Communication Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 209–214
DOI: 10.15439/2023F4426
ISSN 2300-5963 ACSIS, Vol. 37

A machine learning approach for automatic testing

Felix Petcusin
Computer Science and Information Technology,
University of Craiova
Craiova, Romania
felix.petcusin@edu.ucv.ro

Cosmin Stoica Spahiu
Computer Science and Information Technology,
University of Craiova
Craiova, Romania
cosmin.spahiu@edu.ucv.ro

Liana Stanescu
Computer Science and Information Technology,
University of Craiova
Craiova, Romania
liana.stanescu@edu.ucv.ro.

***Abstract*—Systems' complexity has exponentially increased in recent years. Security and safety have become crucial in critical systems, and end-users now demand clear traceability to ensure protection against errors and external attacks. Meeting this requirement necessitates significant effort in testing. Although automated test sequences can handle a large portion of testing, it is crucial to identify as many errors as possible within the initial hours or days of the testing period. This paper introduces a machine learning-based solution that utilizes learned patterns to determine the test order. It analyzes which functionalities are more susceptible to errors and recursively generates the test sequence to be executed at each step.**

***Index Terms*—automated tests, machine learning, tests prioritization**

I. Introduction

TODAY, people's safety, entertainment, business decisions, and lives rely heavily on computers and various software tools. Therefore, it is crucial to ensure their proper functioning. The most effective approach to achieve this is by testing the products before they are released on the market. Software testing has become an essential component of any software project being the only way to guarantee high-quality applications that meets customer requirements, almost defects-free.

The complexity of software systems is increasing rapidly, with approximately one third of development costs currently being spent on electric/electronic development, a figure that continues to rise. Multiple variants of components are developed and tested through a series of prototyping phases, often with different schedules. Consequently, the level of complexity in specification activities has surpassed what can be effectively handled by traditional testing methods systems that are reliant on human input [1].

Most system development projects include a separate stage devoted to requirements specification, another stage for development and another for testing. All functional requirements must be validated and verified against the implementation. Testing is crucial especially for safety related industry, and software requirements are usually categorized as either "Integration Test" or "Software Test" requirements based on the content of requirements information. Specific testing techniques and methods are employed depending on this classification.

It is crucial to conduct testing before deploying software to end users to identify and fix errors in a timely manner and ensure the software is functional as intended. While testing aims to minimize errors, it is almost impossible to achieve a software product that is 100% error-free. Product quality is dependent on various parameters, such as performance, reliability, correctness, testability, and reusability, which can only be ensured through testing. Although testing can be time-consuming and expensive, it is better to invest in it early rather than after customer issues have arisen [2]. Balancing project costs and benefits/quality is important to make a development business case feasible. The key elements that indicate how much testing is enough are provided test coverage, time, and cost.

Manual testing was the traditional process, but it was time-consuming and limited by the available timeframe [3]. Hence, most testing activities have shifted to automated software testing execution with different tools that are more efficient, reduce time and cost, and increase test coverage. There is a wide range of testing tools available on the market that can be customized based on the complexity of the projects [4].

Recently, the authors have proposed a new paradigm in testing by extending classic tools to incorporate artificial intelligence (AI). This approach offers several benefits, such as faster and easier test creation, simpler test execution and analysis, and reduced test maintenance [5]. AI has transformed the testing approach by simplifying test documentation steps, decreasing maintenance effort, and providing new ways to interpret the results.

Thematic track: Software Engineering for Cyber-Physical Systems

The aim of the current work paper is to present the results obtained by a prototype system that uses Machine Learning algorithms for test-cases prioritization. The input is represented by an existing database of testcases together with the results obtained from past releases, in an Automotive project. The system will analyze existing testcases (executed in past releases), together with the newly created ones in order to classify the ones with higher risk of failure. They will be executed first in the list.

The remainder of this article is organized as following: chapter 2 presents the motivation leading to the present work and summarizes the prior research in this domain, chapter 3 presents an original proof-of-concept application which was developed and the results obtained during experiments and in the final chapter there are presented the conclusion and future directions.

II. Related Work

In recent years, several machine learning (ML) algorithms have been used to solve a few particularly difficult problems in the field of automate classifications for systems. Two aspects are considered: requirements and tests classification.

An example of such a problem is the identification and classification of non-functional requirements in requirements documents. ML-based solutions have shown promising results that go beyond those of traditional Natural Language Processing (NLP) approaches.

In [6], the authors work on automatic classification of requirements by performing a systematic review of 24 ML-based solutions for identifying and classifying NFRs. The authors selected 24 research papers that use 16 different ML algorithms. These algorithms can be divided into three categories: supervised learning (7 algorithms), unsupervised learning (4 algorithms), and semi-supervised learning (5 algorithms). The supervised learning algorithms were used in 17 papers (71%), with SVM being the most popular algorithm in 11 studies (45.8%). The authors come to the following conclusions: ML-based solutions have potential in classifying and identifying NFRs; collaboration between RE and ML researchers is needed to address open challenges in the development of ML systems for real-world use; The use of ML in RE opens up exciting possibilities for the development of novel expert and intelligent systems to support RE tasks and processes.

Another article [7] presents an approach to automatically classify content elements of a requirements specification into requirements or information. The presented approach could be used in the following ways: classification of content elements in previously unclassified documents; Perform an analysis on a previously classified document and assist the user in identifying elements that are not correctly classified.

The authors propose Convolutional Neural Networks, a machine learning algorithm that is receiving more and more attention in the field of natural language processing. To train the neural network, the authors used a collection of over 10,000 content elements extracted from 89 requirements specifications from their industry partner. By using 90% of the content elements as training data and the remaining 10% as test data, the authors' approach was able to achieve a stable classification accuracy of about 81%.

In [13], the authors introduce a methodology for automatically assessing the quality of requirements based on input from field experts who utilize the methodology. The main objective of this methodology is to predict the quality of new requirements. To accomplish this, the experts provide an initial set of requirements that have been previously classified according to their quality and deemed appropriate. For each requirement in the set, the authors extract metrics that quantify the quality value of the requirement. The methodology suggests employing a Machine Learning technique called rule inference to learn the value ranges for these metrics and determine how they should be combined to interpret the quality of requirements, as perceived by domain experts

Strictly related to tests classification, the state-of-the-art approach involves using machine learning algorithms for test creation and maintenance. This has led to improvements in reducing maintenance efforts and enhancing product quality. Machine learning can be used throughout the software testing life cycle, from test creation to issue management. So, how can machine learning be applied in testing? There are several ways to enhance the process:

A. *Handling issues:*

ML algorithms can be used to classify issues based on severity levels, predict assignees for issues based on past experience, cluster issues based on common features, and prioritize cases based on their relation to issues. Various tools have been developed in this area [8]:

- classify issues according to severity levels
- predict assignee of an issue according to previous experience
- cluster issues to see whether they heap together on specific features
- prioritization of cases by relating to issues

B. *Software maintenance*

Machine learning techniques can reduce maintenance efforts by automating the review process. Self-healing methods can save time by automatically detecting and suggesting resolutions for broken test cases caused by code changes [9].

C. *New tests generation*

As software systems become more complex, it becomes challenging to define tests that can check the entire product spectrum. Investing in an ML system that can automatically generate pattern-based tests is a cost-effective solution. Once the ML system is trained, it can learn patterns and generate tests automatically in the long run. Innovative solutions for automated test generation have been proposed in [10] and [11] for embedded systems, while [12] proposes a hybrid ML algorithm to manage test scenarios for printed-circuit boards.

Experimental results have shown an increase in fault identification from 57.3% to 78.9%

III. The Method

The testing phases for successive software versions in complex projects are in most cases particularly challenging, both in terms of cost and time. Several thousand test cases can be executed during a full test cycle, and the execution time can be extended to weeks. Given this long duration of testing, a serious bug significantly increases the cost of the project if it is discovered in the final phase of the timeframe. This includes implementation costs and retesting costs. A big advantage would be to find a way where topics with a high chance of failing to be executed first. This would make it possible to evaluate first those functions where there is a high risk of failure, then those with lower risk, and so on. At the end of the test phase, only the functions with the lowest risk of failure remained.

This article presents an original solution that performs an automated classification of tests based on their scope and determines step by step which tests need to be run next. The decision is based on previous experience and the results of the current cycle, based on tests already carried out. The solution uses machine learning algorithms for automated classification, result analysis, and determination of the test sequence for each next step. The application has been designed as a tool that supports the testing process in the following areas:

- Automated classification of test cases
- Risk assignment - learning phase
- Selection of the test sequence.

A. Testcases definition

The application was designed to be compatible with systems where tests are designed based on a template that provides detailed information about the scope and steps of the tests. The following elements need to be defined for each test specification and will be considered as input:

Test scope description: This specifies what will be verified in the current test and is expressed as free text. For example, "The test is verifying the system's reaction in case of a damaged LED."

Test preconditions: This describes the initial state of the system, also in free text. For example, "The system is running with no active errors."

Test steps description: This provides a step-by-step specification of the tasks that need to be performed to verify the test scope. For example:

Step 1: Start the system diagnosis and disconnect the load.

Step 2: Turn on the light.

Step 3: Verify if the system detects any errors.

Pass/fail criteria: This defines how to interpret the results of the previous steps. For example, "If the system detects the fault, the test is considered passed."

Automation: This refers to a script that can be executed automatically to perform the specified steps and evaluate the results.

During a regular test cycle, the tests are grouped into sequences based on functionalities. Each sequence is executed sequentially, and the results are analyzed and documented. This means that each test is assigned to only one main functionality and is executed when that functionality is the focus.

The proposed solution use the TensorFlow algorithm to parse each test individually. Based on the analysis of the defined scope and description, it generates a list of functionalities that are directly or indirectly verified. This classification creates a graph where all the tests are linked to each other based on predefined keywords, which represent the functionalities in focus.

B. Dataset

We have generated a dataset using an existing set of real Testcases used for the validation of an Electronic Control Unit (ECU) in the automotive industry. Each individual Testcase in the dataset possesses the following attributes:

- Test scope description
- Test preconditions
- Test steps description
- Pass/fail criteria

Starting from these attributes we have generated two new attributes that are used for Testcases prioritization.

The first attribute, called Test Priority is obtained by concatenating the text information from the Test scope description, Test preconditions and Test steps descriptions. The second attribute is called Test Added Value and is obtained based on the Pass/fail criteria and the results of the interpretation of the Testcases executed in each of the five releases. The Test Added Value is a label having values with 0 and 1.

0 - Testcase low value added

1 - Testcase high value added

These labels from the Test Added Value were automatically assigned via a script to each of the Testcases. Preprocessing was performed on the Test Priority attribute to eliminate extraneous information, such as logical expressions and statements that could not be converted into lexical tokens.

The resulting dataset comprises 5000 Tescases , which serve as the raw input for our Test prioritization process.

C. The Application description

We utilized two computational methods for our dataset. The first method is checking the traditional "bag of words" (BoW) representation, where word occurrences are counted to create a vector for each sentence. The second method utilizes "word embeddings," a newer approach that assigns each word a vector preserving semantic meaning. Our experiments aimed to evaluate the potential of word embeddings (VE) compared to the classic BoW representation when combined with deep neural networks for automated Testcases prioriza-tion.

IV. Experiments and Results

The experiments performed followed a two-step processing pipeline:

Text vectorization: Each Testcase document was transformed into a numerical vector using either the BoW or VE method.

Deep learning classification: A suitable deep neural network (NN) was defined, trained, and validated to classify the vectorized representations obtained in the previous step.

For both models, we applied the standard approach of cross-validation. Our dataset was split into training and testing sets, with 75% of the examples used for training and the remaining 25% for testing to measure model accuracy. The split was performed using the "train_test_split" method from the scikit-learn package.

A. First Model:

Initially, we made the BoW representation of text. To accomplish this, we constructed a vocabulary from our dataset consisting of a unique word list. Each word was assigned an index, and every Testcases (example) was then associated with a vector of dimensions equivalent to the vocabulary size, which was 2135 in our specific case. Within the vector, each element indicates the count of occurrences for the corresponding word in our dataset.

For the Testcases classification using deep neural networks (NN), we utilized Keras. The NN architecture consisted of an input layer, one hidden layer with 10 nodes, and an output layer. The hidden layer employed a densely-connected NN layer of type layers. Dense with the ReLU activation function. Since we were dealing with a binary classification problem, we used the sigmoid activation function with a dimensionality of 1 for the output layer. The optimization of the NN was performed using the Adam algorithm, and binary cross-entropy served as the loss function.

Using the constructed model, we trained it using our training data. The training process involved 10 samples per gradient update, and we performed 20 iterations. The first layer had 21,360 parameters, while the second layer had 11 parameters. The total number of parameters was calculated as follows: each feature vector had 2135 dimensions, which required weights for each feature dimension and each node, resulting in 2135 * 10 (adding 10 times bias for each node). The layer had 10 weights and one bias. During the training process, all 21,371 parameters were determined. The results are presented in Figure 1.

To evaluate the performance of the trained network, we measured accuracy on both the training and test sets, as well as the training and validation loss. The first model achieved an accuracy of 85%.

B. Second Model

In our second model, we employed word embeddings to represent the requirements, departing from the previous model that used the Bag-of-Words (BoW) approach to map each requirement to a single feature vector. Instead, we represented each word as a numeric vector. There are two options to acquire word embeddings: training them separately on the new corpus or using pre-trained versions. In our experiment, we opted to utilize pre-trained GloVe word embeddings, specifically the glove6B.50d.txt file, which encompasses 400,000 unique words and has a total size of 822MB. However, we filtered the embeddings to include only the words present in our dataset.

To prepare the data for our word embeddings model, we utilized the pre-processed test and training data and performed tokenization. The Keras Tokenizer utility class was employed to convert the dataset into a list of integers. Each integer in the list corresponds to a word in the dictionary that

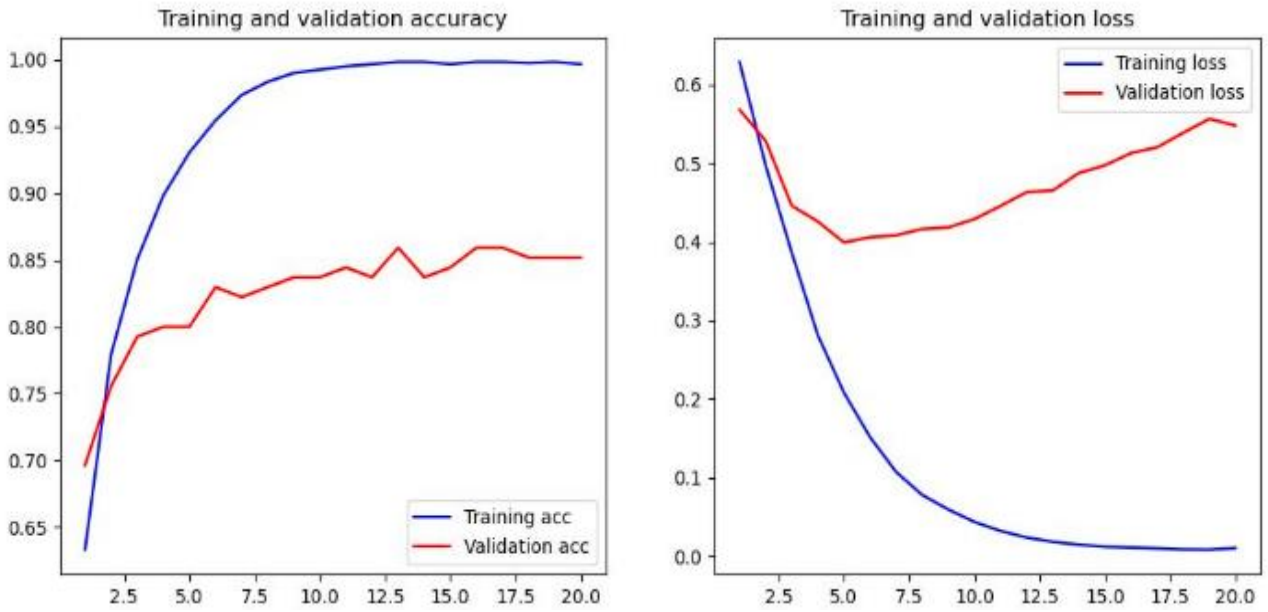

Fig. 1 Results obtained using first model

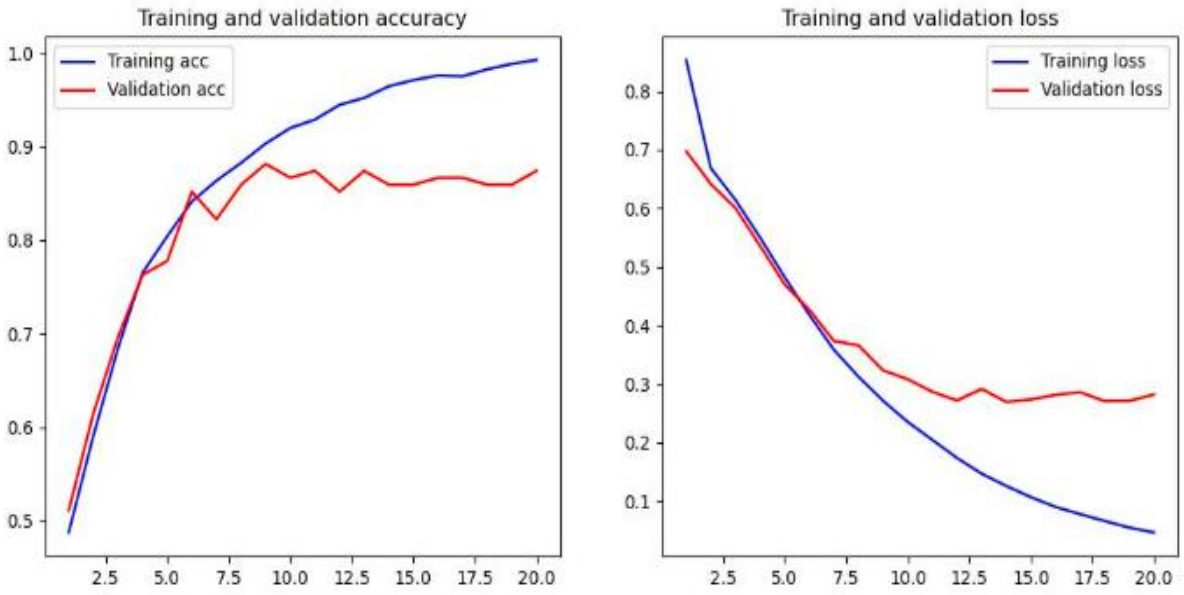

Fig. 2 Results obtained using the second model

represents the entire corpus. As the length of each requirement (example) may differ, we padded the word sequences with zeros to ensure a consistent length of 100 for our experiment.

For the architecture of our second model, we employed a deep neural network (NN) comprising an input layer, three hidden layers, and an output layer. The first layer utilized the layers.Embedding data type, enabling us to map the examples represented as lists of integers to a suitable representation for processing by the subsequent GlobalMaxPool1D layer. The embedded layer was configured with the following parameters:

Input dimension: 2135, representing the vocabulary size.

Output dimension: 50, indicating the size of the dense vector.

Input length: 100, denoting the length of the word sequence.

The second hidden layer was of type GlobalMaxPool1D, employing default parameters to downsample the incoming feature vectors by selecting the maximum value across each feature dimension.

The third hidden layer was of the Dense Layer type, similar to the first model.

As we were dealing with a binary classification problem, akin to the first experiment, we employed the sigmoid activation function with an output dimension of 1 for the NN model's output layer. Binary cross-entropy was used as the loss function to measure the discrepancy between the actual output and the predicted output.

We optimized our NN using the Adam optimizer. With the model created, we proceeded to train it using our training data. We used 10 samples per gradient update and conducted 50 iterations. The results obtained are depicted in Figure 2.

To evaluate the performance of the trained network, we measured the accuracy on the training and test sets, as well as the training and validation loss. This model showcased improvement over the first model, achieving an accuracy of 89%.

V. Conclusion

In the last years, artificial intelligence started to transform the testing paradigm in ways that could not have been considered possible some years ago. In the current paper it is presented an innovative solution applied in the testing domain based on machine learning algorithms for tests execution.

It is considered the first results and experiments towards automating classification of new written Testcases in software engineering for automotive industry towards test execution. We have investigated the potential of combining deep NN models with word representations for improving the performance of this task.

Our results are preliminary, nevertheless, we were able to obtain an improvement in performance for the word embeddings approach, as compared with the baseline bag of words approach. In the near future we plan to strengthen our results by expanding our experiments to include different and larger data sets, as well as to use different and suitably trained deep NN architectures.

References

[1] M.Weber, J.Weisbrod (2003) Requirements engineering in automotive development: experiences and challenges. IEEE Software,2003, 20(1):16–24

[2] F. Azaïs, S. Bernard, M. Comte, B. Deveautour, S. Dupuis, H. El Badawi, M.-L. Flottes, P. Girard, V. Kerzerho, L. Latorre, F. Lefèvre, B. Rouzeyre, E. Valea, T. Vayssadel, A. Virazel, "Development and

Application of Embedded Test Instruments to Digital Analog/RFs and Secure ICs", IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp. 1-4, 2020

[3] A. Adekanmi, “Research on software testing and effectiveness of automation testing”, 2019

[4] H. Gamido, M. Gamido “Comparative Review of the Features of Automated Software Testing Tools”, International Journal of Electrical and Computer Engineering, vol 9, pp. 4473-4478, 2019

[5] https://www.functionize.com/machine-learning-in-software-testing

[6] Binkhonain M., Zhao L, “A review of machine learning algorithms for identification and classification of non-functional requirements”, 2019, Expert Systems with Applications: X, 1, doi: 10.1016/j.eswax.2019.100001

[7] Winkler J., Vogelsang A. (2016) Automatic Classification of Requirements Based on Convolutional Neural Networks, In: IEEE 24th International Requirements Engineering Conference Workshops (REW), 39–45 doi: 10.1109/REW.2016.021.

[8] K. Sneha, G. M. Malle, "Research on software testing techniques and software automation testing tools," International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), pp. 77-81, 2017

[9] https://www.infoq.com/news/2021/03/machine-learning-testing/

[10] S.Roy, S. K. Millican, V.D. Agrawal, "Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator", 20th International Conference on Embedded Systems (VLSID) 2021, pp. 316-321, 2021

[11] S. Roy, S.K. Millican, V.D. Agrawal, "Principal Component Analysis in Machine Intelligence-Based Test Generation", Microelectronics Design & Test Symposium (MDTS), pp. 1-6, 2021

[12] M. Liu, F. Ye, X. Li, K. Chakrabarty, X. Gu, “Board-Level Functional Fault Identification Using Streaming Data”, Computer-Aided Design of Integrated Circuits and Systems, vol 40, no. 9, pp. 1920-1933, 2021

[13] Parra E., Dimou C., Llorens J., Moreno V., Fraga, A. (2015) A methodology for the classification of quality of requirements using machine learning techniques, Information and Software Technology, 67:180–195, doi: 10.1016/j.infsof.2015.07.006.

[14] M.N. Velev, C. Zhang, P. Gao, and A.D. Groce, “Exploiting Abstraction, Learning from Random Simulation, and SVM Classification for Efficient Dynamic Prediction of Software Health Problems,” 16th International Symposium on Quality Electronic Design (ISQED ’15), March 2015, pp. 412–418