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

Abdirahman Osman Hashi[1]
Istanbul Technical University
Istanbul, Turkey
Wadani12727@gmail.com

Octavio Ernesto Romo Rodrigue
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 mode. 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 andtesting. 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.

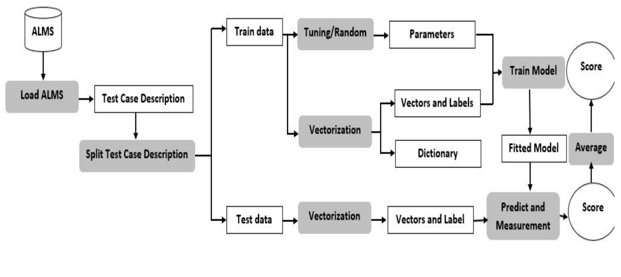


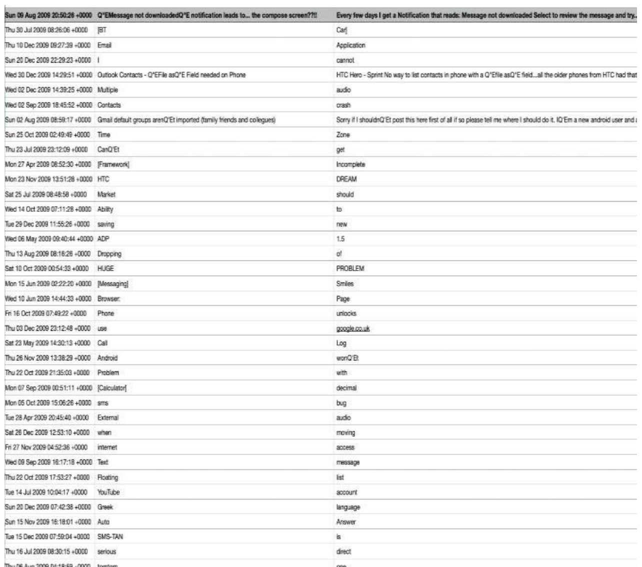
Fig. 1 Proposed Framework

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.



Sun 09 Aug 2009 20:50:26 +0000	Q"EMessage not downloadedQ"E notification leads to... the compose screen??!!	Every few days I get a Notification that reads: Message not downloaded Select to review the message and try...
Thu 30 Jul 2009 08:26:06 +0000	[BT	Car]
Thu 10 Dec 2009 09:27:39 +0000	Email	Application
Sun 20 Dec 2009 22:29:23 +0000	I	cannot
Wed 30 Dec 2009 14:29:51 +0000	Outlook Contacts - Q"EFile asQ"E Field needed on Phone	HTC Hero - Sprint No way to list contacts in phone with a Q"Efile asQ"E field...all the older phones from HTC had that
Wed 02 Dec 2009 14:39:25 +0000	Multiple	audio
Wed 02 Sep 2009 18:45:52 +0000	Contacts	crash
Sun 02 Aug 2009 08:59:17 +0000	Gmail default groups arenQ'Et imported (family friends and collegues)	Sorry if I shouldnQ'Et post this here first of all if so please tell me where I should do it. IQ'Em a new android user and
Sun 25 Oct 2009 02:49:49 +0000	Time	Zone
Thu 23 Jul 2009 23:12:09 +0000	CanQ'Et	get
Mon 27 Apr 2009 08:52:30 +0000	[Framework]	Incomplete
Mon 23 Nov 2009 13:51:28 +0000	HTC	DREAM
Sat 25 Jul 2009 08:48:58 +0000	Market	should
Wed 14 Oct 2009 07:11:28 +0000	Ability	to
Tue 29 Dec 2009 11:55:26 +0000	saving	new
Wed 06 May 2009 09:40:44 +0000	ADP	1.5
Thu 13 Aug 2009 08:16:26 +0000	Dropping	of
Sat 10 Oct 2009 00:54:33 +0000	HUGE	PROBLEM
Mon 15 Jun 2009 02:22:20 +0000	[Messaging]	Smiles
Wed 10 Jun 2009 14:44:33 +0000	Browser:	Page
Fri 16 Oct 2009 07:49:22 +0000	Phone	unlocks
Thu 03 Dec 2009 23:12:48 +0000	use	google.co.uk
Sat 23 May 2009 14:30:13 +0000	Call	Log
Thu 26 Nov 2009 13:38:29 +0000	Android	wonQ'Et
Thu 22 Oct 2009 21:35:03 +0000	Problem	with
Mon 07 Sep 2009 00:51:11 +0000	[Calculator]	decimal
Mon 05 Oct 2009 15:06:26 +0000	sms	bug
Tue 28 Apr 2009 20:45:40 +0000	External	audio
Sat 26 Dec 2009 12:53:10 +0000	when	moving
Fri 27 Nov 2009 04:52:36 +0000	internet	access
Wed 09 Sep 2009 16:17:18 +0000	Text	message
Thu 22 Oct 2009 17:53:27 +0000	Floating	list
Tue 14 Jul 2009 10:04:17 +0000	YouTube	account
Sun 20 Dec 2009 07:42:38 +0000	Greek	language
Sun 15 Nov 2009 16:18:01 +0000	Auto	Answer
Tue 15 Dec 2009 07:59:04 +0000	SMS-TAN	is
Thu 16 Jul 2009 08:30:15 +0000	serious	direct
Thu 06 Aug 2009 04:18:59 +0000	tomtom	one
Wed 05 Aug 2009 15:46:58 +0000	NEw	bug

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 demonstrates how data become after transformation and it is clean that is unlike in Fig. 2. The main reason that we did data transformation is to get each test case description in one row so that it can be easy for the model to categorize based on the word count in the training model.

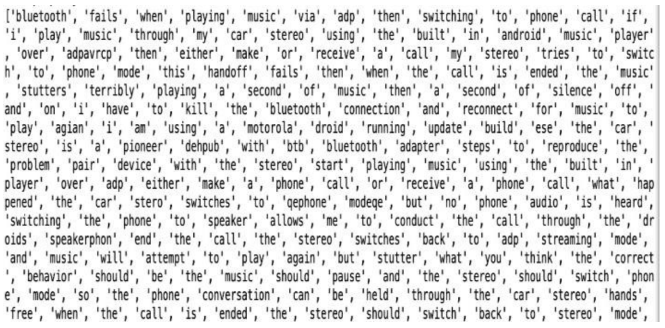
2) Data Cleaning

The second step that was data cleaning by removing numbers, punctuations and capitalization from the test case descriptions. We know that as long as we have a cleaned data, we can have a successful project as it is the start point. And that makes me to take a proper care to clean the data by removing unnecessary information such as numerical words or alphanumerical.

Fig. 3 Data transformed

3) Lemmatizing and Steaming

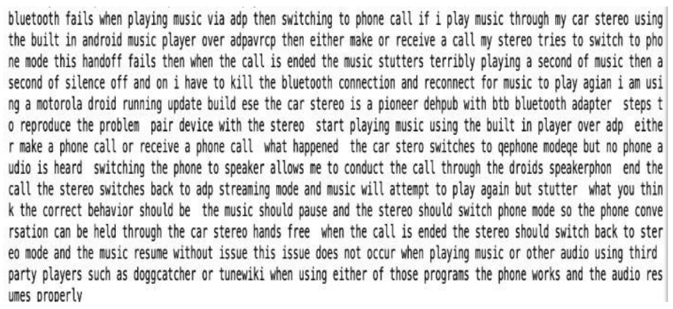
The idea of steaming is to find the core root of the word whereas the lemmatizing is to reduce the inflected words to ensure the root word where it belongs in the language. Before we vectored the words, we have to return every verb by it is base-form which means it is core root. The upcoming figure four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



```
['bluetooth', 'fails', 'when', 'playing', 'music', 'via', 'adp', 'then', 'switching', 'to', 'phone', 'call', 'if',
'i', 'play', 'music', 'through', 'my', 'car', 'stereo', 'using', 'the', 'built', 'in', 'android', 'music', 'player'
, 'over', 'adpavrcp', 'then', 'either', 'make', 'or', 'receive', 'a', 'call', 'my', 'stereo', 'tries', 'to', 'switc
h', 'to', 'phone', 'mode', 'this', 'handoff', 'fails', 'then', 'when', 'the', 'call', 'is', 'ended', 'the', 'music'
, 'stutters', 'terribly', 'playing', 'a', 'second', 'of', 'music', 'then', 'a', 'second', 'of', 'silence', 'off', '
and', 'on', 'i', 'have', 'to', 'kill', 'the', 'bluetooth', 'connection', 'and', 'reconnect', 'for', 'music', 'to',
'play', 'agian', 'i', 'am', 'using', 'a', 'motorola', 'droid', 'running', 'update', 'build', 'ese', 'the', 'car', '
stereo', 'is', 'a', 'pioneer', 'dehpub', 'with', 'btb', 'bluetooth', 'adapter', 'steps', 'to', 'reproduce', 'the',
'problem', 'pair', 'device', 'with', 'the', 'stereo', 'start', 'playing', 'music', 'using', 'the', 'built', 'in', '
player', 'over', 'adp', 'either', 'make', 'a', 'phone', 'call', 'or', 'receive', 'a', 'phone', 'call', 'what', 'hap
pened', 'the', 'car', 'stero', 'switches', 'to', 'qephone', 'modeqe', 'but', 'no', 'phone', 'audio', 'is', 'heard',
'switching', 'the', 'phone', 'to', 'speaker', 'allows', 'me', 'to', 'conduct', 'the', 'call', 'through', 'the', 'dr
oids', 'speakerphon', 'end', 'the', 'call', 'the', 'stereo', 'switches', 'back', 'to', 'adp', 'streaming', 'mode',
'and', 'music', 'will', 'attempt', 'to', 'play', 'again', 'but', 'stutter', 'what', 'you', 'think', 'the', 'correct
', 'behavior', 'should', 'be', 'the', 'music', 'should', 'pause', 'and', 'the', 'stereo', 'should', 'switch', 'phon
e', 'mode', 'so', 'the', 'phone', 'conversation', 'can', 'be', 'held', 'through', 'the', 'car', 'stereo', 'hands',
'free', 'when', 'the', 'call', 'is', 'ended', 'the', 'stereo', 'should', 'switch', 'back', 'to', 'stereo', 'mode',
```

Fig. 4 Text case Description

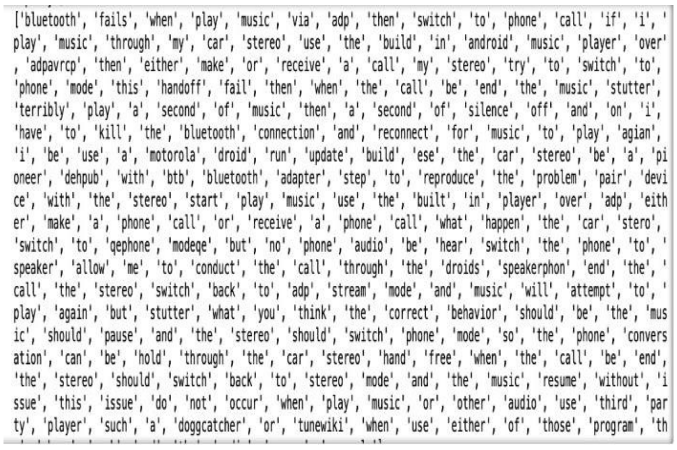
Fig. 4 shows one of the test case descriptions that we have in our dataset. However, we have not done lemmatizing it yet, but we have done it only for data transformation and data cleaning which makes now easily to be readable. Nevertheless, the next upcoming figure will demonstrate how the same text 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 adp then switching to phone call if i play music through my car stereo using
the built in android music player over adpavrcp then either make or receive a call my stereo tries to switch to pho
ne mode this handoff fails then when the call is ended the music stutters terribly playing a second of music then a
second of silence off and on i have to kill the bluetooth connection and reconnect for music to play agian i am usi
ng a motorola droid running update build ese the car stereo is a pioneer dehpub with btb bluetooth adapter  steps t
o reproduce the problem  pair device with the stereo  start playing music using the built in player over adp  eithe
r make a phone call or receive a phone call  what happened  the car stero switches to qephone modeqe but no phone a
udio is heard  switching the phone to speaker allows me to conduct the call through the droids speakerphon  end the
call the stereo switches back to adp streaming mode and music will attempt to play again but stutter  what you thin
k the correct behavior should be  the music should pause and the stereo should switch phone mode so the phone conve
rsation can be held through the car stereo hands free  when the call is ended the stereo should switch back to ster
eo mode and the music resume without issue this issue does not occur when playing music or other audio using third
party players such as doggcatcher or tunewiki when using either of those programs the phone works and the audio res
umes 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 it is core root, but the next step is to do that action which lemmatizing or steaming.



```
['bluetooth', 'fails', 'when', 'play', 'music', 'via', 'adp', 'then', 'switch', 'to', 'phone', 'call', 'if', 'i', '
play', 'music', 'through', 'my', 'car', 'stereo', 'use', 'the', 'build', 'in', 'android', 'music', 'player', 'over'
, 'adpavrcp', 'then', 'either', 'make', 'or', 'receive', 'a', 'call', 'my', 'stereo', 'try', 'to', 'switch', 'to',
'phone', 'mode', 'this', 'handoff', 'fail', 'then', 'when', 'the', 'call', 'be', 'end', 'the', 'music', 'stutter',
'terribly', 'play', 'a', 'second', 'of', 'music', 'then', 'a', 'second', 'of', 'silence', 'off', 'and', 'on', 'i',
'have', 'to', 'kill', 'the', 'bluetooth', 'connection', 'and', 'reconnect', 'for', 'music', 'to', 'play', 'agian',
'i', 'be', 'use', 'a', 'motorola', 'droid', 'run', 'update', 'build', 'ese', 'the', 'car', 'stereo', 'be', 'a', 'pi
oneer', 'dehpub', 'with', 'btb', 'bluetooth', 'adapter', 'step', 'to', 'reproduce', 'the', 'problem', 'pair', 'devi
ce', 'with', 'the', 'stereo', 'start', 'play', 'music', 'use', 'the', 'built', 'in', 'player', 'over', 'adp', 'eith
er', 'make', 'a', 'phone', 'call', 'or', 'receive', 'a', 'phone', 'call', 'what', 'happen', 'the', 'car', 'stero',
'switch', 'to', 'qephone', 'modeqe', 'but', 'no', 'phone', 'audio', 'be', 'hear', 'switch', 'the', 'phone', 'to', '
speaker', 'allow', 'me', 'to', 'conduct', 'the', 'call', 'through', 'the', 'droids', 'speakerphon', 'end', 'the', '
call', 'the', 'stereo', 'switch', 'back', 'to', 'adp', 'stream', 'mode', 'and', 'music', 'will', 'attempt', 'to', '
play', 'again', 'but', 'stutter', 'what', 'you', 'think', 'the', 'correct', 'behavior', 'should', 'be', 'the', 'mus
ic', 'should', 'pause', 'and', 'the', 'stereo', 'should', 'switch', 'phone', 'mode', 'so', 'the', 'phone', 'convers
ation', 'can', 'be', 'hold', 'through', 'the', 'car', 'stereo', 'hand', 'free', 'when', 'the', 'call', 'be', 'end',
'the', 'stereo', 'should', 'switch', 'back', 'to', 'stereo', 'mode', 'and', 'the', 'music', 'resume', 'without', 'i
ssue', 'this', 'issue', 'do', 'not', 'occur', 'when', 'play', 'music', 'or', 'other', 'audio', 'use', 'third', 'par
ty', 'player', 'such', 'a', 'doggcatcher', 'or', 'tunewiki', 'when', 'use', 'either', 'of', 'those', 'program', 'th
```

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 by it is core root. For instance, if we take a look the work ‘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 –word or TFIDF 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 categories 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.

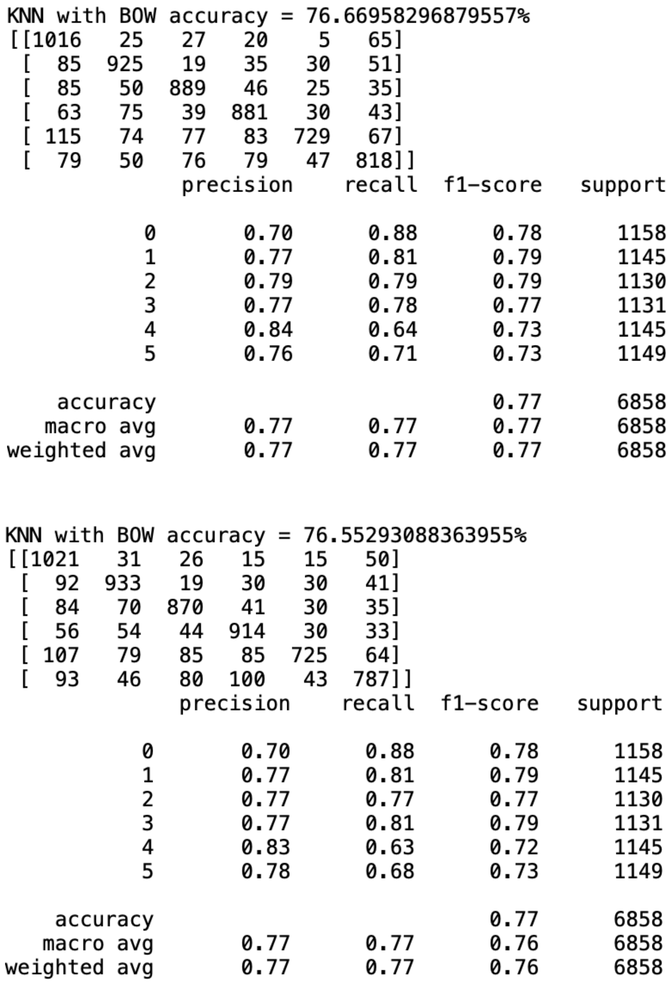
TABLE I. DATASET DESCRIPTION

eam ID		
	Test Domain	***Test Cases***
0	Multimedia	2286
1	Multimedia	2286
2	Android OS & Linux Kernel	2286
3	Android OS & Linux Kernel	2286
4	Cellular & Connectivity	2286
5	Cellular & Connectivity	2286
	Total test cases	**13716**

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 minskowsi 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.

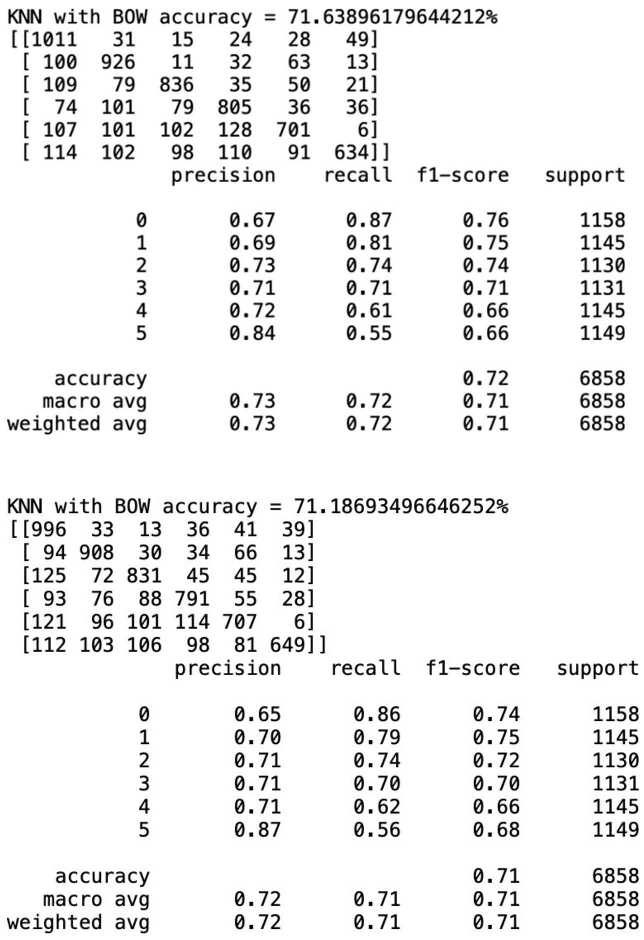


```
KNN with BOW accuracy = 76.66958296879557%
[[1016   25   27   20    5   65]
 [  85  925   19   35   30   51]
 [  85   50  889   46   25   35]
 [  63   75   39  881   30   43]
 [ 115   74   77   83  729   67]
 [  79   50   76   79   47  818]]
              precision    recall  f1-score   support

           0       0.70      0.88      0.78      1158
           1       0.77      0.81      0.79      1145
           2       0.79      0.79      0.79      1130
           3       0.77      0.78      0.77      1131
           4       0.84      0.64      0.73      1145
           5       0.76      0.71      0.73      1149

    accuracy                           0.77      6858
   macro avg       0.77      0.77      0.77      6858
weighted avg       0.77      0.77      0.77      6858
```

```
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   54   44  914   30   33]
 [ 107   79   85   85  725   64]
 [  93   46   80  100   43  787]]
              precision    recall  f1-score   support

           0       0.70      0.88      0.78      1158
           1       0.77      0.81      0.79      1145
           2       0.77      0.77      0.77      1130
           3       0.77      0.81      0.79      1131
           4       0.83      0.63      0.72      1145
           5       0.78      0.68      0.73      1149

    accuracy                           0.77      6858
   macro avg       0.77      0.77      0.76      6858
weighted avg       0.77      0.77      0.76      6858
```

Fig. 7 Knn-Bow minkowski (Lem-Ste)

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 me tric is applied.



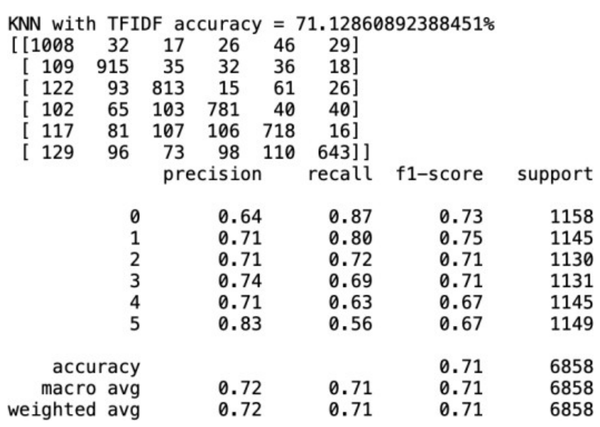
```
KNN with BOW accuracy = 71.63896179644212%
[[1011   31   15   24   28   49]
 [ 100  926   11   32   63   13]
 [ 109   79  836   35   50   21]
 [  74  101   79  805   36   36]
 [ 107  101  102  128  701    6]
 [ 114  102   98  110   91  634]]
              precision    recall  f1-score   support

           0       0.67      0.87      0.76      1158
           1       0.69      0.81      0.75      1145
           2       0.73      0.74      0.74      1130
           3       0.71      0.71      0.71      1131
           4       0.72      0.61      0.66      1145
           5       0.84      0.55      0.66      1149

    accuracy                           0.72      6858
   macro avg       0.73      0.72      0.71      6858
weighted avg       0.73      0.72      0.71      6858
```

```
KNN with BOW accuracy = 71.18693496646252%
[[996  33  13  36  41  39]
 [ 94 908  30  34  66  13]
 [125  72 831  45  45  12]
 [ 93  76  88 791  55  28]
 [121  96 101 114 707   6]
 [112 103 106  98  81 649]]
              precision    recall  f1-score   support

           0       0.65      0.86      0.74      1158
           1       0.70      0.79      0.75      1145
           2       0.71      0.74      0.72      1130
           3       0.71      0.70      0.70      1131
           4       0.71      0.62      0.66      1145
           5       0.87      0.56      0.68      1149

    accuracy                           0.71      6858
   macro avg       0.72      0.71      0.71      6858
weighted avg       0.72      0.71      0.71      6858
```

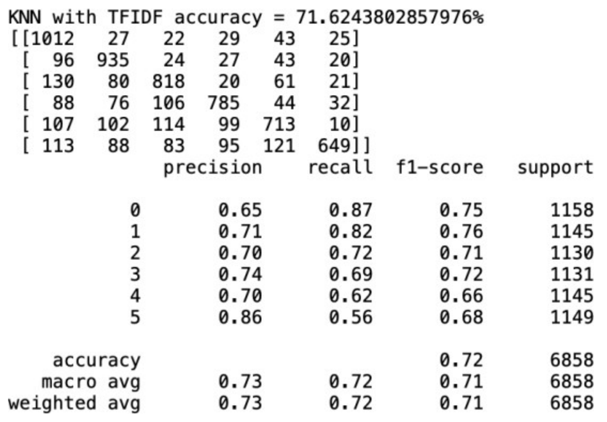
Fig. 8 Knn-Bow cosine (Lem-Ste)

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.



```
KNN with TFIDF accuracy = 71.12860892388451%
[[1008   32   17   26   46   29]
 [ 109  915   35   32   36   18]
 [ 122   93  813   15   61   26]
 [ 102   65  103  781   40   40]
 [ 117   81  107  106  718   16]
 [ 129   96   73   98  110  643]]
              precision    recall  f1-score   support

           0       0.64      0.87      0.73      1158
           1       0.71      0.80      0.75      1145
           2       0.71      0.72      0.71      1130
           3       0.74      0.69      0.71      1131
           4       0.71      0.63      0.67      1145
           5       0.83      0.56      0.67      1149

    accuracy                           0.71      6858
   macro avg       0.72      0.71      0.71      6858
weighted avg       0.72      0.71      0.71      6858
```



```
KNN with TFIDF accuracy = 71.6243802857976%
[[1012   27   22   29   43   25]
 [  96  935   24   27   43   20]
 [ 130   80  818   20   61   21]
 [  88   76  106  785   44   32]
 [ 107  102  114   99  713   10]
 [ 113   88   83   95  121  649]]
              precision    recall  f1-score   support

           0       0.65      0.87      0.75      1158
           1       0.71      0.82      0.76      1145
           2       0.70      0.72      0.71      1130
           3       0.74      0.69      0.72      1131
           4       0.70      0.62      0.66      1145
           5       0.86      0.56      0.68      1149

    accuracy                           0.72      6858
   macro avg       0.73      0.72      0.71      6858
weighted avg       0.73      0.72      0.71      6858
```

Fig. 9 Knn-TF-IDF cosine (Lem-Ste)

Overall, we have two different vector representations to be fitted the K-nearest neighbor classifier, here the results based on the distance metric and word root which has proportional effect to the accuracy of the model.

TABLE II. RESULT

Model	**Results**			
	Distance Metric	***Word root***	***Accuracy (%)***	***F1-scores***
KNN-BoW	minkowski	Steaming	**76.55**	**77**
KNN-BoW	cosine	Lemmatizing	**71.18**	**71**
KNN-BoW	minkowski	Lemmatizing	**76.66**	**77**
KNN-BoW	cosine	Steaming	**71.63**	**72**
KNN-TF-IDF	cosine	Steaming	**71.62**	**72**
KNN-TF-IDF	cosine	Lemmatizing	**71.12**	**71**

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 naïve 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

[1] Shimagaki, J., Kamei, Y., Ubayashi, N. and Hindle, A., 2018, October. Automatic topic classification of test cases using text mining at an Android smartphone vendor. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1- 10).

[2] Hasan, D. A., Hussan, B. K., Zeebaree, S. R., Ahmed, D. M., Kareem, O. S., & Sadeeq, M. A. (2021). The impact of test case generation methods on the software performance: A review. International Journal of Science and Business, 5(6), 33-44.

[3] Shan, L. and Zhu, H., 2009. Generating structurally complex test cases by data mutation: A case study of testing an automated modelling tool. The Computer Journal, 52(5), pp.571-588.

[4] Edwards, S.H., 2004, March. Using software testing to move students from trial-and-error to reflection-in-action. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (pp. 26-30).

[5] Zhang, X.Y., Zheng, Z. and Cai, K.Y., 2018. Exploring the usefulness of unlabelled test cases in software fault localization. Journal of Systems and Software, 136, pp.278-290.

[6] Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P., Bertolino, A. and Li, J.J., 2013. An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8), pp.1978-2001.

[7] Stocks, P.A. and Carrington, D.A., 1993, May. Test templates: A specification-based testing framework. In Proceedings of 1993 15th International Conference on Software Engineering (pp. 405-414).IEEE.

[8] Ammann, P. and Offutt, J., 1994, June. Using formal methods to derive test frames in category-partition testing. In Proceedings of COMPASS'94-1994 IEEE 9th Annual Conference on Computer Assurance (pp. 69-79). IEEE.

[9] Hartman, A. and Nagin, K., 2004. The AGEDIS tools for model based testing. ACM SIGSOFT Software Engineering Notes, 29(4), pp.129-132.