

Machine-Learning-Based Android Malware Detection

for the course

IT303: Software Engineering

Submitted by

Sunil Kumar(181IT148)

P Narendra(181IT229)

Samarth S Hadimani(181IT240)

IV SEM B.Tech (IT)

Under the guidance of

Dr. Biju R. Mohan

Dept of IT, NITK Surathkal

And

Under the mentorship of

Dr. Manjunath Vanahalli

Dept of IT, NITK Surathkal

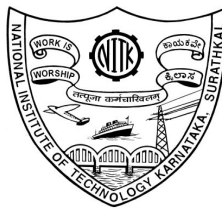
*in partial fulfillment for the award of the degree
of*

Bachelor of Technology

in

Information Technology

at



Department of Information Technology

National Institute of Technology Karnataka, Surathkal.

October 2020

INDEX

Page No

1.Introduction	1
2.Literature Survey	2
3.Methodology	3
4.Result	5
5.Conclusion	8

1.Introduction

Today Mobile Technology is being extensively used around the world. Since 2008 the usage of mobile technology is growing very fast. All the confidential and private information like photographs, videos, banking details, etc are easily stored on the mobile.

Many mobiles are available with many operating systems. Android is an open-source mobile operating system available on many smartphones. According to Google 1.3 million, Android Devices are being activated each day. Nowadays, the mobile device becomes an essential part of the life of the people. Most of the activities in our day-to-day life are being completed with the help of the mobile device. It is necessary to protect the data stored in the mobile from malware applications. Because of the presence of the multi-vulnerabilities in the android mobile devices, the attackers may easily hack the data stored in the mobile, which leads to the possibility of fraudulent activities. The permissions asked during installation may provide the ability to access the mobile data without much difficulty. Hence Android Malware Detection system is necessary for mobile devices.

Malware threats are expected to increase with the extension of the functionality in mobile phones. Many harmful applications contain malware and put the user's data and device at risk. These applications contain malware categories like spyware, Trojans, phishing apps, etc. There are various Android Malware Detection Techniques available for detecting malware. In this project, the existing Android Malware Detection Techniques are compared to design more robust and efficient techniques.

2.Literature survey

Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye, "*Significant Permission Identification for Machine-Learning-Based Android Malware Detection*" in this IEEE paper they used SVM and a large dataset to test the proposed MLDP model. They have shown that it is possible to reduce the number of permissions to be analyzed for mobile malware detection while maintaining high effectiveness and accuracy. SIGPID has been designed to extract only significant permissions through a systematic three-level pruning approach. It can detect 93.62% of malware in the dataset and 91.4% unknown/new malware.

Aung, Zarni, and Zaw, Win, "*DETECTING MALWARE USING PERMISSIONS*" in Research Gate Paper they proposed a framework that intends to develop a machine learning-based malware detection system on Android to detect malware applications and to enhance the security and privacy of smartphone users. This system monitors various permission-based features and events obtained from the android applications and analyses these features by using machine learning classifiers to classify whether the application is benign or malware. The performances of machine learning techniques were evaluated using the true positive rate, false-positive rate, and overall accuracy.

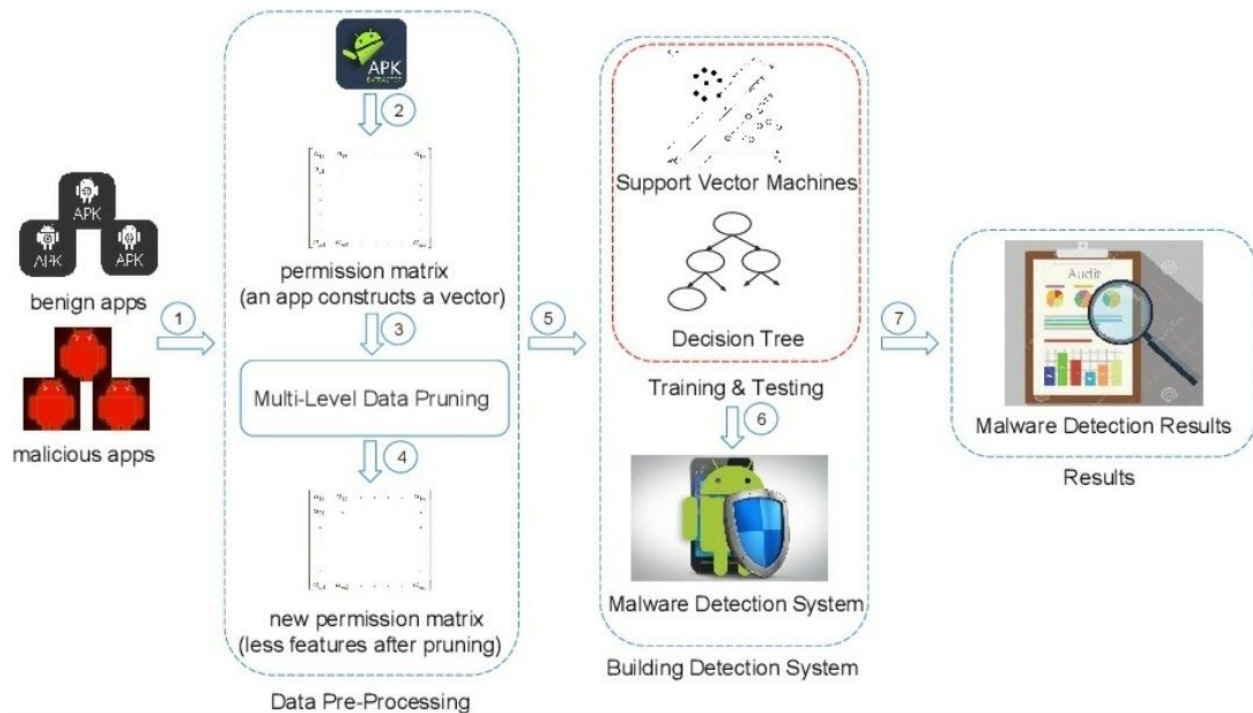
3.Methodology

AcquiringAPKdataset:-The first module of the malware detection system is the dataset. The data set represents a representative collection of malware and benign ware. A proper data set is essential for the analysis of the behavior of the malware. Examples of both the malware and the benign apps for proper detection are taken under consideration.

Preprocessing of data:-

- **Feature Extraction:-**To this module the APK dataset we have obtained from the last module is given as input. The feature extractor is the component that extracts the desired features from the malware and benign apps. The mechanism that extracts the static features is called a static feature extractor. Static features can be extracted from the manifest file, dex file, and byte code. The most widely used tool is the APK tool. From the APK tool, we obtain the APK file, manifest file, classes.dex, and small file. We perform reverse engineering on the dataset followed by parsing to extract the static features.
- **FeatureSelection:-**To this module the extracted features we have obtained from the last module are given as input. But we need to select those among them which will provide better accuracy in the classification process. Using Scikit-Learn's PCA estimator, The fit learns some quantities from the data, most importantly the components and explained variance. To see what these numbers mean, let's visualize them as vectors over the input data, using the components to define the direction of the vector, and the explained variance to define the squared-length of the vector.

Machine learning and Malware Detection:- The selected features from the feature selector are given as input. The detection method or the classifier is used to determine whether an app is a malware or not. Based on the features, the theclassifier classifies apps as malware or benign. Mostclassifiers use machine learning. Classifiers based on machine learning use one or multipleclassifiers. Layered classifiers may also be used in the detection process. We now build the model using KNN and SVM classification algorithms.



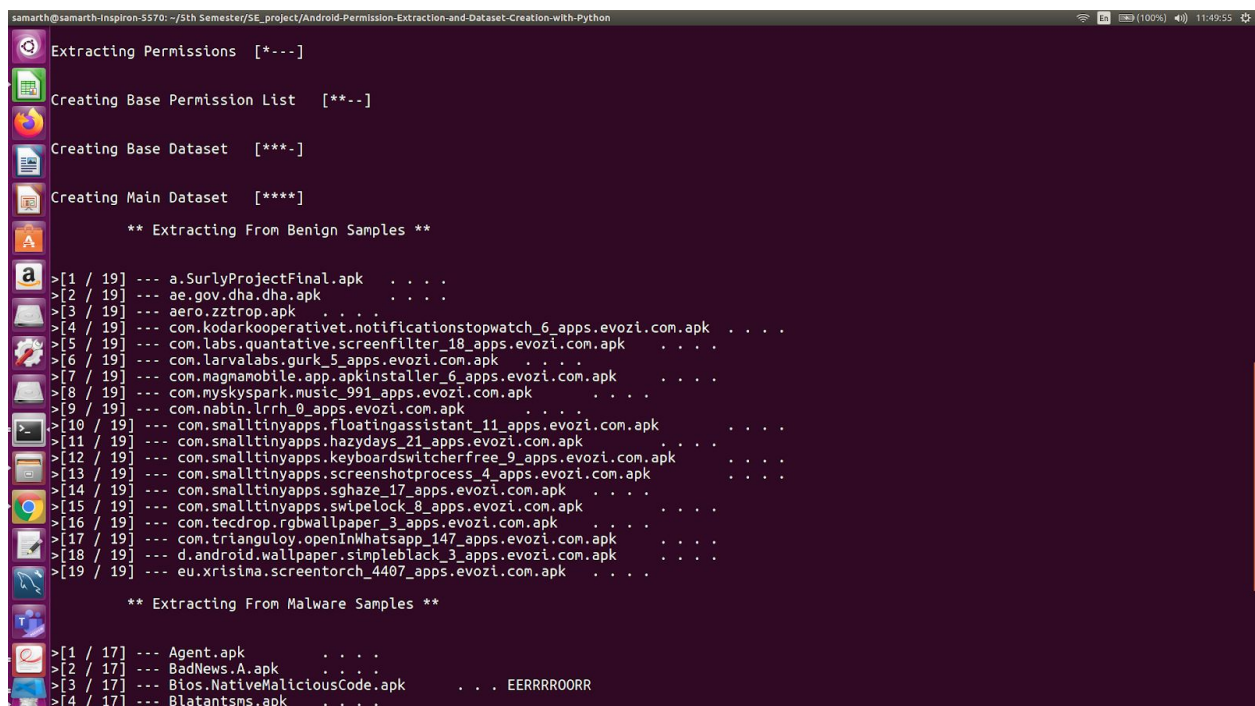
Malware Detection Algorithm:-The detection method or the classifier is used to determine whether an app is malware or not. Based on the features, the classifier classifies apps as malware or benign ware. Most classifiers use machine learning. Classifiers based on machine learning use one or multiple classifiers. Layered classifiers may also be used in the detection process. We now build the model using KNN and SVM classification algorithms.

1. Input = preprocessed vector
2. X = feature extraction.final perm vector
3. Y = feature extraction.final binary class vector
4. $\text{model} = \text{KneighborsClassifier}(\text{arg})$
5. or $\text{model} = \text{svm.SVC}(\text{arg})$
6. $X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}} = \text{train test split}(X, Y, \text{test size})$
7. $\text{model.fit}(X, Y)$
8. $\text{model.score}(X, Y)$
9. $\text{predicted} = \text{model.predict}(X_{\text{test}})$
10. $\text{accuracy} = \text{accuracy score}(y_{\text{test}}, \text{predicted})$
11. $\text{report} = \text{classification report}(y_{\text{test}}, \text{predicted})$
12. Output = classification report

4.Results and Screenshots:-

Feature extraction:-

The features that can be extracted by not executing the application are called static features. The mechanism that extracts the static features is called a static feature extractor. Researchers may consider only a single category of static features and others consider a set of multiple static features. Common static features are permission, API call, string extracted, native commands, XML elements, metadata, intents, broadcast receivers, hardware components, etc. Static features can be extracted from the manifest file, dex file, and byte code.



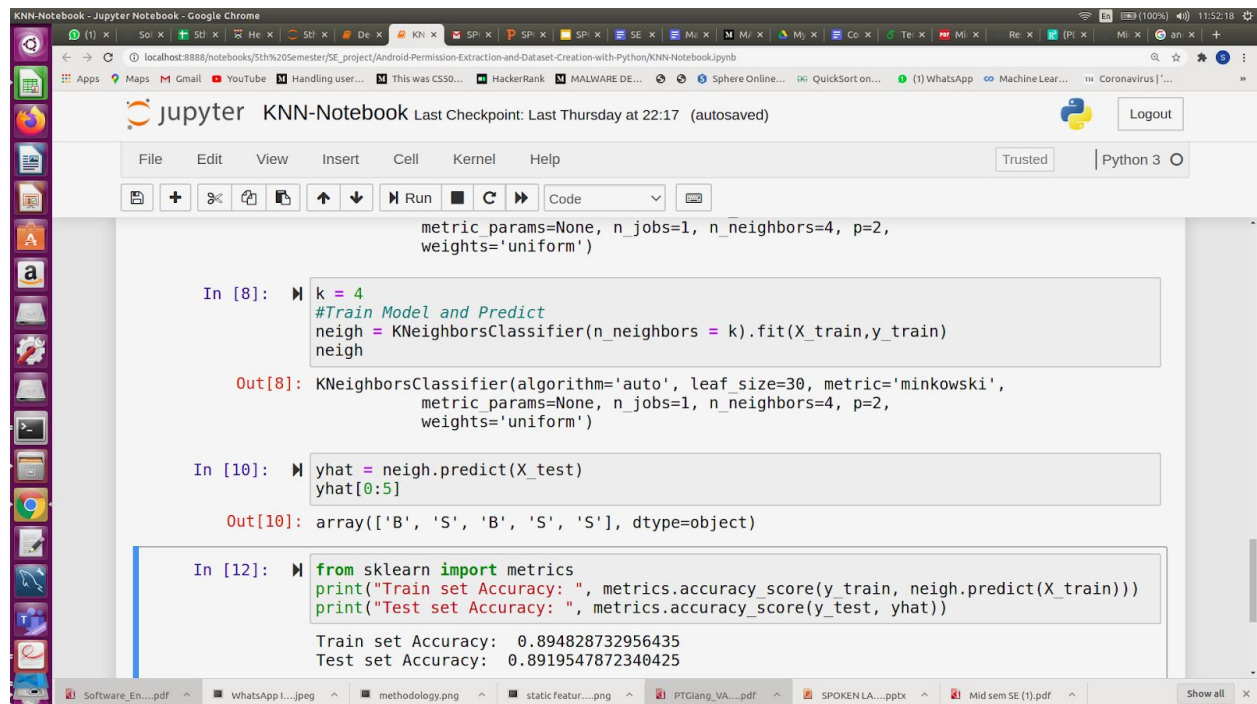
```
samarth@samarth-Inspiron-5570: ~/5th Semester/SE_project/Android-Permission-Extraction-and-Dataset-Creation-with-Python
Extracting Permissions  [*--]
Creating Base Permission List  [***]
Creating Base Dataset  [***]
Creating Main Dataset  [****]

** Extracting From Benign Samples **
>[1 / 19] --- a.SurlyProjectFinal.apk . . . .
>[2 / 19] --- ae.gov.dha.dha.apk . . . .
>[3 / 19] --- aero.zztrop.apk . . . .
>[4 / 19] --- com.kodarkooperativet.notificationstopwatch_6_apps.evozi.com.apk . . . .
>[5 / 19] --- com.labs.quantative.screenfilter_18_apps.evozi.com.apk . . . .
>[6 / 19] --- com.larvalabs.gurk_5_apps.evozi.com.apk . . . .
>[7 / 19] --- com.magmamobile.app.apkinstaller_6_apps.evozi.com.apk . . . .
>[8 / 19] --- com.myskyspark.music_991_apps.evozi.com.apk . . . .
>[9 / 19] --- com.nabin.lrrh_0_apps.evozi.com.apk . . . .
>[10 / 19] --- com.smalltinyapps.floatingassistant_11_apps.evozi.com.apk . . . .
>[11 / 19] --- com.smalltinyapps.hazydays_21_apps.evozi.com.apk . . . .
>[12 / 19] --- com.smalltinyapps.keyboardswitcherfree_9_apps.evozi.com.apk . . . .
>[13 / 19] --- com.smalltinyapps.screenshotprocess_4_apps.evozi.com.apk . . . .
>[14 / 19] --- com.smalltinyapps.sghaze_17_apps.evozi.com.apk . . . .
>[15 / 19] --- com.smalltinyapps.swipelock_8_apps.evozi.com.apk . . . .
>[16 / 19] --- com.tecdrop.rgbwallpaper_3_apps.evozi.com.apk . . . .
>[17 / 19] --- com.trianguloy.openInkWhatsapp_147_apps.evozi.com.apk . . . .
>[18 / 19] --- d.android.wallpaper.simpleblack_3_apps.evozi.com.apk . . . .
>[19 / 19] --- eu.xrisima.screentorch_4407_apps.evozi.com.apk . . . .

** Extracting From Malware Samples **
>[1 / 17] --- Agent.apk . . . .
>[2 / 17] --- BadNews.A.apk . . . .
>[3 / 17] --- Bios.NativeMaliciousCode.apk . . . EERRRROORR
>[4 / 17] --- Blatantsms.apk . . . .
```

The Extracted Features will be stored in Data.csv File.

KNN



KNN-Notebook - Jupyter Notebook - Google Chrome

Localhost:8888/notebooks/5th%20Semester/5E_project/Android-Permission-Extraction-and-Dataset-Creation-with-Python/KNN-Notebook.ipynb

jupyter KNN-Notebook Last Checkpoint: Last Thursday at 22:17 (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3

```
metric params=None, n_jobs=1, n_neighbors=4, p=2,
weights='uniform')

In [8]: k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh

Out[8]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric params=None, n_jobs=1, n_neighbors=4, p=2,
weights='uniform')

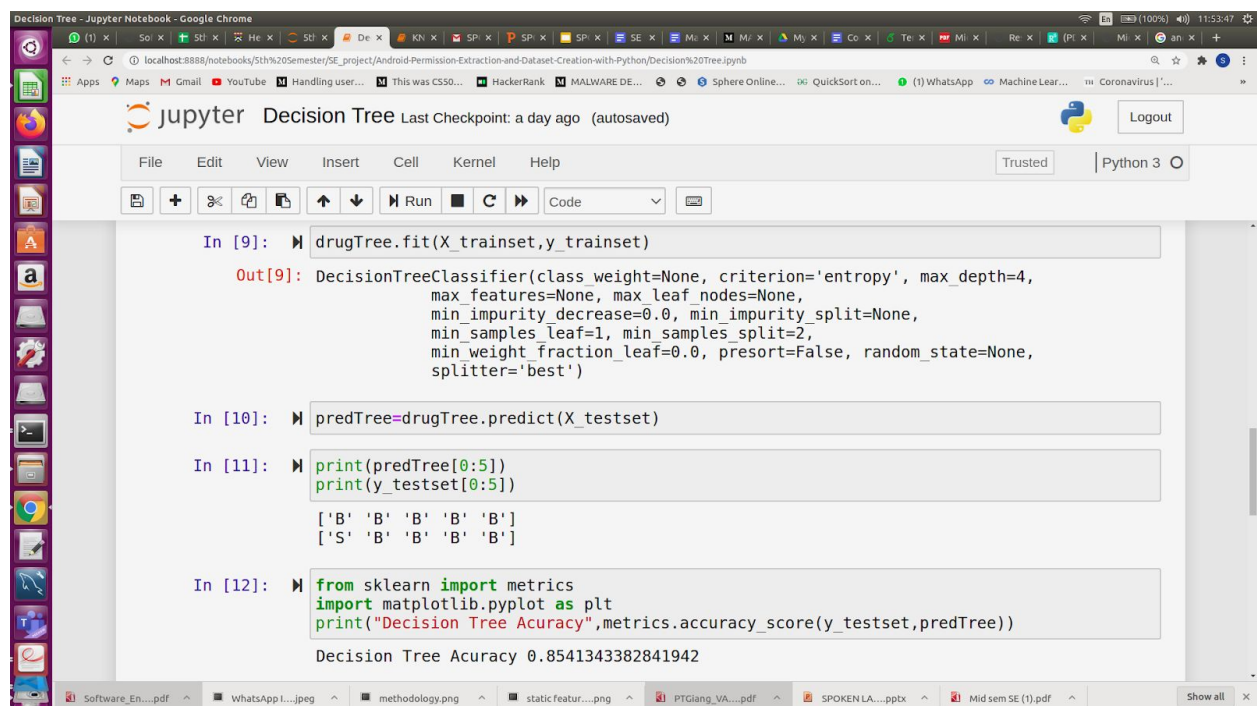
In [10]: yhat = neigh.predict(X_test)
yhat[0:5]

Out[10]: array(['B', 'S', 'B', 'S', 'S'], dtype=object)

In [12]: from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))

Train set Accuracy: 0.894828732956435
Test set Accuracy: 0.8919547872340425
```

Decision Tree



Decision Tree - Jupyter Notebook - Google Chrome

Localhost:8888/notebooks/5th%20Semester/5E_project/Android-Permission-Extraction-and-Dataset-Creation-with-Python/Decision%20Tree.ipynb

jupyter Decision Tree Last Checkpoint: a day ago (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3

```
In [9]: drugTree.fit(X_trainset,y_trainset)

Out[9]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')

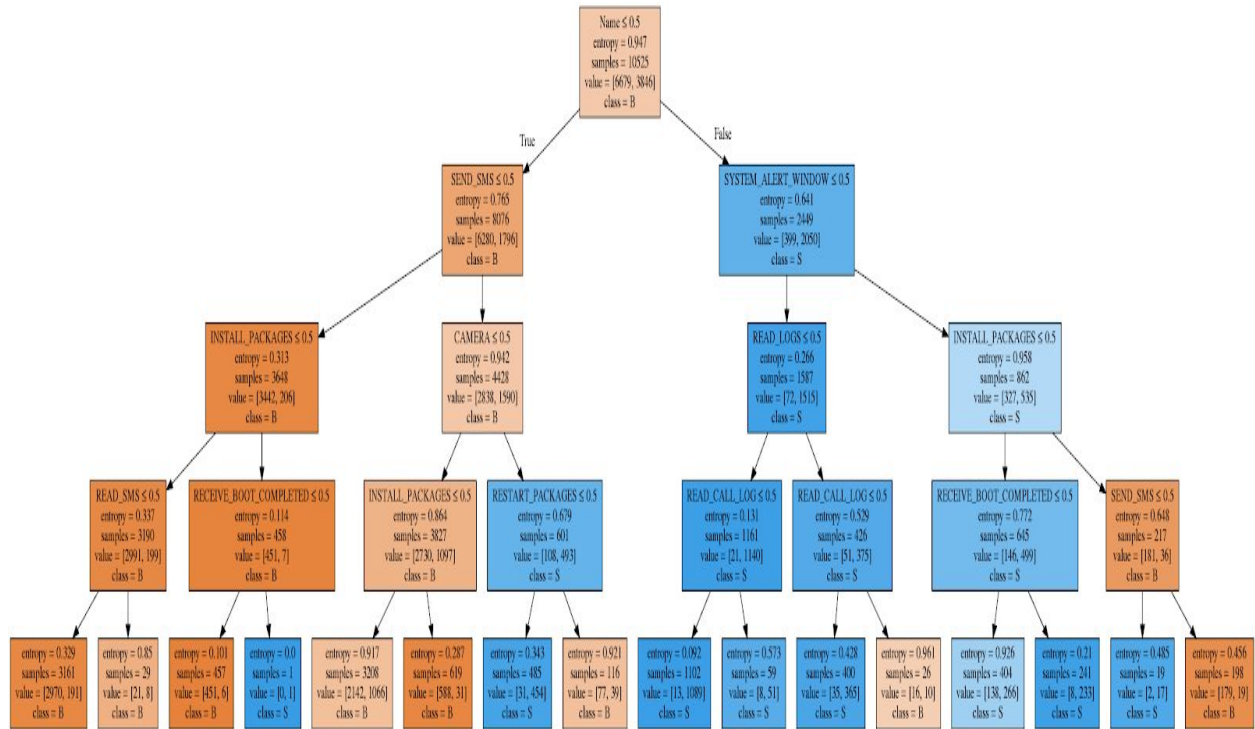
In [10]: predTree=drugTree.predict(X_testset)

In [11]: print(predTree[0:5])
print(y_testset[0:5])

['B' 'B' 'B' 'B' 'B']
['S' 'B' 'B' 'B' 'B']

In [12]: from sklearn import metrics
import matplotlib.pyplot as plt
print("Decision Tree Acuracy",metrics.accuracy_score(y_testset,predTree))

Decision Tree Acuracy 0.8541343382841942
```

Decision Tree for Malware Detection

5.Conclusion

In this paper, we have shown that it is possible to reduce the number of permissions to be analyzed for mobile malware detection while maintaining high effectiveness and accuracy.

SIGPID has been designed to extract only significant permissions through a systematic three-level pruning approach. Based on our dataset, which includes over 2000 malware, we only need to consider 22 out of 135 permissions to improve the runtime performance. We achieved over 89.5% detection accuracy in the KNN algorithm and 85 % accuracy by using Decision Trees. The extracted significant permissions can also be used by other commonly used supervised learning algorithms to yield good results.