Automatic Directory Classification of Test Cases Based on Machine Learning Algorithms at an Android Smartphone Vendor

Abdirahman Osman Hashi
Istanbul Technical University
Istanbul, Turkey
Wadani12727@gmail.com

Octavio Ernesto Romo Rodriguez
Istanbul Technical University
Istanbul, Turkey
oct_romo@hotmail.com

Abstract—Software test cases is an important study issue that has piqued the interest of many academics who are attempting to create or suggest a heuristic strategy that might lessen the laborious manual effort that software engineers expend while classifying test cases. The goal is to ensure that all features and apps have been tested and verified. In order to achieve that, there must be a good framework that can suggest or match the feature labels with their test cases in a chronological way. Failing to do so will result in inaccurately labeled test cases. Therefore, the key objective of this paper is to propose a method that can do an automatic directory classification of test cases based on their test case description by applying the K-nearest neighbor classifier. Bag-of-word (Bow) and Term Frequency-Inverse Document Frequency were used as a vector representation and fitted the KNN classifier. The experimental result shows that using KNN-BOW has a good score compared to KNN-TF-IDF as it outperformed and achieved 77% accuracy in comparison with the 71% that KNN-TF-IDF achieved. Because of that, KNN-BOW is a good option for the directory classification based on test case descriptions. The proposed method has a contribution to the domain and makes sure that using machine learning algorithms can make easy directory classification of test case descriptions.

I. INTRODUCTION
Software testing is the main component or principal element in developing software efficiently and making certain of its correctness regarding the operation that is expected to be carried out under distinctive input variables [1]. Numerous techniques or strategies exist for performing software testing. The most common ones are black box and white box testing. Other classifications of the software testing methods can be done by referring to how the testing is carried out. According to this, classification test cases can be either manual testing or automated testing [2].

Manual testing is performed by preparing test cases manually and is more prone to human errors, whereas automatic testing is carried out by recording the various test cases on the basis of what actions the user has performed. This saves a lot of time in writing test cases manually and improves the efficiency. Besides that, manual testing is not suitable for intensive software, such as those companies that are manufacturing Android phones [4]. Because we know that Android is a broad software platform that consists of many different layers composed of applications, drivers, operating system, components, and kernel. It is clear that this software has its own complexity and Android manufacturers need comprehensive tests in order to make sure the system is working as it is expected to carry out and to fit the requirements of the organization in terms of hardware support and software [2].

To address this issue, many researchers proposed several methods, such as using manual tags or manual assignment for feature labels, or even using an application lifecycle management system. Nevertheless, these proposed solutions are still leading to inaccurately labeled test cases. Therefore, the key objective of this paper is to propose a method that can do an automatic directory classification of test cases based on their test case description. The idea is to reduce the tedious manual effort that software developers currently spend classifying test cases. The goal is to ensure that all features and apps have been tested and verified. In order to achieve that, there must be a good framework that can suggest or match the feature labels with their test cases in a chronological way. Hence, the result of this proposed method will be the categorization of the directory structure of the test cases by determining for each test case which directory it must be or belongs to be in. The prediction will be based on the whole component that the test case belongs to, and this will ensure whether the test cases were classified correctly or not.

This paper is structured with five sections. The following section provides related work of topic classification test cases. The third section describes how it will look like the proposed methodology that is going to be implemented in third section. The fourth section presents the output of the proposed framework and analyzation. Finally, the fifth section presents conclusion and future work.

II. BACKGROUND & RELATED WORK
As classifying software test cases is an important research domain area, many researchers propose different methods, such as manual tagging, using application lifecycle management, and automatic topic classification. Each proposed method has its own pros and cons in terms of time-consuming tasks and inaccurately labeled task cases. For instance, tests are labelled manually at the studied Android smartphone vendor, and they have defined tags and feature labels. These features need to be categorized in an accurate way, unless the test cases are
unsuccessful, which will cause the developer to not know which features and apps were tested [1]. Meanwhile, application lifecycle management is another method that has been proposed to tackle this issue. ALM, in short form, is a way of managing a pre-defined process to facilitate software development from start to finish, such as product release or end product support. The predefined process is such as defect tracking, fixing, or testing, and it needs to be connected through a web interface or be its own dedicated window application form [1], [2].

In a nutshell, it’s well-known that there is a great amount of research that has been published in the area of automatic test case generation methods in the past few years [2], [5]. Hence, we can classify test case generation into three main types: requirement-based, program-based, and random-based.

A. Requirement based

Requirement-based test case generation can also be called specification test case generation [3], as the test case can be a semi-formal or formal specification of the required data or a function of the software under test [2]. The formal specification can be driven by different formalisms of software requirements such as logic programs, finite state machines, and first-order logic ones [6] and [7]. On the other hand, the semi-formal specification can also be driven by the diagram notation of software systems. In the dataflow diagram, it has defined the structure requirement as a hierarchy for the test case benchmark [8].

B. Random based

Random-based test cases are related to a certain class of probabilistic models that are generated during the execution time of software operations. Through a random sampling, it usually selects test cases over the input space of the software based on a certain probabilistic distribution [7]. By applying previous software operations at random, it can be recognized as a simple random testing method, whereas applying a stochastic model can be defined as a sophisticated one. And the sophisticated ones have been applied to various models such as Markov Chains and Bayesian Networks. In terms of fault detection, reliability testing, and functional validation and verification, the sophisticate [9].

C. Program based

Program-based test cases are based on analyzing the source code of the program under test without taking into consideration the execution of the program. And it doesn’t take into consideration the behavior of the program during the execution time as a dynamic model. Because of that, it can be defined as a static test creation or generation method. It is also path-oriented because it always takes a certain path as an input during test case generation. It is also noteworthy to mention that some researchers define it as a goal-oriented method. Because it is able to determine which path causes the branch or the statement to be executed [8].

Feature labelling is also another way that researchers tackled this issue, but it has a challenge in coming up with appropriate features that each team should have. It has to be unique for each feature for each team, and it has to be labeled manually, which needs domain knowledge and investigation. This makes it similar to other tagging tasks which are time-consuming [1].

The proposed methodology is based on the requirement to summarize test case descriptions. However, feature labels cannot be assigned manually as it can cause human error, and that is why we need to analyze the text of the test case description to assign each feature label under which category it belongs to. Meanwhile, text case description has gathered Android smartphone vendors' data, and the data that has gathered is the one we have to analyze and make the directory classification based on the text analysis.

However, the method presented in this work is inspired by the idea of an application lifecycle management system, which other researchers have improved and come up with the idea of topic classification. Because of that, we are continuing this work and trying to expand the areas that other researchers left in the feature label only to be applied in the test case descriptions.

III. METHODOLOGY

The main focus of the work is towards developing a method that can do an automatic directory classification of test cases based on test case description, and therefore, we know that research methodologies are developed to achieve the work’s objectives. As it describes the guidance that can be given to achieve the system objective, we will explain the whole process in detail.

A. Research Framework

In this framework, we are dealing with the test case descriptions as mentioned before. Because of this, we first split the raw data into two parts: training and testing. The ratio of each part is half of the raw data, which means 50% for training and 50% for testing [1]. We tried to do looping for the data split. However, we found out that there is duplicate data when we are using a loop. Because of that, we followed the previous researchers who split the dataset into (50) training and testing. After splitting the dataset, the next process is to find the best performing parameters by tuning parameters from the training dataset. The vectorizing of training text was the next step, which intends to covert text into a bag-of-words. As we have test case descriptions, we need to remove words that contain numeric characters and words that appear less than 5% of the time in each test case description. Not only bag-of-words, but we also did the same for converting text into TF-IDF, which stands for term frequency-Inverse Document Frequency. Here is the figure demonstrating the whole process.

Fig. 1 Proposed Framework
Fig. 1 shows what the proposed framework looks like and it also shows how they interconnect each step while also illustrating how the data can come from the training row dataset and testing row dataset. However, there are certain things that have to be done before fitting the model, and we would like to discuss those things before we train the model. Here is the list of certain things we have to do before we train the model:

- Data Transformation.
- Data Cleaning
- Lemmatizing and Steaming.
- Vectorization.

1) Data Transformation

The first step that we have done was data transformation. As probably formatted data improves the quality of the data, it was compulsory to do so. And this transformed data will make easier to fit the model. Here is the actual figure of the dataset before I've done data transformation.

Fig. 2 Actual dataset

Fig. 2 shows what the actual dataset looks like. We can see that the test case description is in different columns, which makes it complex to fit the training model. Because of that, we can see there is a high demand for data transformation in order to improve the structure of the data, which has a direct effect on the data quality. The upcoming figure will show us what the data looks like after we have done data transformation.

Fig. 3 Data transformed

3) Lemmatizing and Steaming

The idea of steaming is to find the core root of the word whereas the lemmatizing is to reduce the inflected words to ensure the root word where it belongs in the language. Before we vectored the words, we have to return every verb by its base-form which means it is core root. The upcoming figure four shows how the original test case description was and the next figures will demonstrated how it changed after we have done the lemmatizing.

Fig. 4 Text case Description

Fig. 4 shows one of the test case descriptions that we have in our dataset. However, we have not done lemmatizing it yet, but we have done it only for data transformation and data cleaning which makes now easily to be readable. Nevertheless, the next upcoming figure will demonstrate how the same text case description will look like after lemmatizing. However, before lemmatizing we have to tokenize the words and then do the lemmatizing.
Bluetooth fails when playing music via aidp then switching to phone call if I play music through my car stereo using the built-in main music player over bluetooth then either does not or becomes a call or my stereo turns on and I did not make this handoff fails then when the call is ended the music stops after a second of silence end and I have to kill the Bluetooth connection and reconnect for music to play again or I use my network music range update build on the car stereo in a pioneer device with a Bluetooth adapter since I reproduce the problem pair device with the stereo start playing music using the built-in player and either m makes a phone call or receive a phone call what happened the car stereo switch to高低两格 but no phone a call is heard switching the phone to speaker allows me to conduct the call through the hands speaker end and the call the stereo switch back to ap connecting mode and music will attempt to play again but stutter what you think if the correct behavior should be the music should pause and the stereo should switch phone mode so the phone can—but that can be held through the car stereo hands free when the call is ended the stereo should switch back to play as music resumes without issue this issue does not occur when playing music through Bluetooth audio third party players such as DirectAV or Turnkey when using either of those programs the phone works and the audio runs properly

Fig. 5 Text case Description Tokenized

Fig. 5 shows how the same test case description looks like after we have done tokenization word by word that makes easy to be either lemmatized or steamed. We can see from the test case description that words are not the core root. For instance, the word ‘playing’ has not yet transformed to its core root, but the next step is to do that action which lemmatizing or steaming.

Fig. 6 Text case Description Lemmatized

Fig. 6 illustrates how the test case description looks like after we have done lemmatizing. It is clear that every word has turned back to its core root. For instance, if we take a look at the word ‘play’ it was ‘playing’ before we have done the lemmatization. However, it can be seen now that every word has turned back by it is base form after used WordNet Lemmatized and Snowball Steamer from Corpus dictionary. And it makes easy to either create the bag-of-words or TF-IDF as long as we have the core root of each word.

B. Train a model with best parameters

We used K-nearest neighbor as our statistical model and fed the vectors which is the best parameters that come from either bag-of-words or TF-IDF. In our model construction, we used python scikit-learn library. In order to measure the distance of a concerned vector against other vectors in the training dataset in bag-of-words, we used minkowski metric to measure it. We know there in no need for cosine distance in bag-of-words because documents sizes are relatively similar and that is why there is no need for size normalization of cosine distance. On the other hand, in order to measure the distance of a concerned vector against other vectors in the training dataset in TF-IDF, we used cosine metric, because there is a need for size normalization when we are measuring the cosine distance. While training the model, we found out that the best performing K is 4 via tuning and thus we used in this model K equal to four.

C. Vectorizing the testing dataset

It is not similar to the train dataset, during testing dataset; we are not supposed to check the word accuracy. Because we know that during the training it fitted the model only the words that does not appeared in the test dataset and word count vector are based on the corpus dictionary in the training part.

D. Predict and Measurement

In order to make sure how our categorization has done correctly, we have to evaluate our model prediction result and make sure the measurements. As we said, K-nearest neighbor is our statistical model in order to categorize the most voted categories as predicted label. There could be certain points that the classifier cannot categorize the labels and such output is unpredictable and it does not contribute or affect the performance of the model. In term of performance, we will discuss in the result section.

IV. RESULT AND DISCUSSIONS

In the subsequent section, results and discussions are going to be discussed and we will present the accuracy that achieved by each model with several of metrics that we tried it. Knn-Bow has achieved the best accuracy with its minkowski metric while Knn-TF-IDF has also proved its aptitude in term of a achieving a good accuracy but not much as Knn-Bow. Here is the details of each of them.

A. Dataset Description

The dataset that is used in this study was collected by two Android smartphone vendors who asked that we keep their names anonymous. The dataset consists of various test cases, but previous researchers had picked six test cases that came from six teams in the company out of 50. The reason they chose is that these test cases were managed in ALMS (the application lifecycle management system) [1]. The following table shows us the different domains that we have in the dataset.

<table>
<thead>
<tr>
<th>Team ID</th>
<th>Test Domain</th>
<th>Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Multimedia</td>
<td>2286</td>
</tr>
<tr>
<td>1</td>
<td>Multimedia</td>
<td>2286</td>
</tr>
<tr>
<td>2</td>
<td>Android OS &amp; Linux Kernel</td>
<td>2286</td>
</tr>
<tr>
<td>3</td>
<td>Android OS &amp; Linux Kernel</td>
<td>2286</td>
</tr>
<tr>
<td>4</td>
<td>Cellular &amp; Connectivity</td>
<td>2286</td>
</tr>
<tr>
<td>5</td>
<td>Cellular &amp; Connectivity</td>
<td>2286</td>
</tr>
<tr>
<td></td>
<td>Total test cases</td>
<td>13716</td>
</tr>
</tbody>
</table>

B. Results and Discussions

The anticipated proposed methodology for classifying test cases based on categories applies to the K nearest neighbor classifier that uses different metrics to calculate the distance
between the real vectors by summing up their absolute differences. In this work, we used minskowski and cosine metrics as the metric of the K nearest neighbor.

Here is the result that we found after we did data cleaning, lemmatizing, and steaming and applied it to the K-nearest neighbor with a bag of words. Although we tried different metrics for KNN- BOW, such as minskowski and cosine, we found that using minskowski is a good option as a metric. This means using Minkowski as a metric is outperformed by using cosine as a metric. Here are the two outputs.

**KNN with BOW accuracy = 76.66958296879557%**

| [1016 | 25 | 27 | 20 | 5 | 65] |
| [85 | 925 | 19 | 35 | 30 | 51] |
| [85 | 50 | 889 | 46 | 25 | 35] |
| [63 | 75 | 39 | 881 | 30 | 43] |
| [115 | 74 | 77 | 83 | 729 | 67] |
| [79 | 50 | 76 | 79 | 47 | 818] |

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.70</td>
<td>0.88</td>
<td>0.78</td>
</tr>
<tr>
<td>1</td>
<td>0.77</td>
<td>0.81</td>
<td>0.79</td>
</tr>
<tr>
<td>2</td>
<td>0.79</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>3</td>
<td>0.77</td>
<td>0.78</td>
<td>0.77</td>
</tr>
<tr>
<td>4</td>
<td>0.84</td>
<td>0.64</td>
<td>0.73</td>
</tr>
<tr>
<td>5</td>
<td>0.76</td>
<td>0.71</td>
<td>0.73</td>
</tr>
</tbody>
</table>

**accuracy**

macro avg: 0.77
weighted avg: 0.77

**KNN with BOW accuracy = 76.55293088363955%**

| [1021 | 31 | 26 | 15 | 15 | 50] |
| [92 | 933 | 19 | 30 | 30 | 41] |
| [84 | 70 | 870 | 41 | 30 | 35] |
| [56 | 44 | 4914 | 30 | 33] |
| [107 | 79 | 85 | 85 | 725 | 64] |
| [93 | 46 | 80 | 100 | 43 | 7871] |

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.70</td>
<td>0.88</td>
<td>0.78</td>
</tr>
<tr>
<td>1</td>
<td>0.77</td>
<td>0.81</td>
<td>0.79</td>
</tr>
<tr>
<td>2</td>
<td>0.77</td>
<td>0.77</td>
<td>0.77</td>
</tr>
<tr>
<td>3</td>
<td>0.77</td>
<td>0.81</td>
<td>0.79</td>
</tr>
<tr>
<td>4</td>
<td>0.83</td>
<td>0.63</td>
<td>0.72</td>
</tr>
<tr>
<td>5</td>
<td>0.78</td>
<td>0.68</td>
<td>0.73</td>
</tr>
</tbody>
</table>

**accuracy**

macro avg: 0.77
weighted avg: 0.77

Moreover, Fig. 8 shows us that using steaming is a good choice in terms of accuracy and f1-score compared to the lemmatizing. Nevertheless, the overall good accuracy is achieved by using lemmatization in a Minkowski metric.

On the other hand, KNN-TF-IDF has applied the same data after cleaning, lemmatizing, and steaming by applying the cosine metric. We found out that using steam is also superior to using lemmatization in terms of accuracy and f1- score. As the upcoming figure illustrates.

In Fig. 7, it can be seen that using lemmatization is a good option in terms of the accuracy, which is outperformed by steaming by using Minkowski as a metric. However, as we can see, F1’s results in both metrics are also most similar. This shows us how they are also close to each other. On the other hand, the upcoming figure will demonstrate how the result is after the cosine metric is applied.
will be WC for F1 scores of 0.71 (B, C, E, F), whereas the rest to 0.88 for both WC and LD performance, yet the separation achieved. As adopted from [1], their model has scored from 0.3 F1 score, and we compared that result with the result that we which makes B, C, E, and F have good performance in terms of classified their performance results in terms of Name-LDA or Name-WC, we find that WC is better than LDA in their model, this shows us analyzing the text has achieved the best accuracy in terms of classifying different modules.

The best accuracy is achieved with the Knn-Bow compared to the Knn-TF-IDF. Yet, using Minkowski as a distance metric with lemmatizing is a good option, which scored 76.66 as accuracy and 77 as f1-score. However, applying Knn-bow with cosine as a distance metric has achieved a low accuracy compared to applying Knn-TF-IDF using cosine as a distance metric has scored 71.62 as an accuracy, but both achieved the same f1-score, which is 72.

C. Comparative analysis

However, in comparing with previous researchers who have classified their performance results in terms of Name-LDA or Name-WC, we find that WC is better than LDA in their model, which makes B, C, E, and F have good performance in terms of F1 score, and we compared that result with the result that we achieved. As adopted from [1], their model has scored from 0.3 to 0.88 for both WC and LD performance, yet the separation will be WC for F1 scores of 0.71 (B, C, E, F), whereas the rest will go through LD (A, D).

On the other hand, our model has scored 76.66 as accuracy and 77 as f1-score, which outperformed the result of the WC performance that was adopted by [1] researchers. This shows us analyzing the text has achieved the best accuracy in terms of classifying different modules.

V. CONCLUSION

Automatic directory classification based on machine learning is proposed for this methodology, which can play an essential role for test case classification that spends a lot of time for testers in order to make sure which feature or apps are working as they are expected to carry out. The K nearest neighbor is used to be the algorithm that has done the classification and fitted two different vector representations: bag-of-words and TF-IDF, which stands for term frequency-inverse document frequency. We found out that using KNN-BOW has a good result and outperformed using KNN-TF-IDF. Because of that, KNN-BOW is a good option for directory classification based on test case descriptions. Moreover, we found out how much such a system could be deployed by an Android smartphone vendor and how much work developers would have to invest in order to make a working system.

VI. FUTURE WORK

This proposed method can be further improved and applied to other fields such as reinforcement learning and online learning, or it can be applied to other machine learning algorithms in order to improve the result.

Meanwhile, domain-specific specialization in the NLP and IR pipelines has the potential to increase both machine learning and run-time performance. Words like 802.11 and H264 should be considered domain knowledge and not separated by a naive tokenizer. It is feasible that the feature label might be enhanced by collecting user suggestions and collaborating with the teams, albeit each team will have their own set of features.

REFERENCES