Nigerian Journal of Technology (NIJOTECH)
Vol. 39, No. 2, April 2020, pp. **542 - 552**

Print ISSN: 0331-8443, Electronic ISSN: 2467-8821
www.nijotech.com
http://dx.doi.org/10.4314/njt.v39i2.25

ANOMALY DETECTION OF ANDROID MALWARE USING ONE-CLASS K-NEAREST NEIGHBOURS (OC-KNN)

B. A. Gyunka[1,*] and S. I. Barda[2]
[1,] DEPARTMENT OF BRANCH OPERATIONS, CENTRAL BANK OF NIGERIA, KANO, KANO STATE, NIGERIA
[2,] DEPARTMENT OF STATISTICS, CENTRAL BANK OF NIGERIA, KANO, KANO STATE, NIGERIA
E-mail addresses: [1] *gyunkson@gmail.com,* [2] *barda65@icloud.com*

ABSTRACT
The advent of the Android Operating System has recorded a remarkable ground-breaking opportunities in the Technological world. However, this great breakthrough also has a very dark side – an uncontrollable rapid continuous releases of malware in the wild, targeted at the platform and all its information and human assets. The misuse-based approaches adopted by many detection systems do no longer have the rigidity and the tenacity to accommodate the rapid successive releases of malware that come in great volume in order to keep up with active defenses against unknown and novel attacks. Systems that are capable of offering anomaly protection are thus in dire need. This study developed a normality model that is based on One-Class K-Nearest Neighbour (OC-kNN) Machine Learning approach for anomaly detection of Android Malware. The OC-kNN was trained, using WEKA 3.8.2 Machine Learning Suite, through a semi-supervise procedure that contained mostly benign and a very few outliers Android application samples. The OC-kNN had 88.57% true performance accuracy for normal instances while 71.9% was recorded as true performance accuracy for outliers (unknown) instances. The false alarm rates for both normal and outlier's instances were recorded as 28.1% and 11.5%. The study concluded that a One-Class Classification model is an effective approach to be used for the detection of unknown Android malware.

Keywords: *Android; Machine Learning, Malware, One-Class Classification, Anomaly Detection, Outlier Detection, Novelty Detection, Concept Learning, k-NN*

1. INTRODUCTION

Machine learning approaches involving the conventional binary or multi-class classification algorithms are commonly known to classify an unknown object into one of several pre-defined categories or classes used in the training phase [1]. For times when the unknown object fails to match with any of those pre-defined categories, a problem arises which makes the system unable to correctly identify or classify these new and unknown instances, thus allowing them to slip away quietly without a definite tag. This problem is known as a zero-day attack (novel threat) which necessitates the need for an anomaly detection model that can mostly be derived from a one-class classification technique (OCC) [2]. Other research themes used for OCC are Outlier Detection, Novelty Detection [3], [4], and Concept Learning [1, 5, 6] defined this novelty detection approach as concept learning techniques that proceed by recognizing only the positive instances of a concept rather than focusing on the difference that exist between the positive and negative instances thus making the technique to have very negligible need for the presence of any negative instances.

Problems that involve outliers or novelty detection can be viewed as binary classification problems that allow behaviour to be classified as normal when it satisfies the rules of normalcy, but when the behaviour derails from those rules and becomes abnormal, it is then classified as suspicious. Anomalies are occurrences or events which do not

* Corresponding author, tel: +234 806 209 4806

match, quantitatively, with the pattern of what is considered to be normal, according to a domain expert [7]. In Machine Learning or Data Science, they are referred to as data points that do not conform to an expected pattern of the items in the data set. In learning situations, anomaly detection has become the most beneficial given that it allows the machine to appropriately approximate the underlying distribution of the provided regular instances or normal behaviour, in order to produce a concise model of normality [8, 9].

Finding and detecting these irregular patterns, also known as outliers, is known as Anomaly detection [10]. One-Class Classification (OCC), also called Single-Class Classification (SCC), is the simplest way of employing supervised classification methods for the purpose of anomaly detection. One-class algorithms are based on recognition given that their main purpose is to recognize data from a particular class while rejecting data from all other classes. However, the anomaly detection employs mostly a semi-supervised technique in which the training sets for the learning algorithm consist of only the normal data class without any anomalies. This is important because the classical method to anomaly detection is to work out a precise description of normal data, so any new instance or example that arrives is contrasted with the model of normality and then an anomaly score is worked out to describe the degree of deviation from the average data instance, an so if the deviation exceeds a predefined threshold, the instance is then considered an anomaly or an outlier [8]. When the model is built based on the class of normal data set (that is normal class model), any deviation from this model will be classified as anomalies. These techniques are known as one-class classification. This study focuses on the use of One-Class k-Nearest Neighbours for the detection of novel Android malware. The process of semi-supervised learning was adopted as WEKA 3.8.2 machine learning suite was employed.

2. LITERATURE REVIEW

2.1 Mobile Operating Systems

The advent of mobile devices have given a great technological boost to the way people do businesses and in the way they relate with fellow humans and nature in general. However, these hand held devices' interactions and relationships with their users is hugely affected by the nature, users' friendliness, and security of the Operating Systems (OS) running in them. A mobile OS is a software or program that powers Smartphones and Tablets and creates an environment for other applications and programs to run on [11]. The functions and features that are found on a mobile device, such as thumb wheel, keyboards, email, and text messaging are all determined by the operating system. Different kinds of mobile OS exist which includes Apple iOS, Google Android, Nokia Symbian, Hewlett-Packard WebOS (formally Palm OS), BlackBerry RIM (Research in Motion), Microsoft Windows Phone (Microsoft Windows 8), Ubuntu and Firefox [12–14]. Table 1 provides more distinctive properties and differences that exist between major Mobile OS and it also shows that the Android OS has the highest future prospects, in terms of market shares, as can also be observed in Figure 1.

Majority of the existing mobile OS have direct connection to specific hardware, with little or no flexibility. However, users have the ability to jailbreak or root some devices in order to gain full control to install a different OS or unlock restricted applications. Due to the open-source nature and user friendliness of the Android OS, its popularity has risen far above all other mobile OS as observed in the continuous increase in the volume of Mobile OS Market shares. A lot of malware are therefore targeted at this platform given that it has drawn the highest attention in the world of mobile devices' users.

2.1.1 The Android Operating System

The Android operating system (OS) is the most popular amongst all other mobile operating systems, as shown in Table 2 and Figure 1. It was built based on Linux 2.6 kernel and managed by the Open Hands*et al*iance [15]. The open nature of Android OS has attracted the attention of immeasurable number of developers who are working on different kinds of projects, both legitimate and illegitimate, on the platform. The Linux kernel serves as a layer of abstraction between the hardware and the rest of the hardware stack and it also enables access to essential services such as security, memory management, process management, network stack, and driver model. It also creates a support for the Dalvik virtual machine's functionality. The Libraries, which is the next layer up after the kernel, is divided into Android Runtime and applications libraries.

Table 1: Mobile Operating Systems Comparison [12]

	MOBILE OPERATING SYSTEMS							
Parameter	Android	iOS	System	Blackberry	Windows Phone	WebOS	Ubuntu	Firefox
OS Family	Linux	Darwin	RTOS	QNX	Window CE-7 Window NT-8	Linux	Linux	Linux
Vendor	Open Handset Alliance, Google	Apple, Inc	Accenture on behalf of Nokia (historically Symbian Ltd. And Symbian Foundation)	Blackberry Ltd.	Microsoft	Open WebOS community contributors, LG Electronics, Previously HP (Hewlett-Packard) & Palm	Canonical Ltd. Ubuntu community	Mozilla Foundation
Environment (IDE)	Eclipse (Google)	XCode (Apple), AppCode	QT, Carbide.C++, Vistamax, Eclipse	Eclipse, BlackBerry JDE	Visual Studio	Eclipse	Ubuntu SDK	WebIDE
SDK Platform	Linux, Mac OS X and Windows	Mac OS X using iOS SDK	Windows XP Professional SP2; Vista & 7 for some SDKs	Linux, Windows, Mac OS X	Windows	OS X, Ubuntu, Windows	Ubuntu Desktop using Ubuntu SDK	All where Firefox is available
License	Free and opensource, but usually bundled with proprietary apps and drivers	Proprietary EULA except for open source components	Proprietary, previously licensed under EPL	proprietary	Proprietary	Apache License	Free and open-source, mainly the GPL	Free and open-source, mainly the MPL; apache
Written In	C, C++, Java	C, C++, Objective-C, Swift	C, C++, Java ME, Python, Ruby, Flash Lite	C, C++, HTML5, Java script, CSS, Action Script, Java	C#, VB.NET, F#, C++, Jscript	JavaScript, CSS, HTML, C and C++	HTML5, QML, C, C++	HTML5, CSS, JavaScript, C++
Initial Release	September 23, 2008	June 29, 2007	1997	January, 1999	October 21, 2010	June, 2009	October 20, 2004	April 23, 2013
Runs on	Smartphones, Tablet, Computers, TVs, Cars and wearable devices	iPhone, iPad, iPod Touch	Smartphones	Smartphones	Personal Computers, Smartphones, Server Computers and Embedded Devices	TVs and Smart Watches	Personal Computers, Servers, Smartphones, Table computers (Ubuntu Touch), Smart TVs (Ubuntu TV)	Smartphones, Tablet and Computers
Application Store	Google Play	App Store	Nokia Ovi Store	BlackBerry World	Windows Phone Store	Palm App Catalog	Ubuntu Store	Firefox Marketplace, Web URL
GUI	Android	Cocoa Touch	Avkon	Cascades	Visual Studio	Graphical (Luna)	Ubuntu SDK	Firefox browser, Firebug
Future Prospect	Very High	High	Low	Low	Medium	Low	Low	Low

The Android runtime is made up of the Dalvik Virtual Machine (DVM) and the core libraries. The VM gives the Android device the ability to run instances of multiple applications as separate processes [16],

having unique UID/GID (User ID/Group ID) and data storage. This prevents each application from having rights to access data storage of another application except through a different procedure or an express permission by the user of the device [17].

Moreover, apps that have been signed with the same private key or share the same public certificate can share the same UID and process in Android. Although dalvik VM is very instrumental in aiding processes and applications isolations, it however does not play a role in Android security as it is not a security boundary. More so, the Dalvik VM is only found in older versions of Android but from version 5.0, the DVM has been substituted with ART – Android Runtime [19]. Table 3 shows all the different versions of the Android OS since inception to 2018.

2.2 K-Nearest Neighbour (k-NN) Algorithm

KNN is known typically as lazy learner. It is so called, not because it's obvious simplicity, but for the reason that it does not learn a discriminative function from the data provided for training but it rather memorizes the given dataset for training, thus KNN does not have any training time. What it does is to store the data meant for training and then wait until data for testing is provided and the classification is performed based on the most related data in the training data that was stored [23]. It doesn't use the training data points to do any generalization (i.e., it keeps all the training data). The training phase for K-NN is extremely fast because it does not really have a training phase. For K-NN to make prediction, it searches for the nearest neighbours in the entire training dataset.

Table 2: Worldwide Smartphone Shipment OS Market Share Forecast [18]

Year	2017	2018	2019	2020	2021	2022	2023
Android	85.1%	85.1%	86.7%	86.6%	86.9%	87.0%	87.1%
iOS	14.7%	14.9%	13.3%	13.4%	13.1%	13.0%	12.9%
Others	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
TOTAL	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Table 3: List of Android version releases [20]

S/N	VERSION	VERSION NAME	RELEASED DATE
1.	Android 1.5	Cupcake	April 27, 2009
2.	Android 1.6	Donut	September 15, 2009
3.	Android 2.0-2.1	Eclair	October 26, 2009
4.	Android 2.2-2.2.3	Froyo	May 20, 2010
5.	Android 2.3-2.3.7	Gingerbread	December 6, 2010
6.	Android 3.0-3.2.6	Honeycomb	February 22, 2011
7.	Android 4.0-4.0.4	Ice Cream Sandwich	October 18, 2011
8.	Android 4.1-4.3.1	Jelly Bean	July 9, 2012
9.	Android 4.4-4.4.4	KitKat	October 31,2013
10.	Android 5.0-5.1.1	Lollipop	November 12, 2014
11.	Android 6.0-6.0.1	Marshmallow	October 5, 2015
12.	Android 7.0-7.1.2	Nougat	August 22, 2016
13.	Android 8.0-8.1	Oreo	August 21, 2017
14.	Android 9.0	Pie	August 6, 2018

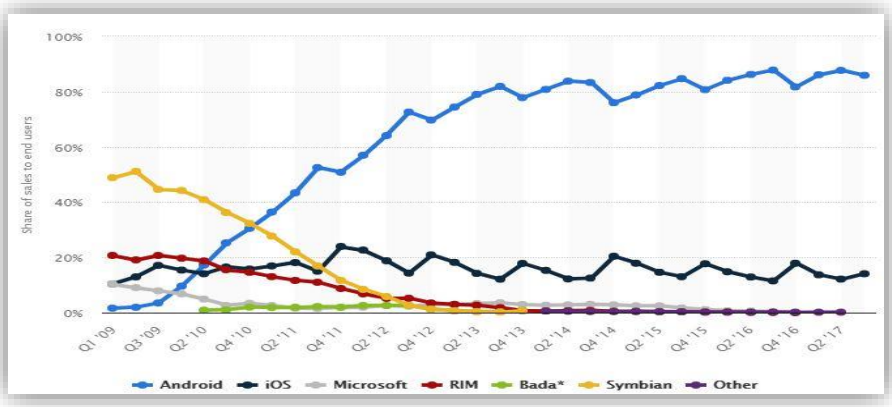

Figure 1: Global sales of Smartphones by OS since 2009 – 2018, [21]

In this work, K-NN was implemented in the WEKA environment using IBK classification filter on the given dataset. Unlike eager learners, the lazy learner takes less time in the training phase but more time in predicting.

2.3 One-Class Classification (OCC) Approach

The main difference that exist between OCC and binary/multi-class classification problems is the fact that in OCC, the negative class is either not present or not properly sampled while focus is placed mostly on the positive class (that is, the target class) or cases – resulting in definite determination of only one side of the classification boundary by using the positive data [1], [5]. This characteristic makes the OCC problem much harder than the problem of conventional two-class classification. What OCC does mostly is to define a classification boundary around the positive (or target) class in such a way that it can accept as many objects as possible from the positive class while the chances of accepting non-positive (or outlier) objects is highly minimized. Thus, all One-Class classifiers are trained on target dataset only and are tested on both target data and other non-target data that remains [22]. Pandey (2017) noted that binary or multi-class classifiers are mostly discriminatory in nature, given that they work by learning to discriminate between classes by using all data classes to produce a hyperplane and then use the hyperplane to label samples that are new, while on the other hand, the one-class classifiers work based on recognition given that their aim is to recognize data from a particular class, and then reject data from all other classes.

2.3.1 OCC Algorithms Categorization

A one-class classifier takes as input a labelled data se D and outputs classifications $\{Outlier, inlier\}$. One-class classifiers normally fall into either of these categories: density estimation methods, boundary methods and reconstruction methods [3].

a. Density Estimation Methods: the main aim of this method is to estimate the complete distribution of the target data. A rejection threshold value is then fixed in order to screen out, as rejects, points that are located in the far tails of the distribution. The drawback of these methods is that a sufficiently large sample of inliers (positive instances) is required before a good estimation can be produced [25]. Example of density based methods includes Gaussian distribution, Parzen density estimation, and Gaussian mixture models.

b. Boundary methods: in these methods, construction of a boundary, such as a sphere, around the target data is the main aim rather than estimating the distribution which requires very large number of samples [25]. Any points found falling outside the limits of the boundary are rejected. Given that the interest is on defining this boundary, obtaining large samples to completely represent the inliers class is not necessary. Example of the boundary methods includes k-nearest neighbours (kNN), support vector data description (SVDD) [26], and the Linear Programming Distance Data Description (LP) [27].

c. Reconstruction methods: these methods aim at developing a simplified representation of the data through clusters or principal components. The method is used to model the training data via the use of a generating process [25]. Examples of these methods include k-means, principal components analysis, self-organizing maps and auto-encoder networks [6].

When it comes to one-class classification performance evaluation, true negative and false positive, which are the traditional classifier evaluation metrics, cannot be computed due to the reason that only positive examples (class) exist [28]. The receiver operating characteristic (ROC) curve is one very useful performance assessment tool in classification and anomaly detection tasks [3].

2.3.2 One-Class K-Nearest Neighbours (OC-KNN)

The modification of the usual multi-class or binary Nearest Neighbour classifier enables the formulation of the One-Class k-Nearest Neighbours (OC-kNN) classifier which focuses on learning the target or positive (benign) class only [29]. The operation of the classifier involve storing all the training examples as its model, then for a given example z, the distance to its Nearest Neighbour $y(y = NN(z))$ is calculated as $d(z, y)$. The new sample belongs to the target class when:

$$\frac{d(z,y)}{d(y,NN(y))} < \delta \qquad (1)$$

Where $NN(y)$ is the Nearest Neighbour of y. In other words, it is the Nearest Neighbour of the Nearest Neighbour of z. The default value of δ is 1 but can be chosen to satisfy the needed requirement. The

average distance of the k nearest neighbours is considered for the OC-KNN implementation.

2.4 Related Work

Bezerra, et al., (2019) proposed a host-based method for the detection of Botnets in Internet of Things (IoT) Devices using One-Class Classification (OCC) approach that was able to model only the legitimate behaviour of a device in order to detect any deviations. The proposed system is underpinned by a novel agent-manager architecture based on HTTPS, which is able to stop the IoT device from being overloaded by the training activities. The One-Class algorithms evaluated are Elliptic Envelope, Isolation Forest, Local Outlier Factor, and One-Class Support Vector Machine (SVM). Yerima & Sezer, (2018) proposed a novel classifier fusion approach called DroidFusion that is based on a multilevel architecture which enables effective machine learning algorithm combination in order to produce an improved accuracy. DroidFusion works by training the base classifiers at a lower level in order to create a model and then a set of ranking-based algorithms are applied on their predictive accuracies at the higher level so as to generate schemes for combination in which one was chosen to build a final classification model. The authors utilized five base classifiers: J48, REPTree, Random Tree-100, Random Tree-9, and Voted Perceptron.

Rashidi, Fung, & Bertino, (2018) worked on a framework for the detection of Android malicious application that was based on Support Vector Machine (SVM) and Active Learning technologies. In order to build an active learning model, the authors made use of expected error reduction query strategy so as to combine Android malware new informative instances and to retrain the model in order to be able to do adaptive online learning. To evaluate their model, the authors utilized the DREBIN benchmark malware dataset via a set of experiments and their findings revealed that their framework could detect new malware more accurately. Ucci, Aniello, & Baldoni, (2018) conducted a survey with focus on showing the application and the use of ML methods in the analysis of malware. The Authors observed that machine learning is one of the most common techniques adopted in literatures for the analysis of complex malware. Idrees, Rajarajan, Conti, Chen, & Rahulamathavan, (2017) proposed PIndroid, which was a novel Android malware apps detection framework that uses permissions and intents features for the training of models and further employed classifier fusion technique to combine the classifiers together for an improved performance. Dong,

(2017) worked using permissions as his primary features to develop a novel detection system for Android malware.

The Author combined machine learning algorithms such as Logistic Regression Model, Tree Model with Ensemble techniques, Neural Network and finally an ensemble model to find the patterns and more valuable information. Yerima, Sezer, & Muttik, (2016) proposed a composite classification model using a parallel combination of heterogeneous classifiers for Android malware detection which employed static features. The classifiers deployed are Decision Tree (Tree based), Naïve Baye (probabilistic), Simple Logistics (function-base), PART (Rule-based) and RIDOR (Rule-based). Four classifier combination approaches were compared together, that is Average of Probability, Maximum Probability, Product of Probability, and Majority Vote, using the classification algorithms. The composite model was aimed at enabling an enhancing early detection model for Android malware which has improved accuracy and that can provide a quicker white box analysis by means of more interpretable constituent classifiers.

3. METHODOLOGY

One-class k-Nearest Neighbours (OC-kNN), was used in developing the outlier detection model. Research has shown that OC-kNN and support vector machine (SVM) (and its derivatives like Support Vector Data Description (SVDD)), are the top-most choices for one-class classification problems [4], [25] but OC-kNN has practically proven best suitable for the data set used in this study, hence its selection. k-NN is a distance-based outlier detection technique and it works based on the assumption that normal data points have close neighbours in the "normal" training set, while novel (new) data points are located far from those points [37]. The Waikato Environment for Knowledge Analysis (WEKA) Machine learning analysis suites, version 3.8.3, was used for the learning process. The k-NN classifier (iBk) was selected under OneClassClassifier meta algorithm and the value of k was set to 3 neighbours while Linear Nearest Neighbours Search (LinearNNSearch) was selected as the Nearest Neighbor Search Algorithm, and

Euclidean Distance was selected and used as the Distance Function with a threshold of $\mu = 0.1$. The target was set as *benign* (note that the spelling and alphabet characters have to be exactly the same with the one represented under the class label in the dataset). Figure 3 illustrates the algorithm for the one-class kNN classifier.

3.1 The Detection Framework

Figure 2 shows the proposed detection system which involve experimental dataset collection and feature extractions through Reverse Engineering (RE) procedures; Pre-processing of extracted features through data cleaning and features selection; Training dataset creation through unary features vectors matrix formulation and the Machine Learning experimental phase for unary algorithm training and classification with One-Class Classifier, that is the One-Class kNN. The end result is the creation of the Normality model which was able to classify subsequent future applications as either benign or malicious.

3.2 Experiment: One-class Anomaly Detection Model

One-class k-Nearest Neighbours (OC-kNN), as discussed in Section 2.3.1, was used to develop the outlier detection model. The value of k was set to 3 neighbours, Linear Nearest Neighbours Search (LinearNNSearch) was selected as the Nearest Neighbor Search Algorithm, and Euclidean Distance was selected and used as the Distance Function, and a threshold, $\mu = 0.1$. Figure 3 illustrates the algorithm for the one-class kNN classifier and WEKA machine learning suite was used for feeding data to the algorithm and the model testing. The algorithm is the standard One-Class kNN algorithms used for the classification of Android Applications as either benign or suspicious (unknown). The inputs used for this algorithm, as shown in Figure 3, includes the pre-processed training dataset D_{tr} (that is the unary features matrix), testing dataset D_{te}, the number used for the nearest neighbours K and the number th used as the threshold measure for accepting outliers. I is an instance in the testing set and it is represented by a feature vector $(f_1(I), f_2(I), \dots, f_m(I))$ where $f_i(I)$ is an instance value for a given feature and m represents the number of discriminatory features. An Euclidean distance between an instance I of testing data and all instances of training dataset was computed in steps 5 and 6. Steps 7 to 11 aims at finding K nearest neighbours of N_1 in the training dataset D_{tr}. The average of all K distances was taken in step 12 and was named as D_2. The unary classification was performed in steps 14 to 17. A test instance I is considered as belonging to the target class (Benign) if the result of the ratio of distances D_1 and D_2 is less than the threshold measure th otherwise it will be classified as an outlier (unknown).

3.3 Performance Evaluation Criteria for OCC

For every learning algorithm used for model creation, there are criteria used for evaluating the performance of the model created. Cross-validation technique is mostly used and a Confusion matrix, which is generated from the classifiers responses, provides the parameters for classifiers evaluation. Table 4 illustrates the confusion matrix for the derivation of the performance metrics for OCC. Unlike the case with multi-class classifiers where the complete probability density of both classes must be known in order for the true error rate to be estimated, OCC only makes available the probability density of the positive class. This implies that the only class that can be minimized are the number of the positive class instances that are not classified by the One-Class Classifier (that is the false negatives, F^-). When examples and sample distribution from the outlier class instances are not represented in the dataset, it becomes impossible to estimate the number of outlier that the one-class classifier will classify (that is the false positive, F^+). Furthermore, it can be observed that since $T^+ + F^- = 1$ and $F^+ + T^- = 1$, the main challenge in OCC is that only T^+ and F^- can be conveniently estimated but less or nothing is known about F^+ and T^-.

This makes it thus important to have some limited amount of outliers class data in order to be able to estimate the performance and to generalize the classification accuracy of a one-class classifier [5]. In such scenario of imbalanced dataset, metrics most suitable for the evaluation of the classifiers performance are those that are class-independent which includes precision, recall, F-measure, sensitivity, specificity, geometric mean, ROC curve, AUC, and precision-recall curve [38]. The following performance metrics were used in evaluating the developed model;

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{TP + FN} = Recall = Sensitivity \quad (2)$$

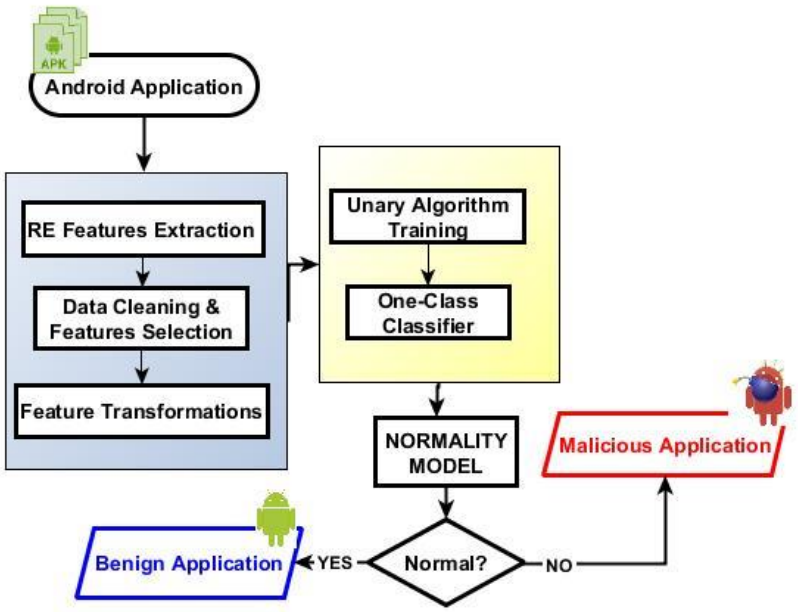

Figure 2: One Class Classification (OCC) Anomaly Detection Framework

STEP1: **Data: Training Dataset D_{tr}, Test Dataset D_{te}, Neighbors K, Threshold th**
STEP2: **Result:** List of class labels for test dataset C_{te}
STEP3: **Algorithm:** OneClassKNN (D_{tr}, D_{te}, th, K)
STEP4: **for each** instance $I \in D_{te}$ **do**
STEP5: $N1 \leftarrow NearestNeighbour(I, D_{tr})$
STEP6: $D_1 \leftarrow Euclidean\_Distance(N, I)$
STEP7: **if** $(K == 1)$ **then**
STEP8: $N2 \leftarrow NearestNeighbour(N1, D_{tr})$
STEP9: $D_2 \leftarrow Euclidean\_Distance(N1, N2$
STEP10 **else**
STEP11: $ND_1 \leftarrow Euclidean\_Distance(D_{tr}, N1)$
STEP12: $D_2 \leftarrow Average(ND_1, ND_2, \ldots, ND_K)$
STEP13: **end**
STEP14: **if** $(\frac{D_1}{D_2} > th)$ **then**
STEP15: $C_{te}.addClass(Unknown)$
STEP16: **else**
STEP17: $C_{te}.addClass(TargetClass)$
STEP18: **end**
STEP19 **End**
STEP20: Return C_{te}

Figure 3: One-Class k-Nearest Neighbours Algorithm

Table 4: One-Class Classification Confusion Matrix [2]

		Predicted class	
		Target Class	Outlier Class
Actual class	Target Class	True Positive, T^+	False Negative, F^+
	Outlier Class	False Positive, F^-	True Negative, T^-

$$True\ Negative\ Rate\ (TNR) = \frac{TN}{TN + FP} = Specificity \quad (3)$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{TP + FP} \quad (4)$$

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{FN + TN} \quad (5)$$

$$Precision = \frac{TP}{TP + FP} \quad (6)$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (7)$$

$$Accuracy\ (ACC) = \frac{TP + TN}{TP + TN + FP + FN} \quad (8)$$

$$Error\ Rate\ (ERR) = 1 - ACC \quad (9)$$

4. RESULTS ANALYSIS AND DISCUSSION

The Confusion Matrix, also known as the contingency table, showed that the model accurately recognized 401 instances as benign but erroneously recognized 41 instances as benign. This implied that a total of 442 instances where recognized as benign. On the other hand, the model accurately predicted 105 instances as outliers and 52 instances were erroneously predicted as outliers, resulting to a total of 157 instances being predicted as outliers by the normality models. Based on the confusion matrix results, Table 5 shows that

the One-Class k-Nearest Neighbours (OC-kNN) normality model has a high true positive recognition rate for benign instances as 0.885 (i.e., 88.5%), false positive recognition rate of 2.81%, an F-Measure of 0.896 (89.6%) and ROC Area of 0.815 (81.5%). For its predictive capacity on available outliers (i.e., instances that are unknown or outside the target class) found in the sample, the model has a true positive detection rate of 0.719 (i.e., 71.9%), a false positive rate of 1.14%, F-Measure of 70.5% and ROC Area of 85.5%.

The cumulative result of the confusion matrix shows that a total of 84.4741% instances were correctly classified while 15.5259% were incorrectly classified. The model took 0.84 seconds to build. The normality model has a detection accuracy of 88.5% for normal instances.

4.1 Comparison of OCC Model with Existing Models

To be able to point out the importance of the results in this study, Table 6 provides a performance comparison with other existing published works on Android malware detection via Machine Learning techniques. Yerima et al., (2016) made use of Naive Bayes, Decision Tree, Simple Logistic, Ridor and PART to form an ensemble model for Android Malware detection while Feng, et al., (2018) focused on Majority Vote.

These models were built mostly based on supervised binary learning. This implies that they are mostly only able to detect known Android malware. The One-Class normality models built in this work has a competitive advantage given that it was developed using normal or benign features, so it has a very strong capacity to detect anything, outliers, that is outside normal. The comparison table shows that the system in this work has 88.5% ability to detect new and unknown malware samples compared to the 96.4% and 97.2% of the other models that were mainly built based on known benign and malware samples.

Table 5: Prediction Accuracy Results for OC-kNN

EVALUATION METRICS					
TP Rate	FP Rate	Precision	F-Measure	ROC Area	Class
0.885	0.281	0.907	0.896	0.815	Benign
0.719	0.115	0.669	0.693	0.896	Outlier
0.845	0.240	0.849	0.847	0.834	Weighted Avg.

Table 6: Performance Evaluation of OCC Model with Existing Models.

Reference	ML Algorithms	Method	TP Rate	FP Rate	Accuracy	Error Rate
[36]	Naive Bayes, Decision Tree (J48), Simple Logistic, Ridor, PART	Ensemble Learning	0.964	0.040	0.962	0.038
[39]	SVM, Naive Bayes, kNN, Decision Tree, Boosted Tree, Extra Trees, Random Forest, Xgboost, Stacking	Majority Vote	0.972	0.023	0.975	0.026
OCC Normality Model	OCC-kNN	One Class Classification	0.885	0.281	0.736	0.264

5. CONCLUSION

Android malware are continually increasing in sophistications and variance, posing a great challenge to existing detection methods. This study took a deep delve into One-Class Classification techniques for anomaly detection of Android malware, as a countermeasure to the emerging unknown malware variances. The One-Class k-Nearest Neighbours that was adopted and trained using unary features from specifically benign Applications, proved to be effective techniques against novel malware. The result showed to be very effective as it recorded accuracy in detection rate of 88.5% for outliers, an Error rate of 2.4% and a false alarm rate of 1.14%. The false alarm was quite very low and insignificant, implying a great strength in the detection accuracy of the normality model. Thus, the studies conclude by recommending the One-Class Classification model for an effective detection of unknown Android malware. Future work will be focused on increasing the feature sets to include APIs for the benign applications in order to have a wider features sample.

6. REFERENCES

[1] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *In Irish Conference on Artificial Intelligence and Cognitive Science*, 2009, pp. 188–197.

[2] D. M. J. Tax, "One-class classification: Concept learning in the absence of counter-examples, PhD Thesis," Delft University of Technology, 2001.

[3] F. Ratle, M. Kanevski, A.-L. Terrettaz-Zufferey, P. Esseiva, and O. Ribaux, "A Comparison of One-Class Classifiers for Novelty Detection in Forensic Case Data ⋆," in *In International Conference on Intelligent Data Engineering and Automated Learning*, 2007, pp. 67–76.

[4] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[5] S. S. Khan and M. G. . Madden, "One-Class Classification: Taxonomy of Study and Review of Techniques," *Knowl. Eng. Rev.*, vol. 29, no. 3, pp. 345–374, 2014.

[6] N. Japkowicz, C. Myers, and M. Gluck, "A novelty detection approach to classification," in *International Joint Conference on Artificial Intelligence*, 1995, vol. 1, pp. 518–523.

[7] J. H. M. Janssens, "Outlier Selection and One-Class Classication, PhD Thesis," Tilburg University, 2013.

[8] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward Supervised Anomaly Detection," *J. Artif. Intell. Res.*, vol. 46, pp. 235–262, 2013.

[9] J. Abah, O. . Waziri, M. . Abdullahi, U. . Arthur, and O. . Adewale, "A Machine Learning Approach to Anomaly-Based Detection on Android Platforms," *Int. J. Netw. Secur. Its Appl.*, vol. 7, no. 6, pp. 15–35, 2015.

[10] M. Ved, "Outlier Detection and Anomaly Detection with Machine Learning," 2018. [Online]. Available: https://medium.com/@mehulved1503/outlier-detection-and-anomaly-detection-with-machine-learning-caa96b34b7f6. [Accessed: 23-Dec-2018].

[11] V. Beal, "Mobile Operating Systems (Mobile OS) Explained," *Hardware & Software*, 2018. [Online]. Available: https://www.webopedia.com/DidYouKnow/Hardware_Software/mobile-operating-systems-mobile-os-explained.html. [Accessed: 18-Jun-2019].

[12] K. Bala, S. Sharma, and G. Kaur, "A Study on Smartphone based Operating System," *Int. J. Comput. Appl.*, vol. 121, no. 1, pp. 17–22, 2015.

[13] V. Kamboj and H. Gupta, "Mobile Operating Systems," *Int. J. Eng. Innov. Res.*, vol. 1, no. 2, pp. 2277–5668, 2014.

[14] Renner Thomas, "Mobile OS - Features, Concepts and Challenges for Enterprise Environments," in *SNET Project Technische Universit at Berlin*, 2014, pp. 1–9.

[15] J. Lessard and G. C. Kessler, "Android Forensics : Simplifying Cell Phone Examinations," *Small Scale Digit. Device Forensics J.*, vol. 4, no. 1, pp. 1–12, 2010.

[16] M. Yates, "Practical Investigations of Digital Forensics Tools for Mobile Devices," *2010 Inf. Secur. Curric. Dev. Conf.*, pp. 156–162, 2010.

[17] H. Andrew, "An Introduction to Android Forensics," 2010. [Online]. Available: http://www.forensicmag.com/article/2010/04/introduction-android-forensics?page=0%2C0&pid=974#.UgPsHZbsZ3I. [Accessed: 27-Feb-2017].

[18] IDC, "Smartphone Market Share," *Device Market Trends*, 2019. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os. [Accessed: 18-Jun-2019].

[19] S. Hahn, M. Protsenko, and T. Müller, "Comparative Evaluation Of Machine Learning-Based Malware Detection on Android," in *Sicherheit 2016: Sicherheit, Schutz und Zuverl{ä}ssigkeit, Beitr{ä}ge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft f{ü}r Informatik e.V. (GI), 5.-7. April 2016, Bonn*, 2016, pp. 79–88.

[20] FutureTurbo, "Android Version Names: Every Os From Cupcake to Android P," *Smartphones*, 2018. [Online]. Available: https://turbofuture.com/cell-phones/Cupcake-Donut-Eclair-Froyo-Gingerbread-Honeycomb-Android-OS-Version-Codenames-and-Why. [Accessed: 29-Mar-2019].

[21] Statista, "Smartphone OS global market share 2009-2018 | Statistic," 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: 26-Jun-2018].

[22] Q. Leng, H. Qi, J. Miao, W. Zhu, and G. Su, "One-Class Classification with Extreme Learning Machine," *Math. Probl. Eng.*, vol. 2015, p. 11, 2015.

[23] S. Asiri, "Machine Learning Classifiers," *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623. [Accessed: 04-Jan-2019].

[24] A. Pandey, "One-Class Classification Algorithms," 2017. [Online]. Available: https://ayush-iitkgp.github.io/posts/one-class-classification-algorithms/. [Accessed: 02-Oct-2019].

[25] L. Swersky, H. O. Marques, J. Sander, R. J. G. B. Campello, and A. Zimek, "On the Evaluation of Outlier Detection and One-Class Classification Methods," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 1–10.

[26] D. M. J. Tax and R. P. W. Duin, "Support Vector Data Description," *Mach. Learn.*, vol. 45, no. 1, pp. 45–66, 2004.

[27] E. Pekalska, D. M. J. Tax, and R. P. W. Duin, "One-Class LP Classifier for Dissimilarity Representations," in *In Advances in Neural Information Processing Systems*, 2003, pp. 777–784.

[28] E. Menahem, L. Rokach, and Y. Elovici, "Combining One-Class Classifiers Using Meta Learning," in *In Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management*, 2013, pp. 2435–2440.

[29] M. Yousef, N. Najami, and W. Khalifav, "A comparison study between one-class and two-class machine learning for MicroRNA target detection," *J. Biomed. Sci. Eng.*, vol. 03, no. 03, pp. 247–252, 2010.

[30] V. H. Bezerra, V. G. T. da Costa, S. Barbon Junior, R. S. Miani, and B. B. Zarpelão, "IoTDS: A One-Class Classification Approach to Detect Botnets in Internet of Things Devices," *Sensors (Switzerland)*, vol. 19, no. 14, pp. 1–26, 2019.

[31] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," in *IEEE Transactions on Cybernetics*, 2018, pp. 1–14.

[32] B. Rashidi, C. Fung, and E. Bertino, "Android malicious application detection using support vector machine and active learning," in *2017 13th International Conference on Network and Service Management, CNSM 2017*, 2018, vol. 2018-Janua.

[33] D. Ucci, L. Aniello, and R. Baldoni, "Survey on the Usage of Machine Learning Techniques for Malware Analysis," *Comput. Secur.*, vol. 1, no. 1, pp. 1–67, 2018.

[34] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," *Comput. Secur.*, vol. 68, pp. 36–46, 2017.

[35] Y. Dong, "Android Malware Prediction by Permission Analysis and Data Mining (Master Thesis)," University of Michigan-Dearborn, 2017.

[36] S. Y. Yerima, S. Sezer, and I. Muttik, "Android Malware Detection Using Parallel Machine Learning Classifiers," *2014 Eighth Int. Conf. Next Gener. Mob. Apps, Serv. Technol.*, no. Ngmast, pp. 37–42, 2016.

[37] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph," in *Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004*, 2004, vol. 3, pp. 430–433.

[38] S. L. Phung, A. Bouzerdoum, and G. H. Nguyen, "Learning Pattern Classification Tasks with Imbalanced Data Sets," in *Pattern Recognition*, 2009, pp. 193–208.

[39] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A Novel Dynamic Android Malware Detection System With Ensemble Learning," in *IEEE Access*, 2018, vol. 6, pp. 30996–31011.