Drebin

November 18, 2018

1 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

```
In [1]: from IPython.display import HTML
        HTML('''<script>
        code_show=true;
        function code_toggle() {
         if (code_show){
         $('div.input').hide();
         } else {
         $('div.input').show();
         }
         code_show = !code_show
        }
        $( document ).ready(code_toggle);
        </script>
        The raw code for this IPython notebook is by default hidden for easier reading.
        To toggle on/off the raw code, click <a href="javascript:code_toggle()">here</a>.''')
```

2 Introduction

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

2.1 Data

I have used the Drebin dataset. Drebin is a publicaly available dataset depitcting Android Malware. It contains 123453 bening applications and 5,560 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

2.2 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 easch 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 and 5600 negative instances.

2.2.1 Machine Learning algorithms:

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

2.2.2 Performance metrics:

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

2.2.3 Tools

- Python version 3.6.3
- Scikit-learn for machine learing

```
In [2]: import os
        import numpy as np
        import pandas as pd
        import random
        from matplotlib import rc
        import scipy.stats as st
        import matplotlib.pyplot as plt
        import time
        from uncertainties import unumpy
        import itertools
        #rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
        #rc('text', usetex=True)
        plt.rcParams['text.usetex']=True
        plt.rcParams['text.latex.unicode']=True
        params = {'legend.fontsize': 'x-large',
                   'figure.figsize': (15, 5),
                  'axes.labelsize': 'x-large',
                  'axes.titlesize':'x-large',
                 'xtick.labelsize':'x-large',
                 'ytick.labelsize':'x-large'}

        from sklearn.svm import SVC
        from sklearn.model_selection import train_test_split, GridSearchCV
        from sklearn.model_selection import StratifiedKFold,KFold
        from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
        from sklearn.metrics import classification_report, recall_score, f1_score
        from sklearn.metrics import roc_curve, auc
        from sklearn.preprocessing import StandardScaler

        #own files
        import reading as read
        import ML_models as ml

        import warnings
        #warnings.filterwarnings('always')  # "error", "ignore", "always", "default", "module"
```

---

3 Malware detection

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

- malware
- non malware.

```
In [3]: BALANCED = True
        a, b = 'IM', ''
        if BALANCED:
            a, b = '','_balanced'
        positive, negative = read.data_balance(negative=a+'BALANCED',
                                          ML_type = 'Detection',
                                          N_family_count = 'ALL',
                                          printing=False)

        pos = read.data_extractor(positive,'positive'+b,mypath='drebin/feature_vectors',load_da
        neg = read.data_extractor(negative,'negative'+b,mypath='drebin/feature_vectors',load_da

        #divide data
        X = np.concatenate((pos, neg), axis=0)
        y = np.hstack((np.ones(len(pos)),np.zeros(len(neg))))
```

4 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.

```
In [4]: #split into test
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

        #Normalize data by dividing by std
        X_train, X_test = ml.scale_set(X_train, X_test)

        from sklearn.naive_bayes import GaussianNB
        clf = GaussianNB()
        clf.fit(X_train, y_train)
        GaussianNB(priors=None)
        #print(clf.predict(X_test))

        pred = clf.predict(X_test) # obliczamy predykcj dla tekstów ze zbioru testowego
        accur = accuracy_score(y_test,pred) # dokladno
        print("accuracy:   %0.3f" % accur)
        print("classification report:",classification_report(y_test,pred)) # wypisz raport kla
        print("confusion matrix") # wypisz macierz (confusion matrix)
        print(confusion_matrix(y_test,pred))
        #print("---------------------------")
```

```
accuracy:   0.751
classification report:              precision    recall  f1-score   support

        0.0       0.72      0.81      0.76      1663
        1.0       0.79      0.69      0.74      1673

avg / total       0.76      0.75      0.75      3336

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

5 SVM

Grid search pameters tested: * C= [10, 10, 100, 1000] (param1) * γ = [1e-4, 1e-3 , 1e-2] (param2)

```
In [5]: SVM_1_performance, SVM_1_roc = ml.model(X, y,
                                                n_splits=5, seed=42,
                                                model_name='SVM', balanced=True)
```

6 Random Forest

Grid search pameters tested: * Number of estimators = [100, 200, 300] (param1) * Maximum depth of tree = [2,3,4] (param 2)

```
In [6]: RM_1_performance, RM_1_roc = ml.model(X, y,
                                              n_splits=5, seed=42,
                                              model_name='RANDOM-FOREST', balanced=True)
```

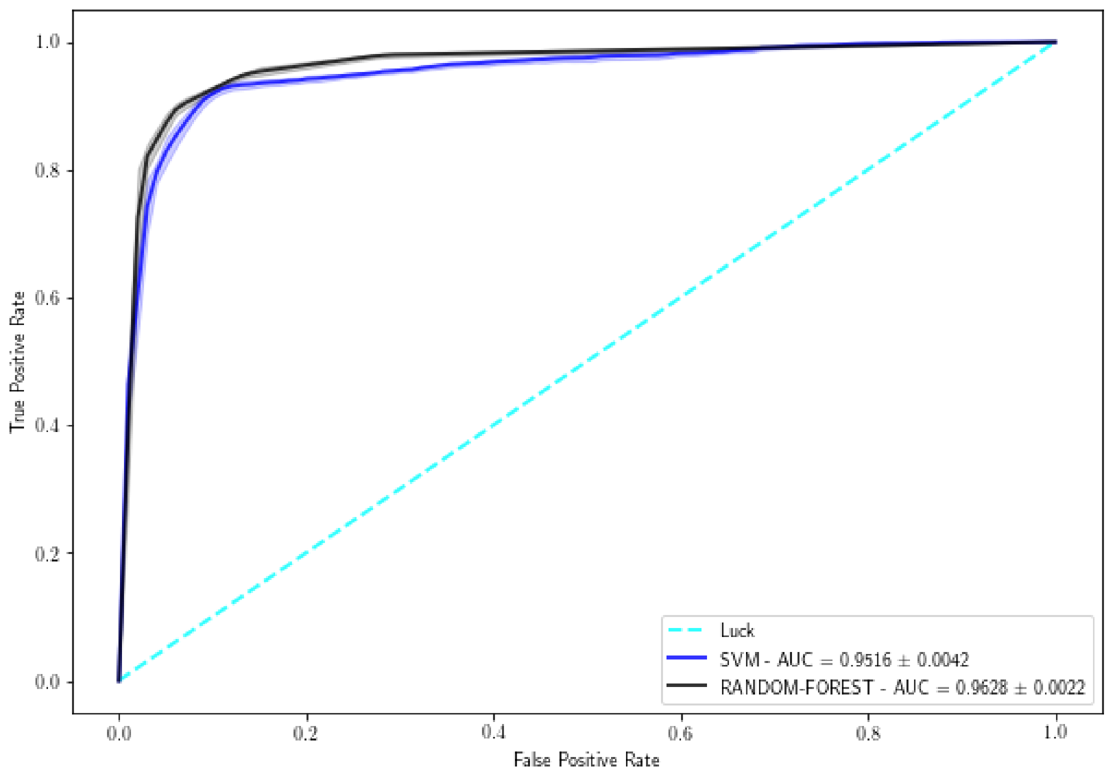
7 Malware detection results

```
In [7]: SVM_1_performance.append(RM_1_performance)
```

```
Out[7]:             Model       accuracy         recall      precision       f1-score  \
        0             SVM  0.724+/-0.029  0.724+/-0.029  0.814+/-0.011    0.70+/-0.04
        0   RANDOM-FOREST  0.918+/-0.005  0.918+/-0.005  0.919+/-0.005  0.918+/-0.005

           param 1  param 2
        0     0.01       10
        0     4.00      100
```

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

```
In [8]: ml.plot_roc([SVM_1_roc, RM_1_roc])
```



8 Malware Classification

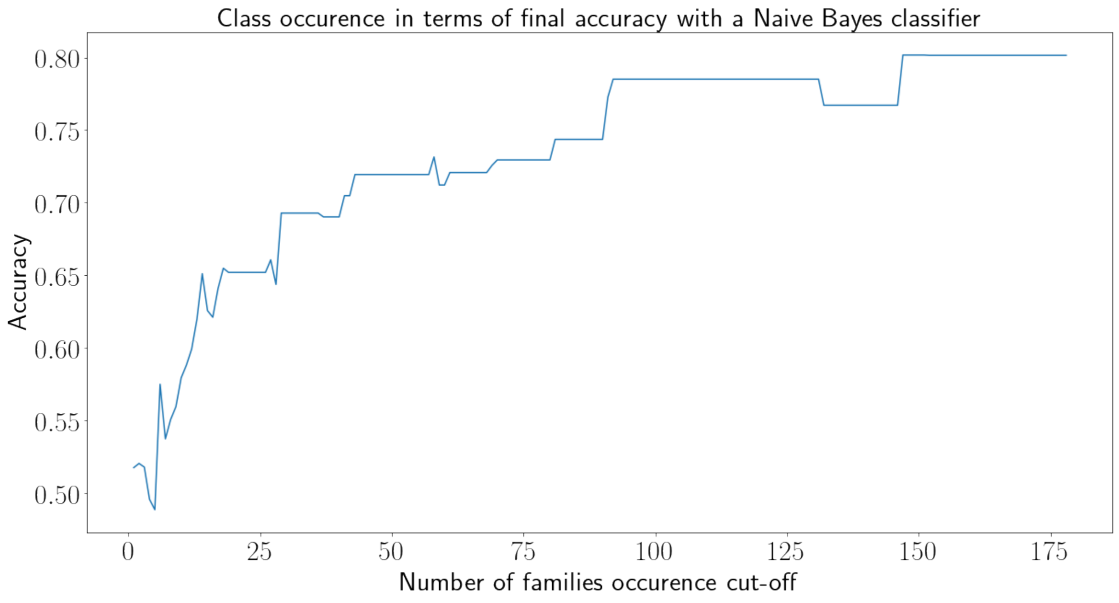
In this case I created a classifier whose goal was determine which malware family a certain feature vector belongs to. Therefore, in the data preprocesing part, I tested where is the threshold to discard families few samples. Only Naive Bayes was trained on this data.

```
In [9]: BALANCED = True
        a, b = 'IM', ''
        if BALANCED:
            a, b = '','_balanced'
        df = pd.read_csv('drebin/sha256_family.csv')
```

```
In [10]: count = df['family'].value_counts()

         xx,yy=[],[]
         for i in range(1,len(count)):

             exclude=i
             positive, negative = read.data_balance(negative=a+'BALANCED',
                                                  ML_type = 'Classification',
                                                  N_family_count = exclude,
                                                  printing=False)

             pos = read.data_extractor(positive,'positive'+b,mypath='drebin/feature_vectors',lo


             df2 = df.groupby("family").filter(lambda x: len(x) > exclude)
             diff = list(set(df.index.values)-set(df2.index.values))
             bad_df = df.index.isin(diff)
             X = pos[~bad_df]
             y=np.array(negative)

             #print("There are {} families with count greater or equalt to {}".format(len(np.u


                 #split into test
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

             #Normalize data by dividing by std
             X_train, X_test = ml.scale_set(X_train, X_test)

             from sklearn.naive_bayes import GaussianNB
             clf = GaussianNB()
             clf.fit(X_train, y_train)
             GaussianNB(priors=None)
             y_predict = clf.predict(X_test) # obliczamy predykcj dla tekstów ze zbioru testow
             accur = accuracy_score(y_test,y_predict) # dokladno
             #print("accuracy:   %0.3f" % accur)
```

```
        #print("classification report:",classification_report(y_test,pred)) # wypisz rapo
        #print("confusion matrix") # wypisz macierz (confusion matrix)
        #print(confusion_matrix(y_test,pred))
        xx.append(i)
        yy.append(accur)
```

```
In [11]: N=30
         plt.figure(figsize=(20,10))
         plt.plot(xx,yy)
         plt.xlabel("Number of families occurence cut-off",fontsize=N)
         plt.ylabel("Accuracy",fontsize=N)
         plt.title("Class occurence in terms of final accuracy with a Naive Bayes classifier",
         plt.xticks(fontsize=N)
         plt.yticks(fontsize=N)
         plt.show()
```



8.0.1 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.

---

9 Conclusion

- Malware detection is most succeful with Random Forests

- 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.