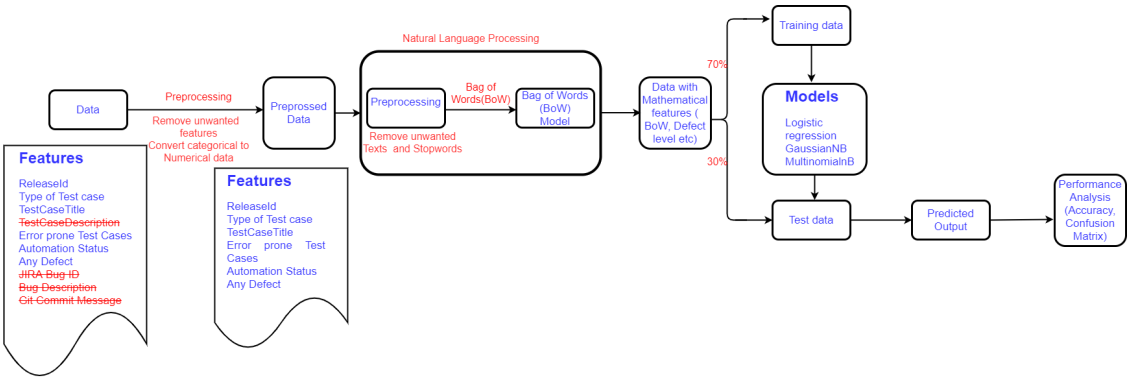

Fig. 1: Flowchart illustrating machine learning based approach for software test case selection.

In another research work, Romano *et al.* [2] developed "SPIRITuS: a Simple Information Retrieval regressIon Test Selection approach" which made a tradeoff between the number of selected test cases from the original test suite and fault detection effectiveness. SPIRITuS was used for test case selection from a large experiment on 389 faulty versions of 14 open-source programs implemented in Java. In this experiment, SPIRITuS was able to select significantly less number of test cases from other approaches at the expense of a slight reduction in fault detection capability.

III. Methodology

Proposed Methodology is illustrated in Figure 1. The follwoing steps are taken to develop a machine learning based software test case selection:

A. Dataset generation. Dataset for test set selection include i) Unique Identifier of Records, ii) Release Identification number, iii) Type of Test Case: 'Sanity' test cases are executed for Sanity of microservice and 'API/Functionality' test cases are executed for core functionality of microservice, iv) TestCaseTitle: Title or summary of test case, v) TestCaseDescription, vi) Error Prone Test Cases: test cases which are covering high error prone area which must be executed in every release, vii) Automation Status: whether test case is automated or not, viii) Any Defect: if there is any defect in the release, ix) Bug ID, x) Bug Description, xi) GIT Commit Message: for a particular release, if there are any commits in GIT, and xii) Target: binary classification of test case selection.

B. Data cleaning. After careful observation, based on the useful information available in the literature and domain experts' knowledges, we drop the following features from our experiments: Unique Identifier of Records, TestCaseDescription, Bug ID, Bug Description, and GIT Commit Message. These features have insignificant contribution to test case selection.

C. Data preprocessing. After dropping the irrelevant features, the dataset consists of one text feature, TestCaseTitle and five categorical features, such as Release Identification number, Type of Test Case, Error Prone Test Cases, Automation Status, and Any Defect, all of which are related to selection of test cases i.e. 'Target' variable. This list of features will be utilized for training classifier models. However, all categorical variables will be converted to their numerical values and text columns into sparse matrix of numerical features.

D. Natural Language Processing (NLP). We compute Bag of Words features from text data to train classifiers. The following NLP based preprocessing tasks are carried out to convert text data of TestCaseTitle to numerical features compatible with classifier models:

1) *Remove unwanted words:* Remove irrelevant characters and words such as special characters and numbers to get clean text for further processing.

Predicted			
		Positive	Negative
Observed	Postive	TP (of TPs)	FN (of FNs)
	Negative	FP (of FPs)	TN (of TNs)

TABLE II: Confusion Matrix for binary classification

2) *Uppercase to lowercase transformation:* Transform all uppercase letters to lowercase because upper and lower case letters have different ASCII codes.
3) *Remove stopwords:* Stopwords are usually the most common words in a language and are irrelevant in predicting the response variables.
4) *Stemming words:* Stemming is the process of reducing words to their stem, base or root form. We use stemming to reduce dimensions of Bag of Words features.

Bag of Words (BoW) describes the occurrence of words within a document which involves a vocabulary of known words and a measure of the presence of known words. BoW ignores the order or structure of words in the document and the model concerns only whether the known words occur in the document. We implemented BoW using python CountVectorizer function available in scikit-learn library.Countvectorizer converts a collection of text documents to a matrix of token counts.

E. Machine Learning Classifiers. We build three machine learning classifier models such as, logistic regression model, Gaussian Naive Bayes, and Multinomial Naive Bayes.

IV. Experimental Results and Discussions

We divide the data into two parts: training and test. 70% data are used for training and the remaining 30% data are used for testing the models.

Classifier	Train accuracy	Test accuracy	F1 score
Regression	90.2	84.2	68.7
GaussianNB	80.4	78.2	70.1
MuitinomialNB	87.6	84.2	70.1

TABLE I: Train and test accracy and F1 score for different classifers

Table I demonstrates train and test accuracy and F1 score for different classifiers. Accuracy is calculated as the number of all correct predictions divided by the total number of the dataset. The best accuracy is 1.0, whereas the worst is 0.0. F1 score is defined as, $\textbf{F1 score} = \frac{\textbf{2*Precision*Recall}}{\textbf{Precision+Recall}}$, $\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP+FP}}$, and $\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP+FN}}$.

In addition, we measure the performance of the classifers in terms of confusion matrix, which is formed from the four outcomes produced as a result of binary classification. A binary classifier predicts all data instances of a test dataset as either positive or negative. This classification (or prediction) produces four outcomes – true positive, true negative, false positive and false negative which are defined as follows.

1) True positive (TP): correct positive prediction
2) False positive (FP): incorrect positive prediction
3) True negative (TN): correct negative prediction
4) False negative (FN): incorrect negative prediction

A confusion matrix for binary classification as demonstrated in Table II is a two by two table constructed by counting of the number of the four outcomes of a binary classifier which are denoted as TP, FP, TN, and FN.

Proceedings DOI: 10.21467/proceedings.115
ISBN: 978-81-947843-5-7

Series: AIJR Proceedings
ISSN: 2582-3922

Classification results for this experiment is represented by confusion matrix which is illustrated in Table III. Results demonstrate that Machine Learning based approach can relieve the domain experts from manual labour for test case selection.

Predicted			
		Positive	Negative
Observed	Postive	89,70,71	10,29,29
	Negative	11,0,0	23,34,34

TABLE III: Confusion Matrix for different classifers

V. Conclusion

Regression testing is conducted after updating any software components. This research demonstrates that machine learning-based approach can reduce the bias and manual labour of domain expert for software regression testing. Prediction performance could be improved if large amount ofd training of data can be increased by adding more releases data. In future, we would investigate the feature selection strategy for natural language processing to optimize the whole software test case selection procedure.

References

[1] Z. Xu, K. Gao, T. M. Khoshgoftaar, and N. Seliya, "System regression test planning with a fuzzy expert system," *Information Sciences*, vol. 259, pp. 532–543, 2014.

[2] S. Romano, G. Scanniello, G. Antoniol, and A. Marchetto, "Spiritus: A simple information retrieval regression test selection approach," *Information and Software Technology*, vol. 99, pp. 62–80, 2018.