

Bachelor of Science in Computer Science
October 2022



Android Malware Detection

Using Machine Learning

Rahul Sai Kesani

This thesis is submitted to the Faculty of Engineering at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science. The thesis is equivalent to 20 weeks of full-time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:

Author(s):

Rahul Sai Kesani

E-mail: rake21@student.bth.se

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Background. The Android smartphone, with its wide range of uses and excellent performance, has attracted numerous users. Still, this domination of the Android platform also has motivated the attackers to develop malware. The traditional methodology which detects the malware based on the signature is unfit to discover unknown applications. In this paper, detection has been tried whether an application is malware or not using Static Analysis (SA). Considered all the permissions that an application asks for and took them as input to feed our machine learning models.

Objectives. The objectives to address and fulfill the aim of this thesis are: To find/create the necessary data set containing malware in the android systems. To test this, different classifiers have been built using different machine learning (ML) algorithms such as Support Vector Machine (SVM) (Linear and RBF), Logistic Regression (LR), Random Forest Algorithm (RF), Gaussian Naive-Bayes (GNB), Decision Tree Method (DT) etc., and also compared their performances. To evaluate and compare each of the chosen models using Accuracy, Precision, F1-Score and Recall methods among the algorithms mentioned in detecting the malware in android with better accuracy in real-time scenarios.

Methods. To answer the research question, 1 method has been chosen which is: To identify malware in android system, the Experiment has been used.

Results. The Sequential Neural Network (SNN) performed well on the dataset with 98.82 percent than the other Machine Learning (ML) algorithms. So, it is the most fruitful algorithm for the Android malware detection. Random Forest (RF), Decision Tree (DT) are the second-best algorithms on the dataset with 97 percent.

Conclusions. Among Logistic Regression, KNN, SVM Linear, SVM RBF, Decision Tree, Random Forest, Gaussian Naive Bayes, and Sequential Neural Network Random Forest is declared as the most efficient algorithm after comparing all the models based on the performance metrics Precision, Recall, F1-Score and also by calculating Accuracy. Random Forest is considered as the most efficient algorithm among the four algorithms when they were compared.

Keywords: Malware, Machine Learning, Random Forest, Sequential Neural Network.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my family and friends for their moral support in helping me finish this thesis.

I would also want to express my gratitude to my examiner, Dr. Prashant Goswami, for his constant assistance and feedback throughout this thesis, his lectures on research technique assisted me in finishing this thesis in a responsible way.

Authors:

Rahul Sai Kesani

CONTENTS

ABSTRACT	III
ACKNOWLEDGEMENTS	IV
CONTENTS	V
1 INTRODUCTION	4
1.1 AIM AND OBJECTIVES	6
1.1.1 <i>Aim and Motivation</i>	6
1.1.2 <i>Problem Statement</i>	6
1.1.3 <i>Outline</i>	7
1.1.4 <i>Research Questions</i>	7
2 BACKGROUND	8
2.1 MACHINE LEARNING	8
2.1.1 <i>Methods in ML</i>	8
2.1.2 <i>Deep Learning</i>	9
2.2 ALGORITHMS	9
2.2.1 <i>Logistic Regression</i>	9
2.2.2 <i>K-Nearest Neighbors</i>	9
2.2.3 <i>SVM Linear</i>	10
2.2.4 <i>SVM RBF</i>	11
2.2.5 <i>Decision Tree</i>	11
2.2.6 <i>Random Forest Method</i>	12
2.2.7 <i>Gaussian Naïve Bayes</i>	13
2.2.8 <i>Sequential Neural Network</i>	13
2.3 PERFORMANCE METRICS	15
2.3.1 <i>Accuracy</i>	15
2.3.2 <i>Precision</i>	15
2.3.3 <i>Recall</i>	15
2.3.4 <i>F1-score</i>	15
3 RELATED WORK	16
4 METHOD	19
4.1 EXPERIMENT	19
4.2 ENVIRONMENTAL SETUP	19
4.2.1 <i>Software Environment</i>	19
4.2.2 <i>Hardware Environment</i>	20
4.3 DATA COLLECTION	20
4.4 IMPLEMENTATION	20
4.5 METHODOLOGY	20
4.6 FEATURE ENGINEERING	22
5 RESULTS AND ANALYSIS	24
6 DISCUSSION	27
6.1 FOR RQ1	27
7 CONCLUSION AND FUTURE WORK	29
7.1 CONCLUSION	29
7.1 FUTURE WORK	29
8 REFERENCES	30
9 APPENDIX	34

LIST OF FIGURES

2.1	K-NEAREST NEIGHBORS	10
2.2	SUPPORT VECTOR MACHINE TERMINOLOGY	10
2.3	SVM RBF	11
2.4	DECISION TREE	12
2.5	RANDOM FOREST METHOD.....	13
2.6	SEQUENTIAL NEURAL NETWORK	14
4.1	HEATMAP CORRELATION.....	23
5.1	CONFUSION MATRICES	24
5.2	CONFUSION MATRICES	25
5.3	LOSS AND ACURACY AFTER EVERY EPOCH	26
9.0	APPENDIX (CODING PART).....	37
9.0	APPENDIX (CODING PART).....	38
9.0	APPENDIX (CODING PART).....	39
9.0	APPENDIX (CODING PART).....	40
9.0	APPENDIX (CODING PART).....	41
9.0	APPENDIX (CODING PART).....	42
9.0	APPENDIX (CODING PART).....	45
9.0	APPENDIX (CODING PART).....	48
9.0	APPENDIX (CODING PART).....	49
9.0	APPENDIX (CODING PART).....	50

LIST OF TABLES

4.1	HARDWARE ENVIRONMENT	20
5.1	PERFORMANCE METRICS	24
5.2	SUMMARY OF SNN	26
9.0	APPENDIX (CODING PART)	34
9.0	APPENDIX (CODING PART)	35
9.0	APPENDIX (CODING PART)	39
9.0	APPENDIX (CODING PART)	43
9.0	APPENDIX (CODING PART)	44

1 INTRODUCTION

In the past few years, mobile devices, like smartphones, tablets, etc. have become popular by increasing the number and complexity of their capabilities. Present mobile devices offer a large number of services and applications as compared to the one's offered by personal computers. Thus, the number of security threats to the mobile devices has increased. And hence the hackers and malicious users are taking advantage of causing threat to the users' personal credentials due to the lack of security mechanisms. In 2016, malware attacks increased to a count 3,246,284 malware samples.

Android is the platform with the highest malware growth rate by the end of 2016. To overcome these security threats, various mobile specific Intrusion Detection Systems (IDSes) were proposed. Most of these IDSes are behavior based, i.e., they don't rely on a database of malicious code patterns, as in the case of signature-based IDSes. In this project, a machine learning based malware detection system has been described for android based smartphones users. This system exploits machine learning techniques to distinguish between normal and malicious applications. Proposed Android Malware Detection System to identify malware with efficiency and effectiveness. To develop a machine learning based malware detection system on Android to detect malware applications and to enhance security and privacy of smartphone users.

Here nowadays, the usage of mobile devices or smartphones has increasing in our daily and almost all of the people around world own a smartphone. According to Global market share, during second quarter of 2018, there was 88% smartphones in the market have been sold towards end users and that is Android systems [34].

Besides, it is becoming more and more popular because of its portability and convenient to use. For an example, the smartphone contains various types of functions and services like it can hold the personal information and access files that usually been stored in the cloud such as bank account information, email details, password and it also allows the user to interact with each other by sending a message or call. However, with the growth of the Android mobile popularity has brought many security concerns and threats from the attacker that might spread the malware that makes the system act differently than it is supposed to behave. The malware usually sent such fraudulent message and charge the user for their fake services.

According to the Security Threat Report (STR) released by Symantec in 2018 [33], the overall target activities that attacked is up by 10 percent in 2017. In fact, by March 12, 2018, there are 4, 964, 460 devices infected by RottenSys malware [36]. This situation desperately needs to find a potential method to detect malware before it harmed more Android smartphones. In this era globalization, people commonly used smartphones in such many ways like using a network connection to interact with the world. For example, online shopping, online banking, and cloud storage. Naturally, there are also has disadvantages by using this kind of network connections towards the user. Like example, the storing of confidential information in smartphones might attract the attacker to use dirty things in order to get user details like spreading malware towards some software or applications that might be installed in their smartphones either they realized or not especially for Android users.

Besides, there are many kinds of existing research that had been proposed to detect the malware by using various types of techniques and methods that implement into the application. For example, Google published an automated scan system for potential malware which is called as Bouncer application (BA) [37]. However, there is still has room for improvement of Android malware detection. The reason is the different type of method and techniques will come out with a different rate of error results.

Furthermore, there is still some false alarm occurred on Android devices that tricked the user. For example, there are 600,000 of Android user that have been downloaded the fake guide applications such

as Pokémon Go and FIFA mobile. This is because, they are mistakenly downloaded the malware application when they want to seek the guide for the games [35].

Research has shown that Android malware analysis can be done in three different ways: The first method involves the deployment of static [15] and dynamic [16]. Investigation of code of application in order to spot components that are malicious before loading the application into any device; The second method involve modification of the Android system in order to put in modules for monitoring and interception of abnormal behaviors that may occur on the device [17,18,19] while the third approach involve engaging virtualization to implement the separation of domains ranging from lightweight isolation of an application on the device to running multiple instances of Android OS on the same device [20,21].

However, recent study has shown that machine learning or “anomaly detection” approaches have now emerged to become a leading and more effective approach for defeating Android malware [22, 23, 24, 25]. Unlike the static analysis techniques that involves the manual examination of the AndroidManifest.xml file, source files and the Dalvik byte code, and the Dynamic analysis (DA) that involves running an application in a controlled environment to study its behavior, the Machine Learning approach involves learning the general rules and patterns from benign and malicious app samples and then allowing data-driven predictions of decisions, such as classification [26]. Machine learning methodologies largely depends on static attributes extracted from an application [27]. The static components of an Android application provide the baseline upon which machine learning approaches are anchored and these static features are carefully gotten through the process of reverse engineering.

Machine learning techniques have been applied widely for the classification of applications, focusing mainly on generic malware detection. The application of machine learning in Android malware detection helps eliminate the difficulty involved with manually crafting and updating detection patterns [22]. Machine Learning is a procedure that analyzes data using software techniques (algorithms) to create a model, as shown in Fig. 1, which is useful for finding patterns and regularities in datasets [28]. It is a process of making machines learn from past experiences (existing data) in order to make decisions on future occurring events or data instances. Feature vectors are very essential elements of Machine Learning and they are usually built for the specific task the Machine Algorithm intent to accomplish. The basic idea behind Machine Learning is to get the probability distribution of data.

Furthermore, there are three basic Learning Methods associated with each Learning Category; Classifications, Clustering, and Regression. Classification is the process used in Supervised Learning in which the data sets are well labelled into groups or classes; Clustering is the process used in unsupervised learning for unlabeled data sets; and Regression is best associated with Re-enforcement learning in which the expected end result is being ranked, graded or estimated. A label is the name of the definite class or group the data instances belongs to. In machine learning, data are represented by a fixed number of features which can either be categorical, nominal, or continuous [29]. This paper gives a thorough review of different existing literatures in the field of Android malware detections using machine learning techniques.

Authors in [30, 31] showed in their works that malware attack methods can be characterized as follows:

- **Information Extraction:** The malware in this category compromises a device and then steals personal information such as IMEI number, user’s personal information and many more.
- **Automatic Calls and SMS:** This group of malware increases a user’s phone bill by placing automatic calls and sending SMS to some premium numbers.
- **Root Exploits:** These set of malwares seek to gain system root privileges in order to take control of the system and modify the system’s configuration and other system information.

- **Search Engine Optimizations:** The malware here artificially searches for a term and simulates clicks on targeted websites in order to increase the revenue of a search engine or increase the traffic on a website.

- **Dynamically Downloaded Code:** This technique enables an installed benign application to download a malicious code and deploys it in the mobile devices without the user being aware.

- **Covert and Overt Communication Channels:** This is a vulnerability that is found in a device that facilitates the information leak between the processes that are not supposed to share the information. This technique is seen as a highly sophisticated.

- **Botnets:** This is a network of compromised mobile devices with a Bot-Master which is controlled by a Command-and-Control servers (C&C). It carries out Spam delivery, DDoS (Distributed Denial of Service) attacks on the host devices.

Malware Authors use many techniques to evade detection. [32] pointed out these concealment techniques to include code obfuscation techniques, encryptions, unnecessary permissions which are not needed by the application, requesting for unwanted hardware, and download or update attacks in which a benign application updates itself or another application with malicious payloads.

1.1 Aim and Objectives

1.1.1 Aim and Motivation

Aim: The thesis aims to implement a machine learning model that can detect the presence of any malware in an Android application accurately. Efficiency is measured based on accuracy and speed.

Objectives: The objectives listed below have been identified to help in fulfilling the aim of this thesis.

1. To find/create the necessary data set containing malware in the Android systems.
2. To divide safe and dangerous features from the data set of malwares in the Android systems.
3. To test this, different classifiers have been built using different machine learning algorithms such as Support Vector Machine (Linear and RBF), Logistic Regression, Random Forest Algorithm, Gaussian Naive-Bayes, Decision Tree Method etc.
4. To evaluate and compare each of the chosen models.
5. To identify the most efficient algorithm among compared models.

Motivation: In June 2021, Android retained its status as the world's most popular mobile operating system, with a market share of about 73 percent. As Android has grown in popularity, so has the amount of malware targeting the platform. Android has been the most popular target for malware makers since 2010, accounting for more than 90% of all mobile malware.

Android applications can be downloaded from a lot of sources including third-party app stores. These application stores serve as a point of entry for malware to spread quickly. The most common strategy used by malware creators is Repackaging, which involves embedding the harmful code segment within a legal application making the application malicious, and distributing it through third-party stores. Their malware programs are designed to get root access, stealing personal information, setting up mobile botnets and many more. So far, there are a large number of malwares discovered, some of which are more dangerous than others. They are harmful because they have the potential to steal user's personal information, as well as sending this data to remote servers. So, there is a need to develop models that can detect the presence of Android malware with accuracy and that is our motivation.

1.1.2 Problem Statement

Smartphones have become the most used device in one's day to day life. They facilitate users with a variety of applications that are enriched with powerful features. It is almost impossible for anyone these days to spend a day without their smartphones. Out of all smartphones, Android smartphones are the ones that are widely used. This increasing popularity of Android smartphones has also attracted malicious attackers. This malicious activity can be done by either a single application or a group of applications working together. The objective of this project is to create a model that can detect such malicious applications.

1.1.3 Outline

Chapter 1 discusses about the Introduction of my thesis, in which the Problem Statement has been discussed of my thesis, Research Questions (RQ), Aim and Motivation. In Chapter 2, provided the Background of my thesis which includes all the necessary information around the concepts and also gave an introduction about the models that were going to used and compared. In Chapter 3, the existing or previous research has been listed that has already been done related to my thesis. In Chapter 4, the Research Methodologies (RM) have been discussed that were used to answer the Research Questions. The Environment has been setup and done Feature Engineering (FE). Chapter 5 looks at the results obtained by training the models and comparing them based on the performance metrics Accuracy, Precision, Recall and F1-Score. In Chapter 6, Results have been discussed and discussed how the Research Questions have been answered using the Research Methodologies that have been chosen. In Chapter 7, some points have been put out related to the Conclusion of our research and also mentioned any Future Works that can be done to improve the thesis.

1.1.4 Research Questions

RQ1: Which is the most effective machine learning algorithm and has the most accuracy in detecting the malware in Android?

Method opted: Experimentation.

Motivation: Experimentation is chosen for this research question because a model has been built by training an algorithm to analyze the performance of algorithms. Experimentation can give better results when compared to reviews because an efficient model has been built. The main aim of the research is to find the most effective and accurate machine learning algorithm for the detection of malware in Android.

2 BACKGROUND

2.1 Machine Learning

Machine learning [11] can appear in many guises. A number of applications, the types of data they deal with, and finally, formalized the problems in a somewhat more stylized fashion. The latter is key if wanted to avoid reinventing the wheel for every new application. The art of machine learning is to reduce a range of fairly disparate problems to a set of fairly narrow prototypes. Much of the science of machine learning is then to solve those problems and provide good guarantees for the solutions. There are seven steps in machine learning which are:

1. Gathering Data
2. Preparing Data
3. Choosing a model
4. Training the model
5. Evaluating the model
6. Tuning the model
7. Predicting results

2.1.1 Methods in ML

There are different types of machine-learning methods [12]. So divided them into categories, depending on whether

- They have been trained with humans or not
 - Supervised
 - Unsupervised
 - Semi-supervised
 - Reinforcement Learning (RL)
- If they can learn incrementally
- If they work simply by comparing new data points to find data points, or can detect new patterns in the data, and then will build a model.

Supervised Learning:

In this type of machine-learning system, the data that you feed into the algorithm, with the desired solution, are referred to as “labels”. Supervised learning groups together a task of classification. The most important supervised algorithms are:

- K-nears neighbors
- Linear regression
- Neural networks
- Support vector machines
- Logistic regression
- Decision trees and random forests

Unsupervised Learning:

In this type of machine-learning system, you can guess that the data is unlabeled. The most important unsupervised algorithms are:

- Clustering: k-means, hierarchical cluster analysis
- Association rule learning: Eclat, apriori
- Visualization and dimensionality reduction: kernel PCA, t-distributed, PCA

As an example, suppose you've got many data on visitor Using of one of our algorithms for detecting groups with similar visitors. It may find that 65% of your visitors are males who love watching

movies in the evening, while 30% watch plays in the evening; in this case, by using a clustering algorithm, it will divide every group into smaller sub-groups.

Reinforcement Learning:

Reinforcement learning (RL) is another type of machine-learning system. An agent “AI system” will observe the environment, perform given actions, and then receive the rewards in return. With this type, the agent must learn by itself. Ties called a policy. You can find this type of learning type in many robotics applications that learn how to walk.

Semi-supervised Learning:

In this method [12] the algorithm is provided with both labelled and unlabeled data. By using both labeled and unlabeled data, the algorithm can learn to label the unlabeled data and use them to predict the results.

2.1.2 Deep Learning

Deep learning (DL) [13,14] is a class of machine learning which performs much better on unstructured data. Deep learning techniques are outperforming current machine learning techniques. It enables computational models to learn features progressively from data at multiple levels. The popularity of deep learning amplified as the amount of data available increased as well as the advancement of hardware that provides powerful computers.

2.2 Algorithms

2.2.1 Logistic Regression

Logistic Regression is a predictive analytic technique built on the notion of probabilities. It is a Machine Learning algorithm that would be used for problems which involves classifying. Here, hypothesis function is taken which will tell us the probability that a sample belongs to a particular class. In logistic regression, sigmoid function has been used as hypothesis function.

$$h_{\theta}(x) = \sigma(x^T \theta)$$
$$\sigma(x^T \theta) = \frac{1}{1 + e^{-x^T \theta}}$$

Ideally, logistic regression classifier is wanted to give an accurate prediction. For that, θ should be found that minimizes the cost function $J(\theta)$. This has been done using gradient descent algorithm

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [(1 - y^i) \log(1 - p^i) + y^i \log(p^i)]$$

where p_i is the probability that i is a malicious application and y_i is the labelled class for that application.

2.2.2 K-Nearest Neighbors

K-Nearest Neighbors is a straightforward machine learning approach that may be applied to regression problems as well as tasks involving classifying [9]. In KNN, K applications have been looked for that are nearest to the given instance application. Also noted down the classes of all those K applications and count how many times each class appeared. The class that appeared the most will be the class of our instance application. Here the K nearest neighbors is chosen using the Euclidean distance (ED).

Minimum error:- 0.59 at K = 37

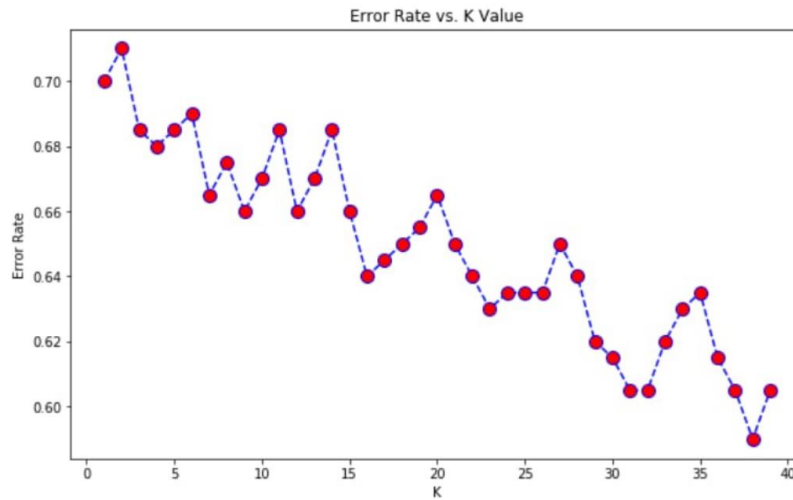


Figure 2.1: Plot between error rate and K

Here K value has to be chosen that gives the least training error rate. There isn't any particular method to do that but when plotted an error rate and k on a 2D-plane, K value has been taken that gives us least error.

2.2.3 SVM Linear

A SVM Linear classifier is built to fit the data you supply and provide a hyperplane that fits well and classify your data into different classes. Following that, you may input some attributes to your classifier to check what the projected class is once you've obtained the hyperplane.

Support Vectors are the points which can be considered as edge cases. They are very nearer to the hyperplane. The two support vectors corresponding to either classes benign and malicious respectively are equidistant to the hyperplane with maximum margin possible [19].

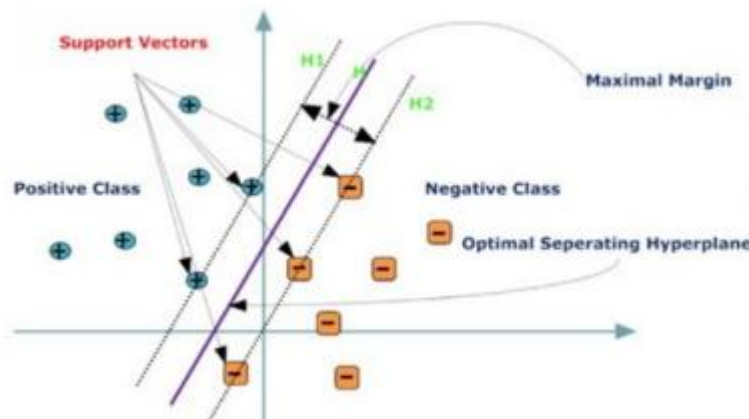


Figure 2.2: Support Vector Machine Terminology

2.2.4 SVM RBF

In classification using Support vector machines, the model with polynomial kernel performs merely when the positive examples and negative examples in the data are overlapping. One way to deal with this overlapping data is to use a support vector machine with a radial kernel generally known as Radial Basis Function (RBF). When RBF kernel is used in SVM, radial kernel behaves like a weighted nearest neighbor model. In other words, the nearest observation has a lot of influence on how the new example has been classified. The value obtained after substituting in radial kernel function is inversely proportional to the closeness [1]. The radial kernel function of two data observations a, b is as below.

$$RBF(a, b) = e^{-\gamma(a-b)^2}$$

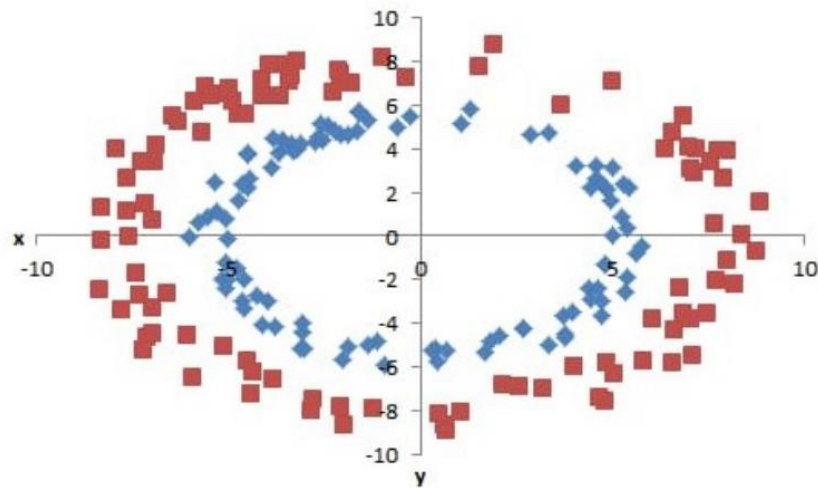


Figure 2.3: Example of a dataset that isn't linearly separable

2.2.5 Decision Tree

Decision Trees are a non-parametric supervised learning approach which can be used for classifying problems as well as regression problems. The sole objective is to construct a machine learning model that guesses the class of a given instance by learning basic decision rules from feature values.

First, entropy will be calculated. It is also known as measure of uncertainty. Then for each attribute A , information gain has been calculated. The attribute with maximum value for information gain will be selected as the root node and this process continues [29]. The formulas for entropy and Information gain are:

$$E(S) = -[p \log(p) + (1 - p) \log(1 - p)]$$

$$Gain(S, A) = E(S) - \sum_v p_v E(S_v)$$

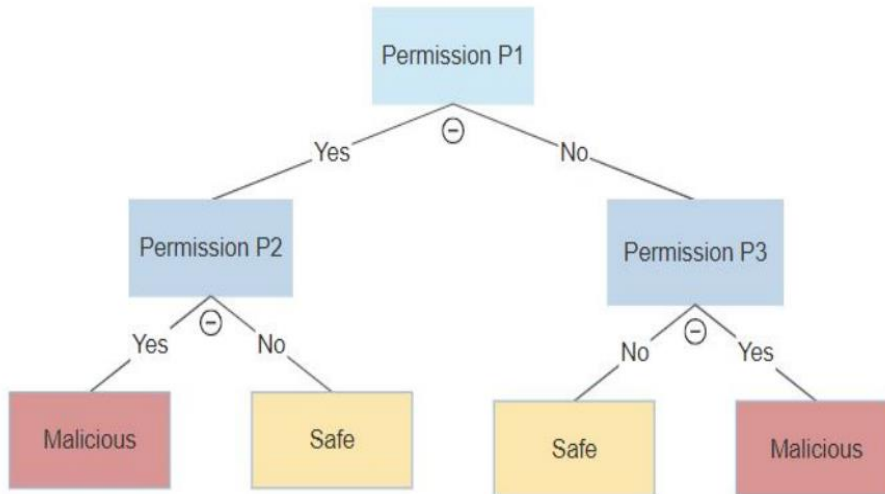


Figure 2.4: Illustration of Decision Tree

2.2.6 Random Forest Method

Random forest algorithm is a supervised learning method of machine learning. It produces a "forest" out of a collection of multiple decision trees. These trees are often trained using an approach called bagging approach. The main idea of this approach is that combining many learning models enhances total output. In simple words, random forest is a method of combining many decision trees to get a more accurate and reliable prediction.

Random Forest is a Machine learning classifier that has various decision trees on different subsets of input datasets and takes the majority of them to improve the performance of the model. The greater number of decision trees that have been taken, the higher the performance of the model will be [18].

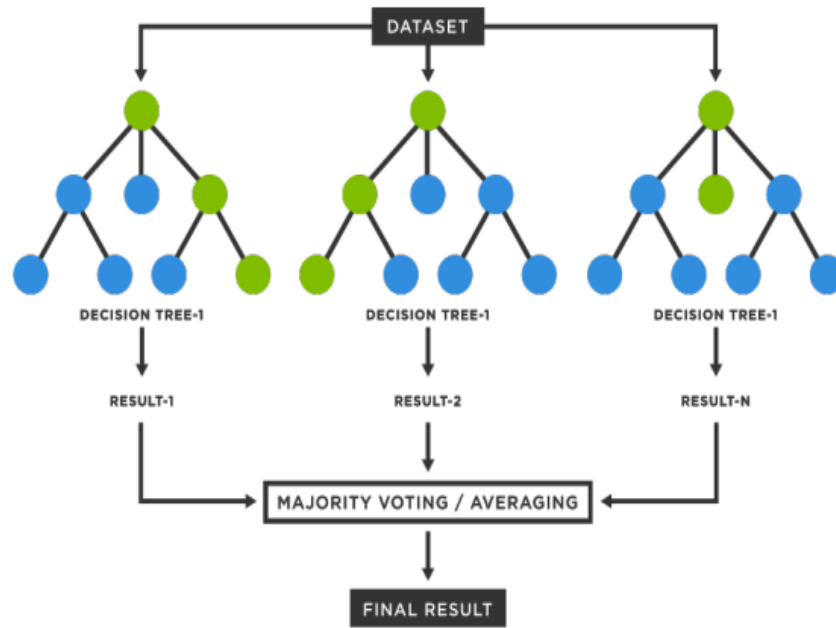


Figure 2.5: Illustration of Random Forest

2.2.7 Gaussian Naïve Bayes

A collection of classification algorithms which are based on a theorem named after Bayes together forms a Bayes classifier. It isn't a single algorithm. It is a group of algorithms sharing one common principle i.e., every pair of features which are used in classification are independent of each other.

For two events E_a , E_b Bayes theorem tells that:

$$P(E_a|E_b) = P(E_b|E_a) * \frac{P(E_a)}{P(E_b)}$$

Using this principle for classification task, I would say that:

$$Pr(class|attributes) = Pr(attributes|class) * \frac{Pr(class)}{Pr(attributes)}$$

As there were two classes malicious and safe, the probabilities will be calculated for the application to be in malicious class and to be in safe class. The class for which the probability value is higher, is the class to which the application belongs to.

$$\hat{class} = \underset{class}{\operatorname{argmax}} P(class|attribute)$$

2.2.8 Sequential Neural Network

Sequential neural network has been taken with 3 hidden layers. It has 1 input layer and 1 output layer also. The input layer consists of 54 features. The first hidden layer has 16 nodes. Second layer has 8 nodes. And the final hidden layer has 4 nodes. The three hidden layers have relu activation function and output layer has sigmoid function.

Divided the input data into batches of size 32. It is called mini-batch mode. Rmsprop optimizer has been used for the optimization to be preventing the decline of learning rate of our model [7]. 30 epochs have been taken. Epoch means passing the whole training data into our neural network once. In 10 epochs, whole training data has been passed 10 times.

After every epoch, the average loss is computed and the neural network back-propagates to the first hidden layer and updates the weights such that the loss is minimized. Cross entropy had been used as our loss function.

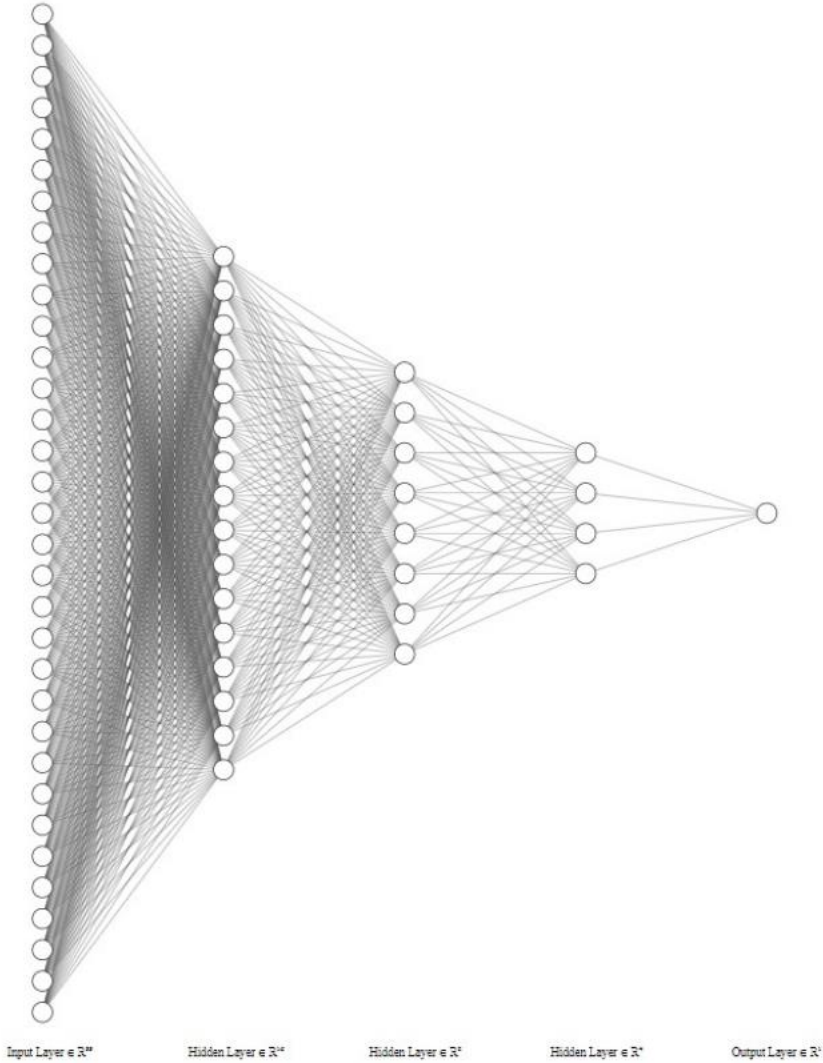


Figure 2.6: Illustration of Sequential Neural Network showing all its layers

2.3 Performance Metrics

2.3.1 Accuracy

Accuracy can be computed by dividing the number of correctly classified applications by total number of applications.

$$Accuracy = \frac{No.ofcorrectlyclassifiedapplications}{Totalno.ofapplications}$$

2.3.2 Precision

Precision can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that are predicted to be in that particular class.

$$Precision = \frac{TP}{TP + FP}$$

2.3.3 Recall

Recall can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that should actually be in that particular class.

$$Recall = \frac{TP}{TP + FN}$$

2.3.4 F1-score

F1-score is a metric involving both Precision and Recall and can be computed as:

$$F1 - score = \frac{2 * Precision * Recall}{Recall + Precision}$$

3 RELATED WORK

The techniques have been studied that are proposed to identify Android malwares. In his work, Anshul et al. [1] presented an idea to detect Android Malwares by Network traffic (NT) analysis. Their approach is used to identify malware on Android that is operated by a remote server. These malwares either accept orders from the server or leak sensitive data to it. First, they analyzed the network traffic of android malwares and then the traffic of normal applications. They discovered the characteristics that distinguish malware traffic from non-malware traffic. And in the second phase, they built a classifier using these network traffic features which can detect the malwares.

In another work, Anshul et al. [2] proposed a technique called the Perm Pair (PP) method. They approached the goal by considering every pair of permissions as the possible input feature and finally decided on each pair, if that combination is vulnerable. Their method includes data sets from 3 different sources called Genome, Debris and Koodoos. Their approach had 3 phases. In the first phase, they constructed 4 different graphs by extracting permission pairs from each application. Out of the 4 graphs, 3 graphs are for malwares and 1 graph is for benign applications. In the second phase, they dealt with merging 3 malicious graphs into a single malicious graph. At the end of this phase, they ended up with two graphs, one for malicious and one for benign. In the third and final phase of their method, they developed a model that calculates two scores called normal score and malicious score for every application and decides whether a particular application is malware or not.

The most commonly used properties in static and dynamic Android malware detection are permissions and network traffic features respectively. Static permissions cannot identify sophisticated malware, which is capable of update attacks. And coming to dynamic network traffic, it cannot detect malware samples without a network connection. Therefore, a hybrid model integrating both of these properties is proposed. They extracted both permissions and network traffic features and made them into a single vector. Using the K-medoids method, they partitioned the vectors into K clusters. And they used the K-Nearest Neighbors method, to classify whether a particular application is malicious or not. They made sure that K is odd, just to make sure out of K nearest neighbors, the count of malicious and benign neighbors is not the same.

In another work, Zhenlong Yuan et al. [3] proposed a technique to associate static features with dynamic features and then classify the given android applications as malicious or safe. They got the features they used as input to their model in three stages:

- Static Phase
- Sensitive APIs
- Dynamic Phase

Static phase includes the permissions that are obtained by unzipping the apk file and parsing xml files obtained later. Another file classes.dex accounts for the sensitive api calls. To obtain the dynamic features, they used to install the apk files on droid box, which is an extended sandbox of Taint droid, which is itself capable of recording the network traffic, information leak and other run time analytics that could play a major role in determining an app as malware or not. They implemented an online android detection engine based on deep learning models. They are able to develop a deep learning model that outperforms many states of art machine learning techniques.

In another work, Suleiman et al. [4] examined the use of varied machine learning techniques in a parallel classification approach to Android malware detection. The proposed method made use of a variety of properties, API calls, instructions, and other API-related topics were provided. The recent rise of Android usage Malware and its ever-increasing ability to be detected, current signature-based techniques are avoided. The classification approach adapted by them is a realistic option. The method that not only gives a supplemental tool but also it has the potential to improve Android malware detection, but it also has some drawbacks. It only permits different classifiers' strengths to be combined.

Assisting with further phases of analysis Moreover the planned from a performance standpoint, the technique is great since it is economical when it comes to classifying an unknown instance because the app's static features are consumed and also chosen features are consumed. Models for constituent categorization have a minimal computational cost before making a categorization judgement.

Xingquan Zhu et al. [5] proposed a feature-based learning method that takes input as the permissions and api calls an app is intended to make during its life time. The proposed frame work consists of 4 different phases. They are:

- App Analyser (decompresses) the apk file
- Fetches permissions and api calls
- Feature generator (Generate binary datasets)
- Data mining models (SVM, Bagging, Decision Tree)

Though this approach doesn't involve any dynamic analysis, this approach of complimenting the permissions with the api calls improves the performance of the model and helps in the better performance of the model.

To obtain input parameters from bundled Android apps, Justin et al. [6] used an apk called Androguard. This androguard is used to obtain permission features from the jar files included in the android apk file. They then utilized the Scikit learn framework, a python library. It gives a straightforward interface to LIBSVM, to train a SVM having 1 class, utilizing these obtained features. They aimed to reduce the high false positive rates generally seen in these android malware detection techniques. They are only reported after testing data as malware if it differs sufficiently from the training data, which is ideal for their purposes because the range of benign applications are more widely available than malicious applications. For the Android operating system, they introduced a revolutionary malware detection solution which works on the principle of ML.

In Zi Wang et al. [7], they offered DroidDeepLearner (DDL), an Android malware characterization and identification strategy that uses a deep learning algorithm to distinguish malicious Android apps from benign ones. They carried out a series of tests to ensure that the suggested technique was both legitimate and effective, and compared the performance of the proposed DroidDeepLearner with that of SVM, which is one of the most effective and commonly used algorithms for malware detection. They ran a variety of experiments using real Android app datasets to validate the malware detection technique, and the findings suggest that the scheme can accurately identify malware in the Android. When compared to traditional solutions which works on SVM as backbone, experiment results suggest that the proposed system is capable of accurately identifying malware.

In Hyo-Sik Ham et al. [8], a study is undertaken on the detection of malware using an Android device, which is the most commonly used as an attack point by attackers. This paper presents a feature selection and experimentation technique for lowering the number of malware detections that are false positives and improving device performance degradation that has been identified as a problem in the current system Machine learning based mobile malware detection. This model also analyses malware detection methods and performance of machine learning classifiers. First, they must compare the agent's resource consumption to that of an existing agent that is monitoring all features. In the future, more improved variations of Android malware could be collected to investigate their attributes. Also, research could be conducted on identifying new malwares that doesn't exist in the known directory.

As there are a many number of permissions an application requests for during the installation, It is important to pick out the most prominent permissions among those that affect the malicious activity of the application. Feature selection is the phase that facilitates the developers with this task. The dataset generated after this feature selection accounts for the more accurate results. In this paper, Pehlivan et al. [9] used a tool called Waikato Environment for Knowledge Analysis (WEKA) to achieve feature selection. The metrics that are employed in evaluation of effective feature selection are TruePositiveRate (TPR), FalsePositiveRate (FPR) and Precision. They also took accuracy as one of their metrics. A feature

is employed in the classification dataset on the basis of Gain Attribute Evaluation (GAE) attribute. The higher is this attribute, the more impact the feature keeps on the output. They considered this gain attribute to decide whether a feature is to be included in the training data set or not.

In another work, Fereidooni et al. [10] used a tool named unitPdroid written in python 3 to extract the prominent features i.e., a technique used in feature engineering. The permissions have been extracted from the class.xml file unzipped from the apk file of the android application. The permissions are then processed and modified into binary features so that it could be easier to train our classification models. This approach also incorporates the suspicious API calls as a feature that can affect the malicious activity of the application. This paper employed an approach called dalvik bytecode analysis to determine if an application is malware. This analysis is based on the kind of information that the application is forwarding to the remote servers. If the application forwards the sensitive information or anything serious, it's rated more probably as the malware. They are able to achieve results more efficiently than many conventional state of art techniques.

4 METHOD

In this chapter, getting into the experimentation and methods that are used to solve the research questions mentioned earlier. The feature engineering will also be discussed and the whole methods used to train the models on the data set.

4.1 Experiment

The most suitable research method to answer Research Question 1 is performing an experiment. The main goal of this experiment is to train the models on the selected data set and evaluate them.

4.2 Environmental Setup

4.2.1 Software Environment

Python:

Python programming language is considered as one of the most famous and versatile open-source programming languages. Python programming language is considered relatively programmer-friendly as it is closer to human languages how it is written. There are many libraries in python to help the programmers build the neural networks and help them in any deep learning process such as object detection. A few python libraries have been used by the researchers to complete the experiment procedure.

Matplotlib:

Matplotlib (MP) [16] is a basic visualizing/plotting library of python programming. It is a tool used to create different types of visualization reports such as line plots, histograms, pie charts, bar charts and many more. Matplotlib also supports 3-dimensional plotting.

NumPy:

NumPy (NP) is a python library which is widely used handle arrays with arrays. NumPy can provide the programmer with an array object that is 50 times faster than the traditional python list. NumPy arrays can be created by converting existing tuples or lists to arrays using `np.array` or by initializing fixed length arrays using NumPy functions.

SciPy:

SciPy (SP) [6] is a computational python library that uses NumPy underneath. It provides more utility functions for optimization, stats and signal processing. SciPy stands for Scientific Python.

Pandas:

Pandas is one of the most important libraries in Python programming for Data Analysis (DA) and Data Science (DS). Pandas is majorly used to handle tabular data. The name Pandas is derived from the term "panel-data". Pandas library can be used for visualization, building and training deep learning/machine learning models and also data manipulation [34].

Seaborn:

Seaborn is a library mainly based on matplotlib. Seaborn is chosen because it combines aesthetic appeal seamlessly with technical insight. Seaborn is used for data visualization. Seaborn is considered as a high-level interface compared to matplotlib.

scikit-learn:

scikit-learn [5] is a library built on NumPy, SciPy and matplotlib. It is considered as a simple and an efficient tool for data mining and data analysis. Scikit-learn is mainly used for Classification, Regression and Clustering.

4.2.2 Hardware Environment

The hardware environment which was used to train the models is shown in the table below:

System	Aspire E5-575 series
CPU	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
GPU	
Installed RAM	12.00 GB
Operating System	Windows 10 Pro

Table 4.1: Hardware Environment

4.3 Data Collection

Any machine learning model needs a dataset over which it can be trained. So, data collection is one of the most important steps. In this project the dataset has been downloaded from Kaggle. It has 138047 records with each record containing the information about all the permissions that a particular application asks for. The dataset has 57 columns which are considered features for our machine learning model. Also made sure that the dataset contained enough examples for both the malware and benign applications. There are 41323 examples for benign applications and 96724 applications for malicious applications. So, here the dataset is not skewed. Each column contains information of whether that application asks for particular permission or not. A value of 1 means that the application takes that permission and a value of 0 means that application does not take that permission.

For training and testing purposes, dataset has been split into two parts. 80% of the dataset has been used for training the machine learning model and the remaining 20% dataset was used for testing every machine learning model and calculating the performance of each model with metrics such as accuracy, f1-score, precision and recall.

4.4 Implementation

The implementation section is the main part of the thesis after results. In this section, gone through the process of installing software's and training our models with the train data sets so that they can detect the malware in the test data set. There are two main steps that have to be followed in the implementation section:

- Installing
- Training

4.5 Methodology

Installed all the software and frameworks mentioned in software environment section for the implementation of thesis. Proper hardware has been used mentioned in the hardware environment section and gathered the data as explained in data collection and proceeded with Training.

In Training, columns of features have been selected, that are mainly used to determine the attribute values:

- Machine
- SizeOfOptionalHeader
- Characteristics
- MajorLinkerVersion
- MinorLinkerVersion
- SizeOfCode
- AddressOfEntryPoint
- BaseOfData
- ImageBase
- SectionAlignment
- FileAlignment
- SizeOfImage
- SizeOfHeaders
- CheckSum
- SizeOfStackReserve
- SizeOfHeapReserve
- LoaderFlags
- SectionsNb
- SectionsMeanEntropy
- SectionsMeanRawsize
- SectionsMeanVirtualsize
- ResourcesNb
- ResourcesMeanEntropy
- ResourcesMeanSize
- LoadConfigurationSize and 30 more.

The training is done by considering all the below attributes which determines the Android App malware by considering the following items:

- Category
- Rating
- Number of ratings
- Price
- Dangerous Permissions count
- Safe Permissions count
- Hardware controls: control vibrator(S)
- Hardware controls: record audio(D)
- Network communication: full Internet access(D)
- Network communication: receive data from Internet(S)
- Network communication: view Wi-Fi state(S)
- Network communication: view network state(S)
- Phone calls: read phone state and identity(D)
- Services that cost you money: directly call phone numbers(D)
- Storage: modify/delete USB storage contents modify/delete SD card contents(D)
- System tools: automatically start at boot(S)
- System tools: modify global system settings(D)
- Your accounts: discover known accounts(S)
- Your location: coarse(network-based) location(D)
- Your personal information: read contact data(D)
- Class
- App_Upper

The attribute values have been determined by the columns of features like Machine, SizeOfOptionalHeader (SOH), Characteristics, MajorLinkerVersion (MLV), etc. So, the main aim of the research is to find the attribute values of Safe Permissions attributes and Dangerous Permission attributes and accurate machine learning algorithm for the detection of malware in Android.

So from all the above attributes, Dangerous Permissions Count by class and Safe Permissions Count by class are the parameters that helped in the detection of the malware because these are the parameters that gave the better results for the malware detection when compared to all the above parameters that are mentioned. So, these are the only main parameters that are helped in the detection of malware.

After considering all the attributes and confusion matrix has been determined and using sklearn framework that imported Logistic Regression, K-Nearest Neighbors Classifier, SVC linear, SVC rbf, Gaussian Naïve Bayes, Neural Networks, Decision Tree Classifier, Random Forest Classifier and applied them across the confusion matrix and obtained results individually for respective algorithms.

4.6 Feature Engineering

The feature set used for training has a big impact on machine learning. Several research have found that certain features are helpful in training machine learning-based malware classifiers. That is the reason feature engineering was used in our implementation. In supervised learning, Feature Engineering (FE) will be used, which is the process of selecting, manipulating, and changing raw data into features.

Feature Engineering was used for the following reasons:

- To remove imputation
- Handling outliers
- To achieve Normalization
- Null Value Handling
- To remove invalid data
- To achieve scaling

Firstly, if a permission is not asked by any application in our dataset, that column contains all 0's. That column has been dropped because it doesn't help in predicting the class of any application.

If any column contains less than 1000 1's or less than 1000 0's, that column has been dropped because of the same reason. Dropped the columns that don't have Numerical values or columns containing NaN values.

In the Price column, if the price of any application is greater than 0, value to 1 has been changed and for remaining applications, value to 0 has been changed. Here, 0 indicates that the application is free and 1 indicates that the application is paid.

After the process of feature selection, left with 54 columns which are going to be considered as features that are used for training the machine learning models.

Computed the correlation coefficients for each feature pair. Correlation coefficient between two variables tells us how those two variables are dependent on each other. Heatmap is a pictorial representation of the correlation matrix. In figure-1, heatmap of 54 features can be seen.

5 RESULTS AND ANALYSIS

In Table-5.1, performance metrics for various models can be seen. The performance metrics include accuracy, precision as well as recall. It also has f1-score.

S.No	Classifier Type	Accuracy	Precision	Recall	F1-score
1	Logistic Regression	91.02	0.91	0.91	0.91
2	K-Nearest Neighbours	92.56	0.90	0.93	0.92
3	SVM Linear	93.85	0.94	0.92	0.93
4	SVM RBF	94.79	0.94	0.93	0.93
5	Gaussian Naive Bayes	53.11	0.71	0.64	0.52
6	Decision Tree	97.14	0.95	0.96	0.96
7	Random Forest	97.45	0.97	0.97	0.97

Table 5.1: Performance metrics of different models

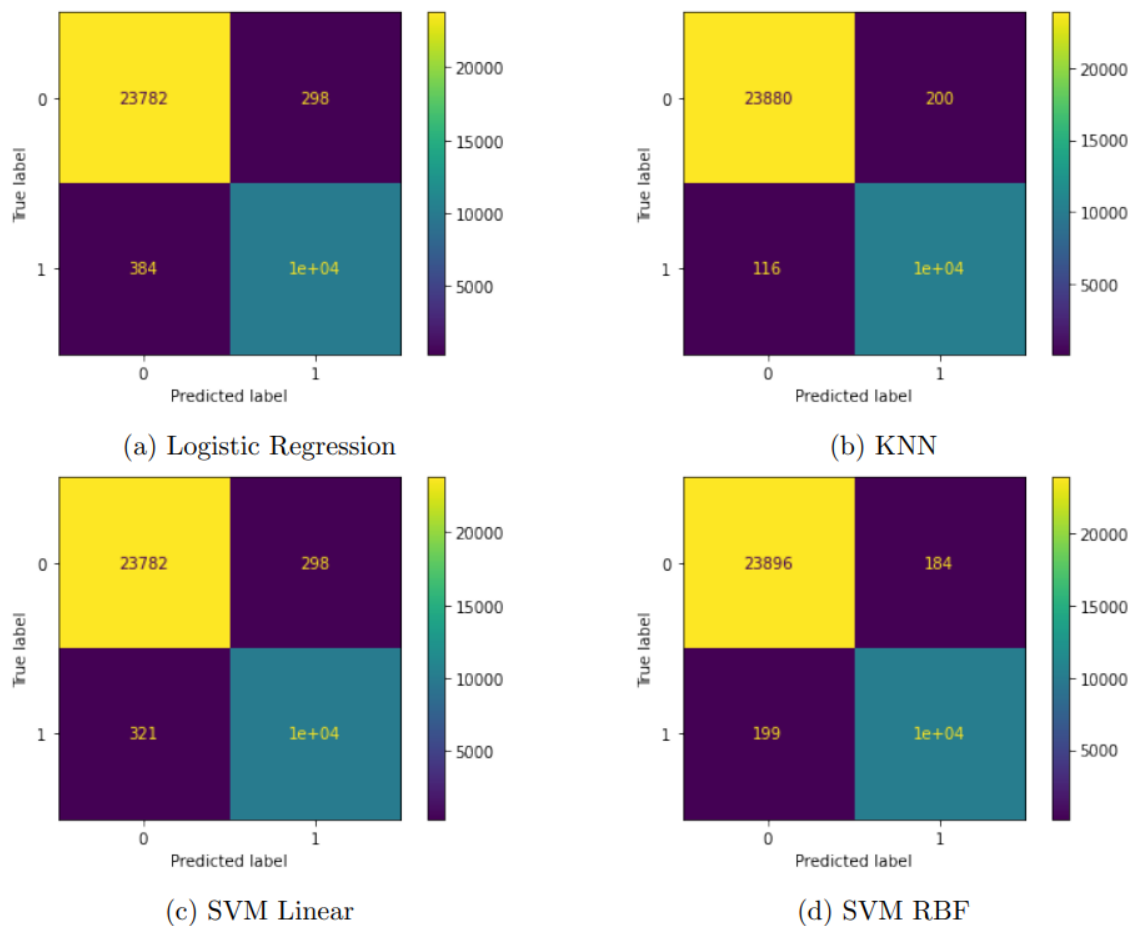


Figure 5.1: Confusion Matrices

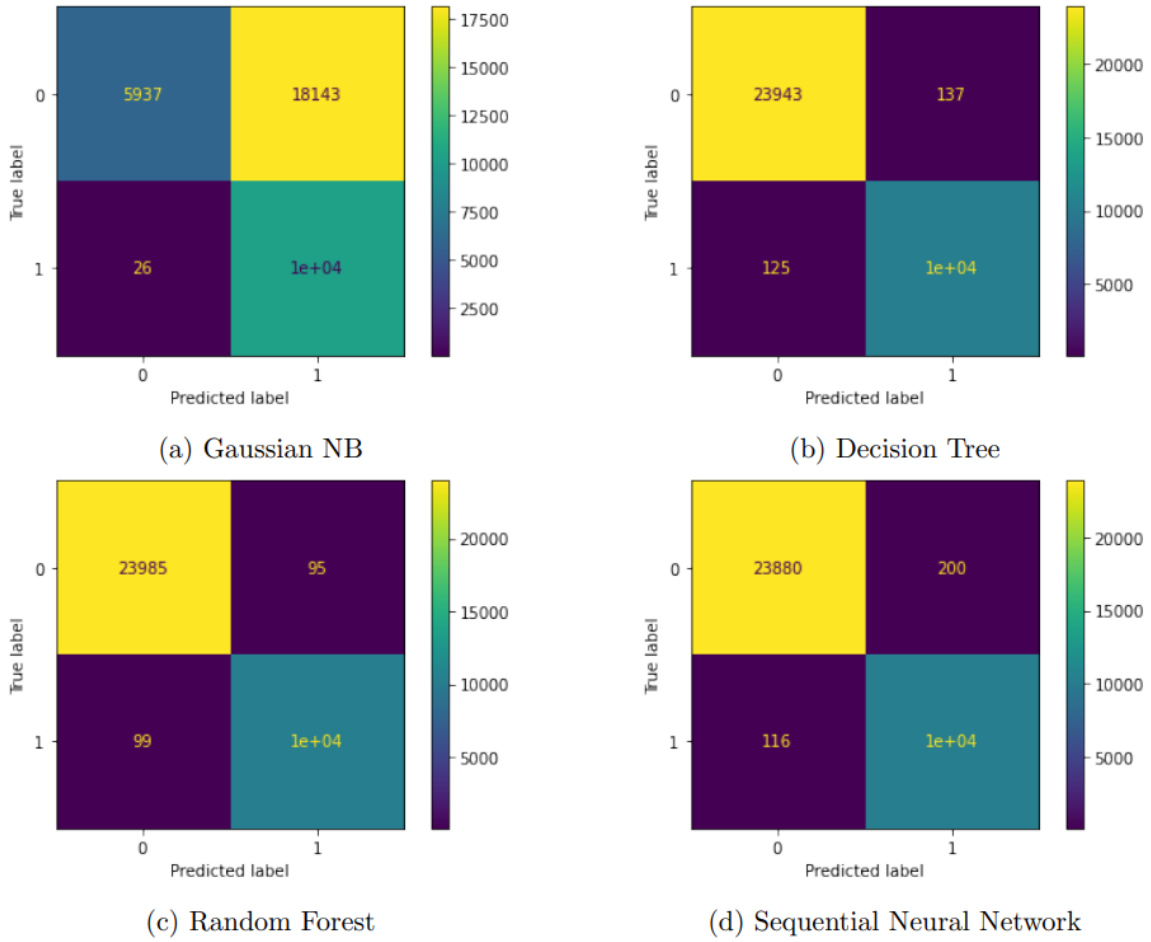


Figure 5.2: Confusion Matrices

In Figure-5.1 and Figure-5.2, Based on Dangerous Permissions Count, confusion matrices have been identified for various models. Confusion matrix compares the predicted classes with actual classes of applications. The values have been obtained for True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) from these confusion matrices.

Accuracy can be computed by dividing the number of correctly classified applications by total number of applications.

$$Accuracy = \frac{No.ofcorrectlyclassifiedapplications}{Totalno.ofapplications}$$

Precision can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that are predicted to be in that particular class.

$$Precision = \frac{TP}{TP + FP}$$

Recall can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that should actually be in that particular class.

$$Recall = \frac{TP}{TP + FN}$$

F1-score is a metric involving both Precision and Recall and can be computed as:

$$F1 - score = \frac{2 * Precision * Recall}{Recall + Precision}$$

For Sequential Neural Network (SNN), 3 hidden layers have been taken of length 16, 8 and 1.

Relu activation function was used for the hidden layers and sigmoid function for the output layer.

Layer	Size	No. of Param
dense	16	880
dense-1	8	136
dense-2	4	36
dense-3	1	5

Table 5.2: Summary of Sequential Neural Network

Cross entropy (CE) value has been taken for our loss function and chose rmsprop as our optimizer. After 10 epochs, accuracy obtained of 98.82 %.

```
Epoch 1/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0870 - accuracy: 0.9724
Epoch 2/10
3236/3236 [=====] - 5s 2ms/step - loss: 0.0545 - accuracy: 0.9868
Epoch 3/10
3236/3236 [=====] - 5s 2ms/step - loss: 0.0556 - accuracy: 0.9870
Epoch 4/10
3236/3236 [=====] - 5s 2ms/step - loss: 0.0553 - accuracy: 0.9874
Epoch 5/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0535 - accuracy: 0.9876
Epoch 6/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0501 - accuracy: 0.9878
Epoch 7/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0478 - accuracy: 0.9879
Epoch 8/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0477 - accuracy: 0.9879
Epoch 9/10
3236/3236 [=====] - 6s 2ms/step - loss: 0.0483 - accuracy: 0.9881
Epoch 10/10
3236/3236 [=====] - 5s 2ms/step - loss: 0.0488 - accuracy: 0.9882
```

Figure 5.3: Loss and Accuracy after every epoch

6 DISCUSSION

Many papers and articles have used machine learning algorithms for the detection of malware in Android. The articles were found by searching Google Scholar with some keywords such as Machine learning, Malware in Android, Prediction of Malware in Android system Regression analysis, SVR, Machine Learning performance metrics and so on. All the important highlights related to this research are noted down that consists of the use of classification in the prediction. In this literature review, various algorithms have been found that are used for the detection of malware in Android such as Logistic Regression, KNN, SVM Linear, SVM RBF, Decision Tree, Random Forest, Gaussian Naive Bayes, and Sequential Neural Network.

Table 5.1 shows the results of experimentation. The model has been found built with Random Forest algorithm has the highest accuracy and ahead of all other algorithms in other metrics. Gaussian Naïve Bayes got the least performance with the least values in all performance metrics. Although decision tree and random forest got the almost the same accuracy, random forest has higher values in other metrics, therefore it achieves the highest performance among all the algorithms. Therefore, based on the results of experimentation, Random Forest can be called as the most efficient algorithm in the detection of malware in Android followed by Decision tree algorithm.

Gaussian Naïve bayes Algorithm has the least performance and is not much suitable for the detection of malware in Android. Followed by Gaussian Naïve Bayes, the next followed by Logistic Regression, KNN, SVM Linear, SVM RBF has better performance. The lower performance of Gaussian Naïve Bayes model was possibly due to lack of enough training data. Too little training data leads to poor performance.

The attribute values have been determined by the columns of features like Machine, SizeOfOptionalHeader, Characteristics, MajorLinkerVersion, etc. So, the main aim of the research is to find the attribute values of Safe Permissions attributes and Dangerous Permission attributes and accurate machine learning algorithm for the detection of malware in Android.

Based on the Training the columns of features have been identified from them their determined the attributes Rating, Number of ratings, Price, Dangerous Permissions count, Safe Permissions count, Hardware controls: control vibrator(S), Hardware controls: record audio(D), Network communication: full Internet access(D), Network communication: receive data from Internet(S), Network communication: view Wi-Fi state(S), Network communication: view network state(S), Phone calls: read phone state and identity(D), Services that cost you money: directly call phone numbers(D), Storage: modify/delete USB storage contents modify/delete SD card contents(D), System tools: automatically start at boot(S), System tools: modify global system settings(D), Your accounts: discover known accounts(S), Your location: coarse(network-based) location(D), Your personal information: read contact data(D), Class. From the above attributes Dangerous Permission Count by class, Safe Permissions Count by class will determine the identification of malware in the Android.

6.1 For RQ1

RQ1: Which is the most effective machine learning algorithm and has the most accuracy in detecting the malware in Android?

Method opted: Experimentation.

Motivation: Experimentation is chosen for this research question because a model has been built by training an algorithm to analyze the performance of algorithms. Experimentation can give better results when compared to reviews because an efficient model has been built. The main aim of the research is to find the most effective and accurate machine learning algorithm for the detection of malware in Android.

Discussion: An Experiment has been conducted to analyze the data and it. The experiment research method consists of a few steps which include Data Collection and Training the models. After training the models, the results are evaluated on the basis on the metrics Accuracy, Precision, Recall and F1-score.

Accuracy Comparison

The accuracy obtained by the Logistic regression was 91.02 percent, after the training, 97.14 percent by the Decision Tree, 93.85 percent by SVM Linear, 94.79 percent by SVM RBF, 92.56 percent by K-Nearest Neighbors, 53.11 percent by Gaussian Naïve Bayes and by Random Forest classification with 97.45 percent.

Precision Comparison

After the training, 91 percent was the precision obtained by the Logistic regression, Decision Tree with 95 percent, SVM Linear with 94 percent, 94 percent by SVM RBF, 90 percent by K-Nearest Neighbors, 71 percent by Gaussian Naïve Bayes and by Random Forest classification with 97 percent.

Recall Comparison

After the training, 91 percent was the recall obtained by the Logistic regression, Decision Tree with 96 percent, SVM Linear with 92 percent, 93 percent by SVM RBF, 93 percent by K-Nearest Neighbors, 64 percent by Gaussian Naïve Bayes and by Random Forest classification with 97 percent.

F1-score Comparison

After the training, 91 percent was the precision obtained by the Logistic regression, Decision Tree with 96 percent, SVM Linear with 93 percent, 93 percent by SVM RBF, 92 percent by K-Nearest Neighbors, 52 percent by Gaussian Naïve Bayes and 97 percent by Random Forest classification.

Among the various Machine Learning approaches on the dataset, Random Forest classification had the highest F1-score. The lowest F1-score on the dataset was achieved by the Gaussian Naïve Bayes. Second place on the dataset was achieved by the Decision Tree. All of the chosen methods worked extremely well on the dataset. On the dataset, the selected algorithms had good accuracy, precision, and F1-score. The recall for SVM RBF, K-Nearest Neighbors, SVM Linear, Random Forest classification and Decision Tree algorithms was above 91 percent on the dataset, but the recall was just 64 percent for the Gaussian Naïve Bayes. Sequential Neural Network on the dataset was based on the accuracy results beat other ML classifiers. The second-best algorithm was Random Forest classification. Sequential Neural Network was 98.82 percent accurate, on this dataset. On the other hand, on the dataset, Decision Tree was 97.45 percent accurate. The Sequential Neural Network performed the best, as a result. When all of these comparisons are taken into account, the most efficient ML method among the eight algorithms for the Android malware detection is Sequential Neural Network.

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

As Android malwares are increasing day to day, wanted to present a model that can detect malwares accurately. Considered the permissions asked by an android application as the features to this machine learning model. This comes under Static Analysis. The idea of considering these permissions as features worked well. Got good accuracy from most of the machine learning models that were implemented. The Sequential Neural Network got the highest accuracy of 98.82%. Out of all the remaining models, Random Forest and Decision tree performed well. And Gaussian Naive Bayes model got the least accuracy among all the models that were implemented.

A set of Android applications operating together can carry out a malicious activity. Called them colluding apps. In this, the malicious activity is carried out by more than one application. Each application participating in collusion does a small part of the malicious action. These applications communicate with each other through covert channels. Sometimes when a malicious activity cannot be performed by a single application, it might be possible that a group of applications coordinating with each other can perform that malicious activity. This phenomenon is called Application Collusion. It is an emerging threat.

The reason behind why this was called as an emerging threat is because most of the Android malware detectors scan the applications individually when determining whether it is a malware or not. But as the malicious activity here is being carried out by a group of applications, those traditional detectors cannot detect this. So, a model is needed to detect these colluding applications. Till now, very little research has been done on this and there is scarcity for datasets.

7.2 Future Work

Till now, very little research has been done on this and there is scarcity for datasets. Tried to create or obtain a few applications that can perform collusion, so that some research can be done on them which may eventually help in creating a model that can detect colluding applications. Firstly, a template has to be obtained for collusion. Then, should to try to split a malicious task into various steps and make each application perform one of the steps. By doing this, collusion can be achieved.

Also have to try to increase an accuracy of neural network model for all types of datasets. Wanted to try to search the dataset and design the system for colluding apps. Also wanted to explore more techniques through which Android malwares can be detected and implement them. Also wanted to work further for implementing above work that would help for detecting Android malware with further more accuracy that helps people.

8 REFERENCES

- [1] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 66–71, 2014.
- [2] A. Arora, S. K. Peddoju, and M. Conti, "Permpair: Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.
- [3] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.
- [4] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 37–42, 2014.
- [5] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300–305, 2013.
- [6] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in 2012 European Intelligence and Security Informatics Conference, pp. 141–147, 2012.
- [7] Z. Wang, J. Cai, S. Cheng, and W. Li, "Droiddeeplearner: Identifying android malware using deep learning," in 2016 IEEE 37th Sarnoff Symposium, pp. 160–165, 2016.
- [8] H.-S. Ham and M.-J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in 2013 International Conference on ICT Convergence (ICTC), pp. 490–495, 2013.
- [9] U. Pehlivan, N. Baltaci, C. Acarturk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based android malware detection," in 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 1–8, 2014.
- [10] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5, 2016. *Computer Science and Information Systems*, 2nd ed., Springer, 2008.
- [11] Introduction to Machine Learning Alex Smola and S.V.N. Vishwanathan published by the press syndicate of the university of cambridge The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 2008.
- [12] Machine Learning: Step-by-Step Guide To Implement Machine Learning Algorithms with Python[Online]. Available: <https://www.pdfdrive.com/machine-learning-step-by-step-guide-to-implement-machine-learning-algorithms-with-python-e158324853.html>
- [13] Deep Learning Techniques: An Overview[Online]. Available: https://www.researchgate.net/publication/341652370_Deep_Learning_Techniques_An_Overview

- [14] “dEEP LEARNING FOR THESIS.” [Online]. Available: <https://www.mathworks.com/discovery/deep-learning.html>
- [15] Enck, W. (2011). Defending users against smartphone apps: Techniques and future directions. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7093 LNCS, 49–70. https://doi.org/10.1007/978-3-642-25560-1_3
- [16] Bläsing, T., Batyuk, L., Schmidt, A. D., Camtepe, S. A., & Albayrak, S. (2010). An android application sandbox system for suspicious software detection. *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010*, 55– 62. <https://doi.org/10.1109/malware.2010.5665792>
- [17] Backes, M., Gerling, S., Hammer, C., Maffei, M., & Philipp, von S.-R. (2012). AppGuard — Real-time policy enforcement for third-party applications. Saarbrücken, Germany.
- [18] Nauman, M., Khan, S., & Zhang, X. (2010). Apex. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security - ASIACCS '10*, 328. <https://doi.org/10.1145/1755688.1755732>
- [19] Xu, R., Saïdi, H., Anderson, R., & Saidi, H. (2012). Aurasium: Practical Policy Enforcement for Android Applications. In *Proceedings of the 21st USENIX Security Symposium* (pp. 539–552).
- [20] Andrus, J., Dall, C., Hof, A. V., Laadan, O., & Nieh, J. (2011). Cells. *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, 173. <https://doi.org/10.1145/2043556.2043574>
- [21] Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., & Peter, M. (2011). L4Android: A Generic Operating System Framework for Secure Smartphones. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 39–50. <https://doi.org/10.1145/2046614.2046623>
- [22] Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *Symposium on Network and Distributed System Security (NDSS)* (pp. 23–26). <https://doi.org/10.14722/ndss.2014.23247>
- [23] Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Roli, F. (2017). Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection, 1– 14. <https://doi.org/10.1109/tdsc.2017.2700270>
- [24] Ucci, D., Aniello, L., & Baldoni, R. (2018). Survey on the Usage of Machine Learning Techniques for Malware Analysis. *Computers and Security*, 1(1), 1–67. <https://doi.org/10.1016/j.cose.2018.11.001>
- [25] Rescuers, V. (2018). How Cybercriminals became ‘The New Mafia.’ Retrieved January 31, 2018, from <http://www.virusrescuers.com/how-cybercriminals-became-the-new-mafia/>

- [26] Coronado-De-Alba, L. D., Rodriguez-Mota, A., & Ambrosio, P. J. E.-. (2016). Feature selection and ensemble of classifiers for Android malware detection. In 2016 8th IEEE Latin-American Conference on Communications (LATINCOM) (pp. 1–6). <https://doi.org/10.1109/latincom.2016.7811605>
- [27] Hahn, S., Protsenko, M., & Müller, T. (2016). Comparative evaluation of machine learning-based malware detection on Android. In Sicherheit 2016: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-7. April 2016, Bonn (pp. 79–88). https://doi.org/10.1007/978-3-662-01089-1_31
- [28] Russell, I., & Markov, Z. (2017). An Introduction to the Weka Data Mining System. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE '17 (pp. 742–742). <https://doi.org/10.1145/3017680.3017821>
- [29] Guyon, I., & Elisseeff, A. (2006). Feature Extraction, Foundations and Applications: An introduction to feature extraction. Studies in Fuzziness and Soft Computing, 207, 1–25. https://doi.org/10.1007/978-3-540-35488-8_1
- [30] Raveendranath, R., Rajamani, V., Babu, A. J., & Datta, S. K. (2014). Android malware attacks and countermeasures: Current and future directions. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014, 137–143. <https://doi.org/10.1109/iccicct.2014.6992944>
- [31] Richter, L. (2015). Common Weaknesses of Android Malware Analysis Frameworks. In IT Security Conference, University of Erlangen-Nuremberg during summer term 2015 (pp. 1–10). Erlangen.
- [32] Baskaran, B., & Ralescu, A. (2016). A Study of Android Malware Detection Techniques and Machine Learning. In Proceedings of the 27th Modern Artificial Intelligence and Cognitive Science Conference 2016, Dayton, OH, USA, April 22-23, 2016. (pp. 15–23).
- [33] Symantec, “Internet Security Threat Report,” 2018.
- [34] Statista, “Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018,” Statista, 2019.
- [35] D. Palmer, “FalseGuide malware dupes 600,000 Android users into joining botnet,” ZDNet, 2017.
- [36] Feixiang He, Bohdan Melnykov, and Elena Root, “RottenSys: Not a Secure Wi-Fi Service At All,” Check Point Research, 2018. <https://research.checkpoint.com/2018/rottensys-not-secure-wi-fi-service/>

[37] Gregg Keizer, “Google reveals Android malware ‘Bouncer,’ scans all apps, Computerworld.,” Computerworld, 2012.

9 APPENDIX

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('Book.csv')
df.head(7)
```

	App	Package	Category	Description	Ra
0	Canada Post Corporation	com.canadapost.android	Business	Canada Post Mobile App gives you access to som...	
1	Word Farm	com.realcasualgames.words	Brain & Puzzle	Speed and strategy combine in this exciting wo...	
2	Fortunes of War FREE	fortunesofwar.free	Cards & Casino	Fortunes of War is a fast-paced, easy to learn...	
3	Better Keyboard: Avatar Purple	com.cc.betterkeyboard.skins.avatarpurple	Libraries & Demo	Skin for Better Keyboard featuring a glossy fe...	
4	Boxing Day	indiaNIC.android.BoxingDay	Lifestyle	Boxing Day by Christopher Jaymes<p>Based on a ...	
5	Ms Claus Live Wallpaper	tmc.christmaslady.livewallpaper	Personalization	Ms Claus Live Wallpaper<p>Find more Free apps ...	
6	Solitaire-Spider-Freecell	ru.uralgames.solitaire.android	Cards & Casino	This solitaire game includes three classic gam...	

7 rows × 184 columns

```
df.shape
```

```
(29999, 184)
```

```
dropper = [ ]
```

```
for col in df.columns[10:]:
```

```
    if (df [col].value_counts( )[0] == 29999 or df [col].value_counts( )[1] < 1000):
```

```
        dropper.append(col)
```

```
df = df.drop(df [dropper], axis = 1)
```

```
df = df.drop('Related apps', axis = 1)
```

```
df.head()
```

	App	Package	Category	Description	Rating
0	Canada Post Corporation	com.canadapost.android	Business	Canada Post Mobile App gives you access to som...	3.1
1	Word Farm	com.realcasualgames.words	Brain & Puzzle	Speed and strategy combine in this exciting wo...	4.3
2	Fortunes of War FREE	fortunesofwar.free	Cards & Casino	Fortunes of War is a fast-paced, easy to learn...	4.1
3	Better Keyboard: Avatar Purple	com.cc.betterkeyboard.skins.avatarpurple	Libraries & Demo	Skin for Better Keyboard featuring a glossy fe...	3.6
4	Boxing Day	indiaNIC.android.BoxingDay	Lifestyle	Boxing Day by Christopher Jaymes<p>Based on a ...	0.0

```
df.isnull().sum()
```

```
App
Package
Category
Description
Rating
```

```

Number of ratings
Price
Dangerous permissions count
Safe permissions count
Hardware controls : control vibrator (S)
Hardware controls : record audio (D)
Hardware controls : take pictures and videos (D)
Network communication : full Internet access (D)
Network communication : receive data from Internet (S)
Network communication : view Wi-Fi state (S)
Network communication : view network state (S)
Phone calls : read phone state and identity (D)
Services that cost you money : directly call phone numbers (D)
Storage : modify/delete USB storage contents modify/delete SD card contents (D)
System tools : automatically start at boot (S)
System tools : modify global system settings (D)
System tools : prevent device from sleeping (D)
System tools : retrieve running applications (D)
System tools : set wallpaper (S)
Your accounts : discover known accounts (S)
Your location : coarse (network-based) location (D)
Your location : fine (GPS) location (D)
Your personal information : read contact data (D)
Your personal information : write contact data (D)
Class
dtype: int64

```



```
df=df.dropna()
```

```

df['App_Upper'] = df['App'].apply(lambda message: sum(1 for c in str(message) if c.isupper)
df['Pack_Upper'] = df['Package'].apply(lambda message: sum(1 for c in str(message) if c.isupper)
df['Description_Upper'] = df['Description'].apply(lambda message: sum(1 for c in str(message) if c.isupper)
df['Pack_Periods'] = df['Package'].apply(lambda message: sum(1 for c in str(message) if c in '.,:;')
df['Desc_len'] = df['Description'].apply(lambda message: len(str(message).split()))

```

```
df['App_Free'] = df['App'].str.contains('free').astype(int)
```

```
df = df.drop(['App', 'Package', 'Description'], axis = 1)
```

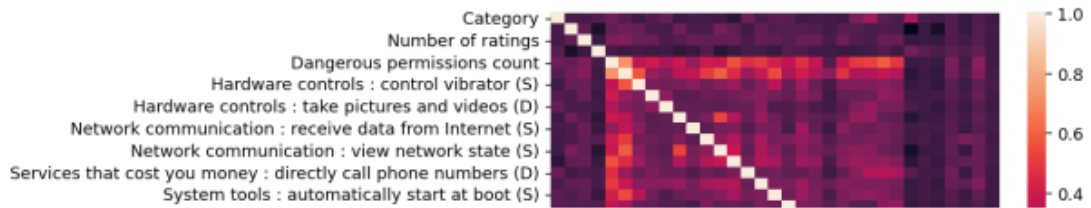
```
sns.countplot(data = df, x = 'Class')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f4c4e5f8990>



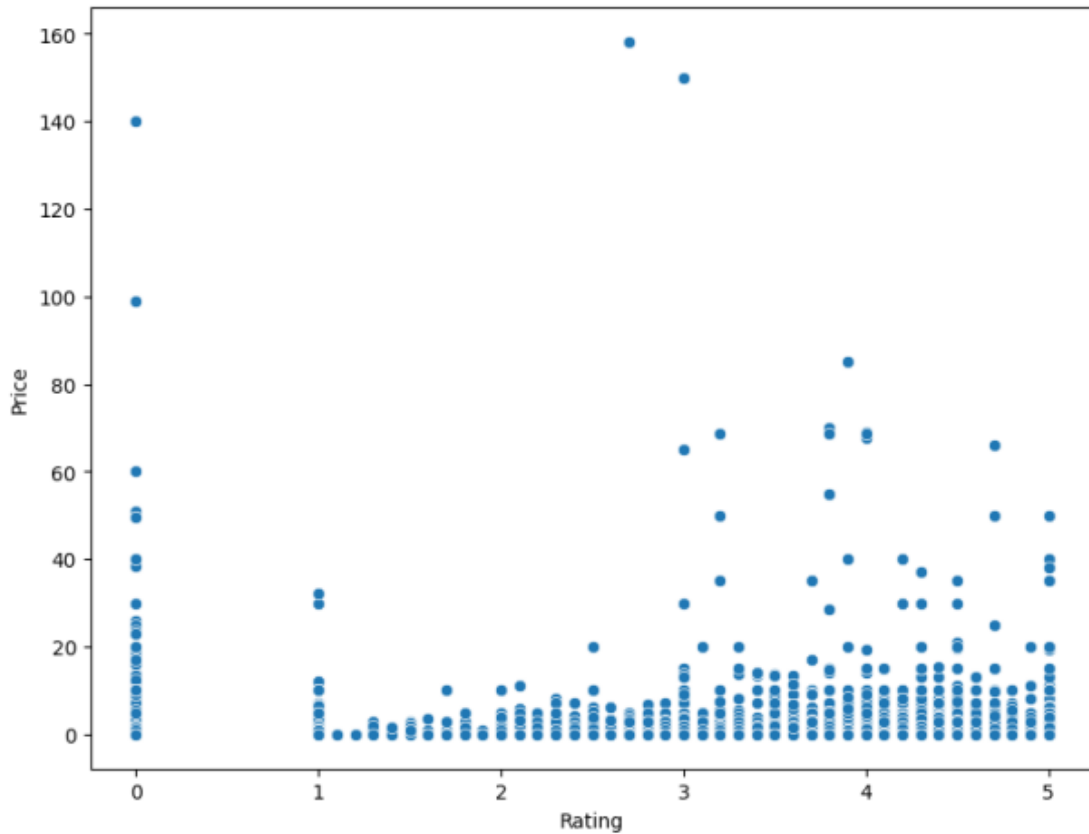
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_cat = df.copy()
df['Category'] = le.fit_transform(df['Category'])
plt.figure(dpi = 130)
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c4f29c250>
```

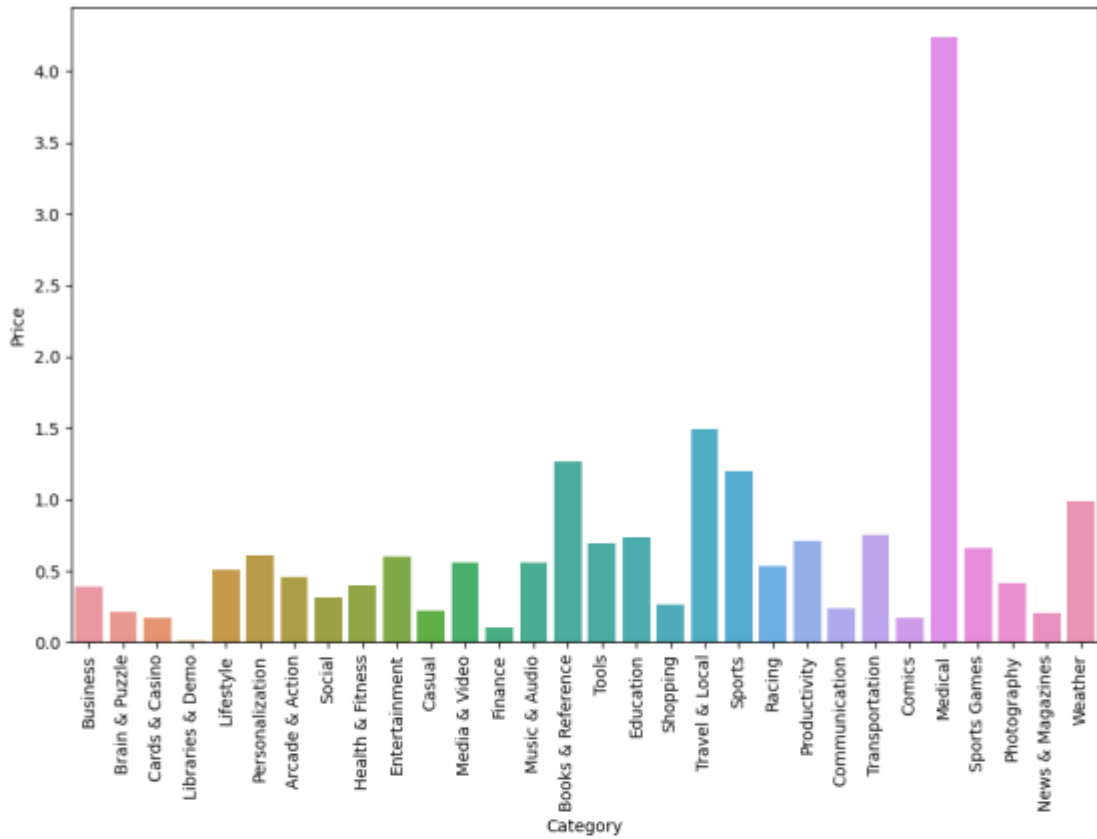


```
plt.figure(figsize = (9, 7), dpi = 100)  
sns.scatterplot(data = df, x = 'Rating', y = 'Price')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c4ba574d0>
```



```
plt.figure(figsize = (11, 7), dpi = 100);  
sns.barplot(data = df_cat, x = 'Category', y = 'Price', estimator=np.mean, ci = None)  
plt.xticks(rotation = 90);
```

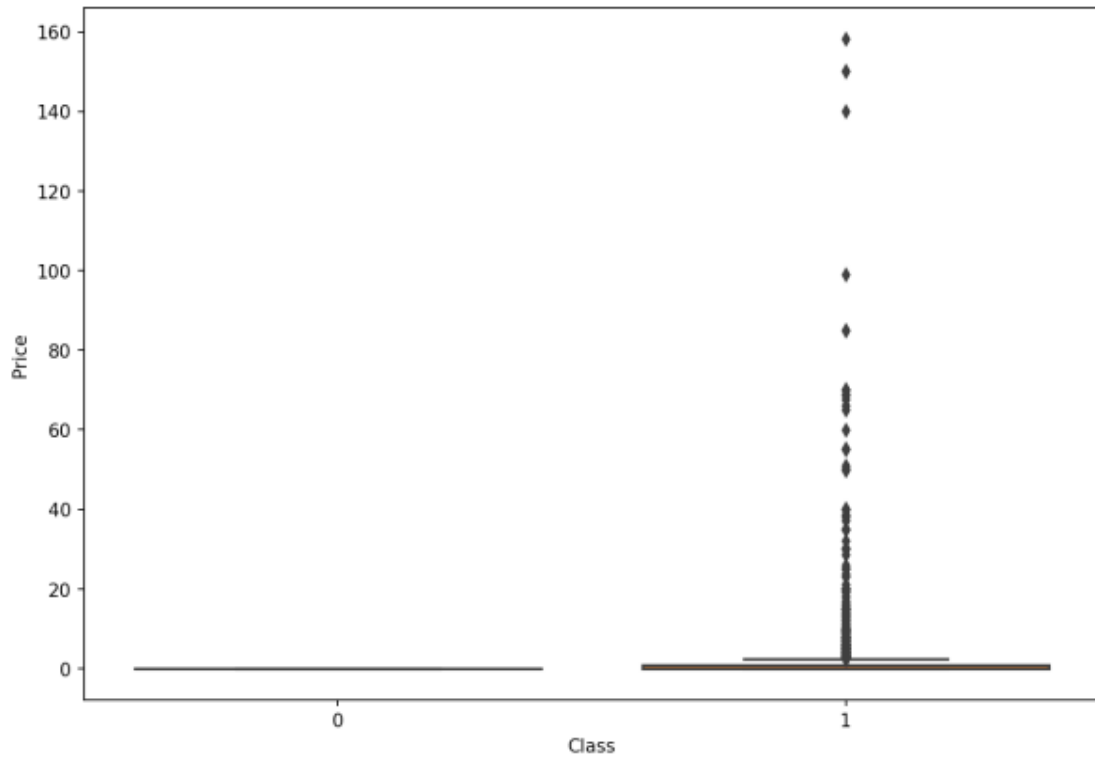


```
df_cat[(df_cat['Category'] == 'Medical') & (df_cat['Price'] > 10)][['Category', 'Price', 'Class']]
```

	Category	Price	Class
3731	Medical	39.95	1
7099	Medical	49.95	1
9596	Medical	59.95	1
13640	Medical	65.97	1
15534	Medical	39.99	1
17009	Medical	39.99	1
22121	Medical	65.00	1
25513	Medical	19.99	1
27009	Medical	23.67	1
29339	Medical	19.99	1

```
plt.figure(figsize=(10, 7), dpi = 150)
sns.boxplot(data = df, x = 'Class', y = 'Price')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c419424d0>
```



```
df[(df['Price'] > 0) & (df['Class'] == 0)]
```

```

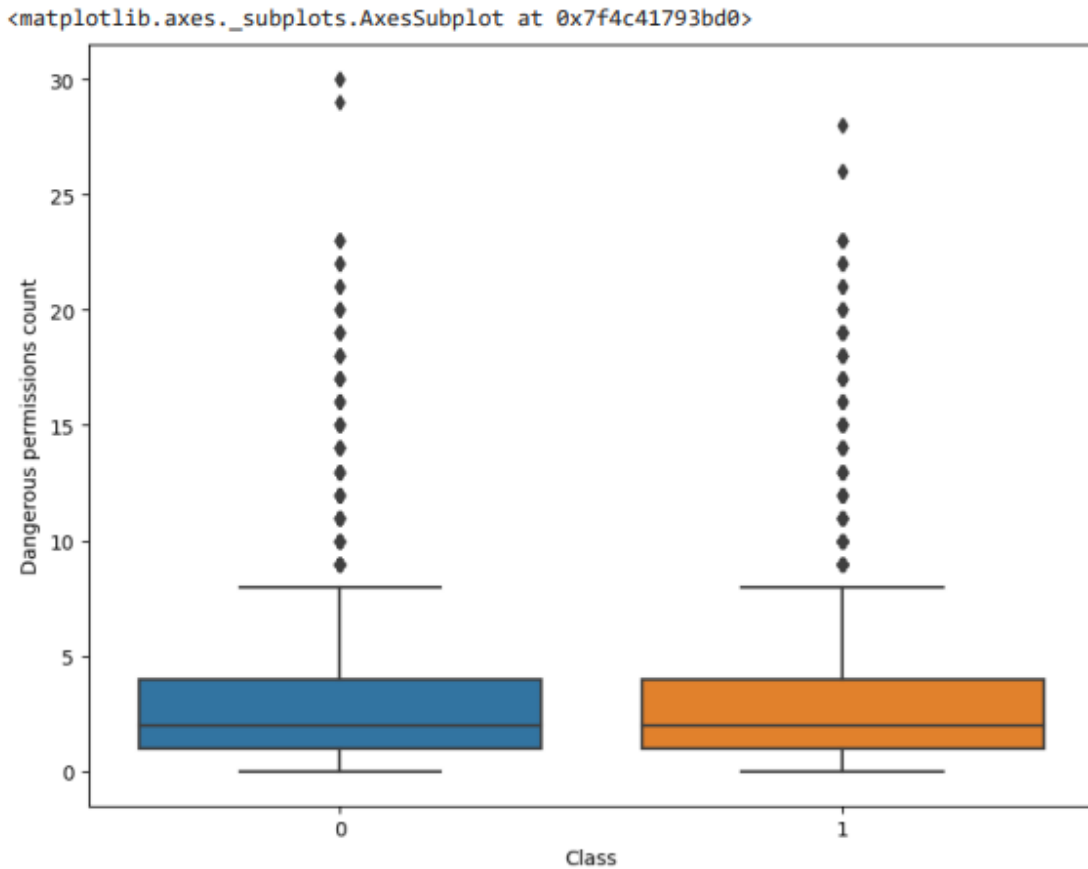
df['Price > 0'] = df['Price'].apply(lambda x: 1 if x > 0 else 0)

```

```

plt.figure(figsize=(9, 7), dpi = 100)
sns.boxplot(data = df, x = 'Class', y = 'Dangerous permissions count')

```



```

df[df['Dangerous permissions count'] > 10]['Class'].value_counts()

```

```

1    550
0    285
Name: Class, dtype: int64

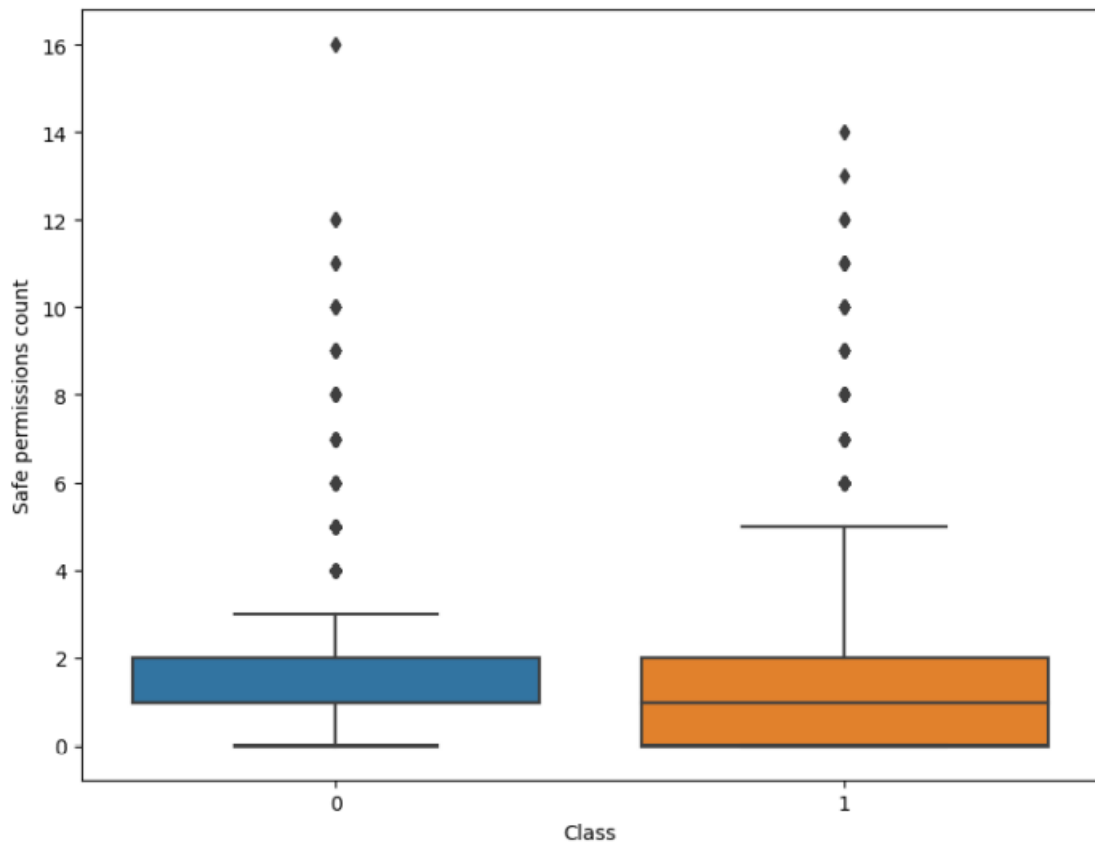
```

```

plt.figure(figsize=(9, 7), dpi = 100)
sns.boxplot(data = df, x = 'Class', y = 'Safe permissions count')

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c41793a10>
```



```
df[df['Safe permissions count'] > 5]['Class'].value_counts()
```

```
1    448
0    240
Name: Class, dtype: int64
```

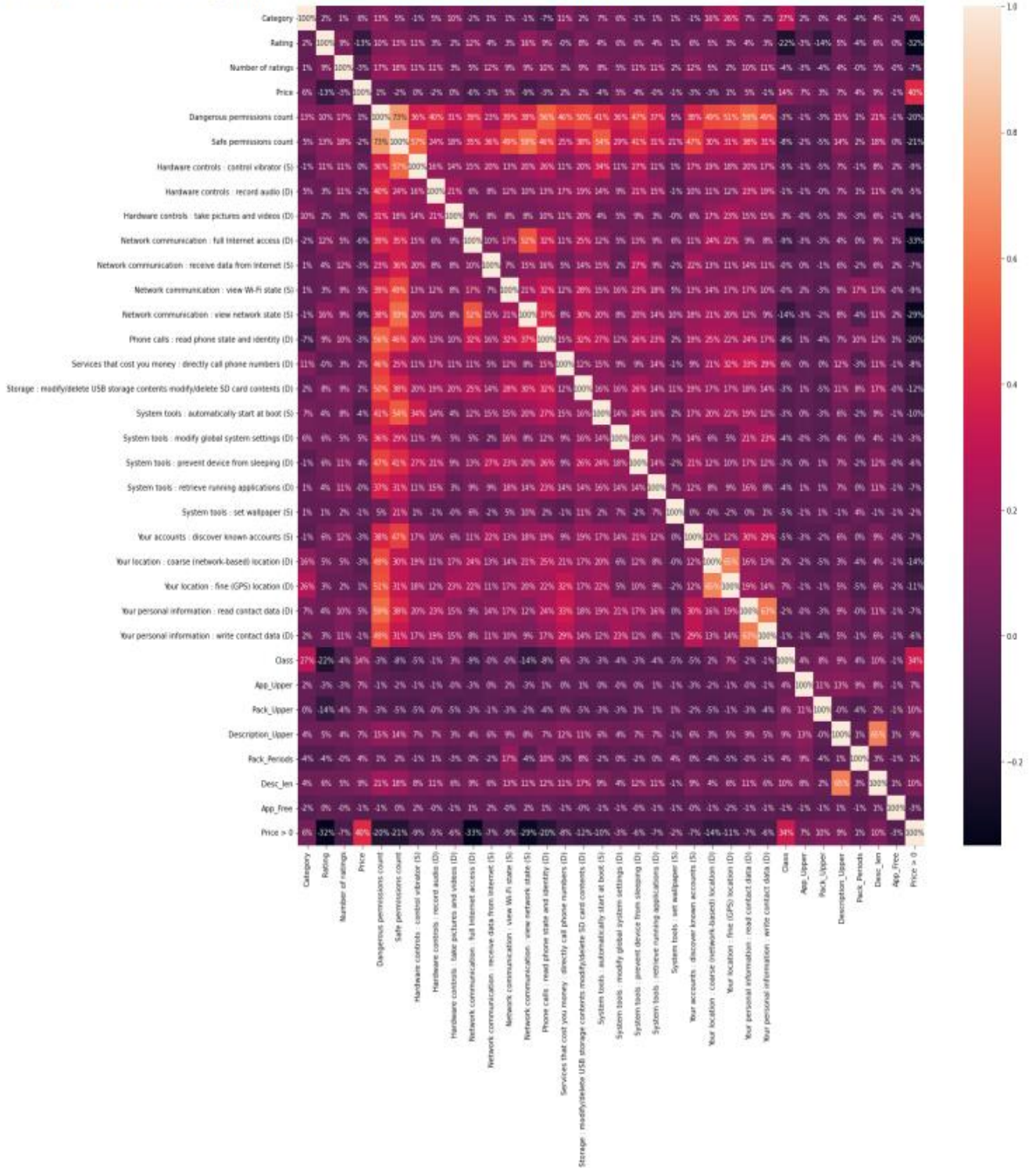
```
df.corr()
```

	Category	Rating	Number of ratings	Price	Dangerous permissions count	Safe permissions count
Category	1.000000	0.015339	0.006504	0.056117	0.126663	0.054688
Rating	0.015339	1.000000	0.086127	-0.132429	0.096680	0.130381
Number of ratings	0.006504	0.086127	1.000000	-0.026005	0.166111	0.184608
Price	0.056117	-0.132429	-0.026005	1.000000	0.008823	-0.024297
Dangerous permissions count	0.126663	0.096680	0.166111	0.008823	1.000000	0.730684
Safe permissions count	0.054688	0.130381	0.184608	-0.024297	0.730684	1.000000
Hardware controls : control vibrator (S)	-0.010867	0.110666	0.111889	0.002827	0.359345	0.569877
Hardware controls : record audio (D)	0.046653	0.027858	0.108273	-0.016967	0.401390	0.241236
Hardware controls : take pictures and videos (D)	0.104415	0.019849	0.028452	0.001740	0.308893	0.180464
Network communication : full Internet access (D)	-0.017902	0.122071	0.051344	-0.063096	0.388429	0.352415
Network communication : receive data from Internet (S)	0.011791	0.035127	0.115118	-0.030787	0.231947	0.359013
Network communication : view Wi-Fi state (S)	0.007193	0.031506	0.088972	0.046025	0.392403	0.492839
Network communication : view network state (S)	-0.008883	0.155461	0.094865	-0.088126	0.382881	0.588918
Phone calls : read phone state and identity (D)	-0.074639	0.085429	0.104141	-0.031650	0.556294	0.456666
Services that cost you money : directly call phone numbers (D)	0.113617	-0.001215	0.030771	0.018925	0.460439	0.246860

numbers (U)						
Storage :						
modify/delete USB storage contents	0.017291	0.083524	0.093607	0.016458	0.500267	0.384947
modify/delete SD card contents (D)						
System tools :						
automatically start at boot (S)	0.070804	0.040813	0.079141	-0.041504	0.412164	0.537745
System tools :						
modify global system settings (D)	0.060167	0.055140	0.054640	0.049564	0.357548	0.293208
System tools :						
prevent device from sleeping (D)	-0.011756	0.056299	0.112802	0.035752	0.474745	0.408169
System tools :						
retrieve running applications (D)	0.012128	0.044659	0.114700	-0.001463	0.365887	0.308628
System tools : set wallpaper (S)	0.006883	0.007336	0.019515	-0.007029	0.047674	0.211459
Your accounts :						
discover known accounts (S)	-0.011995	0.061129	0.118919	-0.026071	0.380246	0.474255
Your location :						
coarse (network-based) location (D)	0.158924	0.050465	0.049593	-0.030370	0.493283	0.298582
Your location : fine (GPS) location (D)	0.261800	0.027495	0.022224	0.005408	0.514761	0.313276
Your personal information : read contact data (D)	0.072426	0.037624	0.103173	0.048971	0.587879	0.377675
Your personal information : write contact data (D)	0.022697	0.028720	0.106485	-0.008581	0.491276	0.309144
Class	0.273943	-0.215724	-0.035859	0.135833	-0.026271	-0.082721
App_Upper	0.017812	-0.026912	-0.034058	0.070185	-0.009142	-0.017271

```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, fmt='.0%')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f4c4164ff90>



X=df.drop("Class",axis=1)

```

Y=df['Class']

X.shape

(29791, 33)

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state =

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.metrics import confusion_matrix, plot_confusion_matrix

def models(X_train,Y_train):

    #Using Logistic Regression
    from sklearn.linear_model import LogisticRegression
    log = LogisticRegression(random_state = 0)
    log.fit(X_train, Y_train)
    plot_confusion_matrix(log, X_test, Y_test)

    #Using KNeighborsClassifier
    from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
    knn.fit(X_train, Y_train)
    plot_confusion_matrix(knn, X_test, Y_test)

    #Using SVC linear
    from sklearn.svm import SVC
    svc_lin = SVC(kernel = 'linear', random_state = 0)
    svc_lin.fit(X_train, Y_train)
    plot_confusion_matrix(svc_lin, X_test, Y_test)

    #Using SVC rbf
    from sklearn.svm import SVC
    svc_rbf = SVC(kernel = 'rbf', random_state = 0)
    svc_rbf.fit(X_train, Y_train)
    plot_confusion_matrix(svc_rbf, X_test, Y_test)

    #Using GaussianNB
    from sklearn.naive_bayes import GaussianNB
    gauss = GaussianNB()
    gauss.fit(X_train, Y_train)

```

```

plot_confusion_matrix(gauss, X_test, Y_test)

#Using Neural Networks
# model.fit(X_train,Y_train,epochs=10, batch_size=32)

#Using DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
tree.fit(X_train, Y_train)
plot_confusion_matrix(tree, X_test, Y_test)

#Using RandomForestClassifier method of ensemble class to use Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =
forest.fit(X_train, Y_train)
plot_confusion_matrix(forest, X_test, Y_test)

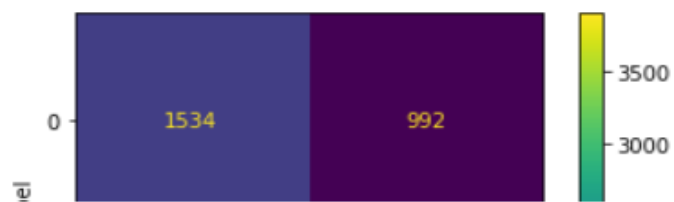
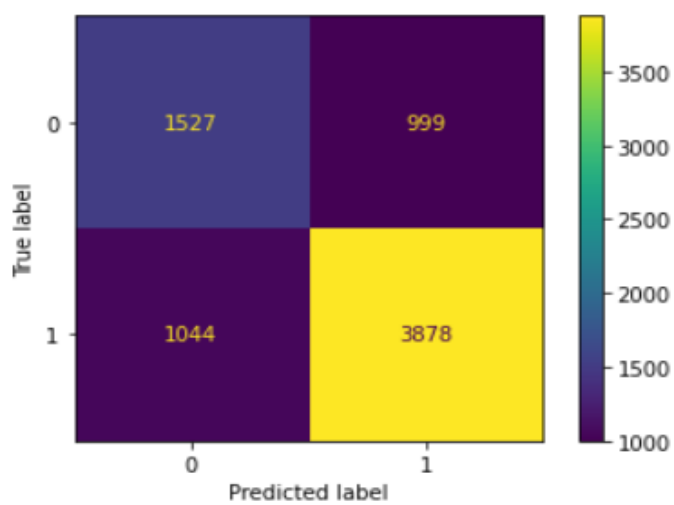
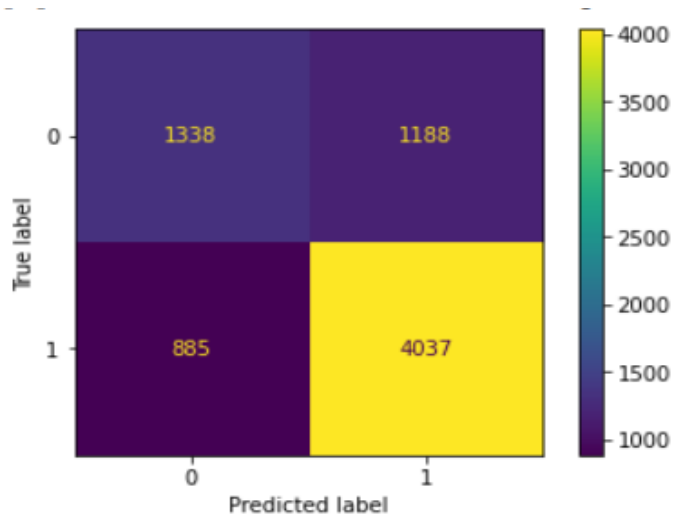
#print model accuracy on the training data.
print('[0]Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:', svc_lin.score(
print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:', svc_rbf.score(X_t
print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_train))
print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
print('[6]Random Forest Classifier Training Accuracy:', forest.score(X_train, Y_train))

return log, knn, svc_lin, svc_rbf, gauss, tree, forest

model = models(X_train,Y_train)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
warnings.warn(msg, category=FutureWarning)
[0]Logistic Regression Training Accuracy: 0.7119903325426308
[1]K Nearest Neighbor Training Accuracy: 0.8170791746855839
[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.71682406122723
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.7739784272479077
[4]Gaussian Naive Bayes Training Accuracy: 0.5222217249250325
[5]Decision Tree Classifier Training Accuracy: 0.991227677572394
[6]Random Forest Classifier Training Accuracy: 0.9775768697131092

```



```

from sklearn.metrics import confusion_matrix
for i in range(len(model)):
    cm = confusion_matrix(Y_test, model[i].predict(X_test))

    TN = cm[0][0]
    TP = cm[1][1]
    FN = cm[1][0]
    FP = cm[0][1]

    print(cm)
    print('Model[{}] Testing Accuracy = "{}!"'.format(i, (TP + TN) / (TP + TN + FN + FP)))
    plot_confusion_matrix(model[i], X_test, Y_test)
    print()# Print a new line

```

