[5]

Android Malware Detection using Machine Learning Algorithms on the Drebin Dataset

HOMEWORK 1

Sayo Michael Makinwa | Machine Learning | 18.11.2018

Sapienza University of Rome

1.0 Introduction

A malware is generally a software that achieves deliberate malicious operations by disrupting proper functionality of the system it is running on or collects information without authorisation to send to the attacker.

Mobile devices have become so sophisticated over the years and they are now by the far the most widely used systems. It is now quite typical for payments to be made via smartphones, important email communications and a host of other things are increasingly preferably done with mobile phones, hence, it has become even more typical for mobile devices to be targeted with malwares.

There are quite a number of smartphone platforms, but Android is by far the most used because it is open source and free. This has also attracted developers to build a lot of useful applications for the Android platform and as such, a huge android community has been built over the years. This growth has also ensured made sure that a lot of malwares have been built on the android platform.

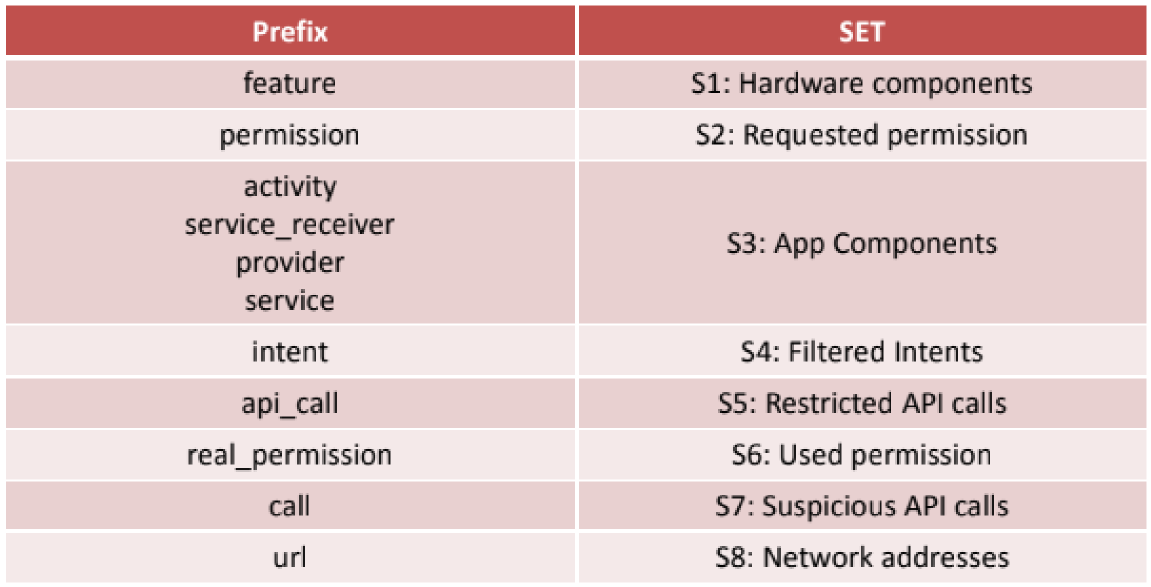
Over the years, researchers have made approaches to detect and prevent the operations of malwares, but it has really been a round robin as attackers have also found new ways of getting to their target users nonetheless. Of the most successful approaches that have been made over the years is the application of machine learning to the detection and classification of malwares.

This is a report about applying Support Vector Machine, Random Forest Classifier, and Naïve Bayes Classifier to the detection of malwares. The Drebin dataset containing details of 129,013 Android applications manifest files of which 5,560 are listed to be malwares was used. The algorithms had good predictions overall, with Random Forest Classifier having the best results.

2.0 Methodology

2.1 Drebin

The Drebin approach to malware detection and classification was to gather as many features as possible from application’s manifest and code, organize and embed these features in a joint vector space where each feature is grouped into sets. Then machine learning techniques are applied to identify patterns in these features. Typically, the features collected from each application have the properties: feature, permission, activity, service_receiver, provider, service, intent, api_call, real_permission, call, and url. These are further grouped into sets of hardware components, requested permission, app components, filtered intents, restricted API calls, used permission, suspicious API calls and network addresses.



Prefix	SET
feature	S1: Hardware components
permission	S2: Requested permission
activity service_receiver provider service	S3: App Components
intent	S4: Filtered Intents
api_call	S5: Restricted API calls
real_permission	S6: Used permission
call	S7: Suspicious API calls
url	S8: Network addresses

Fig. 2.1.1 Drebin Dataset Organisation [3]

The dataset gathered was worked on in this homework, using the set grouping as described in Fig. 2.1.1. The actual content of the features could not be used because they are so large, some of them having different values running into thousands, and not a significant number of them, relative to the total number of the dataset, are the same across other application files, and so, building a one hot encoder for this results and exponential growth in the size of the feature vectors to be used for the algorithms. Instead, a count was made for the number of occurrence of the feature properties in each feature set, and so, a feature vector of size eight was used, each feature having count values. So, each file has an input vector that looks like [2, 11, 5, 3, 7, 6, 11, 26], with the output being *TRUE* (is a malware) or *FALSE* (not a malware).

2.2 Support Vector Machines

The choice to use this algorithm was because it is special in the way it works, using maximum margins as support vectors used as decision boundaries in the data for its classification. These support vectors are at the boundaries of the classification as they are the least likely of their classes, and so, in a sense, the algorithm can discard the other data points after getting these support vectors and can classify new data points strictly based on these support vectors. Paying particular attention to the least likely elements of each class gives Support Vector Machines its edge over other algorithms as it enables it to perform better in classifying datasets with noise.

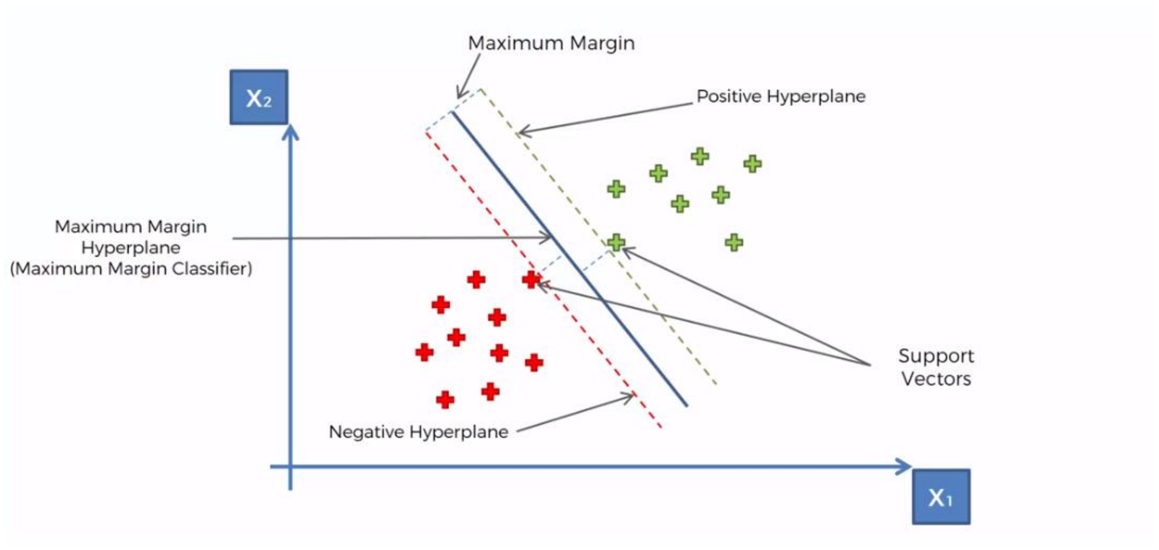


Fig. 2.2.1 SVM Intuition [7]

2.3 Random Forest Classifier

This is an ensemble learning method that combines many decision trees together. It recursively selects subsets of the datasets for which it builds different decision trees, each with its prediction. It leverages the power of decision trees and the power of numbers to eliminate errors each individual tree could have made in its chosen subset of data

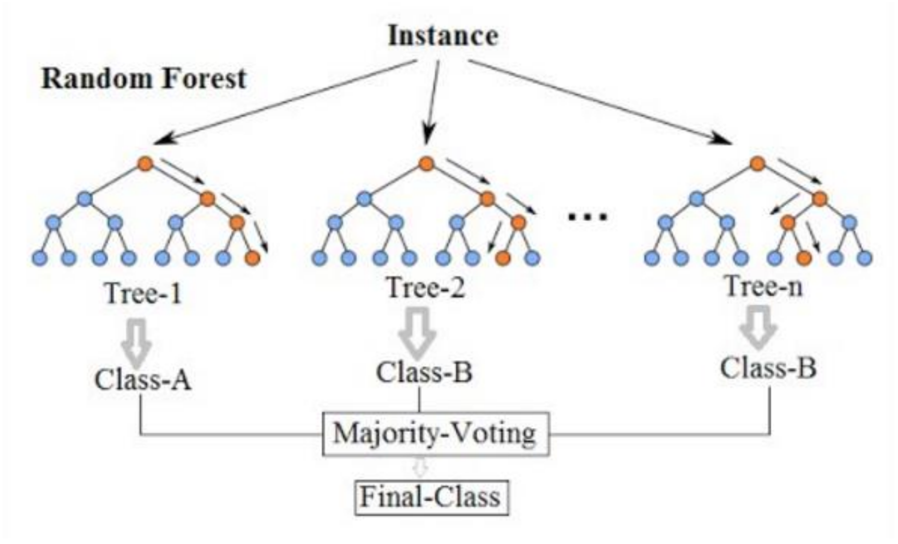


Fig. 2.3.1 Random Forest Intuition [6]

2.4 Naïve Bayes Classifier

Naïve Bayes Classifier is a supervised learning approach that takes advantage of Bayes' conditional probability theory. The algorithm calculates the marginal likelihood that a new data point belongs to a particular class based on its position and the other data points that are immediately close to it with respect to the others. It calculates this marginal likelihood for each class that exists in the problem domain and classifies it along with the class with which its probability is higher.

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

Fig. 2.4.1 Bayes' Conditional Probability

The algorithm usually starts with an independence assumption and that may sometimes lead to wrong classification if there is not enough data to correct the assumption.

3.0 Evaluation of Results

To replace the placeholder text on this page, you can just select it all and then start typing. But don't do that just yet!

First check out a few tips to help you quickly format your report. You might be amazed at how easy it is.

70% of the data was used as the training set and 30% used as testing set. The following metrics are based on the performance on the 30% testing set

3.1 The Dataset

As pointed out in the methodology, classes of feature sets were used for the prediction. In order to learn which sets are the most significant in predicting whether or not an application is a malware, the full dataset is separated, using the ground truth, into two classes: malware and not malware, then the means were calculated for each feature set. This analysis was done independent of the classification algorithms used for the prediction. This is important as algorithms like Polynomial Support Vector Machines do not work based on weighted features and hence, cannot provide feedback on which features are the most significant in the prediction.

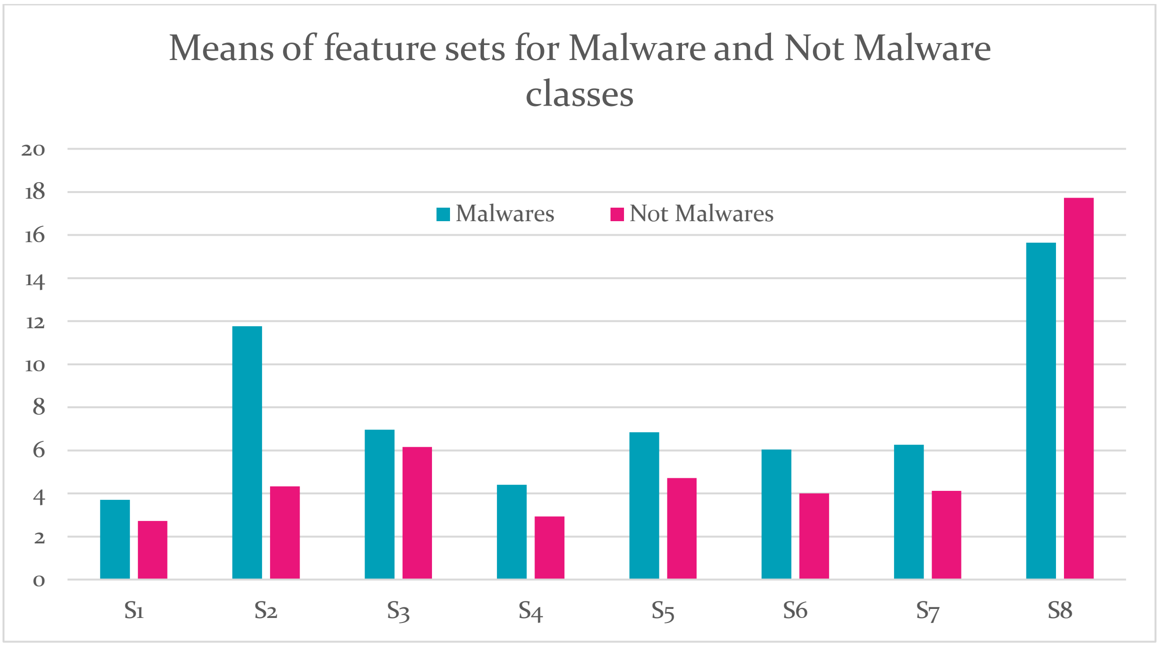


Fig. 3.1.1 Means of feature sets for Malware and Not Malware classes

The chart in Fig. 3.1.1 shows that it is in fact the S2 feature set (number of requested permissions) that should play the most important role in the prediction. This means that Malware applications typically request for a lot more permissions than usual. The Chart also shows that malwares are expected to make more suspicious API calls (set S7), use more permissions (set S6), make more restricted API calls (set S5), and so on. Another interesting thing is that applications that are not malwares averagely have more network addresses than malwares (set S8).

3.2 Support Vector Machine

The linear kernel of the Support Vector Machine was really terrible at picking out malwares and generalized that there are no malwares, having 100% false negatives. This is because there is a very small sample of malwares in the dataset (only about 3%!), and since linear models cannot fit well in this case, it could not learn what malwares look like. The polynomial kernel with degree of 3 performed much better (no better performances with higher degrees) and was able to learn a bit what malwares look like. Polynomial SVM performed wonderfully in predicting files that are not malwares (99.8%), but was very awful in picking out malwares for this dataset (16%). Another possible reason for the performance is the fact that actual contents of the features could not be used because then, the feature vectors would become too large to be reasonable.

			Learned Function Output	
			Positives (Malware)	**Negatives (Not Malware)**
			321	38,383
			True Positives (TP)	**False Negatives (FN)**
Ground Truth	**Positives (Malware)**	1,669	267 (16%)	1,402
			False Positives (FP)	**True Negatives (TN)**
	Negatives (Not Malware)	37,035	54	36,981 (99.8%)

Table 3.2.1 Analytical Performance of Polynomial SVM with degree of 3

Precision Score	Recall Score	F1 Score
0.96	0.16	0.27

Table 3.2.2 Performance of Polynomial SVM with degree of 3, generated with sklearn metrics

3.3 Random Forest Classifier

The Random Forest Classifier produced a fantastic result. It is no doubt the most suited for this particular dataset of all three algorithms. It produced 99.7% true negatives, 78.8% true positives (despite the skewed nature of the dataset) and had a precision score of almost 1. Training the classifier was also much faster than the SVM.

Precision Score	Recall Score	F1 Score
0.988	0.788	0.849

Table 3.2.4 Performance of RFC, generated with sklearn metrics

			Learned Function Output	
			Positives (Malware)	Negatives (Not Malware)
			1,437	37,267
			True Positives (TP)	False Negatives (FN)
Ground Truth	Positives (Malware)	1,677	1,322 (78.8%)	355
			False Positives (FP)	True Negatives (TN)
	Negatives (Not Malware)	37,027	115	36,912 (99.7%)

Table 3.2.3 Analytical Performance of RFC

3.4 Naïve Bayes

The Naïve Bayes Classifier turns out the worst of the three algorithms tried. One important thing to note is its over classification of files as malware which is in direct contrast to the tendencies of Support Vector Machine that tended more towards “Not Malware” classification. This poor detection of malware would also be more improved with more actual malware files in the dataset.

			Learned Function Output	
			Positives (Malware)	Negatives (Not Malware)
			3,179	35,525
			True Positives (TP)	False Negatives (FN)
Ground Truth	Positives (Malware)	1,661	823 (49.5%)	838
			False Positives (FP)	True Negatives (TN)
	Negatives (Not Malware)	37,043	2,356	34,687 (93.6%)

Table 3.2.5 Analytical Performance of Naïve Bayes Classifier

Precision Score	Recall Score	F1 Score
0.917	0.788	0.495

Table 3.2.6 Performance of Naïve Bayes Classifier, generated with sklearn metrics

4.0 Limitations

The major limitation is the fact that actual content of the features could not be used because they are so large, some of them having different values running into thousands, and not a significant number of them, relative to the total number of the dataset, are the same across other application files, and so, building a one hot encoder for this results and exponential growth in the size of the feature vectors to be used for the algorithms. This limited what was used to counts of properties in each feature set; the actual content would have given more information to the algorithm.

Another limitation was the amount of malware files present in the dataset, relative to the total; the number is so small that the algorithms could not adequately learn what a malware looks like, or perhaps they learned "too much" of what is not a malware.

5.0 Conclusion

The Drebin work shows clearly what a real world dataset can be like – inconsistent. This homework has shown that in cases like this, Random Forest Classifiers are perhaps the best performers. Support Vector Machine and Naïve Bayes Classifiers can also perform well when the classes in the datasets are a bit more balanced.

References

1. Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz y, Kamer Ali Yuksely, Seyit Ahmet Camtepe, and Sahin Albayrak: Static Analysis of Executables for Collaborative Malware Detection on Android. 2009
2. Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck: DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. 2014

3. Dott. Luca Massarelli: Machine Learning for Malware Analysis. A.Y. 2018-2019
4. Geethu M Purushothaman, G Gopinadh, Nihar SNG Sreepada: Malware Detection in Android. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 4, April 2014
5. https://blog.telium.com.br/ataque-cibernetico-wana-cry-saiba-como-se-prevenir/
6. https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d
7. https://www.udemy.com/machinelearning/