CIS745 - Assignment 1

By: Aya Migdadi

1. Exercise 1

Exercise 1

1.1 Permissions Extraction

Permissions extraction phase is based on extract permission information from Malware and Benign applications by converting APKs files to smali files in their respective folders, and then create text files for each folder.

2.4 Dataset creation

The generated data will be in separated text files, so I used pandas library to collect all permissions of Malware and Benign in one Comma Separated Values (.csv) file to store them in one place ready to be fed into ML algorithms.

The data frame created has 129 rows (Apk file) and 147 columns (permissions extracted)

Formatting

The data is formatted in following way

	android.permission.READ_CONTACTS	android.permission.SEND_SMS	android.permission.READ_SMS	android.perm
00357b0e208c20df3182d54cb2ba15bf.apk	1.0	1.0	1.0	
02548535ff1cc285fddf699f2d77bcba.apk	1.0	1.0	1.0	
0639a74f508591f99a7d2309f5825fea.apk	1.0	1.0	1.0	
073a2f2d51c7dc00eb21e27cb8fa80f3.apk	1.0	1.0	1.0	
08634da89ce3e70a81bdf128b998a89e.apk	1.0	1.0	1.0	
...	...	...	...	
cmsecurity.applock.theme.valentineshearts.apk	0.0	0.0	0.0	
cn.xiaochuankeji.tieba.apk	0.0	0.0	0.0	
co.bitx.android.wallet.apk	1.0	0.0	0.0	
co.gotitapp.android.apk	0.0	0.0	0.0	
codematics.universal.tv.remote.control.apk	0.0	0.0	0.0	

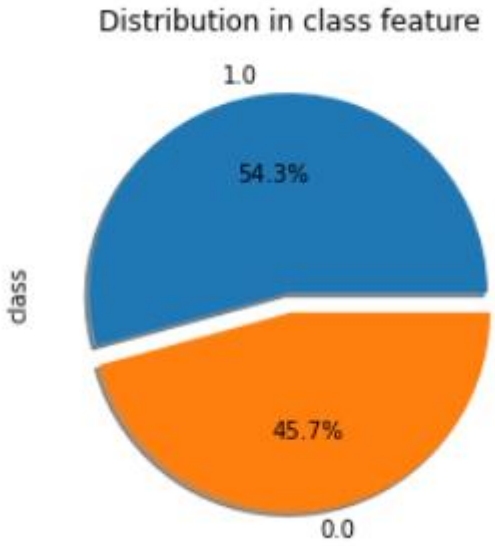
129 rows × 147 columns

The all dataset consists of 129 applications (70 Malware & 59 Benign)

The first column contains name of respective application and last column "CLASS" contains information if the application if from benign or malware family of training set. [0=Benign, 1=Malware]

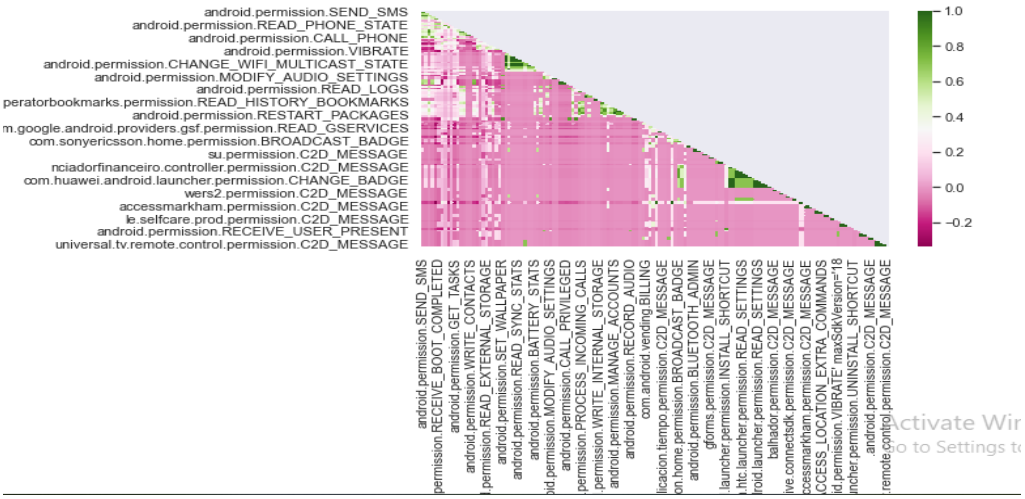
In between there are all the permissions (common + all found in 1st phase) with respective information bit, [0=The application does not use this permission, 1=This permission is used in the application]

2.5 Data Exploration



The pie chart shows unbalanced data set (54.3 malware and 45.7 benign), so I used Cohen's Kappa metric at evaluation step.

2.6 Features Selection



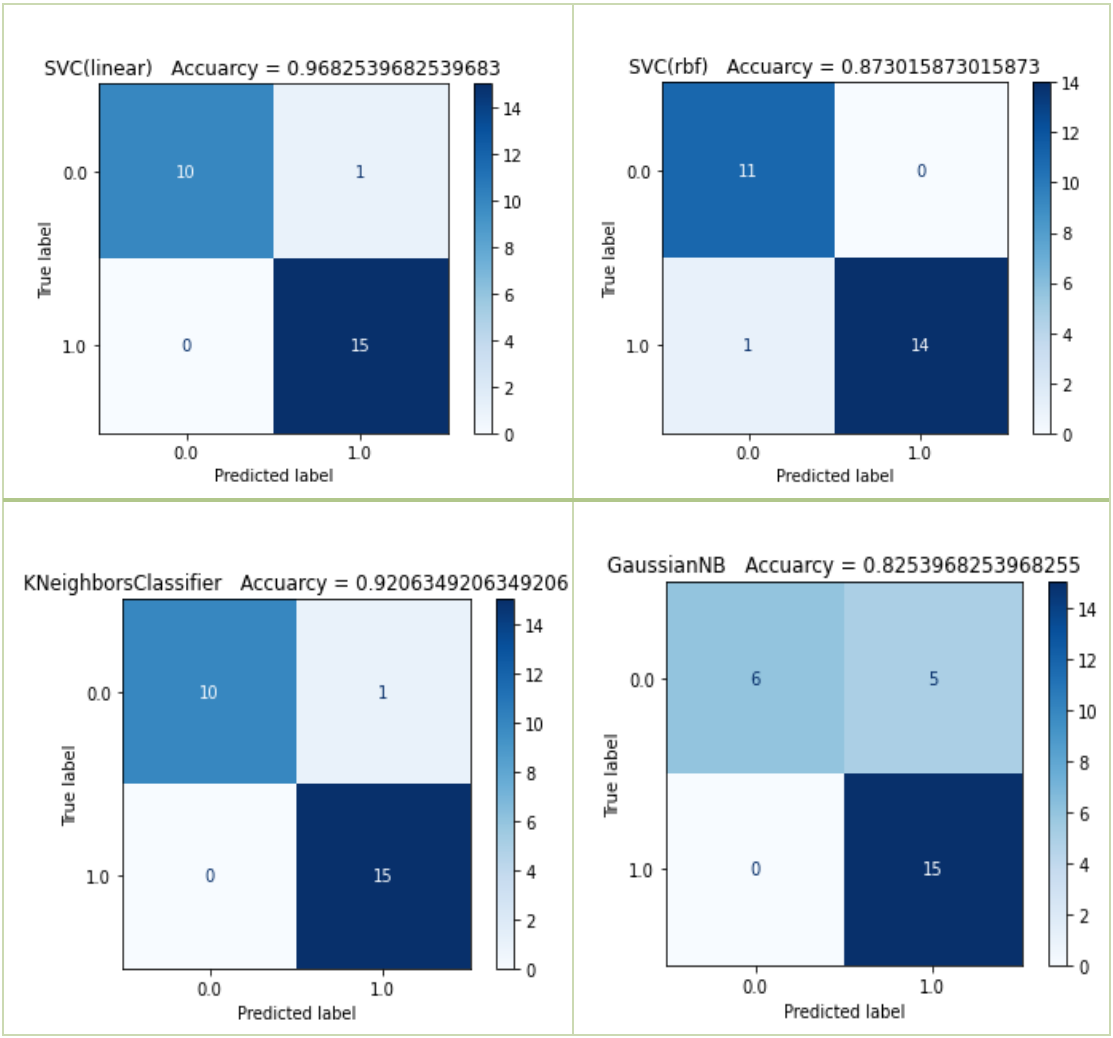
From the color of the graph we can see that there isn't much correlation between the variables. So, I have to keep all the 147 columns.

2.7 Modelling

I used SVM (with linear and RBF kernels), KNN and Bayesian classifiers.

I used k-fold cross-validation procedure with k-fold equal 3 for estimating the performance of the models.

The figure below shows the confusion matrix of the models.



The table below shows the final result of all models.

	accuracy	Precision	Recall_score	Specificity_list	True_pve Rate	False_pve Rate	F1_Score	AUC	Cohen's Kappa
SVC(linear)	0.968254	0.966667	96.969697	97.222222	96.969697	2.777778	0.966583	0.968013	1.191434
SVC(rbf)	0.873016	0.784632	100.000000	76.111111	100.000000	23.888889	0.879048	0.989562	1.333468
KNeighborsClassifier	0.920635	0.903030	93.265993	91.111111	93.265993	8.888889	0.917065	0.963777	1.442152
GaussianNB	0.825397	1.000000	61.952862	100.000000	61.952862	0.000000	0.756054	0.845118	0.241618

Conclusion:

- The table shows that SVM (with linear kernel) model achieves the highest accuracy followed by KNeighborsClassifier .
- The result shows that Cohen's Kappa almost the same in SVM and KNN.

2. Exercise 2

2.1 collect smali files

I collected smali files from Malware and Benign APKs folders created from Exercise1. I used OS package with (walk function) for find all smali files in specific subfolders. And, I used Shutil package to copy these files into proper folders (each smali files belong to one apk file in one folder).

2.2 Extract API calls features and Dataset creation

I extract API calls from all smali files that belong to specific APK files one by one. Finally, I create data frame of APKs with number of features ('startService','getDeviceId','createFromPdu','getClassLoader', 'getClass','getMethod','getDisplayOriginatingAddress','getInputStream' ,'getOutputStream','killProcess','getLine1Number','getSimSerialNumb er','getSubscriberId','getLastKnownLocation','isProviderEnabled').

Formatting

The data is formatted in following way

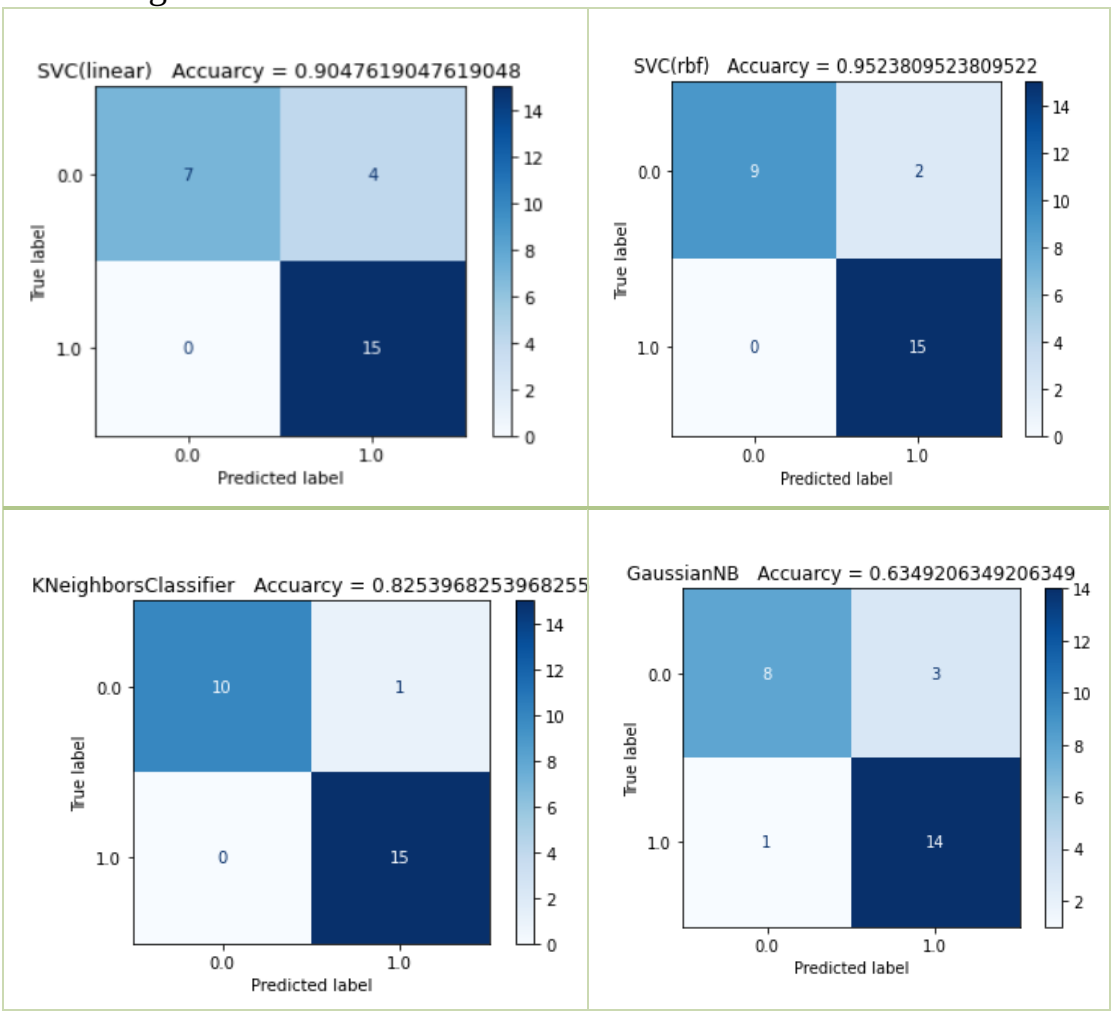
	startService	getDeviceId	createFromPdu	getClassLoader	getClass	getMethod	getDisplayOriginatingAddress	getInp
00357b0e208c20df3182d54cb2ba15bf	0	0	0	0	1	1	0	
02548535ff1cc285fddf699f2d77bcba	1	1	0	1	1	1	0	
0639a74f508591f99a7d2309f5825fea	0	0	0	0	1	1	0	
073a2f2d51c7dc00eb21e27cb8fa80f3	0	0	0	0	1	1	0	
08634da89ce3e70a81bdf128b998a89e	1	0	0	0	1	1	0	
...	...	...	...	...	...	...	...	
cmsecurity.applock.theme.valentineshearts	0	0	0	0	1	0	0	
cn.xiaochuankeji.tieba	1	1	0	1	1	1	0	
co.bitx.android.wallet	1	1	0	1	1	1	0	
co.gotitapp.android	1	1	0	1	1	1	0	
codematics.universal.tv.remote.control	1	1	0	1	1	1	0	

129 rows × 16 columns

The all dataset consists of 129 applications (70 Malware & 59 Benign)

2.3 Modelling

The figure below shows the confusion matrix of the models.



The table below shows the final result of all models.

	accuracy	Precision	Recall_score	Specificity_list	True_pve Rate	False_pve Rate	F1_Score	AUC
SVC(linear)	0.904762	0.962963	82.828283	97.222222	82.828283	2.777778	0.887963	0.961588
SVC(rbf)	0.952381	0.962963	93.265993	97.222222	93.265993	2.777778	0.947090	0.962318
KNeighborsClassifier	0.825397	0.814815	82.828283	83.333333	82.828283	16.666667	0.813228	0.936279
GaussianNB	0.634921	0.606527	86.531987	45.000000	86.531987	55.000000	0.700575	0.871128

Conclusion:

- The table shows that SVM model achieves the highest accuracy followed by KNeighborsClassifier .