Empirical Study on Intelligent Android Malware Detection based on Supervised Machine Learning

Talal A.A Abdullah[1]
Information Technology Department,
Kulliyyah of Information and
Communication Technology
International Islamic University
Malaysia, Malaysia

Waleed Ali[2]
Department of Information
Technology, Faculty of Computing
and Information Technology
King Abdulaziz University, Rabigh
Kingdom of Saudi Arabia

Rawad Abdulghafor[3]
Computer Science Department
Kulliyyah of Information and
Communication Technology
International Islamic University
Malaysia, Malaysia

***Abstract*—The increasing number of mobile devices using the Android operating system in the market makes these devices the first target for malicious applications. In recent years, several Android malware applications were developed to perform certain illegitimate activities and harmful actions on mobile devices. In response, specific tools and anti-virus programs used conventional signature-based methods in order to detect such Android malware applications. However, the most recent Android malware apps, such as zero-day, cannot be detected through conventional methods that are still based on fixed signatures or identifiers. Therefore, the most recently published research studies have suggested machine learning techniques as an alternative method to detect Android malware due to their ability to learn and use the existing information to detect the new Android malware apps. This paper presents the basic concepts of Android architecture, Android malware, and permission features utilized as effective malware predictors. Furthermore, a comprehensive review of the existing static, dynamic, and hybrid Android malware detection approaches is presented in this study. More significantly, this paper empirically discusses and compares the performances of six supervised machine learning algorithms, known as K-Nearest Neighbors (K-NN), Decision Tree (DT), Support Vector Machine (SVM), Random Forest (RF), Naïve Bayes (NB), and Logistic Regression (LR), which are commonly used in the literature for detecting malware apps.**

***Keywords*—Android; malware applications; machine learning**

I. INTRODUCTION

Android constitutes the most common mobile operating system [1] that presently dominates the smartphone market. In the second quarter of 2018, the Android Operating System (AOS) represented the most significant market share amongst other smartphone platforms by approximately 88% worldwide [1]. The popularity of the Android operating system is due to the fact that it constitutes an open-source system with rich SDK libraries, a third-party distribution center, and utilizes Java as a programing language [2].

The fast growth rate of Android applications worldwide has led to a considerable increase in the development and spread of Android malware applications [3]. Android malware can infect any type of application such as bank apps, gaming apps, education, or other lifestyle apps [4] in order to provide unauthorized access and remotely control the system without the user's permission. As more Android malware applications are continuously being developed at an alarming rate, it is important to efficiently and continuously monitor and control their activities.

In recent years, many Android commercial tools and anti-virus programs have been developed to detect android malware applications. Most of these commercial Android malware detection tools are based on using fixed signatures or identifiers. These commercial tools, however, only perform well in detecting the Android malware applications with known signatures or identifiers and may fail to detect the unknown Android malware apps [5] that have been developed more recently, especially zero-day malware apps. In other words, these commercial tools are unable to make accurate decisions when determining whether the new Android app is a malware or not [6][7].

Alternatively, numerous research works [8][9][4][10] focused on training machine learning classification algorithms based on known Android malware apps in order to detect unknown Android malware applications. In fact, machine-learning algorithms have been found to achieve a remarkable accuracy ratio at detecting malicious applications depending on the quality of the extracted features, the dataset, and the methods used in training of the models [6]. In this article, a comprehensive review of Android malware detection approaches based on static, dynamic and hybrid analysis is presented. Furthermore, the article experiments and compares the performances of six commonly used supervised machine learning algorithms.

The rest of the paper is structured as follows: Section II discusses the related work while the major contributions in this study are summarized in Section III. Section IV presents the structure of the Android operating system. The growth of Android malware and some samples are overviewed in Section V. Some supervised machine learning algorithms are overviewed in Section VI. The methodology of Android malware detection based on machine learning is presented in Section VII. The result and discussion are provided in Section VIII, followed by the conclusion and future work in Section IX.

II. Related Work

The ability of machine learning to accurately detect unknown malicious Android applications at an early stage constitutes an attractive advantage that can be utilized to enhance user security and privacy. Several works have applied machine learning through different methods and models to produce better solutions for Android malware detection. In this section, we focus on articles discussing Android malware detection based on machine learning and applying static, dynamic, and hybrid approaches, in addition to other recent articles on different approaches such as ensemble learning and deep learning.

A. Intelligent Android Malware Detection Approach based on Static Analysis

This approach is considered as the most common approach suggested by many researchers as it is simple, fast, and easy to be implemented. The static analysis approach requires only decompiling an Android package (APK) and then extracting the set of Android permissions or API calls invoked throughout the code without running the Android apps.

In [11][12], the authors introduced an Android malware detection system based on permission features. The authors in [11] developed three levels of classifications based on significant permission features that can be efficient in differentiating between benign and malicious apps. In order to leverage the higher computing power of the server, [12] developed a system to extract a number of features and then trained a one-class support vector machine in an offline manner. More than 11,120 Android application samples collected from the DREBIN dataset were used in [13] to evaluate the four machine learning algorithms Random Forest, Decision Tree, Extremely Randomized Tree, and Gradient Tree Boosting, and then a substring-based feature selection method was proposed to identify Android malware applications. In [14], the authors ranked all the individual permissions with their potential risk using the three methods of mutual information, Correlation Coefficient (CorrCoef), and T-test. Furthermore, they employed Sequential Forward Selection (SFS) and Principal Component Analysis (PCA) in order to identify risky permission subsets. Support vector machine, decision trees and random forest were used to detect malware apps based on the identified subsets of risky permissions. More than 30 features from seven (7) categories were collected in [15] which implemented a collection of machine learning algorithms such as Support Vector Machine, Random Forest, Naïve Bayes, and logistic regression. The authors in [11] demonstrated that the best performance was accomplished by Random Forest. However, the dataset used in [11] was relatively small and included only 32 benign apps and five (5) malware apps. Three (3) Bayesian classification approaches for identifying Android malware were analyzed and suggested in [16] [16] which applied a static analysis using a dataset of malware samples containing 49 known Android malware families and a wide variety of benign apps.

Other articles, such as [17][18], used a combination of permissions and API features for building Android malware detection. Authors in [18] experimented on the performance of SVM, J48, and Bagging on real-word Apps for more than 1,200 malware apps and 1,200 benign apps. They obtained 96.39% accuracy in detecting malware apps. In [17], the best accuracy rate was performed by SVM and ensemble learning, with 95.1% and 95.6%, respectively.

B. Intelligent Android Malware Detection Approach based on Dynamic Analysis

In the dynamics-based approach, it is required to use a simulator, an emulator, or even a physical device to run an Android app to monitor its dynamic behavior. Then, the dynamics features are extracted to train the machine learning classifiers in order to be used in Android malware detection.

The intelligent Android malware detection approach based on dynamic analysis has been suggested in several research studies. For instance, [19] applied dynamic analysis using the Random Forest algorithm as a machine learning algorithm and proposed the Conformal Prediction model assessed on 1,866 malware and 4,816 benign applications on a real Android device. DroidDolphin [20] is a dynamic malware analysis framework that uses GUI-based testing, big data analysis, and machine learning to detect Android malware. The framework can be used in conjunction with other existing works to improve the detection rate of malware. Furthermore, [21] developed a dynamic Android malware detection based on API calls and system call traces using 7,520 apps, including 3,780 for training and 3,740 for testing, while [22] implemented a tool to automatically extract dynamic features from Android phones and performed a comparative analysis of emulator-based detection against device-based detection by means of Random Forest, Naive Bayes, Multilayer Perceptron, Simple Logistics, J48 decision tree, PART, and SVM (linear) algorithms.

C. Intelligent Android Malware Detection Approach based on Hybrid Analysis

The hybrid analysis is a combination of static analysis and dynamic analysis that can be integrated to detect Android malware [23].

In [24], developed a MARVIN Android malware detection tool that was utilized to classify apps based on features extracted from static and dynamic analysis with over 135,000 Android apps and 15,000 malware samples and successfully classified 98.24% of malicious apps with less than 0.04% false positives. Subsequently, [25] proposed a novel hybrid Android malware analysis approach called mad4a. In order to achieve a comprehensive analysis and discover more malware apps, mad4a used both static and dynamic advantages to analyze the dataset. Authors in [26] extracted and merged static and dynamic app features and then adjusted the weights to use Weka for training the detection model. The ten-fold cross-validation method achieved an accuracy of 97.4%.

D. Other Advanced Intelligent Techniques

Authors in [27] proposed a hybrid-model approach using a fusion logic algorithm, achieving very high accuracy (96.69%) and a low false-positive rate (2.5%) in predicting unknown malware apps. Another hybrid-model was proposed by [28] for malware detection using the anomaly-based approach with machine learning classifiers. Bayes network and random forest classifiers were used in [28] and produced a 99.97% true-

positive rate. Also, an evolving hybrid neuro-fuzzy classifier was proposed in [29] to enhance the detection accuracy of malware applications that achieved 90% detection accuracy with a dataset of 250 malware apps and 250 benign. The author in [30] suggested a hybrid intelligent Android malware detection approach based on evolving support vector machine with a genetic algorithm (GA) and particle swarm optimization (PSO) in order to enhance detection accuracy of Android malware apps.

Deep learning has been used in Android malware detection by [31][32][9]. However, deep learning requires a great amount of data, more time, and a sophisticated and powerful computer to produce a good result. Ensemble learning produced excellent results in many research studies, such as [14][10][15][16]. Pindroid [17] used a group of permissions and intents supplemented with ensemble methods for accomplishing more accurate malware detection [17]. On another note, [18] produced a hypothesis to detect Android malware in the early stage by means of parallel machine learning classifiers that utilized various algorithms with inherently different characteristics.

The study of [19] adopted a machine learning approach that used the dataflow application program interfaces (API) to collect features and use them to detect malware apps. A thorough analysis was conducted to extract features and improve the k-nearest neighbor classification model. An automated testing tool called WaffleDetector was implemented by [20] to identify Android malware by proposing a group of Android features consisting of sensitive permissions and API to feed machine learning algorithms. Finally, [32] used metadata to categorize malware, and [33] implemented an online machine learning classification. Useful review articles of Android malware detection using machine learning techniques can be found in [34][35][36].

III. Summary of Contributions

This article presents a comprehensive review of Android malware detection approaches based on static, dynamic, and hybrid analysis. In addition, it compares and discusses the performances of six supervised machine learning algorithms, which are commonly used in the literature for detecting malware apps, known as K-Nearest Neighbors (K-NN), Decision Tree (DT), Support Vector Machine (SVM), Random Forest (RF), Naïve Bayes (NB), and Logistic Regression (LR). The significant contributions in this study can be summarized in the following aspects:

- Android architecture, Android malware, and permissions as effective malware predictors are investigated and discussed in this study.
- This work presents a comprehensive review of common Android malware analysis methods that are categorized under static, dynamic, and hybrid approaches.
- More significantly, this paper empirically discusses and compares the performances of six supervised machine learning algorithms commonly used in the literature for detecting malware apps.

IV. Android Architecture

Android is an open-source system that comprises a Linux-based software stack for a wide range of devices and form factors created by Google [37]. The Android operating system is a stack of components that can be defined as consisting of five layers that organize the functions of the system in the form of the Linux kernel layer, hardware abstractor layer, Android libraries layer, Java API framework layer, and system application layer.

A. The Linux Kernel

Android uses a version of the Linux kernel equipped with a few unique additions [38]. The Android kernel is responsible for handling functions such as memory process, device drivers, resource access, power management, and other typical OS duties. It also serves as a layer between the hardware and other software stacks [39].

B. Hardware Abstractor Layer

The hardware abstractor layer (HAL) is defined as [40] a standard interface implemented by hardware vendors that enables Android to be agnostic about lower-level driver implementations. HAL allows the user to implement functionalities without affecting or modifying the higher-level system (“Legacy HALs"). The main hardware abstractor layer contains Application Programming Interfaces (APIs) for the upper layers in order to use hardware in a unified and straightforward way [41]. In Android 8.0 and above, the lower-level layers are rebuilt to fit a new and more sophisticated architecture; however, devices that use Android 8.0 and above should support HALs written in the HIDL language, with a few exceptions [37].

C. Android Libraries

This layer is composed of two modules. The first module contains the Native C/C++ Libraries, such as OpenGL, Webkit, or SSL/TLS, that contain essential application features. Native code is used to program Android-system components and services such as ART and HAL. This code requires native libraries that are mostly written in C and C++ languages [37]. The Android platform provides an API framework that allows applications to interact with the underlying Android system [18].

The second module contains Android Runtime (ART), a modified Java Virtual Machine (JVM) in order to run Android applications that are not implemented in native code. ART constitutes a byte code format designed especially for Android that is optimized for minimizing memory consumption and is written to run multiple virtual machines on low-memory devices by executing DEX files [37]. The Dalvik virtual machine has been designed to work effectively in multiple virtual machines in order to increase stability and reduce memory consumption [15]. ART comes with ahead-of-time compiling (AOT), which performs complete bytecode translation after installation and before running the application. ART also provides improved garbage collection and new debugging features [42].

D. Java API Framework

All Android OS features that are available for use through APIs are programmed using Java [37]. Application Programming Interface (API) refers to a set of tools that provide a communication interface between different software components [7]. The API framework consists of a core set of classes and packages [18]. These APIs are fundamental components for building Android applications, such as the view system to create the user interface (UI) [37]. Top-level system applications are necessary to provide basic functionality like calendar, contacts, and e-mail [37].

E. System Application

The system-applications layer is the top layer that is responsible for interacting between the end-user and the device. System applications are located in order to provide basic functionalities such as managing contacts, sending messages, making calls, and browsing the Web [37][2].

The system application layer contains the four components of activity, services, content provider, and broadcast receiver. Every component fulfills a specific purpose and has its own life cycle. The activity component interacts with the user and represents a single screen with a user interface [37] and is mainly used as an entry point for the application. The services, on the other hand, are a group of the components and processes used for performing specific tasks in the background and do not require a user interface [41]. The content provider is used to manage and share data between multiple applications [38], which allows applications to read and write data (such as contact information) and communicate with each other or interact with other applications in the system. In contrast, the broadcast receiver is used as a mailbox to respond to and receive the broadcast messages of the order or other applications (such as the low battery message) [2].

V. Android Malware

Android malware apps are growing at an alarming rate, regardless of the measures used to reduce infections amongst Android users worldwide [43]. For example, G DATA security experts discovered that there were 8,400 new Android malware samples every day in the first quarter of 2017 [44]. Fig. 1 shows the growth of Android malware apps during recent years.

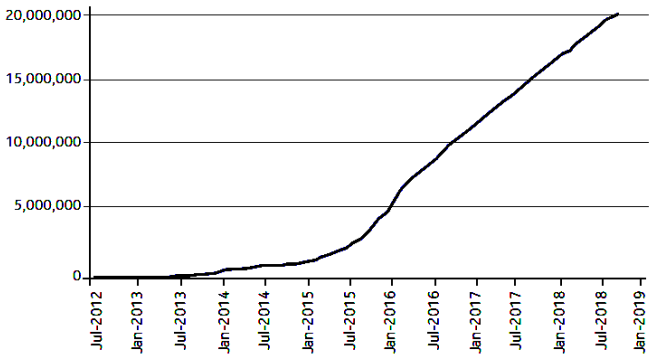

Fig. 1. The Growth of Android Malware Apps between 2012-2018.

TABLE I. Top Android malware Detected in 2018 [46]

Name	Threat	Threat's percentage
Android.Adware.AdultSwine	Moderate	17.29
Android.Adware.Uapush.A	Moderate	13.98
Android.Trojan.Leech.d	High	4.69
Android.Trojan.AndrClicker.D	High	4.41
Android.Spyware.mSpy	High	4.11
Android.MobileSpyware.FlexiSpy	High	3.62
Android.Trojan.Xgen.FH	High	3.12
Android.InfoStealer.Adups	High	3.03
Android.Trojan.Rootnik.i	High	3.01
Android.Trojan.Triada	High	2.76
Android.Trojan.Gmobi.a	High	2.61
Android.BankingTrojan.Marcher.A	High	2.39
Android.BankingTrojan.Acecard.m	High	2.15
Android.Trojan.HiddenApp	High	2.08
Android.Trojan.Sivu.C	High	2.06

There are a variety of attack types ranging from the attack that only is advertising without harming the product or the website to the most sophisticated attack that is capable of accessing personal and sensitive information on the device [23][45]. The majority of Android malware can be categorized into fake installers or SMS trojans. Both of them are using social engineering to trick users into installing malicious apps [2]. Table I shows the top 15 Android malware detected in 2018.

VI. Supervised Machine Learning

Machine learning is defined as the science of computer programming that can learn from data and past experiences [47]. Today machine learning models are used for recommender systems such as online shops, for fraud detection in credit card companies or for medical diagnosis in hospitals [41]. The supervised learning approach is able to automate a decision process from the generalization of known examples and specific input data [41]. In general, the data is labeled and divided into training and testing data. The training data is fed into a supervised machine learning algorithm to train the model. Subsequently, the test data is used to verify the effectiveness of the model by comparing the predicted label with the test label of the data.

In this section, we will describe six (6) supervised machine learning algorithms commonly used in literature for detecting malware apps: K-Nearest Neighbors (K-NN), Decision Tree (DT), Support Vector Machine (SVM), Random Forest (RF), Naïve Bayes (NB), and Logistic Regression (LR).

A. K- Nearest Neighbours

This algorithm classifies cases based on their similarity to other cases. In K-nearest neighbors, data points that are near to each other are set to be neighbors, and the output is predicted by the majority vote of the K-closest neighbors. Thus, the distance between the two cases is a measure of their

dissimilarity. In a classification problem, the K-nearest neighbors algorithms work as follows:

- Choose a value for K
- Calculate the distance of unknown cases in all cases
- Search for the K-samples in the training data that are similar to the measurements of the unknown data point
- Predict the response of the unknown data point using the most popular class responses value from the K-NN.

There are different ways to calculate the similarity between two data points. The most frequently used method is the Euclidean distance, which is computed using the formula (1).

The value K assumes a significant job in impacting the prediction accuracy of the algorithm. However, choosing the K value is not a simple undertaking.

$$\text{ED}(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

x_1 , x_2 are points in the space n (1)

B. Decision Trees

Decision Trees are versatile and very powerful machine learning algorithms that can be used in both regression and classification tasks, and even in multioutput tasks [47]. In order to produce a decision, a hierarchy of if-else questions needs to be answered. For instance, in order to distinguish between four (4) animals such as bears, hawks, penguins, and dolphins, specific questions need to be asked. The first question may be whether the animal has feathers or not, which narrows down the probability from four to just two. If the answer is 'yes', another question follows to distinguish between hawks and penguins, such as whether the animal can fly or not. On the other hand, if the animal does not have feathers, it is possible the choosing animal either dolphins or bears [48].

At the top of the tree, the most significant features for decision nodes are used. The child nodes at the bottom assign the data points to their categories in a more accurate way [41]. The advantages of decision trees are their simplicity, little data preparation, including feature extraction and the interpretability of the model, which results in the ability to visualize the model [49]. Furthermore, decision trees can handle numerical as well as categorical data [50].

C. Support Vector Machine

The support vector machine (SVM) algorithm is counted among the supervised machine learning algorithms that are commonly used in malware detection and other classification and regression problems. SVM is efficiently used in many complex applications with small or medium-sized datasets.

The main principle here is to identify the best hyperplane that can separate the classes. The term 'support vectors' means the data points that are near to the hyperplane and might shift the hyperplane position up or down if removed. Margin in SVM constitutes the distance between the support vector and the hyperplane [7]. SVM generally achieves good accuracy, particularly on clean datasets. Furthermore, it works well with high-dimensional datasets and large datasets that have larger data-training time. SVM represents the training data as points in the dimensional space that are assembled based on their class. Subsequently, each group is separated by a line called a hyperplane. For example, if the dataset has picture samples of cats and dogs, the SVM algorithm will separate all cat pictures in one-dimensional space and all dog pictures in another dimensional space and between them a hyperplane. The new inputs are mapped into the trained space and categorized based on which side of the gap they fall on.

For more confidence and less error generation, the margin function must select the hyperplane in order to ensure that the distance between the nearest training data points in any class is as large as possible [2]. In most cases, the data points are not linearly separable. Thus, the SVM uses kernel functions to transform the data into a higher-dimensional space and then classify them using the same principle as the linear case.

D. Random Forest

Random forest is defined as a collection of decision trees that are slightly different from each other. The idea is that when many decision trees are implemented that are slightly different from each other, different overfitting occurs on parts of the data. The amount of overfitting can be reduced by averaging their results. Thus, we can benefit from the predicting power of decision trees and the result of their overfitting average for best predicting results [48].

The Random Forest algorithm derives its name from infusing randomness into the tree working to guarantee each tree is extraordinary. The algorithm can be described as follows [51]:

- Multiple decision trees are built on 70% of the collected dataset; however, these data are chosen randomly.
- Random variables are selected from out of all the predicted variables. Subsequently, the algorithm determines the best split that matches these selected variables and applies it to split the nodes.
- The wrong classification rate or the prediction error is calculated using the rest of the data.
- After comparing the trained trees classification results and votes, the algorithm chooses the best result as the ultimate result.

As in decision trees, Random Forest removes the irrelevant features as feature selection is necessary when there is a need for dimensionality reduction [7].

E. Naïve Bayes

The Naïve Bayes algorithm is considered as one of the most powerful and straightforward machine learning techniques that depend on the Bayes theorem with an intense independence assumption among predictors [38]. Naïve Bayes algorithm has proven its effectiveness in many applications such as medical diagnosis, text classification, and system performance management [52].

The Naïve Bayes algorithm involves the following concepts that need to be understood.

- Class Probability: Class probability is the probability of a particular class in the dataset, i.e., the possibility of a randomly selected item from the dataset to be in a particular class.
- Conditional Probability: Conditional probability is the probability of the feature value given to the class.

The class probability is calculated as the calculation of samples in the *C* class divided by the overall number of samples of all the classes, as shown in equation (2).

$$p(C) = \frac{\#\ of\ samples\ in\ C}{\#\ of\ samples\ in\ Total} \quad (2)$$

The rate of each sample divided by the rate of samples in that class is called conditional probabilities, as shown in equation (3).

$$p(V|C) = \frac{\#\ of\ instances\ whit\ V\ and\ C}{\#\ of\ instances\ with\ V} \quad (3)$$

Looking at the probabilities, we can compute the likelihood of the samples having a place in a class and make choices utilizing the Bayes theorem, as shown in equation (4).

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \quad (4)$$

The probability of the sample for each class is computed, and the highest probability class is assigned as a result [7].

VII. Methodology

This section describes the methodology used to detect Android malware apps using standard supervised machine learning algorithms. The research follows the four phases of data collection, feature extraction, training of classification models, and performance evaluation.

A. Data Collection

In this study, the Malgenome-215 dataset with 3,799 application samples used by [4] was adopted in our experiments in order to train and evaluate the common classification models. The dataset consists of 2,539 benign apps and 1,260 malware apps. The majority of apps are from 49 different Android malware families collected from the data between August 2010 to the recent one in October 2011.

B. Feature Extraction

Android applications contain critical information that can be extracted to analyze the attitudes of these applications [53]. Android features fall under the three types of permissions, sensitive APIs and dynamic behaviors [9]. Dynamic behaviors are extracted through dynamic analysis, while the rest of the features are extracted by using static analysis, as shown in Fig. 2.

In the dataset [44] used in this study, the static features are extracted using a static python tool from manifest file for permissions and intents, and from the .dex files for API calls. Then, these features are represented in a binary form based on the presence of these features in the Android apps.

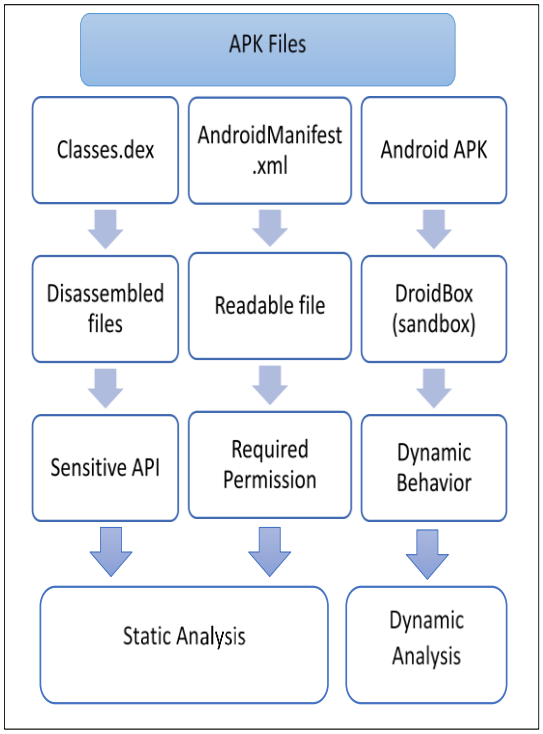

Fig. 2. Android Feature Types.

C. Training of Classification Models

The main goal of the classification model is to predict a class label that is chosen from the predefined possibilities list. Classification problems can be binary classification, which has only two classes to be classified, or multi-class classification, which uses the classification model to predict multiple classes.

From the perspective of machine learning, Android malware detection can be understood as a binary classification problem. To fulfill our objective in this study, we use binary classification to answer the question of whether the Android application is benign or malware based on the static features.

In this study, six (6) common supervised machine learning models are trained based on known Android apps with 215 static features in order to distinguish malware from benign apps. Accordingly, unknown Android malware apps can be detected using the trained, supervised machine learning models of K-Nearest Neighbors (K-NN), Decision Tree (DT), Support Vector Machine (SVM), Random Forest (RF), Naïve Bayes (NB), and Logistic Regression (LR). Each classification algorithm uses different mathematical approaches to distinguish between classes, as mentioned in Section 5.

D. Performance Evaluation

In order to evaluate the performance of six (6) supervised machine learning models, we use four (4) essential metrics, which were commonly used in literature for Android malware detection:

- Accuracy: The ratio of the number of Android apps that are classified correctly as a benign app or as a malware app to the total number of Android apps. It can be computed using equation (5).

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

- Precision: The ratio of malware apps properly detected to the complete amount of applications categorized as malicious. It can be computed using equation (6).

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

- Recall: The ratio of malware apps detected adequately to the total number of malware apps. It can be computed using equation (7).

$$precision = \frac{TP}{TP+FN} \quad (7)$$

- F-Score: The mean of precision and recall. This value shows how precise the model is. It can be computed using equation (8).

$$F - Score = 2 * \frac{precision*recall}{precision+recall} \quad (8)$$

VIII. Results and Discussion

A. Experiments Environment

This study implemented six (6) popular machine learning algorithms (K-NN, DT, SVM, RF, NB, LR) on a malgenome-250 dataset [4] collected from Genome project [54] which contained 3,799 Android applications. This dataset [4] consisted of 2,539 benign samples and 1,260 malware samples from 49 different Android malware families. The experiments were conducted on the Anaconda Jupiter navigator using a laptop with the features shown in Table II. In order to prepare the training dataset, 215 static features of Android applications, including permissions, intents, and API calls, were extracted and converted to binary forms. If a static feature was requested, 1 would be assigned to that feature; otherwise, 0 would be given.

B. Evaluation Methods and Measures

In this paper, the six (6) popular machine learning algorithms were evaluated using the two evaluation methods of holdout and k-fold validation. In holdout validation, the data was divided into 80% for the training dataset and 20% for the testing dataset, while 10-fold was used in k-fold cross-validation. The data was split into 10 folds; each fold was used nine (9) times as training fold and one time as testing fold. Then, the mean of the accuracy of all folds was presented as a final accuracy.

In order to evaluate and measure the performance of the machine learning algorithms, we used the four (4) common measures of Accuracy, Precision, Recall, and F-Score, as described in Section 6.4.

TABLE II. Feature of the Laptop used in the Experiments

CPU	Intel(R) Core (TM) i7-8750H CPU @ 2.20 GHz
Memory	16 GB
OS	Windows 10 Home 64-bits
Platform	Anaconda Jupiter navigator

C. Performance of Machine Learning Algorithms

1) K- Nearest Neighbors (KNN): As mentioned earlier, KNN is considered as one of the most straightforward and powerful classification models. The performance of this algorithm is affected by the k parameter used to finding the k training examples that are closest to the unknown example. Therefore, we trained the KNN model with the changing value of k from 1 to 30. KNN achieved the best accuracy when k = 1 for both holdout and 10-fold cross-validation methods. Fig. 3 shows the accuracy of the k-NN model for 10-fold cross-validation with the changing value of k from 1 to 30.

2) Decision Trees (DTs): Decision trees (DTs) are composed of decision nodes and terminal leaves that are connected through edges. The number of child nodes connected by edges can be binary or non-binary. It can be simply described as a hierarchy of 'if-else' questions leading up to a decision. Decision trees are affected by the maximum depth given to the trees. Therefore, we trained the TDs with changing N-depth with a range from 1 to 30 to get the best result. It was observed that TDs achieved the best accuracy with depth=13 in the holdout method, while the best accuracy was achieved with depth=12 in 10-fold cross-validation, as shown in Fig. 4.

3) Support Vector Machine (SVM): SVM classifies data into distinct classes by maximizing the margin between the separating hyperplane. If the data cannot be separated linearly, the data will be converted into a high *n*-dimensional feature space such that SVM can draw a hyperplane.

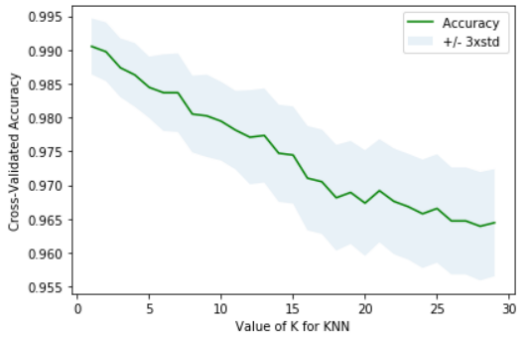

Fig. 3. Accuracy of the k-NN Model for 10-Fold Cross-Validation with Changing the Value of k from 1 to 30.

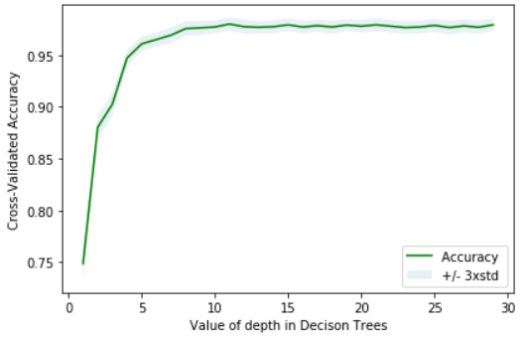

Fig. 4. DTs Accuracy for 10-Fold Cross-Validation when the Depth between 1-30.

TABLE III. THE PERFORMANCE MEASURES ACHIEVED BY SVM WITH LINEAR AND RBF KERNEL FUNCTIONS

Kernel		Accuracy	Precision	Recall	F1-score
RBF scale	Hold-out	0.9816	0.98	0.98	0.98
	10-Fold	0.9897	0.99	0.99	0.99
RBF auto	Hold-out	0.9806	0.98	0.98	0.98
	10-Fold	0.9776	0.98	0.98	0.98
Linear	Hold-out	0.9829	0.98	0.98	0.98
	10-Fold	0.9897	0.99	0.99	0.99

The mathematical function applied for the conversion is called the kernel function and can be RBF or other kernel functions. This experiment examined with Linear and RBF kernel functions. As can be seen from Table III, the best result was produced when the kernel was Linear.

4) Random Forest (RF): Random forest is counted among the ensemble learning algorithms that are constructed from a collection of correlated decision trees after training. The bagging technique is used to obtain a random sample from the features and learn a decision tree classifier for each subset of the data. The performance of the RF algorithm is affected by the n_estimator parameter that represents the number of trees in the forest. Generally, the higher the number of trees, the better to learn the data; however, adding a lot of trees can slow down the training process considerably. Therefore, we trained RF with changing the n_estimator with a range from 10 until 100. Fig. 5 shows the training of the RF model for 10-fold cross-validation with changing n_estimator between 10-100. The best performance was achieved when the n-estimator was 33 in holdout and 68 in 10-fold cross-validation.

5) Naïve Bayes (NB): Naïve Bayes treats each feature independently and evaluates the probability to make predictions based on the Bayes theorem. Naïve Bayes has different models, such as GaussionNB, BernoulliNM, and MultinomialNB. After training the three models, Multinomia1NB performed better than the others, as shown in Table IV.

6) Logistic Regression (LR): Logistic Regression can be applied in both binary classification and multi-class classification. It is useful when the observed dependent variable is categorical. The parameter solver can be changed to different types such as newton-cg, lbfgs, liblinear, sag, and saga, which showed similar results in this experiment. Therefore, we choose the default solver, which is liblinear. The respective results are shown in Table V.

D. Discussion

In this section, we compare the performance of the selected six popular machine learning algorithms (K-NN, DT, SVM, RF, NB, LR) in terms of Accuracy, Precision, Recall, and F1-score.

As it can be observed from Table VI, all algorithms achieved high Accuracy, Precision, Recall and F1-score in terms of predicting and detecting malware apps. The Accuracy range of the applied algorithms was between 0.95 and 0.99. The best accuracy (0.99211) was achieved by Random Forest (RF) in both holdout and 10-fold cross-validation methods. Furthermore, the best Precision (0.99), Recall (0.99), and F1-score (0.99) were achieved by RF.

Among all the applied algorithms, Naïve Bayes (NB) achieved the lowest Accuracy in both holdout and 10-fold cross-validation methods. NB produced Accuracy= 0.9572 in holdout method and Accuracy = 0.9545 in 10-fold cross-validation. Recall and F1-score achieved by were 0.95 in both holdout and 10-fold methods. Precision achieved by NB was 0.95 in holdout and 0.96 in 10-fold. KNN performed better than SVM, and LR with 0.98684 Accuracy in holdout validation and 0.99052 in 10-fold cross validation. The KNN Precision, Recall and F-score measured 0.98 in both holdout and 10-fold cross-validation method. In DT performance, DT accomplished 0.97632 and 0.9797 Accuracy, 0.97 and 0.98 Precision, 0.97 and 0.98 Recall, and 0.97 and 0.98 F1-score in holdout and 10-fold cross-validation, respectively. For LR, the Accuracy (0.96579) performed by LR in holdout was slightly lower than Accuracy (0.97367) in 10-fold cross-validation. Moreover, LR accomplished 0.97 for the remaining measures (Precision, Recall, F1-score) in both holdout and 10-fold cross-validation methods.

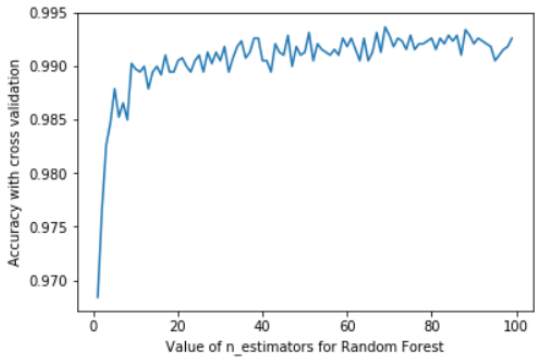

Fig. 5. Accuracy of the RF Model for 10-Fold Cross-Validation with Changing n_estimator between 10-100.

TABLE IV. NB ACCURACY FOR GAUSSIONNB, BERNOULLINM, AND MULTINOMIALNB

		Accuracy	Precision	Recall	F1-score
GaussianNB	Holdout	0.7167	0.84	0.72	0.72
	10-Fold	0.7210	0.84	0.72	0.73
BernoulliNM	Holdout	0.6736	0.42	0.65	0.51
	10-Fold	0.6683	0.45	0.67	0.54
MultinomialNB	Holdout	0.9572	0.95	0.95	0.95
	10-Fold	0.9545	0.96	0.95	0.95

TABLE V. LR SOLVER TYPES MEASURES

Solver		Accuracy	Precision	Recall	F1-score
liblinear	**Holdout**	0.9658	0.97	0.97	0.97
	10-Fold	0.9737	0.97	0.97	0.97
sag	**Holdout**	0.9720	0.96	0.96	0.96
	10-Fold	0.9700	0.97	0.97	0.97
newton-cg	**Holdout**	0.9720	0.96	0.96	0.96
	10-Fold	0.9700	0.97	0.97	0.97
saga	**Holdout**	0.9720	0.96	0.96	0.96
	10-Fold	0.9700	0.97	0.97	0.97
Ibfgs	**Holdout**	0.9720	0.96	0.96	0.96
	10-Fold	0.9700	0.97	0.97	0.97

TABLE VI. THE PERFORMANCE MEASURES OF POPULAR MACHINE LEARNING ALGORITHMS USED IN ANDROID MALWARE DETECTION

Algorithm	Evaluation method	Accuracy	Precision	Recall	F1-score
K-NN	**Holdout**	0.9868	0.98	0.98	0.98
	10-Fold	**0.9905**	0.98	0.98	0.98
DT	**Holdout**	0.9763	0.97	0.97	0.97
	10-Fold	0.9797	0.98	0.98	0.98
SVM	**Holdout**	0.9829	0.98	0.98	0.98
	10-Fold	0.9897	**0.99**	**0.99**	**0.99**
RF	**Holdout**	**0.9921**	**0.99**	**0.99**	**0.99**
	10-Fold	**0.9937**	**0.99**	**0.99**	**0.99**
NB	**Holdout**	0.9572	0.95	0.95	0.95
	10-Fold	0.9545	0.96	0.95	0.95
LR	**Holdout**	0.9658	0.97	0.97	0.97
	10-Fold	0.9737	0.97	0.97	0.97

IX. CONCLUSION AND FUTURE WORK

The ability of machine learning algorithms to learn from the existing data and then generalize from seen examples to unseen examples encouraged us to apply six (6) popular machine learning algorithms in order identify the new and unknown malware apps or zero-day malware apps. This paper reviewed and discussed some common Android malware methods based on machine learning in the form of static, dynamic and hybrid analysis approaches. Furthermore, this study implemented Nearest Neighbors, Decision Tree, Support Vector Machine, Random Forest, Naïve Bayes, and Logistic Regression in order to overcome the difficulties faced by conventional methods to detect unknown and zero-day Android malware apps. The experimental results showed that all six (6) machine learning algorithms performed remarkably well in Android malware detection. In particular, Random Forest achieved the best detection results while Naïve Bayes produced the lowest detection results in Android malware detection.

This paper can be improved further by implementing ensemble learning methods. Furthermore, the performance of machine learning can be enhanced using feature selection techniques.

REFERENCES

[1] Statista, “Mobile OS market share 2018 | Statista.” [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: 27-Mar-2019].

[2] H. A. Alatwi, “Android Malware Detection Using Category-Based Machine Learning Classifiers,” Rochester Inst. Technol., 2016.

[3] R. Samani and G. Davis, “McAfee Mobile Threat Report Mobile Malware Continues to Increase in Complexity and Scope,” McAfee, 2019.

[4] S. Y. Yerima and S. Sezer, “DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection,” IEEE Trans. Cybern., vol. 49, no. 2, pp. 453–466, 2019.

[5] S. Y. Yerima and S. Sezer, “High Accuracy Android Malware Detection Using Ensemble Learning,” Inst. Eng. Technol., no. April, 2015.

[6] D. Youchao, “Android Malware Prediction by Permission Analysis and Data Mining,” Univ. Michigan-Dearborn, 2017.

[7] K. Chumachenko and I. Technology, “Machine Learning Methods for Malware Detection and Classification,” Univ. Aplied Sci., 2017.

[8] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers,” Futur. Gener. Comput. Syst., vol. 78, pp. 987–994, 2018.

[9] Z. Yuan, Y. Lu, and Y. Xue, “DroidDetector : Android Malware Characterization and Detection Using Deep Learning,” vol. 21, no. 1, pp. 114–123, 2016.

[10] 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.

[11] L. Sun, W. Srisa-an, H. Ye, Z. Li, J. Li, and Q. Yan, “Significant Permission Identification for Machine-Learning-Based Android Malware Detection,” IEEE Trans. Ind. Informatics, vol. 14, no. 7, pp. 3216–3225, 2018.

[12] J. Sahs and L. Khan, “A Machine Learning Approach to Android Malware Detection,” 2012.

[13] I. B. Chebbi, W. Boulila, and I. R. Farah, Evaluation of Tree Based Machine Learning Classifiers for Android Malware Detection, vol. 10449. Springer International Publishing, 2015.

[14] D. Feng, W. Wang, J. Liu, X. Wang, X. Zhang, and Z. Han, “Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection,” IEEE Trans. Inf. Forensics Secur., vol. 9, no. 11, pp. 1869–1882, 2014.

[15] H. S. Ham and M. J. Choi, “Analysis of Android malware detection performance using machine learning classifiers,” Int. Conf. ICT Converg., pp. 490–495, 2013.

[16] S. Y. Yerima and S. Sezer, “Analysis of Bayesian Classification based Approaches for Android Malware Analysis of Bayesian Classification based Approaches for Android Malware Detection,” Inst. Eng. Technol., no. April, 2014.

[17] N. Milosevic, A. Dehghantanha, and K. R. Choo, “Machine learning aided Android malware classification.,” Comput. Electr. Eng., vol. 61, pp. 266–274, 2017.

[18] N. Peiravian and X. Zhu, “Machine Learning for Android Malware Detection Using Permission and API Calls,” 2013.

[19] H. Papadopoulos, N. Georgiou, C. Eliades, and A. Konstantinidis, “Android malware detection with unbiased confidence guarantees,” Neurocomputing, vol. 280, pp. 3–12, 2018.

[20] W. C. Wu and S. H. Hung, “DroidDolphin: A dynamic android malware detection framework using big data and machine learning,” Proc. 2014 Res. Adapt. Converg. Syst. RACS 2014, pp. 247–252, 2014.

[21] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, “Identifying Android malware using dynamically obtained features,” J. Comput. Virol. Hacking Tech., vol. 11, no. 1, pp. 9–17, 2015.

[22] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Emulator vs real phone: Android malware detection using machine learning," IWSPA 2017 - Proc. 3rd ACM Int. Work. Secur. Priv. Anal. co-located with CODASPY 2017, pp. 65–72, 2017.

[23] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," Digit. Investig., vol. 13, pp. 22–37, 2015.

[24] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN : Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis," 2015 IEEE 39th Annu. Comput. Softw. Appl. Conf., vol. 2, pp. 422–433, 2015.

[25] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," Digit. Investig., vol. 24, pp. 25–33, 2018.

[26] M. Su, "Machine Learning on Merging Static and Dynamic Features to Identify Malicious Mobile Apps," pp. 863–867, 2017.

[27] H. Y. Chuang and S. De Wang, "Machine Learning Based Hybrid Behavior Models for Android Malware Analysis," Proc. - 2015 IEEE Int. Conf., pp. 201–206, 2015.

[28] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," Soft Comput., vol. 20, no. 1, pp. 343–357, 2016.

[29] A. Altaher, "An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features," Neural Comput. Appl., vol. 28, no. 12, pp. 4147–4157, 2017.

[30] W. Ali, "Hybrid Intelligent Android Malware Detection Using Evolving Support Vector Machine Based on Genetic Algorithm and Particle Swarm Optimization," vol. 19, no. 9, pp. 15–28, 2019.

[31] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," Digit. Investig., vol. 24, pp. S48–S59, 2018.

[32] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android Malware Characterization Using Metadata and Machine Learning Techniques," vol. 2018, 2018.

[33] R. M. Burstall, "Android Malware Classification by Applying Online Machine Learning," Comput. J., vol. 9, no. 1, pp. 15–15, 2012.

[34] A. Bala, S. Malhotra, N. Gupta, and N. Ahuja, "A Survey of Android Malware Detection Strategy and Techniques," Adv. Intell. Syst. Comput., vol. 409, pp. 579–587, 2016.

[35] H. Alireza, Souri1, Rahil, "A state-of-the-art survey of malware detection approaches using data mining techniques.pdf." Alireza Souri, Rahil Hosseini, 2018.

[36] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," Digit. Investig., vol. 13, pp. 22–37, 2015.

[37] Developers, "Platform Architecture | Android Developers," Developers, 2019. [Online]. Available: https://developer.android.com/guide/platform. [Accessed: 01-Apr-2019].

[38] C. Science and V. Grampurohit, "Android App Malware Detection," no. July, 2016.

[39] M. Z. Alkurdi, "Malware Detection for Android Applications Using SimHash Algorithm," 2014.

[40] Source, "Legacy HALs | Android Open Source Project," source, 2019. [Online]. Available: https://source.android.com/devices/architecture/hal. [Accessed: 05-Oct-2019].

[41] J. Thon, "Predictive Identification of Android Malware through Hybrid Analysis created by," 2018.

[42] Android Source, "ART and Dalvik | Android Open Source Project," Android Source, 2019. [Online]. Available: https://source.android.com/devices/tech/dalvik/. [Accessed: 01-Apr-2019].

[43] 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., pp. 37–42, 2014.

[44] C. Lueg, "8,400 new Android malware samples every day," G Data Security Blog, 2017. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day. [Accessed: 15-Mar-2020].

[45] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in Proceedings - IEEE Symposium on Security and Privacy, 2012, no. 4, pp. 95–109.

[46] C. S. Providers and T. Intelligence, "Nokia Threat Intelligence Report – 2019," Netw. Secur., vol. 2018, no. 12, p. 4, 2018.

[47] A. Abraham et al., Hands-On Machine Learning with Scikit-Learn and TensorFlow.pdf. O'Reilly Media, 2014.

[48] A. C. Muller and S. Guido, Introduction to Machine Learning with Python. 2016.

[49] E. Alpaydin, Introduction to Machine Learning Second Edition. 2010.

[50] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "The CART decision tree for mining data streams," Inf. Sci. (Ny)., vol. 266, pp. 1–15, May 2014.

[51] G. Biau, "Analysis of a Random Forests Model," pp. 1–40, 2010.

[52] I. Rish, "An empirical study of the naive Bayes classifier & H," T.J. Watson Res. Cent., pp. 41–46, 1999.

[53] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow APIs and machine learning," Inf. Softw. Technol., vol. 75, pp. 17–25, 2016.

[54] Genome, "Android Malware Genome Project." [Online]. Available: http://www.malgenomeproject.org/. [Accessed: 22-Dec-2019].