Machine Learning Homework 1

- Michal Ostyk-Narbutt 2018/2019
- Matricola: 1854051
- Some parts of the code were taken from my Bachelor thesis: github.com/Ostyk/BSc-thesis
- This homework itself is also available github.com/Ostyk/Drebin-malware

Out[9]:

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Introduction

This report is an approach to tackling the problem of detecting and classifying malware with the use of Machine Learning.

Data

I have used the Drebin dataset which is a publicaly available dataset depitcting Android Malware. It contains 123453 bening applications and 5560 malware. It uses static analysis to extract features from samples.

Details:

- Drebin: (Effective and Explainable Detection of Android Malware in Your Pocket)
- link to original paper https://www.tu-braunschweig.de/Medien-DB/sec/pubs/2014-ndss.pdf (https://www.tu-braunschweig.de/Medien-DB/sec/pubs/2014-ndss.pdf)

Feature engineering and dataset structure

The dataset is aranged into a folder with all the applications (130000+) in seperate text files. These text files depict the details each individual application by a series of categories. There are 8 categories: $S_1 \cdots S_8$

Examples of features one the files:

- feature::android.hardware.touchscreen
- call::getDeviceId
- call::printStackTrace
- permission::android.permission.READ_PHONE_STATE
- api_call::android/net/ConnectivityManager;->getActiveNetworkInfo
- permission::android.permission.INTERNET

In order to be able to train a Machine Learnign algorithm we need to extract the features. So for each file, I counted the number of occurences of every category. Hence the feature vector is of size 1×8, with each element depicting the number of occurences each category.

Moreover, for the purposes of this report, I discarded the results of training on an imbalanced dataset. Hence, all results are obtained on 5600 positive(Malware) and 5600 negative (Non-malware--benign) instances.

Machine Learning algorithms:

- Naive Bayes
- Support Vector Machines (SVM)
- Random Forests

Performance metrics:

- Accuracy
- F1 score (maximised for the when not using a balanced dataset)

I also utilised cross validation to get the most of the rather small dataset.

Tools

- Python version 3.6.3
- Scikit-learn for machine learing

Malware detection

The first problem was malware detection, were the goal was to classify a feature vector as being either

- malware
- non malware.

Bayesian approach

- The best Naive Bayes approach out was Gaussian Naive Bayes. I have also tried Multinomial, and Bernoulli version but gaussian had the best score. However, in comparision to other algorithms such as SVM and Random Forest it did not perform very well. Espcially since there are 314 False Negatives- Malware that went undetected

```
accuracy:   0.751
classification report:              precision    recall  f1-score   su
pport

        0.0       0.79      0.69      0.74      1673
        1.0       0.72      0.81      0.76      1663

avg / total       0.76      0.75      0.75      3336

confusion matrix
[[1157  516]
 [ 314 1349]]
```

SVM

Grid search pameters tested:

- C= [10, 10, 100, 1000] (param1)
- γ = [1e-4, 1e-3 , 1e-2] (param2)

Random Forest

Grid search pameters tested:

- Number of estimators = [100, 200, 300] (param1)
- Maximum depth of tree = [2,3,4] (param 2)

Malware detection results

- with the optimal hyperparamters

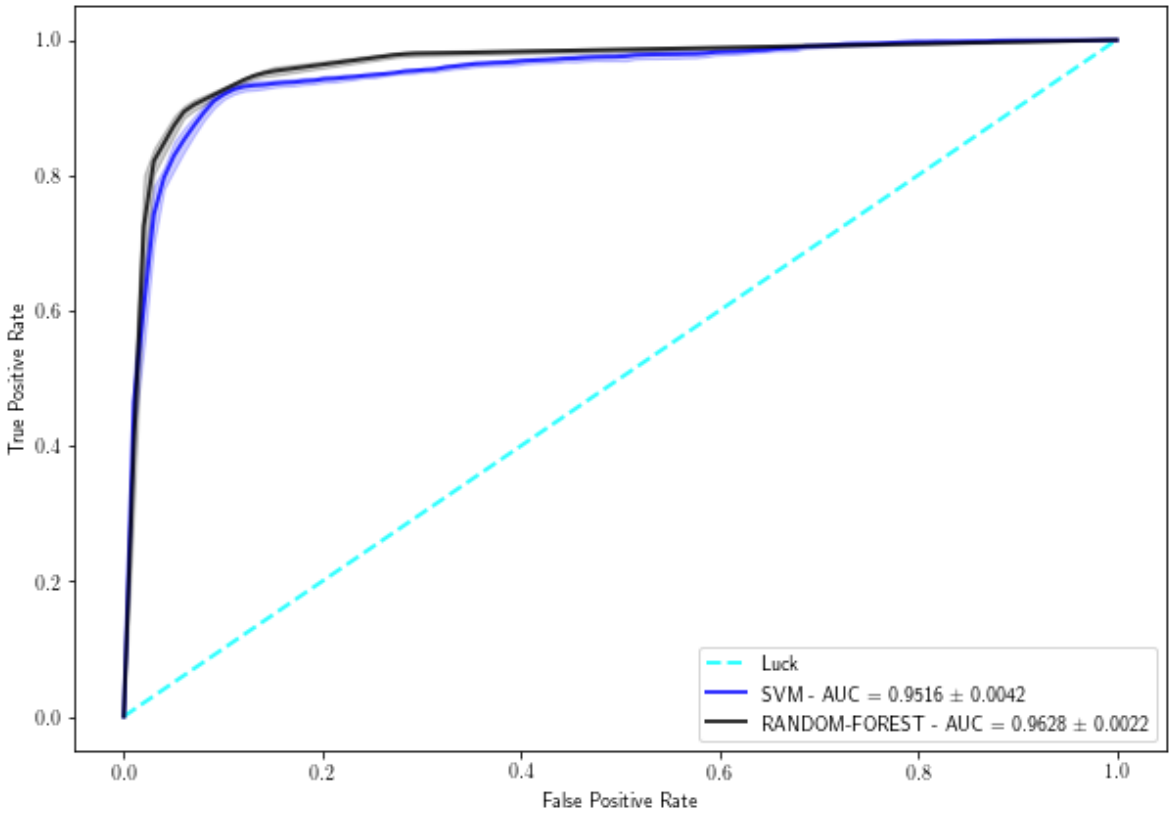
The 5 fold cross validation on a 6 hyperparemeter combinations show that SVM is significantly outperformed by Random Forest but is still better than Naive Bayes.

Out[19]:

	Model	accuracy	recall	precision	f1-score	param 1	param 2
0	SVM	0.724+/-0.029	0.724+/-0.029	0.814+/-0.011	0.70+/-0.04	0.01	10
0	RANDOM-FOREST	0.918+/-0.005	0.918+/-0.005	0.919+/-0.005	0.918+/-0.005	4.00	100

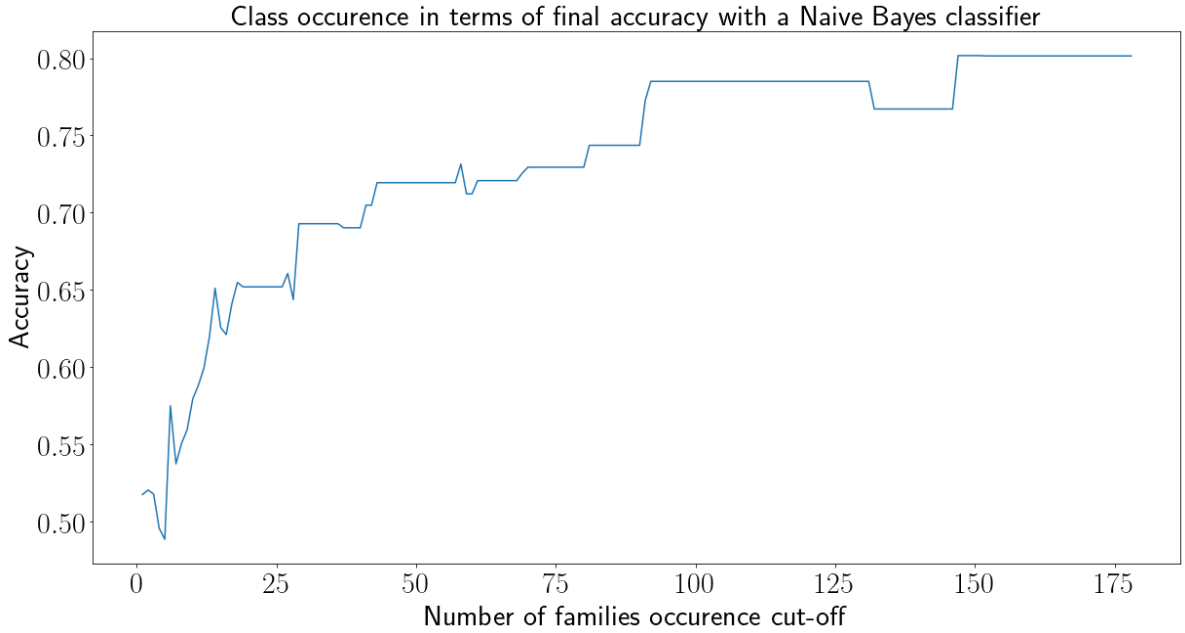
Receiver operating characteristic (ROC) and the area under the curve

- The uncertainties dervive cross validation folds. The AUC for the random forest is only slightly better than the SVM suggesting that the scaling of the data (prior to training) may not be optimal in this case.



Malware Classification

In this case I created a classifier whose goal was determine which malware family a certain feature vector belongs to. There are 179 families in the dataset but are very disproprtionate.Therefore, in the data preprocesing part, I tested where is the threshold to discard families few samples. Only Naive Bayes was trained on this data.



In this particular case since this set is heavily imbalanced in terms of family occurences in the dataset, the more classes we have the better the accuracy we have. Hence proving that proper classification of Malware families will require a lot of samples. The fewer the samples the better chance of incorrect classification.

Conclusion

- Malware detection is most succeful with Random Forest with accuracy $= 91.81 \pm 0.005\%$ and AUC $= 96.28 \pm 0.0022\%$
- SVM also provides a good score but both algorithms only do well with a balanced dataset
- However, in the case of Malware classification it is important to note that if we do not have enough samples of a particular class then the probabilty of missclassifying a sample of that class is very high. The more data the better. However, malware family classification is not as important as simple detection, espacially if its being done online.
- To conclude, Machine Learning is a very powerfull tool in this context of malware detection and classification. Some methods work better than others depending on the feature vectors and hyperparameters. Hence for an optimal solution, many more methods would need to be explored. For example bagging or boosting on SVM could be applied. In terms of pure performance metrics, most algorithms seem to work well in lowering the number of false negatives which are the most important in dealing with Malware.