Android Based Multi-Lingual SMS Spam Prototype Design and Development

KIPKEBUT ANDREW[1], THIGA MOSES[2], OKUMU ELIZABETH[3]
[1, 2, 3] School of Science and Engineering, Kabarak University -Kenya

Abstract- Most spammers are constantly developing new sophisticated methods, rendering previous techniques obsolete. A thoughtful deficiency in most sms spam detection methods is lack of satisfying accuracy, reliability, low performance and comprehensibility especially when individual classifiers are used, these remains important aspects to be considered for an optimal model development. Sms spam detection using machine learning techniques is a new approach especially in ubiquitous computing devices such as mobile phones, moreover the design of short message spam detection techniques in a mobile platform is challenging task due to the non-stationary distribution of the data and the multi-lingual nature of text messages from users. It is in this background that the research proposes a multi-stage ensemble hybrid prototype sms spam detection model for a mobile environment using machine learning techniques. It involves enhanced use of pre-processing techniques, content-based feature engineering techniques, multilingual natural language processing, data training and testing. The effectiveness of the proposed prototype is empirically validated using ensemble classification methods that gave an overall classification accuracy of 98.2606%.

Indexed Terms— Algorithm, Detection, Ensemble Feature engineering, Machine learning, SMS.

I. INTRODUCTION

Mobile communication devices have been the most adopted means of communication both in the developed and developing countries with its penetration more than all other electronic devices put together [1]. Every mobile communication device needs some type of mobile operating system to run its services: voice calls, short message service, camera functionality, and so on. Google Android, Apple IOS and Microsoft Windows Phone are most common types of mobile operating systems [2].According [3] Android's percentage share in the market is increasing at an alarming rate, Google android is rapidly taking its place in the eyes of today's youth and every person today wants affordable and the best operating system which Android guarantees to provide to its users. Dollah et al [4] research on mobile device ownership, the research indicated that 159 out of 225 respondents (70.4%) had Android based device for their mobile phones followed by others (19%), Apple iPhone (11.1%), and Windows Phone (2.2%). The least was Blackberry mobile phone with a percentage of (1.3%) only. The possible factors that led to the high ownership rate of Android based mobile phones may be attributed to the competitive price of these devices. However, in lieu of this finding, the simplicity, reliability and functionality may be best attributed to others, such as, Apple iPhone and windows Phone. A research conducted by [5] on the increasing market penetration of mobile devices, such as smartphones and tablets, poses additional challenges on the design of distributed systems. Due to the heterogeneous environment consisting of both, mobile and fixed devices, a multitude of effects on different scales need to be considered. Microscopic effects, such as an individual user's interaction with the device, as well as macroscopic effects, such as scalability with the number of users have an impact on the system's performance. The combined evaluation of micro- and macroscopic effects requires both, simulations and prototypical deployments. It is also common practice for developers of user-facing software to transform a mock-up of a graphical user interface (GUI) into code. This process takes place both at an application's inception and in an evolutionary context as GUI changes keep pace with evolving features (Moran et al., 2018). Using stable IDE software such as android studio developers can develop applications for phone that accelerate development and help you build the highest-quality apps (Hagos, 2018).

II. RELATED WORK

There is a growing need for automated testing techniques aimed at Android apps. A critical challenge is the systematic generation of test cases. One method of systematically generating test cases for Java programs is symbolic execution. But applying symbolic execution tools, such as Symbolic Pathfinder (SPF), to generate test cases for Android apps is challenged by the fact that Android apps run on the Dalvik Virtual Machine (DVM) instead of JVM. In addition, Android apps are event driven and susceptible to path-divergence due to their reliance on an application development framework [6]. Recent introduction of a dynamic permission system in Android, allowing the users to grant and revoke permissions after the installation of an app, has made it harder to properly test apps. Since an app's behavior may change depending on the granted permissions, it needs to be tested under a wide range of permission combinations. At the state-of-the-art, in the absence of any automated tool support, a developer needs to either manually determine the interaction of tests and app permissions. The study by [7] focuses on mapping the testing techniques for mobile applications. Additionally, the study emphasizes the need for testing metrics to be included and adhere to address mobile application testing lifecycle conformance. The major lags in the mentioned techniques for a smartphone application testing lie in the automation of testing. According to the authors, this is an emerging and future of mobile and other testing, but very few studies have implemented this technique over complex applications. Automated testing techniques perform well over small to medium and simple mobile applications, but very little work is done over the implementation and analysis of this technique over complex mobile applications. On android ecosystem there is a large selection of different testing tools, libraries and frameworks available for Android. It is hard to understand which tool to use for what type of tests whether unit testing, mocking, user interface testing or integration testing [8].

III. METHODOLOGY

In this section the following very important methods are outlined for the design and development of the prototype: - Text preprocessing, feature engineering techniques, model design, development and testing. The Meta model as per the conceptual diagram determines the class of data as either spam or not based on data set provided. Finally an android based prototype is implemented on the client phone to detect the class for the messages.

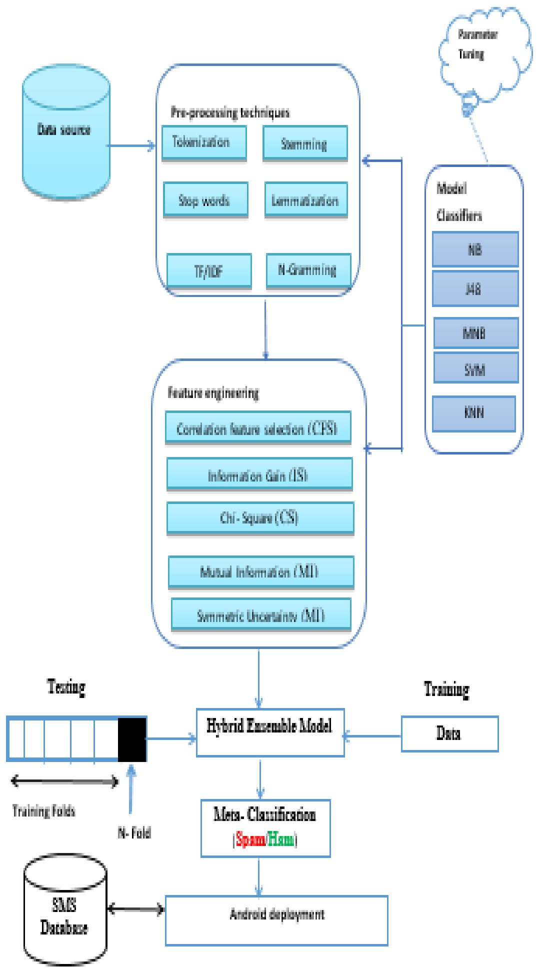

Figure 1: Conceptual framework

A. *Model prototype development and Implementation*

The android-based client-side SMS SPAM detection model implementation of this research is done using open-source machine learning libraries, Java language, JavaScript, XML on android studio IDE with SQLite database for the backend. The model also includes an additional cloud translation API module for multilingual language processing. The final implementation includes the following main modules.

- Listening of incoming messages.
- Enhance Text pre-processing

- Enhanced Feature engineering methods
- Training and testing of the model.
- Message classification and clustering.

As part of software validation, module testing is adopted as a software testing type, it checks individual subprograms, subroutines, classes, or procedures in a program. Instead of testing whole software program at once, module testing recommends testing the smaller building blocks of the program, Module testing avoid redundant activities and checks. This testing is done using JUnit. JUnit is a testing framework for the Java programming language. JUnit has been important in the development of test-driven development [9], it covers faults and defects for a given software.

B. Prototype Design and Implementation

In this section an overview of the ensemble hybrid client side sms spam detection modules are presented as per the model. After having evaluated the model, the researcher implemented it as per the reasons aforementioned previously. The application architecture is shown in figure 2. The architectures involves the initial stage of receiving incoming raw messages which are pre preprocessing (stop words, tokenization and stemming), TF and IDF, training and testing. To achieve this implementation it was necessary to comprehend well not only how the hybridized ensemble model is designed, but also an in depth pre-processing and feature engineering procedures, the steps are illustrated below. The top-level pseudo code developed in Java language. The coding developed in this research work consisted of four modules. The model is implemented at the client-side portable android smart device GSM capabilities. The user must allow the application to listen to incoming messages through phone settings=> apps=>choose Spam detector app=> under permissions=> select allow sms .

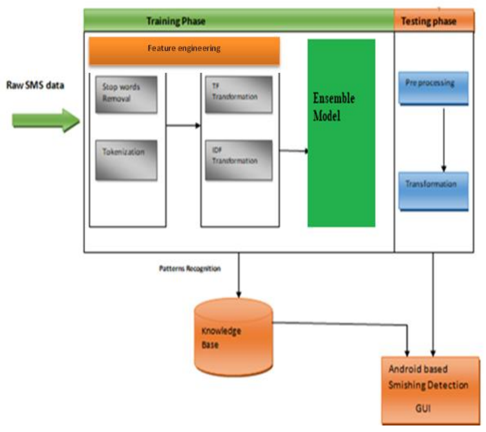

Figure 2: Android based Architecture

The GUI is developed using JAVA and android studio Integrated Development Environment (IDE) with SQLite database which contains the messages for training and testing .The system GUI conforms to user interface design principles that's satisfies the following user interface design principles ,Clear, Consistent, Efficient, Responsive and Reliable. The incoming text message is first translated into English from Swahili text messages when required, if the message is in English then there is no need of translation. Figure 3 Shows how this is done, If a message is detected as SPAM the user has an option of saving it or dismissing it, If the save option is selected the user can view the saved message in a different folder for future reference which also include the Spam contacts associated with the Spam messages received.

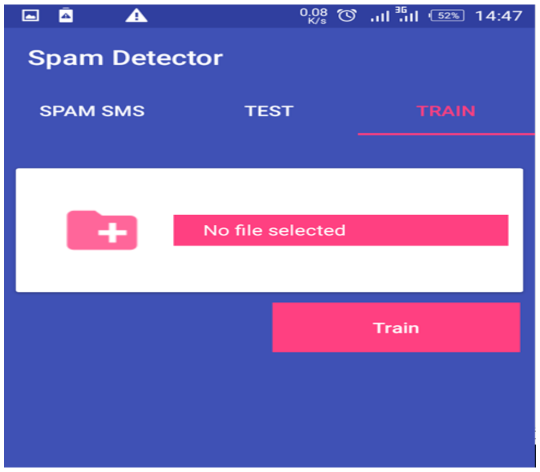

Figure 3 Training and testing example.

IV. RESULTS AND DISCUSSION

In this study it is clear that combining classifiers using stacking algorithm was observed to give a better accuracy compared to bagging and boosting since it provides more diversity of classifiers and also recorded high true positive and low false positive. A stacked model of Naïve Bayes, Artificial Neural Network and Support vector machine recorded the most optimal solution with highest precision of 98.3% and a low false positive of 0.064 %.

Table 1: Detailed accuracy of the Ensemble Hybrid Stacked model

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC	PRC-Area	Class
	0.992	0.073	0.988	0.992	0.990	0.927	0.960	0.987	Ham
	0.927	0.008	0.947	0.927	0.937	0.927	0.959	0.912	Spam
W.Avg	0.983	0.064	0.982	0.983	0.983	0.927	0.960	0.977	

This model also recorded a Kappa statistics of 0.9268, Mean absolute error of 0.0181, MCC of **0.927** ,Root mean squared error 0.13 and ROC of 0.960 .

A. *Prototype module Testing*

Testing is a very important process in any design and development of a software. It uncovers all the bugs generated by the software to make the application a successful product. In this thesis the prototype testing was done as per the modules using **JUnit** a module framework for Java based applications. It is an automation framework for module testing as well as user interface. It contains annotations such as @Test, @Before, @After etc. [10].This process is an important phase in software development lifecycle since it serves as the "Quality Gate" for the android application, the test summary report is an important deliverable which is prepared at the end of a Testing project. Further several metrics were used to help understand the test execution results, the status of test cases, defects among others. Defect Summary-Severity wise; Defect Distribution-Function/Module wise; Defect rejection ratio **(DRR)** and Defect leakage ratio**(DLR)** were also included as part of the software test report that including the use of Charts/Graphs for better visual representation.

Table 2: Test cases plan vs executed report

Tests cases planned	Test cases executed	Test cases passed	Test cases failed
25	20	18	2

The tests planned and tests executed report on table 2 allowed the researcher to optimally track the testing progress as per the test cases plan, number of executions, number of passed and failed test.

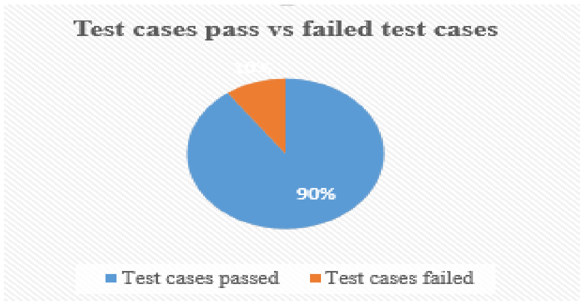

Figure 4: Test Cases vs. failed test cases

In general there were a total of 25 test cases plan as per the KLOC of the application, 20 test case were executed successfully, 18 (90%) of them passed the test and 2 (10%) of them failed the test. The 10 % in the failed test cases was due to server and network issues, and also from unresponsive scripts as illustrated in figure 5, these were resolved by modifying the Ac*tivityTestRule and* then re-executing the failed test cases again [11].

Table 3 : No of defects identified by status and severity

	Critical	Major	Medium	Cosmetic	Total
Closed	8	4	6	0	18
Open	0	0	0	3	3
				Total	21

In table 4 and figure 5 the defects opened (New, Assigned, Reopened, and Blocked) vs. closed report (Resolved, Closed, and canceled) displays the number of defects that were opened compared with the number of Defects that were closed. This report helps to determine the rate at which defects are being opened compared with the rate at which defects are being closed. In general there were 8 (44%) closed critical, 4 (22%) major, 3 (16.6%) medium and zero cosmetic defects .All Critical defects were closed since they represented those features that are most important to the system to function e.g. the training module. The cosmetic test case that represented 100 % of the open defect remained open since it only represented the aesthetic value of the project rather than the functionality.

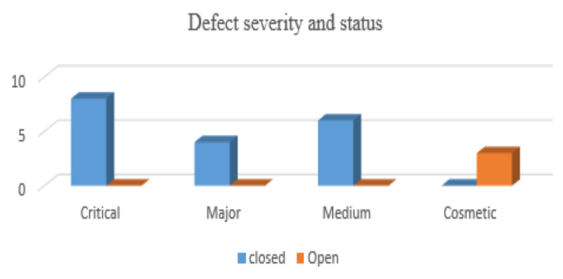

Figure 5: Bar chart representing defect severity.

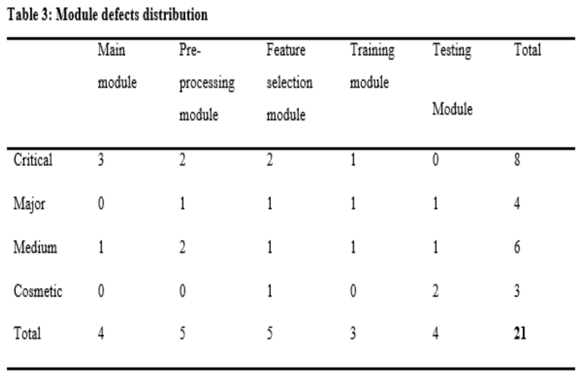
Table 3: Module defects distribution

	Main module	Pre-processing module	Feature selection module	Training module	Testing Module	Total
Critical	3	2	2	1	0	8
Major	0	1	1	1	1	4
Medium	1	2	1	1	1	6
Cosmetic	0	0	1	0	2	3
Total	4	5	5	3	4	**21**

In general the module defect for the main module registered a total of 4 defects (21%) , preprocessing module 5 defects (23%) , feature selection module 5 defects (23%) , training module 3 defects (13%) and testing module 4 defects (21%) of the total defects for critical , major , medium and cosmetic as shown in figure 6. The module defect distribution report is important since it displays the number of defects by status, helping the researcher track the progress of defects for the prototype and also in identifying and prioritizing the defects

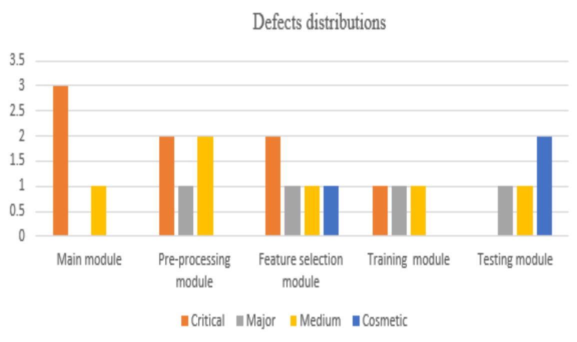

Figure 6: Defects distributions

After identifying the defects the next step was fixing the defects as per priority , once a defect has been resolved and verified, the defect status is changed to closed ,another important metric is to measure and evaluate the quality of a test execution - Defect rejection ratio (DRR) =(number of defects rejected/total number of defects raised)*100 and defect leakage ratio(DLR) =(number of defects missed /total defects of the application)*100 DRR was recorded as 0.0952 (9.52%) and DLR of 0.1423 (14.23%). The smaller value of DRR and DLR is, the better quality of test execution done [12].

V. CONCLUSION

The experiments conducted on the ensemble prototype method produced a better classification performance as compared to other existing model. This improvement was achieved because of the preprocessing techniques and the enhanced feature selection methods adopted. The prototype implementation and testing conducted on android device also proved that the generalized model is usable and can handle diverse text messages through multi-lingual processing.

REFERENCES

[1] Okediran, O. O., Arulogun, O. T., Ganiyu, R. A., & Oyeleye, C. A. (2014). Mobile operating systems and application development platforms: A survey. *International Journal of Advanced Networking and Applications*, *6*(1), 2195.

[2] Novac, O. C., Novac, M., Gordan, C., Berczes, T., & Bujdosó, G. (2017). Comparative study of Google Android, Apple iOS and Microsoft Windows phone mobile operating systems.

In *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)* (pp. 154-159). IEEE.

[3] Jain, V., & Sharma, A. (2013). The consumers preferred operating system: Android or iOS. *International Journal of Business Management and Research (IJBMR)*, *3*(4), 29-40.

[4] Foozy, C.F.M. & Ahmad, Rabiah & Abdollah, Mohd (2014). A framework for SMS spam and phishing detection in Malay language: A case study. *International Review on Computers and Software*. 9. 1248-1255.

[5] Richerzhagen, B., Stingl, D., Ruckert, J., & Steinmetz, R. (2015, August). Simonstrator: Simulation and prototyping platform for distributed mobile applications. In *The 8th EAI International Conference on Simulation Tools and Techniques (ACM SIMUTOOLS 2015)* (pp. 99-108).

[6] Mirzaei, N., Malek, S., Păsăreanu, C. S., Esfahani, N., & Mahmood, R. (2012). Testing android apps through symbolic execution. *ACM SIGSOFT Software Engineering Notes*, *37*(6), 1-5.

[7] Zein, S., Salleh, N., & Grundy, J. (2016). A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software*, 117, 334-356.

[8] Morgado, I. C., & Paiva, A. C. (2019). The iMPAcT tool for Android testing. *Proceedings of the ACM on Human-ComputerInteraction*, 3(EICS), 1-23.

[9] Gromov, M. L., Prokopenko, S. A., Shabaldina, N. V., & Laputenko, A. V. (2019, June). Model Based JUnit Testing. In *2019 20th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM)* (pp. 139-142). IEEE.

[10] Dietterich, T., Domingos, P., Mitchell, T., Page, D., &Shavlik, J. (2016). Learning Bayesian Networks (part 3).Terms via iOS and Android Based Devices. International Journal of Interactive Mobile Technologies, 11(3).

[11] Hagos, T. (2018). Android studio. In *Learn Android Studio 3* (pp. 5-17). Apress, Berkeley, CA.

[12] Mesquita, D. P., Rocha, L. S., Gomes, J. P. P., & Neto, A. R. R. (2016). Classification with reject option for software defect prediction. *Applied Soft Computing*, *49*, 1085-1093.