Detection of Malicious Android Applications

Project report submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Computer Science & Engineering

by

Aditya Rathi: 17UCS010
Parth Jain: 17UCS106

Under Guidance of
Dr. Poonam Gera

Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur

April 2021

The LNM Institute of Information Technology
Jaipur, India

CERTIFICATE

This is to certify that the project entitled “Detection of Malicious Android Applications”, submitted by Aditya Rathi (17UCS010) and Parth Jain (17UCS106) in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by them at the Department of Computer Science and Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2020-21 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my/our opinion, this thesis is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

Date	Adviser: Dr. Poonam Gera

Dedicated to My Family and Friends

Acknowledgment

We would like to thank Dr. Poonam Gera for her guidance and support in completing our project.

Abstract

The threat of Android Applications has increased with the increase in the popularity of smartphones. Android applications do require many permissions by which they provide various services. There can be infectious applications which can ask for unessential permissions to gain elevated privileges over user's data. If infected with malware, users may face disclosure of personal data of which we know the importance.

In this project, we aim to detect such malicious Android applications with the use of Machine Learning algorithms. There are various trends that these apps follow which we can exploit in order to find those apps and obtain high accuracy in detection. We used Permission Usage trend of the application along with the type of intents a component might want to receive. For example, an Malware app asks for those permissions which may not even be required for such a category of app. Since both of these information are specified in the Manifest.xml file of an Android Application Package (APK), we focused to analyze the Manifest file of a large number of Android apps in this project to detect malware Android applications with good accuracy.

Contents

Chapter 1

INTRODUCTION

1.1 The Area of Work

The project is based on predicting the chances of an Android Application being safe for the user or not. If the app is not safe, it is classified as a Malware app. A malware app is a mobile software which is intentionally designed in such a way that may cause harm to the operating system, steal private information, cause damage to a server or client etc. We aim to build a web software to predict such applications.

1.2 Problem Addressed & Brief Overview

Popular approaches such as signature based methods to detect malware apps are effective to find known malware and inadequate for unknown malware. Therefore, we worked on an approach which is based on Permission and Intent usage patterns of an Android app.

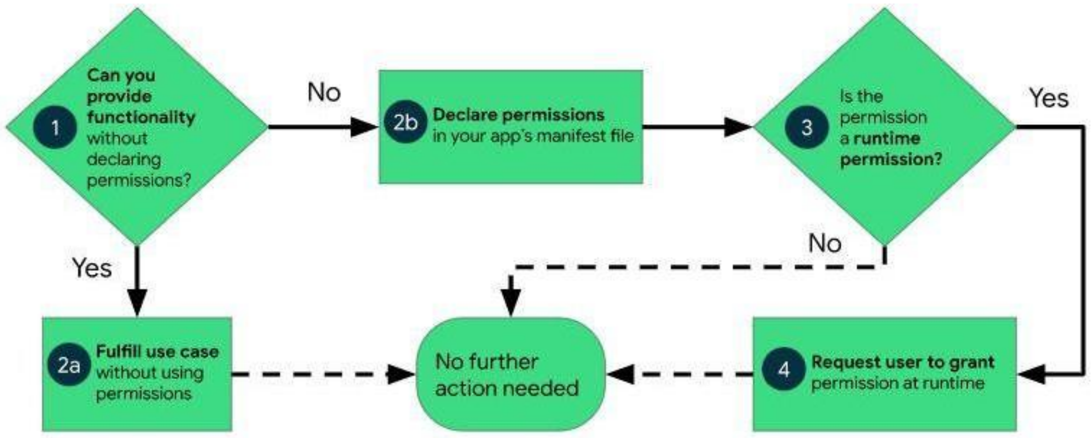
In this project, we aim to build an application which can predict the chances of whether a given Android App is Malware or Benign with the help of Machine Learning algorithms. The algorithm of this application is based on detecting the patterns of Permissions and Intent usage of an Android app. To achieve this, we analysed the Manifest.xml file of a number of Android apps and created a binary dataset and trained a Machine Learning model based on this dataset. The model was then deployed on a Web application.

1.3 Understanding the Approach

We have used Permissions and Intents in order to detect Malware Android apps but let us first understand why these two features are important for us:

1.3.1 What is a Permission in Android?

Android apps offer various functionalities that require access to restricted data or actions. Restricted data include state of system and user's information while Restricted actions include actions like recording audio, connecting to device etc. There are two types of permissions - install time permission (granted at the time of installation) and runtime permissions (requested at the run time of the application). The permissions are declared in the application's manifest file's <uses-permission> tag.

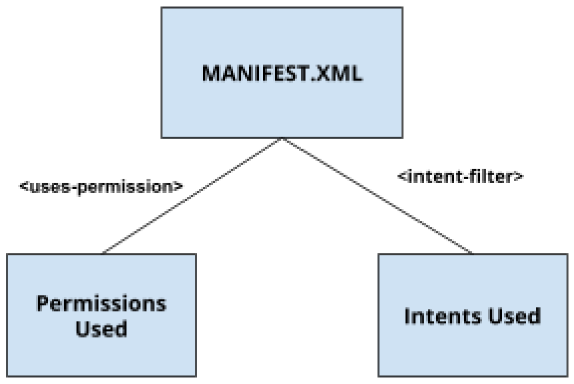

1
Can you provide functionality without declaring permissions?
No
2b
Declare permissions in your app's manifest file
3
Is the permission a runtime permission?
Yes
No
Yes
2a
Fulfill use case without using permissions
No further action needed
4
Request user to grant permission at runtime

Source: Android Documentation

If the app can provide functionality without using any permissions then it should fulfil the use case without requesting the user for the permission. This is where Malware apps come into play. Most of the Malware applications do not want to fulfil a specific case but yet they ask the user for some permissions. The intention behind asking permissions from the user can be to achieve elevated rights to inaccessible information, access user's private data, harming the device with ransomware and many more.

1.3.1 What are Intents in Android?

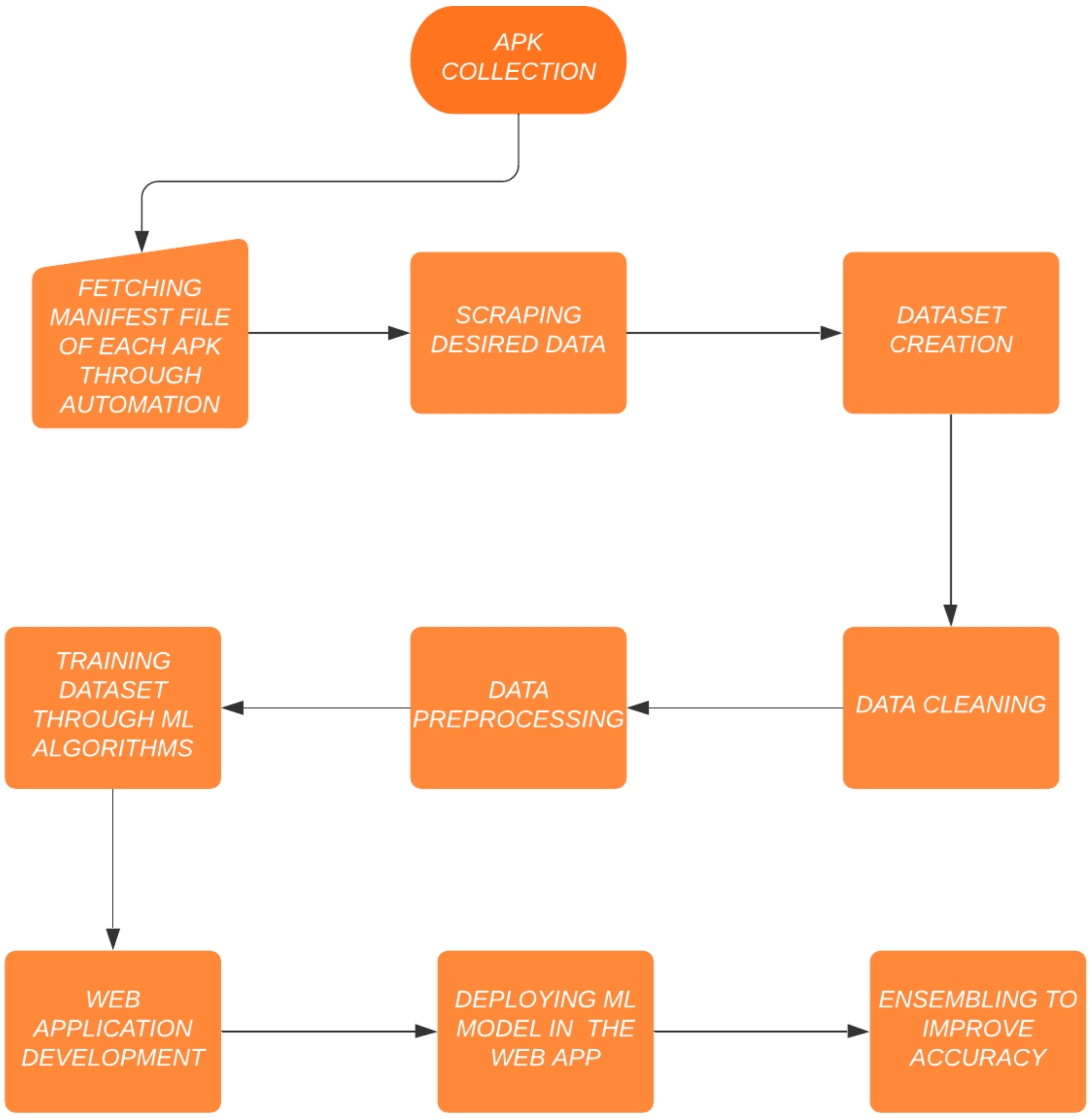
Android Intent is a message object which is passed between components. In general, they are used for navigating between components such as activities, broadcast receivers, content providers or services. There are two types of intents, explicit and implicit. Explicit intent specifies the component and the class that is to be invoked whereas implicit intents do not specify the component. Research works deduced inferred Intents can be used in detection of malicious apps. Therefore, we will also take types of intents used frequently by malicious Android apps into consideration. The intent information is also present in the manifest file inside the <intent-filter> tag.



1.4 Overview & Workflow

We managed to get the access of Malware and Benign APKs dataset from trustable resources. Around 500 benign APKs and 438 Malware APKs were obtained. Automation of uploading APKs and Scraping Manifest data: We had around 938 APKs, therefore we used Selenium Web Driver to automate the extraction of relevant information into the CSV dataset after cleaning of irrelevant information. After the dataset was ready we implemented Machine Learning algorithms and exported 5 Pickle files into which model was saved. Using these Pickle files, the models were deployed in a Flask Web application. The back end of the web application was then developed such that it can upload any APK and extract its permission and intent information using a similar process that we used in dataset creation. Using this input and the trained model we were able to find out the results.

Workflow of the project


APK COLLECTION
FETCHING MANIFEST FILE OF EACH APK THROUGH AUTOMATION
SCRAPING DESIRED DATA
DATASET CREATION
DATA CLEANING
DATA PREPROCESSING
TRAINING DATASET THROUGH ML ALGORITHMS
WEB APPLICATION DEVELOPMENT
DEPLOYING ML MODEL IN THE WEB APP
ENSEMBLING TO IMPROVE ACCURACY

Chapter 2

TECHNOLOGY STACK INVOLVED

Before heading towards the detailed implementation of the project, we will give a brief overview of all the technologies, libraries, frameworks and tools which were used to make an end to end application.

- Selenium
  Selenium is a powerful open-source portable framework which is generally used for automating tests which are carried on Web browsers. We used the Selenium Web Driver so that we could automate the process of uploading 900+ APKs. We also used Selenium for web scraping of the required data.

- SISIK Web Tool
  Sixo Online APK Analyzer (SISIK) is a third party web application which was used to output the Manifest.xml file of any APK on uploading it.

- Pandas
  Pandas is a Python data analysis library which provides flexible data structures designed in such a way that it becomes easy and efficient when working on a relational dataset. The Pandas library was used to convert the raw data into a relational binary dataset in CSV format.

- Scikit-Learn
  Scikit-Learn is a library in Python which provides various classification, clustering and regression algorithms. We used this library to implement five classification algorithms - Random Forest, Gaussian Naive Bayes, Bernoulli Naive Bayes, Logistic Regression and Decision Tree.

- Seaborn
  Seaborn is a library in Python used for Data Visualization. It is used for plotting statistical graphs and uses Matplotlib. It also integrates with Pandas data. It helps us to understand the data and the results in more detail.

- Joblib
  Joblib is a package in Python which was used to save and load the Machine Learning model into a Pickle file. The Python Pickle module is basically used for serializing a Python object. Using Joblib and Pickle, the Machine Learning model was deployed into the Flask application.

- Flask
  Flask is a popular Python based Web framework which is used to build and develop web applications. It is a microframework which does not contain Object Relational Manager (ORM). The Flask framework was used to build the back end logic of our web application. It included the functioning of uploading any APK, fetching it in the database, initializing Selenium to extract the desired data from the APK and using the deployed Machine Learning model, fetching the results.

- HTML and CSS
  HTML is a popular markup language which is used to build the front end structure of any web application. CSS (Cascading Style Sheets) was used to enhance the HTML and add features such as loaders, buttons to improve the user interface.

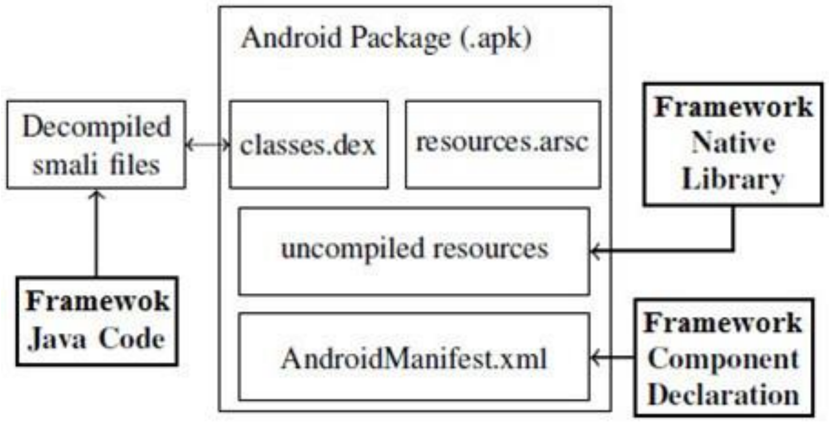
Chapter 3

CREATING DATASET

3.1 Finding the collection of APK Files

To create a dataset, first we needed a collection of benign and malware APKs. We obtained the malware APK files from VirusTotal.com and benign APKs were obtained from Mr. Sen Chen who is an Assistant Professor at NTU Singapore. We gathered a total number of 500 benign applications and 437 malware applications.

3.2 Finding a tool which outputs the manifest from APK file

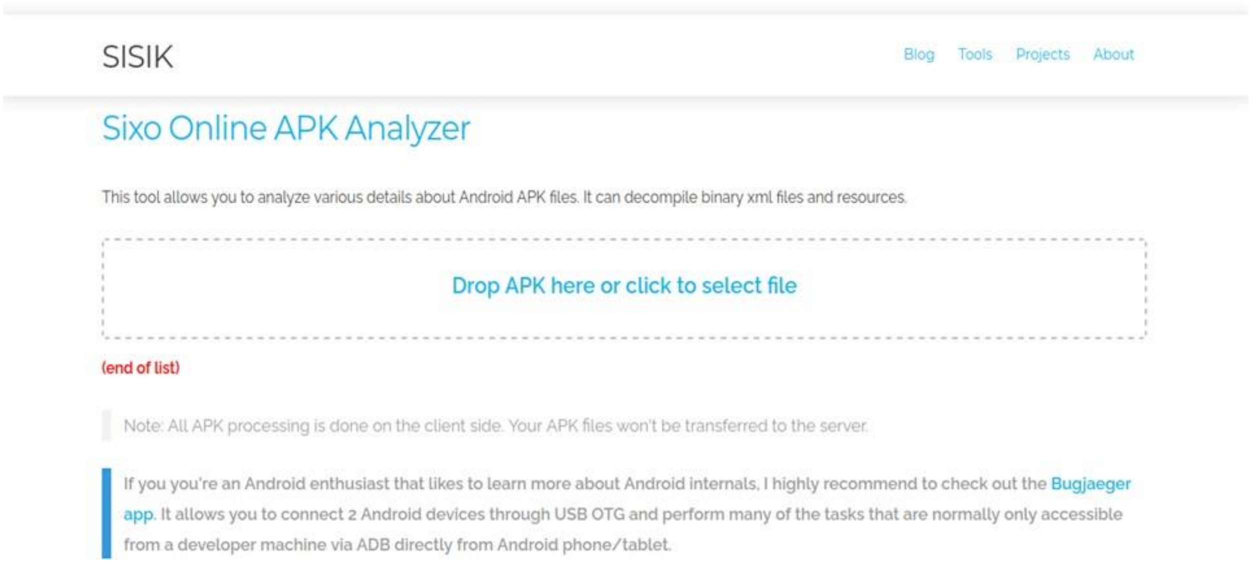


Every android app has a manifest file which contains all the essential information and features of the App. In our project, we are only concerned with the ***Intent Filters*** and the ***Permission Usage*** features.

Permission usage: every permission that the application needs.

Intent Filters: specifies the type of intents (messaging object) that the component of the application would like to receive.

To extract this information from manifest files of each APK, we used the Sixo Online APK Analyzer tool (*https://www.sisik.eu/apk-tool*).


SISIK

Sixo Online APK Analyzer

This tool allows you to analyze various details about Android APK files. It can decompile binary xml files and resources.

(end of list)

Note: All APK processing is done on the client side. Your APK files won't be transferred to the server.

If you you're an Android enthusiast that likes to learn more about Android internals, I highly recommend to check out the Bugjaeger app. It allows you to connect 2 Android devices through USB OTG and perform many of the tasks that are normally only accessible from a developer machine via ADB directly from Android phone/tablet.

Web tool which outputs Manifest.xml file on uploading an APK

3.3 Automation and Web Scraping

Manually uploading the APK file into the APK Analyzer tool and storing the relevant information for thousands of such files would be a tedious job and will require a lot of time, so we used Selenium Webdriver for automating and scraping the information strings from the manifest.

Now, Permission usage and Intent filters items are in the form of text strings. Every APK might have the same or different items. To keep record of this, we thought of creating a binary dataset (value 0 for item does not exist and 1 for item exists for a particular APK).

Every APK will have its own list of items, so the columns represent different items (permission usage item or intent filter item) and row represents different APKs.

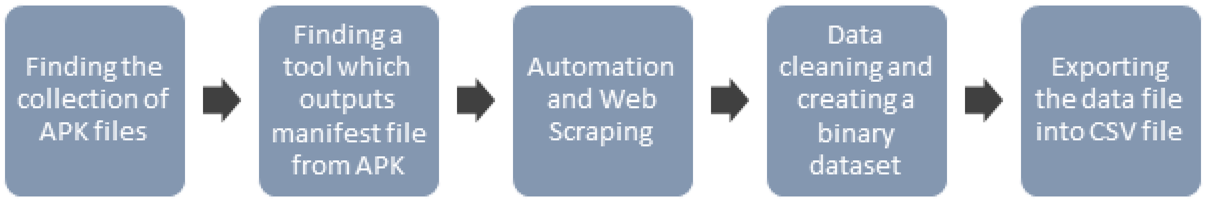
Between every two APK uploads, some time frame was given in which we can scrape the required data. Using Selenium, the string items were scraped and stored in a 2D list. Every sub list of this 2D list represents the string items of a single APK.

3.4 Data cleaning and creating a binary dataset

This 2D list had various redundant strings and unwanted data so there was a need to clean the data up. We used a regex function and stored the values in a set that no values get repeated. The set was ready to be used as the columns of Pandas dataframe.

3.5 Exporting the data file into CSV file

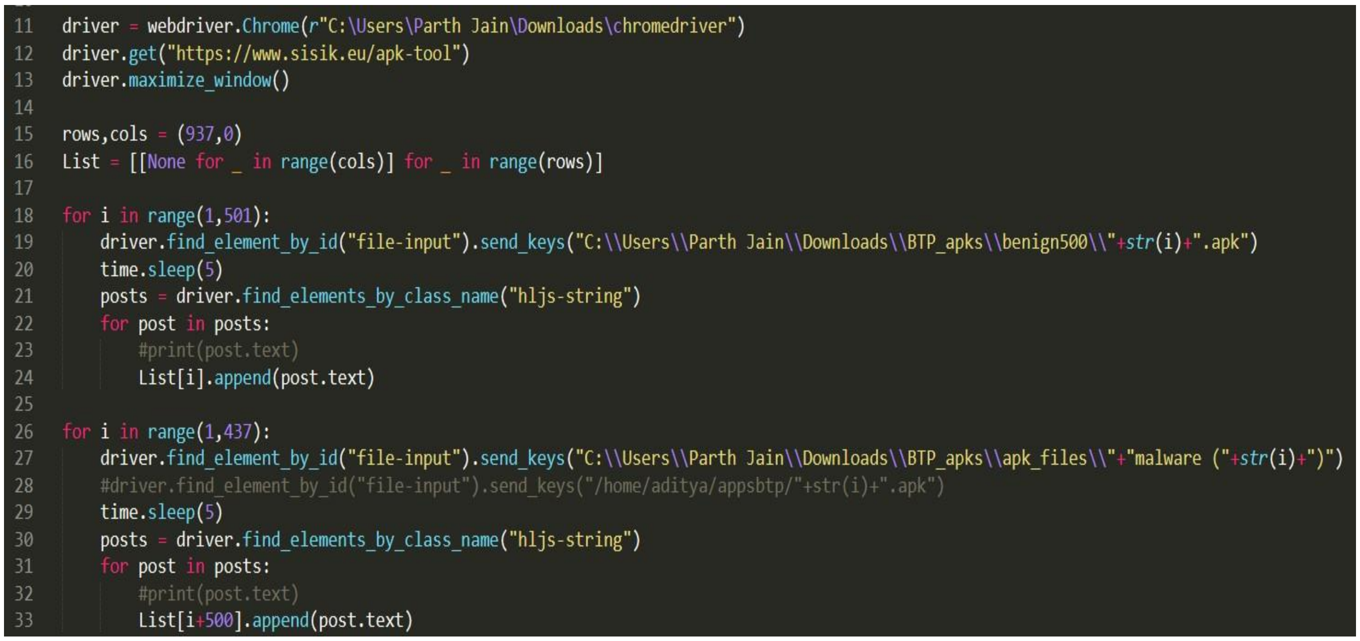
We iterated the 2D list and the set to compare the values. In case of a match, value 1 was assigned to that particular cell, otherwise 0. The data frame was then exported into CSV format.


Finding the collection of APK files
Finding a tool which outputs manifest file from APK
Automation and Web Scraping
Data cleaning and creating a binary dataset
Exporting the data file into CSV file

Process of Creating Dataset

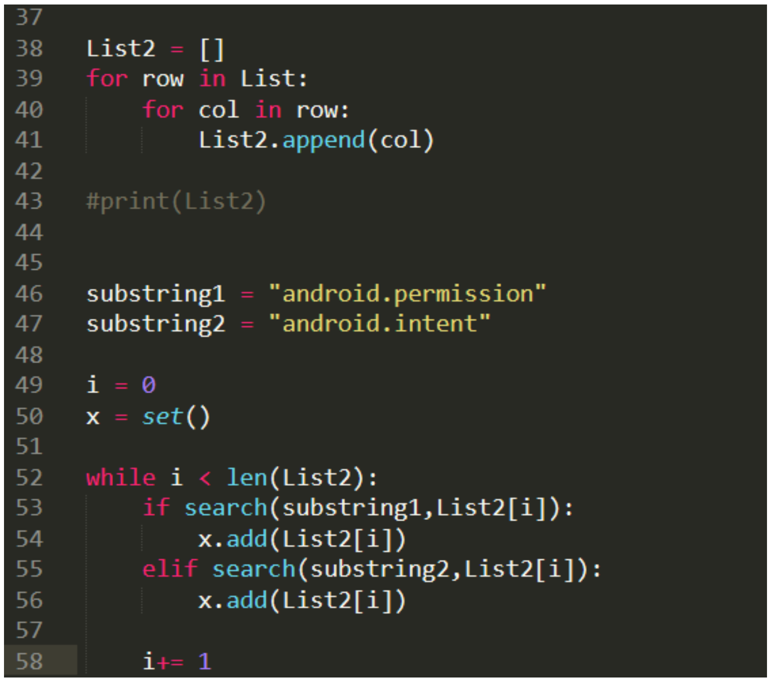
CODE SNIPPETS

1) **Automating the uploads of malware and benign APKs to the SISIK Web Tool and scraping required strings into a 2D list using Selenium Webdriver**



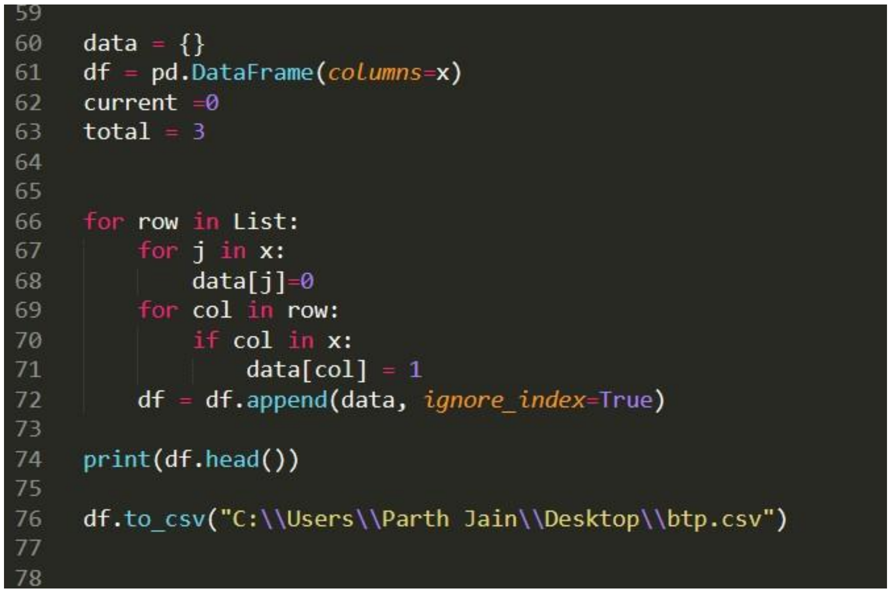
```python
driver = webdriver.Chrome(r"C:\Users\Parth Jain\Downloads\chromedriver")
driver.get("https://www.sisik.eu/apk-tool")
driver.maximize_window()

rows,cols = (937,0)
List = [[None for _ in range(cols)] for _ in range(rows)]

for i in range(1,501):
    driver.find_element_by_id("file-input").send_keys("C:\\Users\\Parth Jain\\Downloads\\BTP_apks\\benign500\\"+str(i)+".apk")
    time.sleep(5)
    posts = driver.find_elements_by_class_name("hljs-string")
    for post in posts:
        #print(post.text)
        List[i].append(post.text)

for i in range(1,437):
    driver.find_element_by_id("file-input").send_keys("C:\\Users\\Parth Jain\\Downloads\\BTP_apks\\apk_files\\"+"malware ("+str(i)+")")
    #driver.find_element_by_id("file-input").send_keys("/home/aditya/appsbtp/"+str(i)+".apk")
    time.sleep(5)
    posts = driver.find_elements_by_class_name("hljs-string")
    for post in posts:
        #print(post.text)
        List[i+500].append(post.text)
```

2) Cleaning the data



```
List2 = []
for row in List:
    for col in row:
        List2.append(col)

#print(List2)


substring1 = "android.permission"
substring2 = "android.intent"

i = 0
x = set()

while i < len(List2):
    if search(substring1,List2[i]):
        x.add(List2[i])
    elif search(substring2,List2[i]):
        x.add(List2[i])

    i+= 1
```

3) Creating the Dataframe using Pandas and exporting the data into CSV file



```
data = {}
df = pd.DataFrame(columns=x)
current =0
total = 3


for row in List:
    for j in x:
        data[j]=0
    for col in row:
        if col in x:
            data[col] = 1
    df = df.append(data, ignore_index=True)

print(df.head())

df.to_csv("C:\\Users\\Parth Jain\\Desktop\\btp.csv")
```

Chapter 4

TRAINING MACHINE LEARNING MODEL

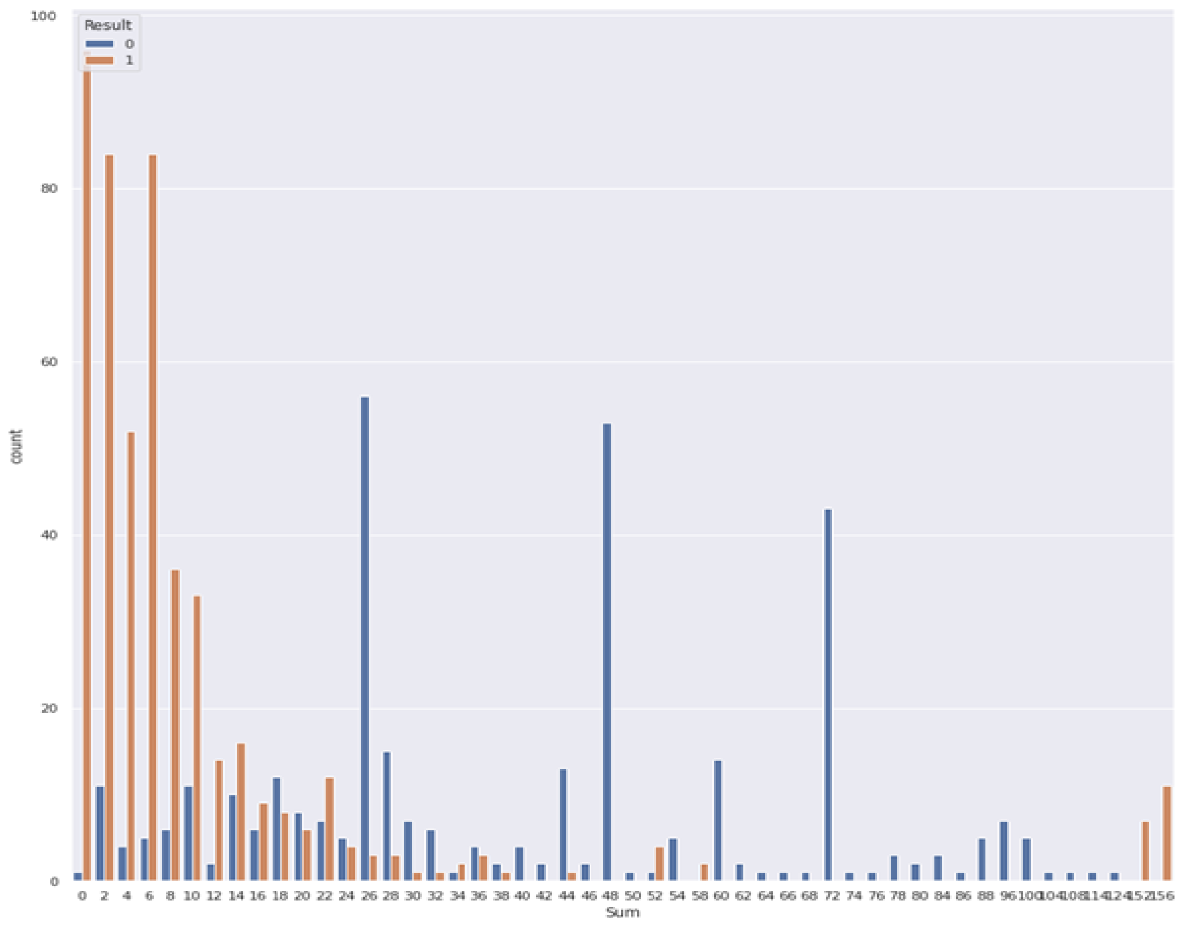
4.1 Study on number of permissions of each application

Before training the machine learning model, we analysed the number of permissions that the apps demanded and we noticed some differences. Malware applications generally require more permissions than benign applications. Most of the benign applications required less than 5 permissions.

In the graph below, ***'0'*** represents malware apps and ***'1'*** represents benign apps.

X -Axis represents the number of permissions.

Y- Axis represents the number of applications.



4.2 Machine learning Model

After this, we employed supervised machine learning methods to classify Android applications into benign and malware apps. Supervised Machine Learning algorithms predict the outcome of unseen data by learning from training data.

We then used ***k-fold cross validation*** to evaluate the performance of our machine learning algorithms. We performed k-fold cross validation with ***k =10***. So, our dataset was split 10 times into 10 different sets, 90% of the total data was used for training and other 10% was used for testing thedata.

We used the ***Google Colab*** platform to train our ML model. Google Colab is an online cloud-based environment that allows anybody to write and execute python code.

4.2.1 Classifiers/Machine learning algorithms used:

1) Naive Bayes

Naïve Bayes classifier is based on the ***Bayes Theorem***. This algorithm assumes that all the variables in the dataset are not correlated to each other. Presence of a particular feature does not affect the other

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

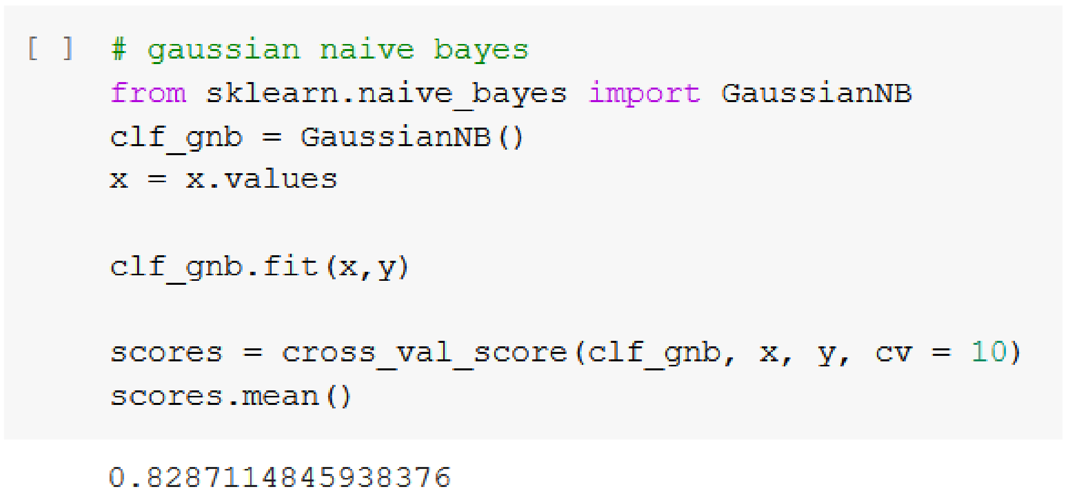
Bayes Theorem

a) Gaussian Naive Bayes

Gaussian Naïve Bayes is a type of Naïve bayes where features are not discrete but are continuous. It is then assumed that these values are sampled from a gaussian distribution also known as normal distribution. In Gaussian Naïve Bayes, the above formula of bayes theorem for probability changes to,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Gaussian Naive Bayes Code



```
[ ]  # gaussian naive bayes
     from sklearn.naive_bayes import GaussianNB
     clf_gnb = GaussianNB()
     x = x.values

     clf_gnb.fit(x,y)

     scores = cross_val_score(clf_gnb, x, y, cv = 10)
     scores.mean()

     0.8287114845938376
```

Accuracy obtained = 0.8287 = 82.87 %

Terminologies to understand Confusion Matrix:

True Negative: Correctly predicting that the app is malware.

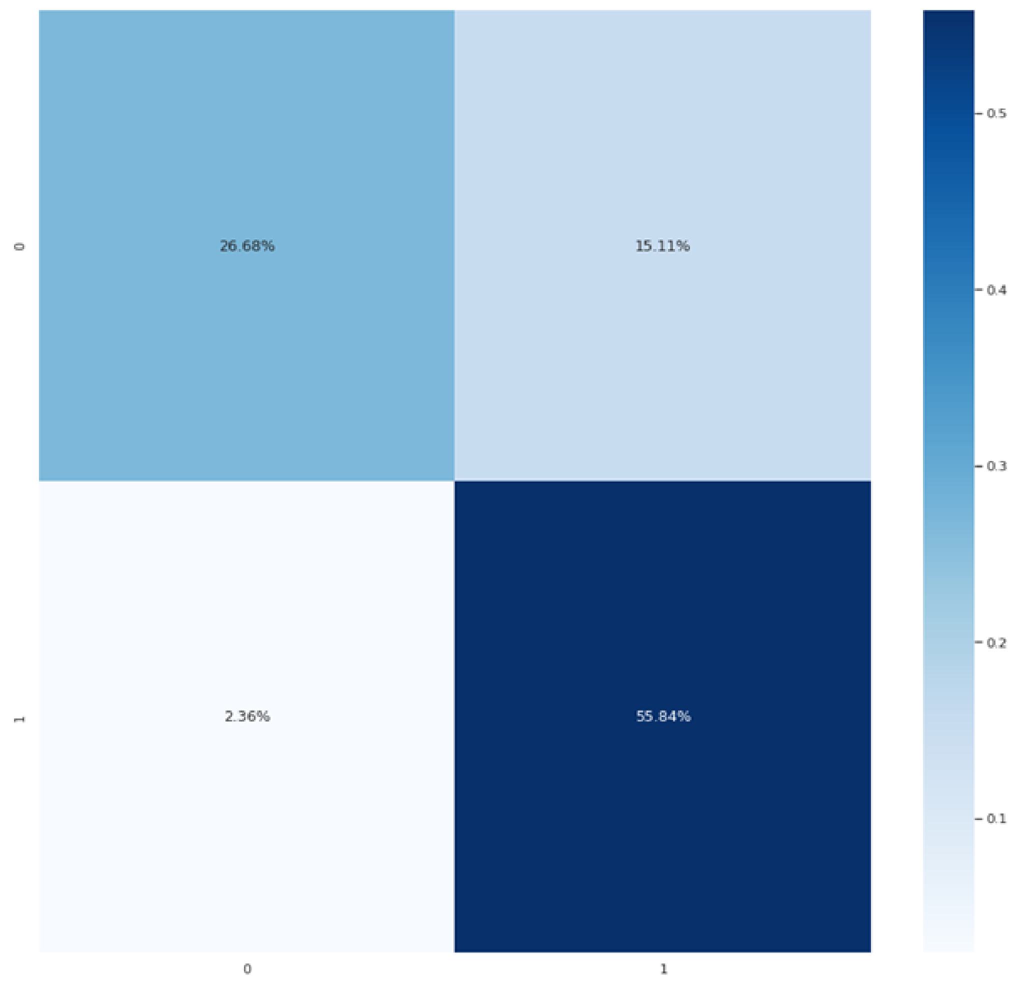
False Negative: Falsely predicting app is malware.

True Positive: Correctly predicting app is benign

False Positive: Falsely predicting app is benign

Confusion Matrix for Naive Bayes:

Square with coordinates (0,0) represents true negative, (0,1) represents false positive, (1,0) represents false negative and finally (1,1) represents true positive.



True Negative (TN) – 26.84 %

False Negative (FN) – 15.11 %

False Positive (FP) – 2.36 %

True Positive (TP) – 55.84 %

True Positive Ratio = TP/(TP+FN) = 0.78

False Positive Ratio = FP/ (FP + TN) = 0.08

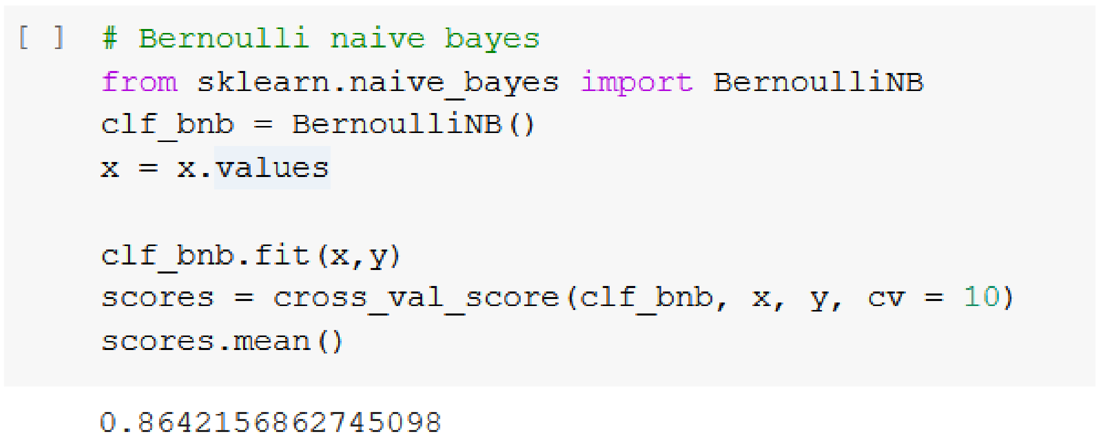
b) Bernoulli Naive Bayes

Bernoulli Naïve Bayes is used for discrete data and it works on Bernoulli distribution.

The Bernoulli distribution

$$p(x) = P[X = x] = \begin{cases} q = 1 - p & x = 0 \\ p & x = 1 \end{cases}$$

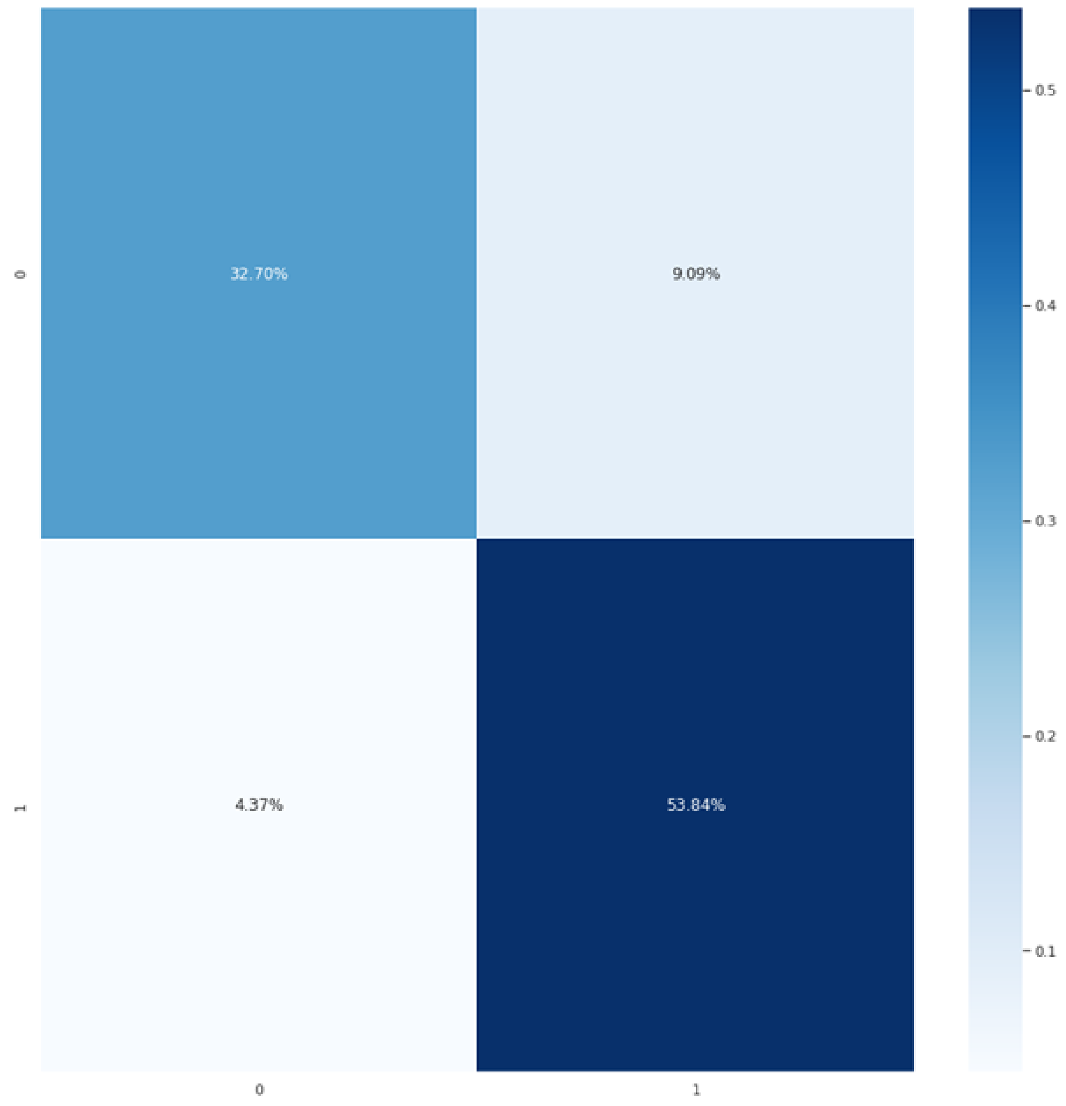
Bernoulli Naive Bayes Code



```
[ ]  # Bernoulli naive bayes
     from sklearn.naive_bayes import BernoulliNB
     clf_bnb = BernoulliNB()
     x = x.values

     clf_bnb.fit(x,y)
     scores = cross_val_score(clf_bnb, x, y, cv = 10)
     scores.mean()

     0.8642156862745098
```

Accuracy obtained = 0.8642 = 86.42 %

Confusion Matrix for Bernoulli Naïve Bayes:



True Negative (TN) – 32.70%

False Negative (FN) – 9.09 %

False Positive (FP) – 4.37%

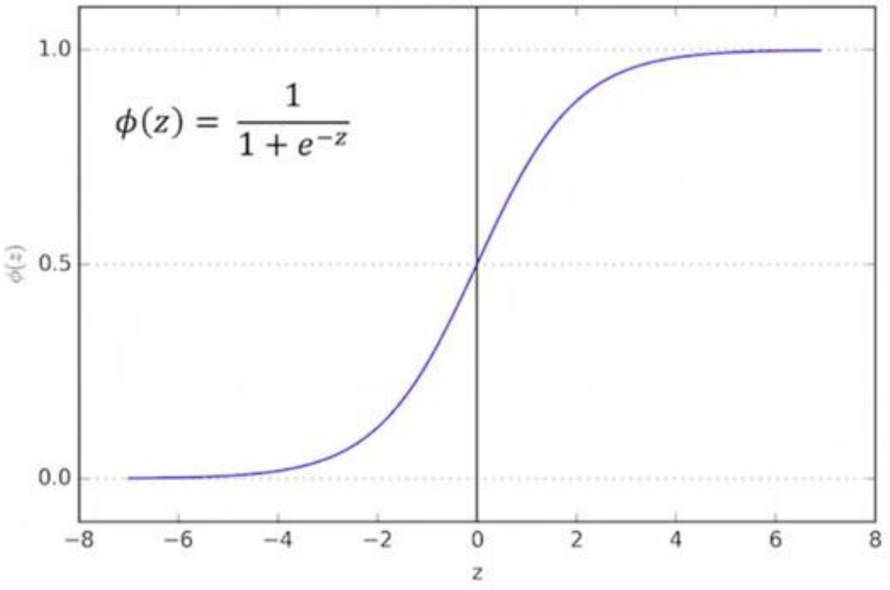
True Positive (TP) – 53.84 %

True Positive Ratio = TP/(TP+FN) = 0.85

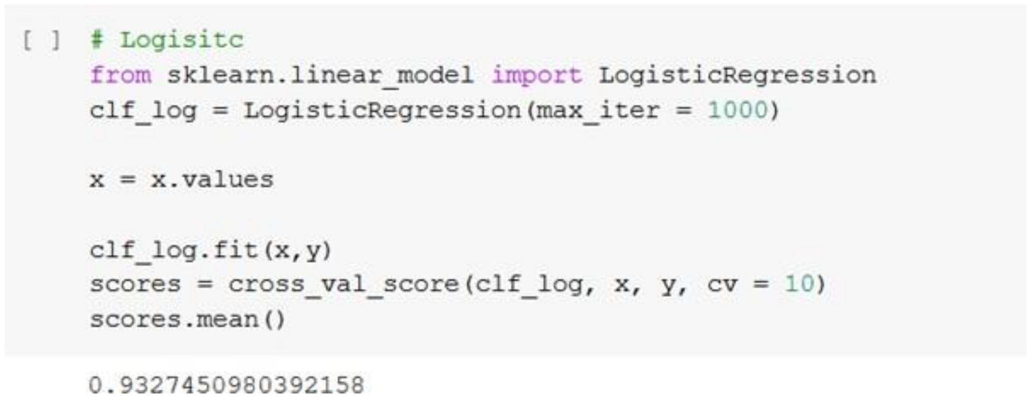
False Positive Ratio = FP/ (FP + TN) = 0.11

2) Logistic Regression

Logistic Regression is a method which is used to predict a dependent variable, given a set of independent variables, such that the dependent variable is categorical. Logistic Regression uses logistic function. Logistic function is a S shaped curve that takes any value and maps it between **0** and **1**.



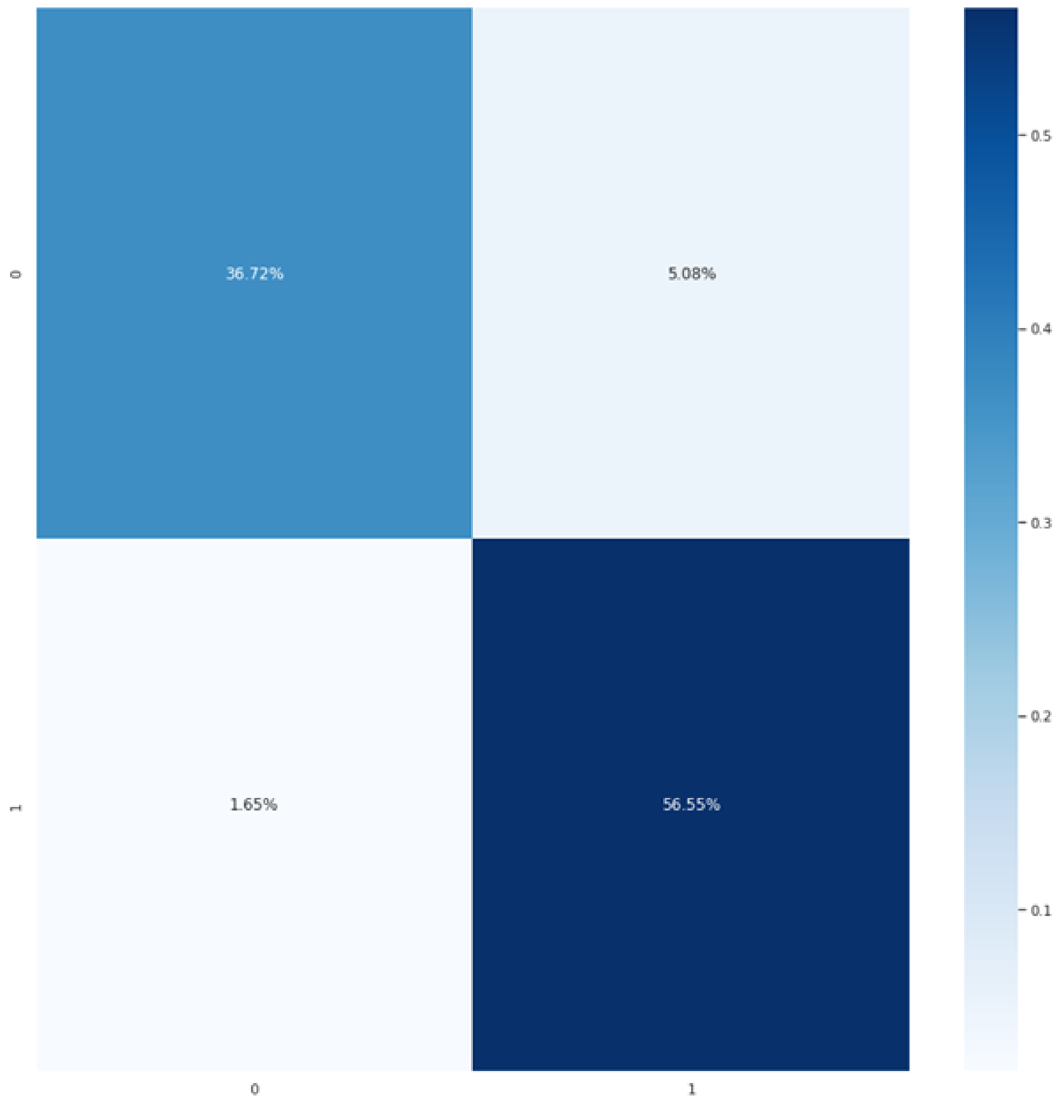
Logistic Regression Code



```
[ ] # Logisitc
    from sklearn.linear_model import LogisticRegression
    clf_log = LogisticRegression(max_iter = 1000)

    x = x.values

    clf_log.fit(x,y)
    scores = cross_val_score(clf_log, x, y, cv = 10)
    scores.mean()

    0.9327450980392158
```

Accuracy obtained = 0.9327 = 93.27 %

Confusion Matrix for Logistic Regression:



True Negative (TN) – 36.72%

False Negative (FN) – 5.08 %

False Positive (FP) – 1.65%

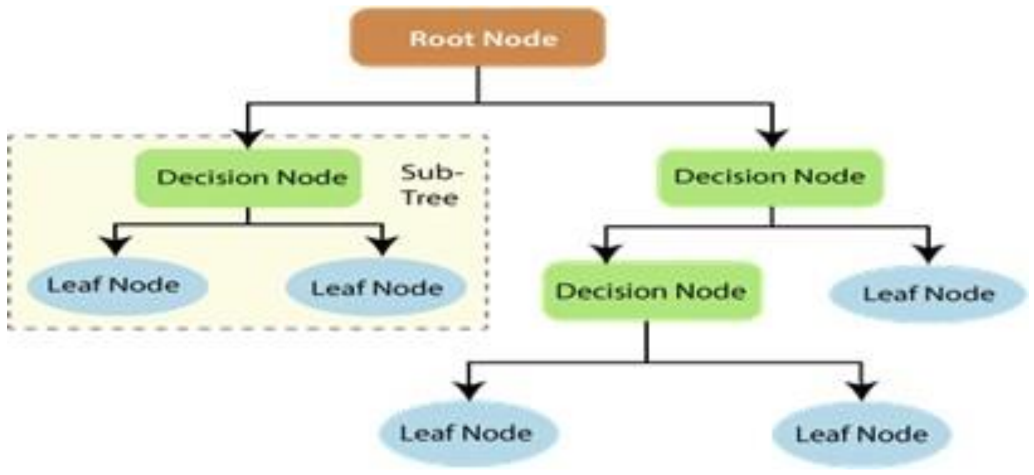
True Positive (TP) – 56.55 %

True Positive Ratio = TP/(TP+FN) = 0.91

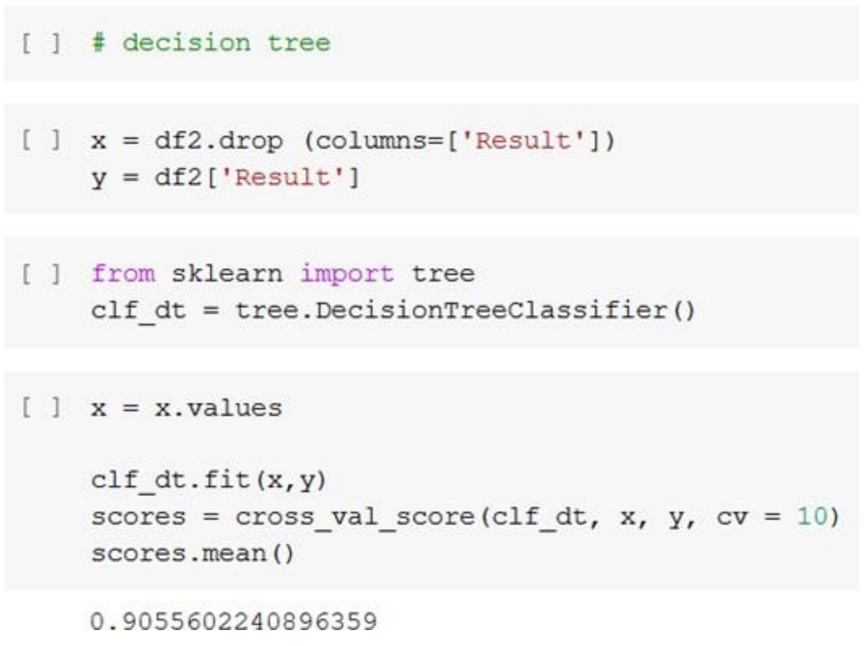
False Positive Ratio = FP/ (FP + TN) = 0.04

3) Decision Tree

In a decision tree, each node is a test or condition on an attribute, each branch represents the outcome of the condition or test and the leaf node represents the decision taken after all computation. Because of the flow chart like structure, decision trees are easy to understand.



Decision tree Code



```
[ ] # decision tree
```

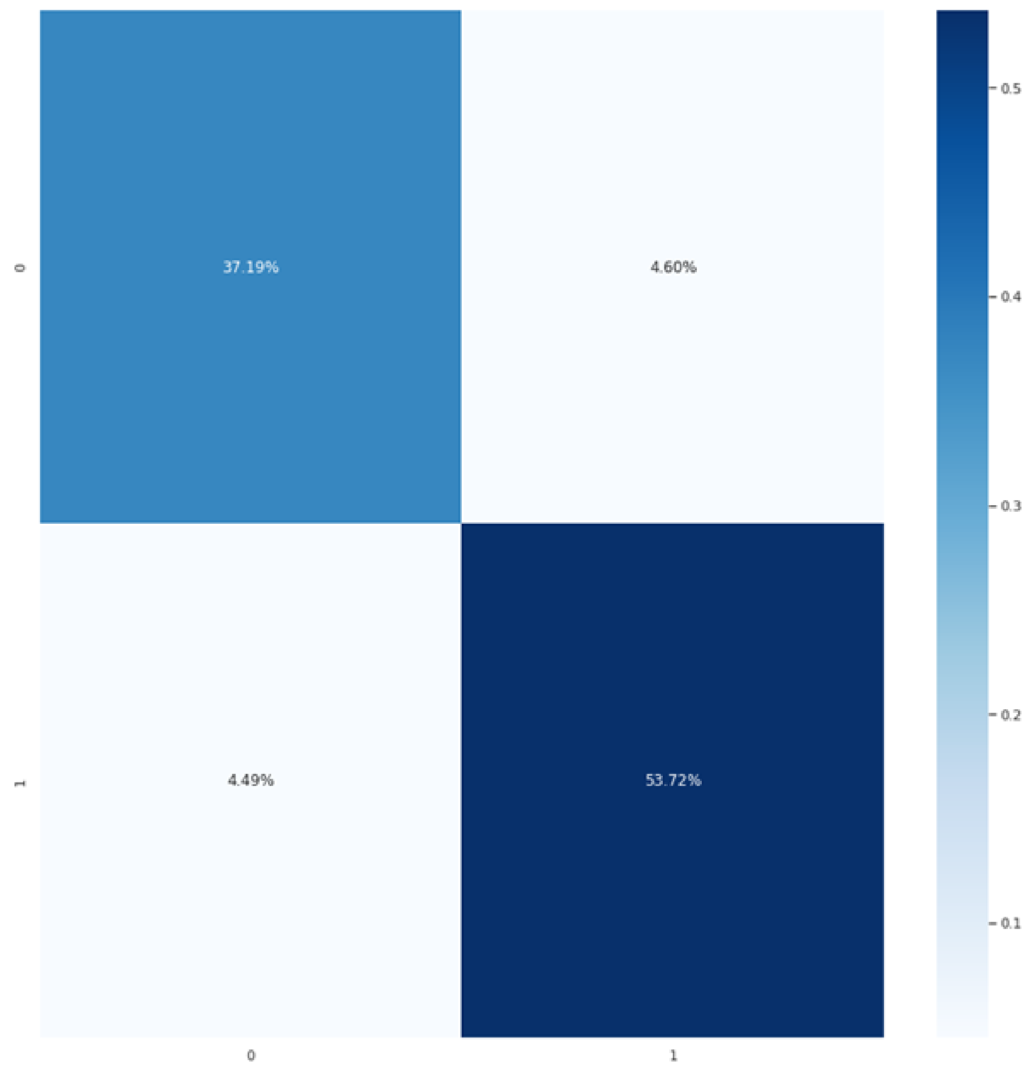
```
[ ] x = df2.drop (columns=['Result'])
    y = df2['Result']
```

```
[ ] from sklearn import tree
    clf_dt = tree.DecisionTreeClassifier()
```

```
[ ] x = x.values

    clf_dt.fit(x,y)
    scores = cross_val_score(clf_dt, x, y, cv = 10)
    scores.mean()

    0.9055602240896359
```

Accuracy obtained = 0.9055 = 90.55 %

Confusion Matrix for Logistic Regression:



True Negative (TN) – 37.19%

False Negative (FN) – 4.60 %

False Positive (FP) – 4.49%

True Positive (TP) – 53.72 %

True Positive Ratio = TP/(TP+FN) = 0.92

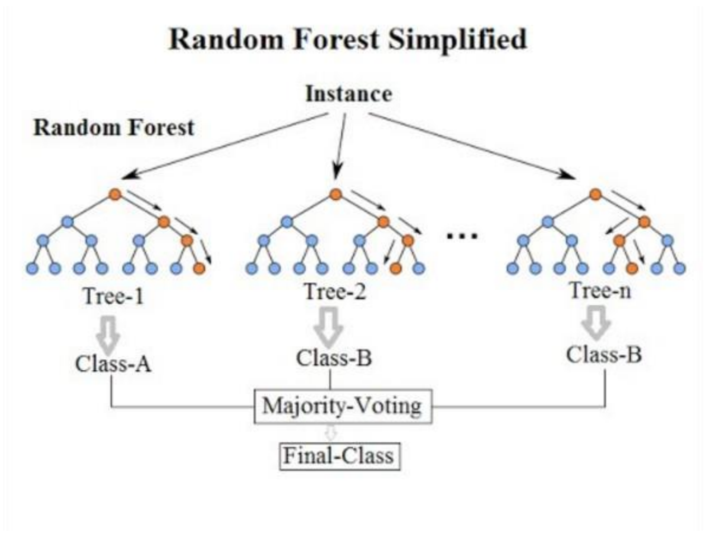
False Positive Ratio = FP/ (FP + TN) = 0.10

4) Random Forest

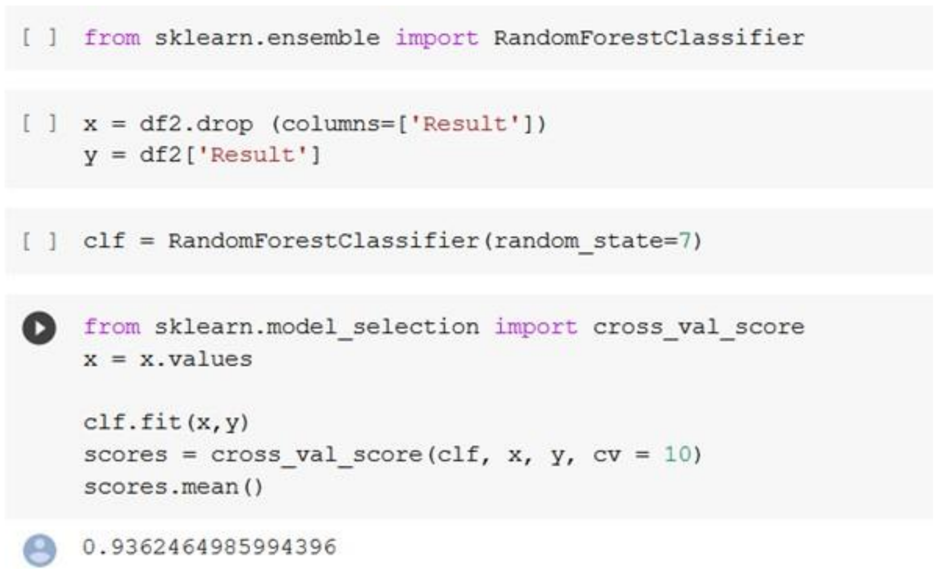
Random Forest is one of the most popular and most powerful machine learning algorithms that is capable of performing both regression and classification tasks.

As the name suggests, this algorithm creates the forest with a number of decision trees. In general, the more trees in the forest, the more robust is the prediction and thus high accuracy.

One of the biggest advantages of Random Forest classifier is that it handles the missing values and maintains accuracy for the missing data and it also doesn't overfit the model.



Random Forest Code



```
from sklearn.ensemble import RandomForestClassifier
```

```
x = df2.drop (columns=['Result'])
y = df2['Result']
```

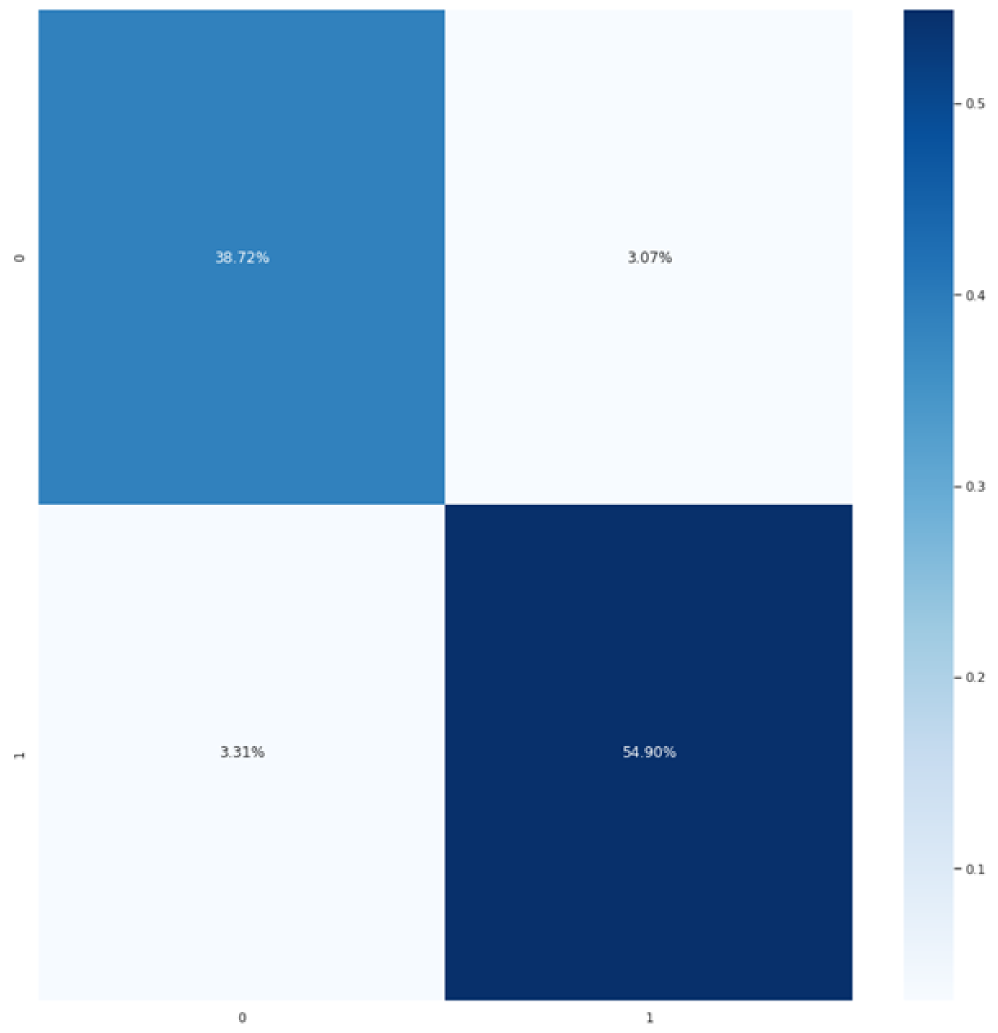
```
clf = RandomForestClassifier(random_state=7)
```

```
from sklearn.model_selection import cross_val_score
x = x.values

clf.fit(x,y)
scores = cross_val_score(clf, x, y, cv = 10)
scores.mean()
```

```
0.9362464985994396
```

Accuracy obtained = 0.9362 = 93.62 %

Confusion Matrix for Random Forest:



True Negative (TN) – 38.72%

False Negative (FN) – 3.07 %

False Positive (FP) – 3.31%

True Positive (TP) – 54.90 %

True Positive Ratio = TP/(TP+FN) = 0.94

False Positive Ratio = FP/ (FP + TN) = 0.07

4.3 Results

Algorithm	True Positive Rate	False Positive Rate	Accuracy
Gaussian Naïve Bayes	**0.78**	**0.08**	**82.6 %**
Bernoulli Naïve Bayes	**0.85**	**0.11**	**86.2 %**
Random Forest	**0.94**	**0.07**	**93.6 %**
Logistic Regression	**0.91**	**0.04**	**93.2 %**
Decision Tree	**0.92**	**0.10**	**90.6 %**

True Positive Ratio is also known as sensitivity or recall. True Positive Ratio tells us how many correct benign samples are there among all the benign samples. On the other hand, False Positive Ratio tells us how many incorrect benign results occur among all the malware samples.

As we can see, logistic regression and decision tree were able to achieve accuracy of more than 90 percent. Random Forest achieved the highest accuracy of 93.6 %. Bayesian based classifiers obtained a little lower accuracy as compared to other classifiers.

Chapter 5

DEVELOPING WEB APPLICATION USING FLASK

After employing various Supervised Machine Learning methods we deployed the model to a web application since the goal was to make a final product which can predict the possibility of any Android app being Malware or Benign. We used Python framework Flask to implement the back end functionality of the application while the front end was implemented by HTML and CSS. The web application is hosted on a local server. We can host the app on a live server in future which would need some changes in the back end. Following is the elaborated description of the implementation of the application.

5.1 Uploading APK from a local directory

Uploading an APK file in Flask needs an HTML form with its enctype attribute assigned to 'multipart/form-data' which posts the file to a URL.

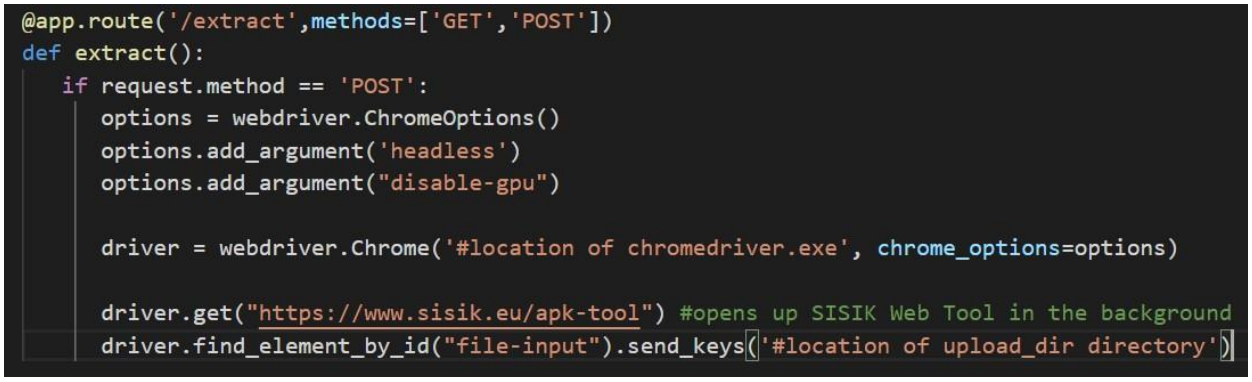
```
   <form id="form"action = "http://localhost:5000/uploader" method ="POST"
   enctype = "multipart/form-data">
   <input type = "file" id = "file" name = "file.apk" />
```

URL handler then extracts the APK from request.files object and uploads it to any desired location. The desired location is the local database directory in our project named 'uploads_dir'. After successful uploading of the APK, the home page upload.html is returned back and the APK is now saved in the local database directory 'uploads_dir' named as 'file.apk'.

```
@app.route('/uploader', methods = ['GET', 'POST'])
def uploader():
   if request.method == 'POST':
      f = request.files['file.apk']
      f.save(os.path.join("uploads_dir", secure_filename(f.name)))
      return render_template('upload.html')
```

5.2 Submitting the APK to SISIK Web tool

The APK was successfully uploaded to the database directory in our local machine. The goal was to fetch the permissions and intent filter data of the uploaded APK into an array. Therefore the APK needed to be submitted to the SISIK Web tool (A website which shows the Manifest.xml file of any APK) which we used earlier in the dataset creation process. We used Selenium Web Driver to get this APK from the ‘uploads_dir’ directory and upload to the SISIK tool. Normally, while using Selenium Web Driver, the browser opens automatically and does what is assigned to it (in this case uploading of APK to SISIK). Thus we hid the web browser by using webdriver.ChromeOptions().add_argument('headless') so that the User Experience would be better.

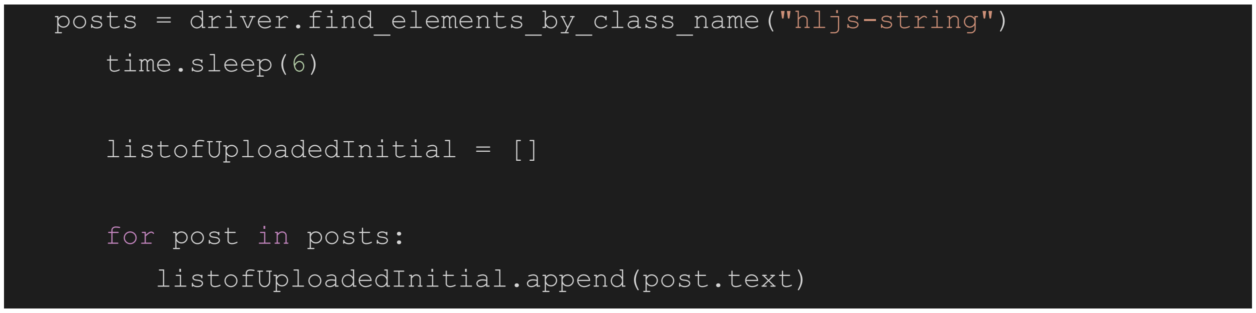


```
@app.route('/extract',methods=['GET','POST'])
def extract():
    if request.method == 'POST':
        options = webdriver.ChromeOptions()
        options.add_argument('headless')
        options.add_argument("disable-gpu")

        driver = webdriver.Chrome('#location of chromedriver.exe', chrome_options=options)

        driver.get("https://www.sisik.eu/apk-tool") #opens up SISIK Web Tool in the background
        driver.find_element_by_id("file-input").send_keys('#location of upload_dir directory')
```

5.3 Scraping the desired data to an array list and filtering out undesired data

With the help of Selenium, web scraping of permissions and intents used which was inside <uses-permission> and <intent-filter> tags, was done. Time frame of 6 milliseconds is given for the driver to fetch data completely. There were a lot of undesired strings fetched into the array list which were cleaned up. Only strings with substring “android.permissons” and “android.intent” were taken.

Unfiltered array list (named as listofUploadedInitial):



```
  posts = driver.find_elements_by_class_name("hljs-string")
    time.sleep(6)

    listofUploadedInitial = []

    for post in posts:
      listofUploadedInitial.append(post.text)
```

Scraping Permission and Intent Strings

Filtered array list (named as listofUploadedFinal):

```
substring1 = "android.permission"
substring2 = "android.intent"

listofUploadedFinal = []
i=0
while i < len(listofUploadedInitial):
   print(i)
   if search(substring1,listofUploadedInitial[i]):
      listofUploadedFinal.append(listofUploadedInitial[i])
   elif search(substring2,listofUploadedInitial[i]):
      listofUploadedFinal.append(listofUploadedInitial[i])
   i+=1
```

Filtering out undesired data

5.4 Getting our input ready

The input for the prediction should be a binary array list. Each index of this array list will mean that particular item (permission and intent strings) present in the corresponding index of listofUploadedFinal array list is actually present in our dataset columns or not. The input list is named as inputList while df_col_list is the dataset column list. The input is ready for the prediction.

```
inputList = []
inputList = inputList+[0]*(len(df_col_list))
for i in range(0,len(df_col_list)):
   if(df_col_list[i] in listofUploadedFinal):
      inputList[i] = 1
   else:
      inputList[i] = 0
inputList.append(sum(inputList))
```

5.5 Saving our ML model as a Pickle file

To use the ML model in the Web Application, it was saved into a Pickle file (.pkl) using the Joblib package in Python. Pickle is a package in Python which is used to serialize objects in Python. We can use the pickle to serialize the models and save the serialized format to a .pkl file.

```
# saving as file
joblib.dump(clf, 'clf_rf.pkl')
```

For every classifying algorithm namely Logistic Regression, Random Forest, Gaussian Naive Bayes, Decision Tree and Naive Bayes, the model was saved into its corresponding Pickle file and ready to be used in the back end of web application.

5.6 Loading ML models to Flask app

Models were loaded in the Flask app using the Joblib package.

```
    model1 = joblib.load('clf_log1.pkl')
    model2 = joblib.load('clf_rf2.pkl')
    model3 = joblib.load('clf_dt3.pkl')
    model4 = joblib.load('clf_bnb4.pkl')
    model5 = joblib.load('clf_gnb5.pkl')
```

5.7 Getting the results

Using scikit-learn functions ‘predict’ and ‘predict_proba’ and passing our ‘inputList’ as an argument we got the results. Using ‘predict’ we get the final outcome that the algorithm has classified into (1 (benign) or 0 (malware)) and ‘predict_proba’ returns the probability.

```
      res1 = model1.predict([inputList]) // benign or malware
      prob1 = model1.predict_proba([inputList]) // probability
```

5.8 Improving accuracy with Voting and Average based Ensembling

To improve the accuracy of the results we have used five ML algorithms and based on their individual votes and average of the probability we achieved our final results.

Voting:

```
malwareVote = 0
benignVote = 0
vote = [res1[0], res2[0], res3[0], res4[0], res5[0]]
for i in range(0,len(vote)):
  if(vote[i] == 1):
    benignVote+=1
  elif(vote[i] == 0):
    malwareVote+=1
```

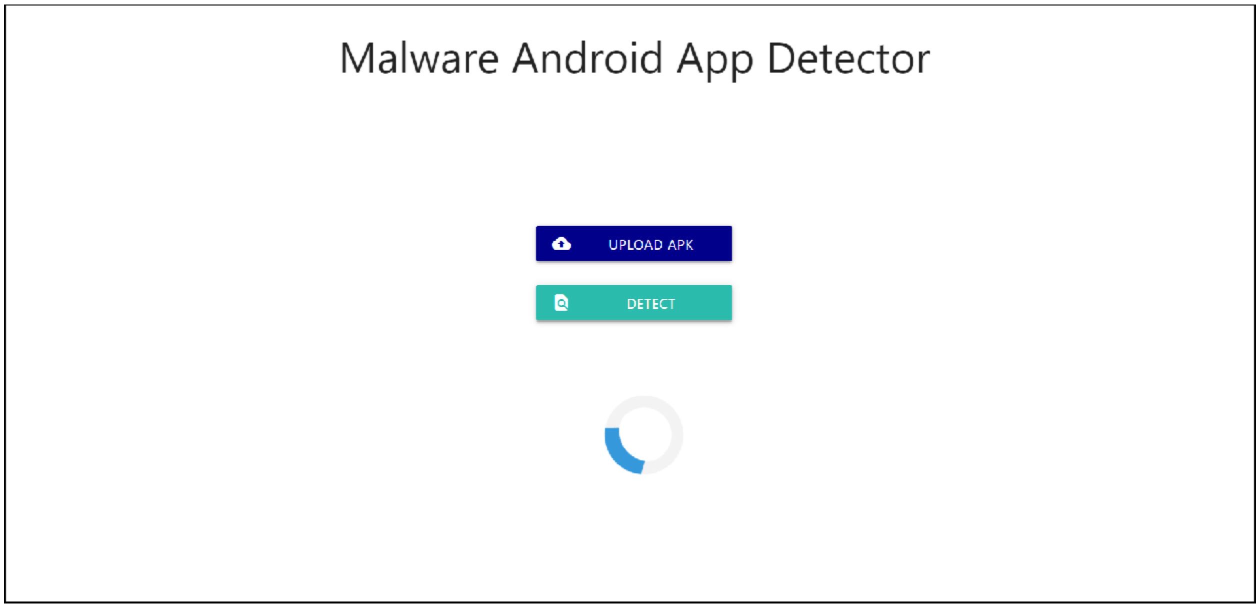
If benignVote is greater than malwareVote then the APK is classified as Benign and vice versa.

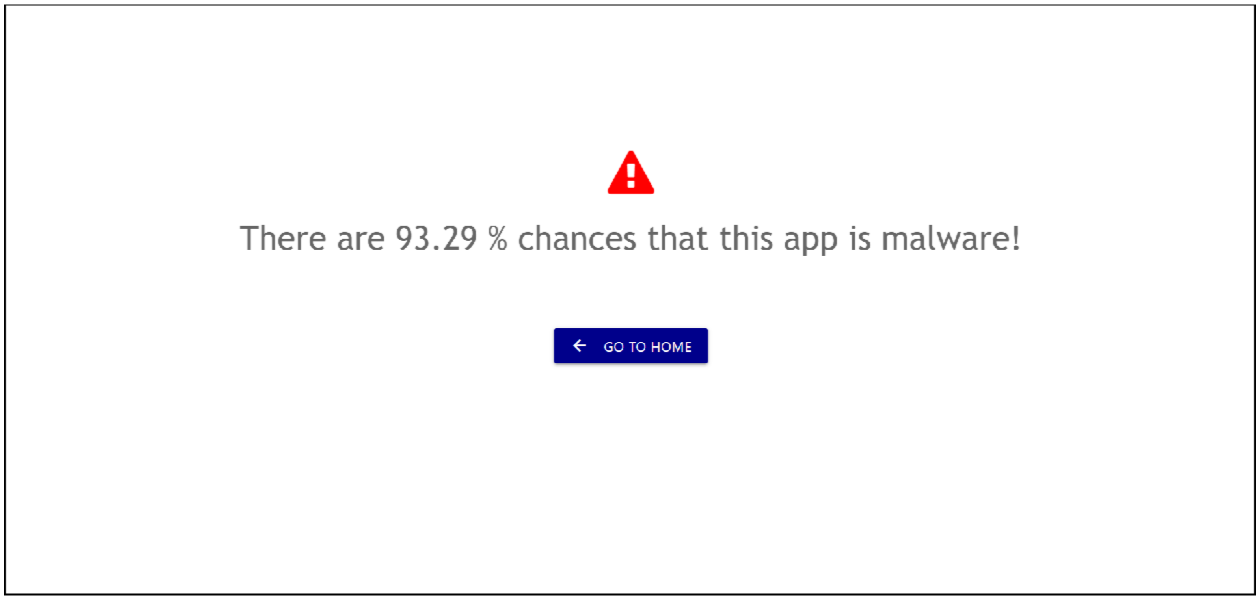
Taking Average:

```
perBenign = (prob1[0][1]*100 + prob2[0][1]*100 + prob3[0][1]*100 + prob4[0][1]*100 + prob5[0][1]*100)/5
perMal = (prob1[0][0]*100 + prob2[0][0]*100 + prob3[0][0]*100 + prob4[0][0]*100 + prob5[0][0]*100)/5
```

We got the final results and they were returned to the HTML.

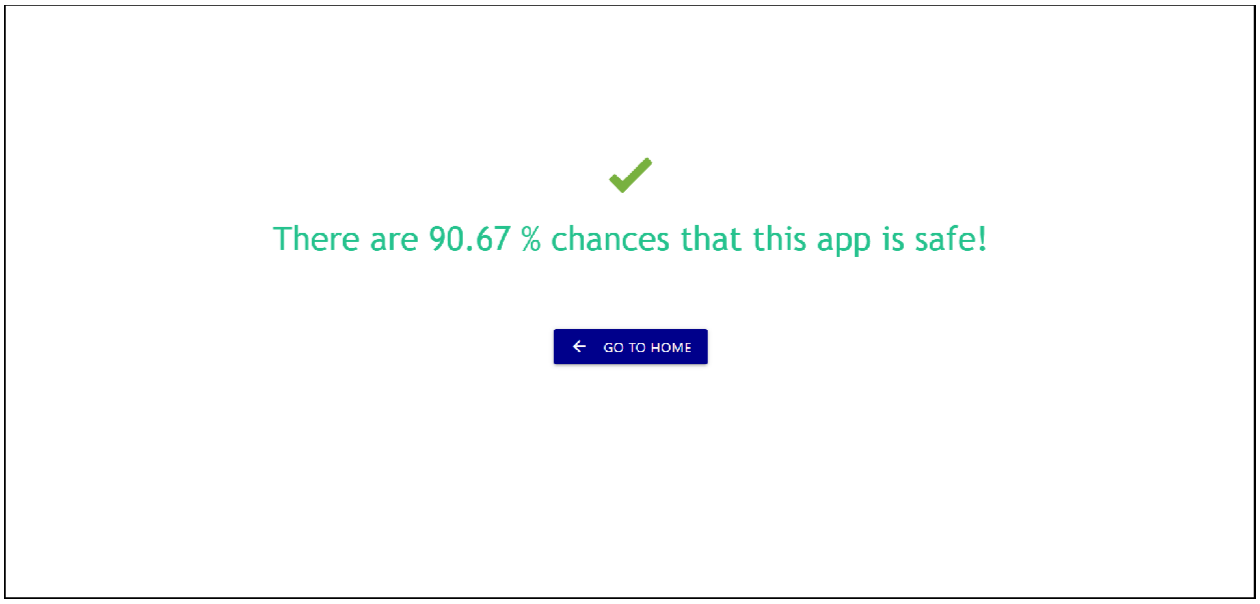
5.9 Screenshots of the final product




There are 93.29 % chances that this app is malware!

When uploaded APK is Malware as predicted by the model

\


There are 90.67 % chances that this app is safe!

When uploaded Android app is safe as predicted by the model

Chapter 5

Conclusion & Future Work

Our web application was successfully able to predict whether an app is malware or benign with an average accuracy of **89.24 %**. For future work, more number of APKs can be taken into consideration and we can perform clustering on the basis of the categories of the apps to achieve better results.

Bibliography

- https://developer.android.com/docs
- https://www.intechopen.com/books/smartphones-from-an-applied-research-perspective/malware-analysis-and-detection-on-android-the-big-challenge
- https://selenium-python.readthedocs.io/api.html
- https://scikit-learn.org/stable/tutorial/basic/tutorial.html
- https://www.researchgate.net/
- https://www.virustotal.com/gui/
- https://flask.palletsprojects.com/en/1.1.x/
- https://developer.android.com/guide/topics/permissions/overview
- https://developer.android.com/guide/components/intents-filters