Homework 1 - Malware Analysis in Android Operating System

Matteo Canavicci - 1548240

November 17, 2018

Contents

1 Introduction

Android is the most popular and common operating system for mobile devices, there are about 3.500.000 applications in the Play Store and many others in third-party markets. With the growth of the number of applications, there is also an increasing number of malicious softwares targeting mobile devices.
Each android application has an XML file called "Android Manifest" that provides to the operating system important informations like the first class to use when starting the app or the type of permissions used in the application. Only permissions provided in the file will be used in the application and only after asking to the user to allow some operations using those permissions. If the application tries to use some other permissions not allowed in the Android Manifest file, the execution fails.
Unfortunately many users tend to grant permissions to unknown applications and this is the main reason the device is infected by a malicious software.

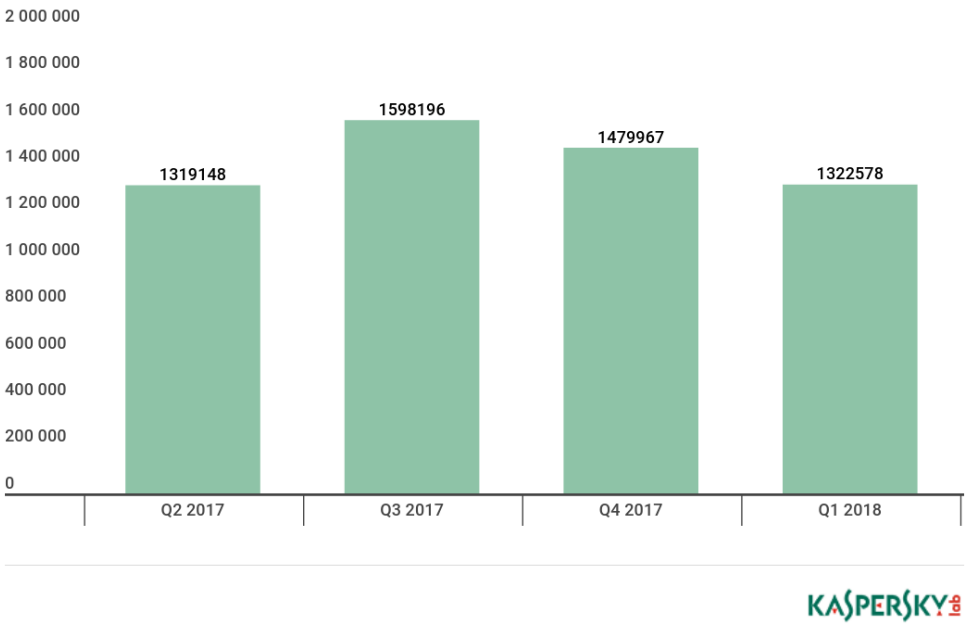

Figure 1: Number of detected malicious installation packages, Q2 2017 – Q1 2018 Reference

1.1 The problem

The first goal of the project is: given all the applications in the dataset, create a binary classifier that can detect if an application is a malware or a non-malware. Morever the malwares need to be classified correctly with the family they belong to.
For doing this it has been used two different approaches and two different kind of classifiers. First of all it has been used the Naive Bayes classifier using the Multinomial distribution, it has been evaluated the performances and then they have been compared with the performances obtained with the support vector machines (SVM).
The naive Bayes classifiers are based on the Bayes' theorem. Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. The naive Bayes approach is good in many use cases like text classification, email spam detection, face recognition, weather predictions and so on.
Like naive Bayes, the support vector machines are supervised learning models. A support vector machine constructs a hyperplane or a set of hyperplanes which can be used for classification.

1.2 Used tools

The programming language used for the project is Python. The main libraries and tools used are:

- Pandas data analysis library - Documentation
- NumPy the fundamental package for scientific computing with Python - Reference
- scikit-learn tools for data mining and data analysis - Documentation
  - sklearn Feature extraction - Documentation
  - sklearn metrics used to calculate evaluation metrics - Documentation
  - sklearn naive bayes and svm for classifiers - MultinomialNB Doc, SVM Doc
- Matplotlib for plotting - Documentation

1.3 DREBIN

Drebin is a lightweight method for detection of Android malware that performs static analysis gathering as many features (e.g. requested permissions, api calls, external urls) from an Android Manifest's application as possible.

1.3.1 The Dataset

The dataset is composed of 123,453 applications from different markets and 5,560 recent malware samples. Each sample text file contains the features of the manifest.xml file one per line with the following structure:

$$\langle \text{type of feature} \rangle :: \langle \text{value of the feature} \rangle$$

From the manifest.xml and from the disassembled code is possible to extract 8 types of features:

1. S1: Requested hardware components (GPS,camera, ...);
2. S2: Requested Permissions (Send sms, access to contacts,...);
3. S3: App components (Activities, Services, Content Providers, Broadcast receivers);
4. S4 Filtered Intents (Inter Process Communications handled by the sample e.g. BOOT_COMPLETED);
5. S5: Restricted API calls (API calls whose access require a permission)
6. S6: Used permissions (Permissions effectively used by the application)
7. S7: Suspicious API calls (API calls who allow access to sensitive data e.g. getDeviceId())
8. S8: Network addresses (Urls embedded in the code)

It has been provided also a dictionary file in csv format containing the SHA1 hash of the malwares and the family they belong to.

2 Data preparation

The dataset is composed of many files and it's difficult to access to all of them in a quick way. So it has been created a new file called "dataset.txt" where it has been inserted all the features of each application one per line. Each line has been labeled with the corrisponding family name if the application is a malware or the label "safeware" if it is a non-malware.The file has been built using the malware dictionary provided in the .csv file. It's possible to choose what features use for the analysis.
For the evaluation it has been used: the requested permisions (S2 in the constants.py file), the used permissions (S6 in the constants.py file) and suspicious api calls (S7 in the constants.py file). Each feature is splitted by a blank space.

2.1 Naive Bayes data preparation

The data contained in the dataset have been stored in a dataframe using the Pandas library. The Pandas dataframe constists of three main components : the data, the index and the columns.The dataframe has the following structure:

Index	Label	Features
0	GinMaster	android.permission.READ_PHONE_STATE ReadWrite ...
1	safeware	Read/Write External Storage android.permission.VIBRATE ...
2	Opfake	android.permission.ACCESS_FINE_LOCATION getDeviceId ...
..	....	.....
..	....	.....
N	safeware	getDeviceId HttpPost getSystemService ...

Since there is some rows where elements are missing, that rows has been removed using the dropna() function. Now for the detection of malware/non-malware the rows have been labeled with 1 for malwares and 0 for safewares. Unfortunately the dataset is unbalanced because after cleaning the empty values rows, we have 117409 non malware and only 5551 malwares. Using the data as they are I will obtain bad performances because if the classifier always predict the same class without performing any analysis of the features I will have an high accuracy but it's not realistic. Indeed the other metrics like precision and recall are lower.

The data has been balanced using the oversampling technique that consists in duplicate random records from the minority class (the malware class in our case), but it can cause overfitting.
The CountVectorizer converts our collection of data to a matrix of token counts. The count is the number of times each word appeared in the document.We can use the counts matrix as training set for the MultinomialNB classifier.
In the case of family classification I don't need the safeware data, so I take only the malware families that have more than 20 samples and in this case I don't need to balance the dataset.

2.2 Support Vector Machines data preparation

Same procedure for preparing data for support vector machines classifier except for the final step, instead of creating the word counts matrix, I made a one hot encoding of the features of each Android application. The one hot encoding is needed because SVM cannot operates on label data directly. It requires all input and output variables to be numeric. In the one hot encoding I've generated a boolean column for each feature in the dataset, in each row I can see if that sample has the feature(1) or not (0).The malwares have been labeled with '1' and non malwares with label '0' in the case of binary classification(malware/safeware). The one hot encoded data have the following structure:

Index	Label	Feature1	.....	FeatureN
0	0	0	..	1
1	1	0	..	0
..	..	..	..	..
..	..	..	..	..
N	0	1	..	1

3 Naive Bayes classifier

The Naive Bayes classifier is a probabilistic classifier which is based on applying the Bayes' theorem with the assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem says that given a class variable y and a feature vector $x_1, ..., x_n$:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

for all i, this relationship is simplified to:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

3.1 Multinomial Naive Bayes

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification.
The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y where n is the number of features and θ_{yi} is the probability $P(x_i|y)$ of feature i appearing in a sample belonging to class y.
The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

3.2 Evaluation with MultinomialNB

For evaluation it has been used Stratified K-Fold cross validation method.This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class. The dataset is partitioned in k disjoint sets $S_1, S_2, ..., S_k$ ($|S_i| > 30$) . For $i = 1$ to k kfold uses S_i as test set and the remaining data as training set Ti. Then it computes the $error_{s_i}$ for each hypothesis of the learning algorithm and return

$$error_{L,D} = \frac{1}{k} \sum_{i=1}^{k} error_{s_i}$$

The performance metrics considered are:

- $Accuracy = 1 - errorRate = \frac{t_p+t_n}{t_p+t_n+f_p+f_n}$,
- $Recall = \frac{t_p}{t_p+f_n}$,
- $Precision = \frac{t_p}{t_p+f_p}$
- $F1Score = \frac{2 \cdot (Precision \cdot Recall)}{Precision+Recall}$

Metrics with the unbalanced dataset:
$Accuracy = 0.95$
$Precision : 0.53$
$Recall : 0.73$
$F1 - score : 0.61$

Metrics with the balanced dataset:
$Accuracy = 0.92$
$Precision : 0.80$
$Recall : 0.83$
$F1 - score : 0.82$

As expected the Accuracy is higher in the test with the unbalanced dataset because the classifier usually predicts the majority class (safwares in our case) but the other metrics are very low. Balancing the dataset we can get better performances, in particular a lower accuracy but better precision, recall and F1-score.

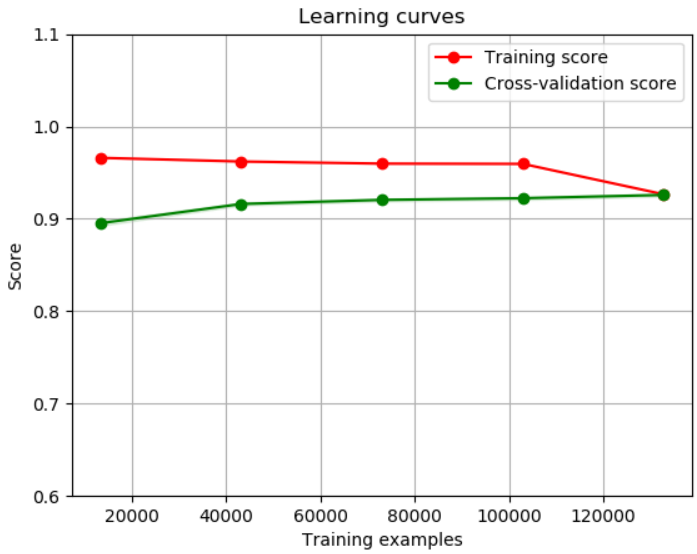

Figure 2: Learning curves of malware detection using MultinomialNB

Family classification performance metrics:
$Accuracy = 0.88$
$Precision : 0.85$
$Recall : 0.80$
$F1 - score : 0.81$

In this case we get good performances selecting malware families having more than 20 samples.

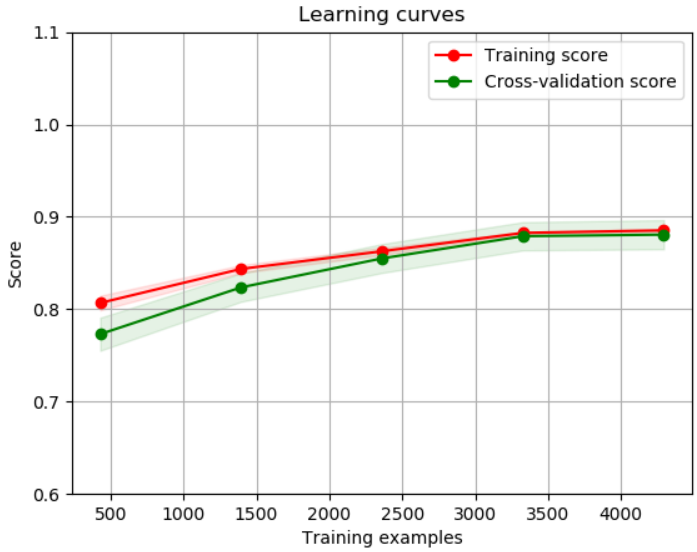

Figure 3: Learning curves of family classification using MultinomialNB

4 Support Vector Machines Classifier

Support vector machines is a supervised machine learning algorithm which can be used for classification. In the SVM we plot each data item in a n-dimensional space (n is the number of features we have) with the value of each feature being the value of the corresponding coordinates. Then it's possible to perform classification finding the hyperplane that differentiate the two classes. What's the best hyperplane to separate the two classes? A good choice could be the hyperplane thata maximizes the distances between the nearest data point and the hyperplane.

4.1 Linear Support Vector Classification (LSVC)

The python library used from Scikit-Learn is LinearSVC that it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples respect to standard SVC. The class has also the multiclass support needed for family classification.
Respect to MultinomialNB function, LinearSVC is slower in terms of running time but I got better performance metrics.

4.2 Evaluation with LinearSVC

As in the MultinomialNB it has been used the Stratified K-Fold for evaluation. The features sets used this time are 'S2' (requested permissions) and 'S7' (suspicious api calls), two instead of three sets rispect to MultinomialNB and the test has been done on half of the dataset sample picked randomly because LinearSVC is slow and memory expensive with large amount of data. I've considered the same performance metrics of MultinomialNB.
Metrics with unbalanced dataset:
$Accuracy = 0.97$
$Precision : 0.86$
$Recall : 0.64$
$F1 - score : 0.73$

Metrics with balanced dataset:
$Accuracy = 0.96$
$Precision : 0.90$
$Recall : 0.81$
$F1 - score : 0.86$

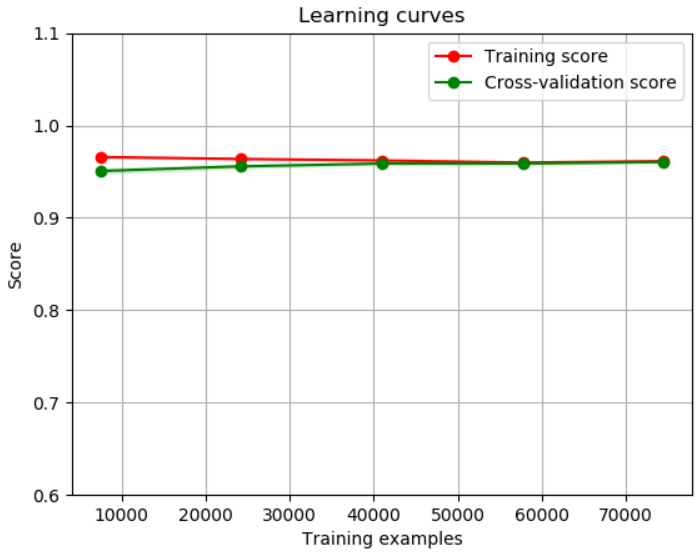

Figure 4: Learning curves of malware detection using LinearSVC

Family classification performance metrics:
$Accuracy = 0.95$
$Precision : 0.87$
$Recall : 0.86$
$F1 - score : 0.86$

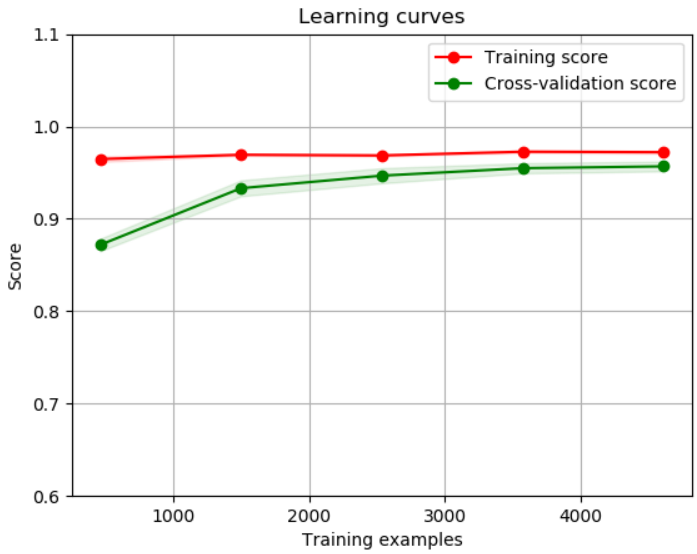

Figure 5: Learning curves of family classification using LinearSVC

5 Software Usage

```
$ python main.py arg1 arg2
arg1  = svm (LinearSVC classifier),
arg1 = bayes (MultinomialNB classifier),
arg2 = True for family classification, False for malware detection
```

6 Conclusion

We can finish saying that a Naive Bayes Classifier is good for text classification, it's fast, easy to implement and it allows to reach good performances classifying the instances. On the other hand the Support Vector Machines provides better performances when classifies the instances but it's very slow with a big dataset.

7 References

- Statistics on Android Malwares
- DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket
- Machine Learning for malware analysis
- Scikit-Learn Documentation
- Linear Classification slides
- Naive Bayes slides
- Evaluation slides