ML HA-1

by Ivan Senilov (1787618)

The goal of this HA is to create a Malware classifier having dataset of features of Android apps with corresponding class of malware (or absence of malware). The DREBIN (https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html) dataset is used for the purpose and needs to be put in `drebin` subfolder with `drebin/feature_vectors` containing files with features.

Unfortunately, local computer could handle only 2000 entries so for the training of the classifiers the Google Cloud VM with 8 cores and 32Gb (12Gb of which were occupied by the data) RAM was used.

Let's import libraries for working with data:

In [1]:

```
import pandas as pd
import numpy as np
```

Load the dictionary with correspondance between file name with features and family of the malware and remove malwares with less than 20 entries:

In [2]:

```
%%time
data = pd.read_csv("drebin/sha256_family.csv")
num = data["family"].value_counts() > 20

for i in num.index:
    if not num[i]:
        data = data[data.family != i]
data = data.reset_index(drop=True)
```

```
CPU times: user 155 ms, sys: 8.25 ms, total: 163 ms
Wall time: 245 ms
```

Let's extract our features populating a dictionary:

In [3]:

```
%%time
N = len(data)  # number of samples to train and test on

X_temp = []
y_temp = []
feature_types = ("api_call", "permission", "real_permission", "feature")  # feature typ
es we are interested in
type_to_vec = {el:[] for el in feature_types}
for i in range(N):
    with open("drebin/feature_vectors/" + data["sha256"][i], 'r') as file:
        lines = {el:[] for el in feature_types}
        for line in file:
            line = line.strip().split("::")
            if line[0] in feature_types:
                lines[line[0]].append(line[1])
                if line[1] not in type_to_vec:
                    type_to_vec[line[0]].append(line[1])
    X_temp.append(lines)
    y_temp.append(data["family"][i])
print(X_temp[1], "\n\n")
```

```
{'api_call': ['java/net/HttpURLConnection', 'android/telephony/TelephonyMa
nager;->getDeviceId', 'android/net/ConnectivityManager;->getActiveNetworkI
nfo', 'android/content/ContentResolver;->openFileDescriptor', 'android/web
kit/WebChromeClient;->onGeolocationPermissionsShowPrompt', 'android/conten
t/Context;->startActivity', 'java/lang/Runtime;->exec', 'android/net/wifi/
WifiManager;->getWifiState', 'android/net/wifi/WifiManager;->setWifiEnable
d', 'android/app/NotificationManager;->notify'], 'permission': ['android.p
ermission.READ_PHONE_STATE', 'android.permission.ACCESS_WIFI_STATE', 'andr
oid.permission.CHANGE_WIFI_STATE', 'android.permission.WRITE_EXTERNAL_STOR
AGE', 'android.permission.ACCESS_NETWORK_STATE', 'android.permission.INTER
NET', 'android.permission.RECEIVE_BOOT_COMPLETED'], 'real_permission': ['a
ndroid.permission.ACCESS_FINE_LOCATION', 'android.permission.READ_CONTACT
S', 'android.permission.ACCESS_WIFI_STATE', 'android.permission.READ_PHONE
_STATE', 'android.permission.VIBRATE', 'android.permission.CHANGE_WIFI_STA
TE', 'android.permission.ACCESS_NETWORK_STATE', 'android.permission.READ_L
OGS', 'android.permission.INTERNET'], 'feature': ['android.hardware.touchs
creen', 'android.hardware.wifi']}


CPU times: user 1.42 s, sys: 562 ms, total: 1.98 s
Wall time: 1min 50s
```

And make a categorical feature vector from them as described in [1]:

In [4]:

```
%%time
X = []

for i in range(len(X_temp)):
    n = np.zeros(1)
    for j in feature_types:
        m = np.zeros(len(type_to_vec[j]))
        for k in range(len(X_temp[i][j])):
            m[type_to_vec[j].index(X_temp[i][j][k])] = 1
        n = np.concatenate((n, m))
    X.append(n)

X = np.asarray(X)
print(X.shape, "\n\n")
```

```
(4785, 137648)


CPU times: user 2.73 s, sys: 6.51 s, total: 9.25 s
Wall time: 9.24 s
```

We have 4,785 sampples with 137,648-dim (!) feature vectors.

Labels are also transformed to categorical one-hot vectors

In [5]:

```
%%time
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
ohe = OneHotEncoder()
y = ohe.fit_transform(le.fit_transform(y_temp).reshape((len(y_temp),1)))
print(y.shape, "\n\n")
```

```
(4785, 24)


CPU times: user 260 ms, sys: 20.1 ms, total: 280 ms
Wall time: 4.88 s
```

Having 24 malware families, train SVM one-vs-all classifier on the data to have a baseline:

In [6]:

```
%%time
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
42)

clf = OneVsRestClassifier(SVC(kernel="linear", random_state=42))
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:", clf.score(X_test, y_test), "\n\n")
print(classification_report(y_test, y_pred, target_names=y_temp), "\n\n")
print(clf.get_params()["estimator"], "\n\n")
```

```
Accuracy: 0.948101265823
```

	precision	recall	f1-score	support
Plankton	1.00	0.87	0.93	30
DroidKungFu	0.96	0.90	0.93	114
Plankton	1.00	0.13	0.24	15
GinMaster	1.00	0.93	0.96	27
FakeDoc	0.99	0.99	0.99	206
GinMaster	0.94	0.76	0.84	21
FakeInstaller	1.00	0.98	0.99	46
Opfake	0.99	0.97	0.98	330
FakeInstaller	0.93	1.00	0.96	13
Opfake	0.78	0.95	0.86	19
BaseBridge	1.00	0.95	0.97	39
BaseBridge	0.97	0.97	0.97	115
Opfake	1.00	0.89	0.94	18
FakeInstaller	1.00	1.00	1.00	16
Adrd	1.00	0.96	0.98	52
Kmin	1.00	1.00	1.00	12
GinMaster	1.00	1.00	1.00	6
Adrd	1.00	0.98	0.99	43
Kmin	1.00	1.00	1.00	20
Geinimi	0.93	1.00	0.97	198
DroidDream	0.98	0.98	0.98	194
FakeInstaller	0.73	0.73	0.73	11
BaseBridge	0.90	1.00	0.95	18
Imlog	1.00	1.00	1.00	17
avg / total	0.97	0.96	0.96	1580

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)


CPU times: user 43min 17s, sys: 1.12 s, total: 43min 18s
Wall time: 43min 20s

/home/isenilov/miniconda3/lib/python3.6/site-packages/sklearn/metrics/clas
sification.py:1428: UserWarning: labels size, 24, does not match size of t
arget_names, 4785
  .format(len(labels), len(target_names))
```

On all 4,785 samples we have 94.8% accuracy while training with default hyperparameters (printed below the stats).

Let's try random forest classifier

In [7]:

```
%%time
from sklearn.ensemble import RandomForestClassifier

clf2 = OneVsRestClassifier(RandomForestClassifier(random_state=42))
clf2.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:", clf2.score(X_test, y_test), "\n\n")
print(classification_report(y_test, y_pred, target_names=y_temp), "\n\n")
print(clf2.get_params()["estimator"], "\n\n")
```

```
Accuracy: 0.924683544304
```

	precision	recall	f1-score	support
Plankton	1.00	0.87	0.93	30
DroidKungFu	0.96	0.90	0.93	114
Plankton	1.00	0.13	0.24	15
GinMaster	1.00	0.93	0.96	27
FakeDoc	0.99	0.99	0.99	206
GinMaster	0.94	0.76	0.84	21
FakeInstaller	1.00	0.98	0.99	46
Opfake	0.99	0.97	0.98	330
FakeInstaller	0.93	1.00	0.96	13
Opfake	0.78	0.95	0.86	19
BaseBridge	1.00	0.95	0.97	39
BaseBridge	0.97	0.97	0.97	115
Opfake	1.00	0.89	0.94	18
FakeInstaller	1.00	1.00	1.00	16
Adrd	1.00	0.96	0.98	52
Kmin	1.00	1.00	1.00	12
GinMaster	1.00	1.00	1.00	6
Adrd	1.00	0.98	0.99	43
Kmin	1.00	1.00	1.00	20
Geinimi	0.93	1.00	0.97	198
DroidDream	0.98	0.98	0.98	194
FakeInstaller	0.73	0.73	0.73	11
BaseBridge	0.90	1.00	0.95	18
Imlog	1.00	1.00	1.00	17
avg / total	0.97	0.96	0.96	1580

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

```
CPU times: user 12min 42s, sys: 30.8 s, total: 13min 13s
Wall time: 13min 14s
```

```
/home/isenilov/miniconda3/lib/python3.6/site-packages/sklearn/metrics/clas
sification.py:1428: UserWarning: labels size, 24, does not match size of t
arget_names, 4785
  .format(len(labels), len(target_names))
```

Random Forest of 10 trees yields 92.5%.

Making the labels matrices dense in order to make Keras work with them.

In [8]:

```
y_train = y_train.todense()
y_test = y_test.todense()
```

The third option is deep neural network:

In [9]:

```
%%time
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras.utils import to_categorical


inputs = Input(shape=(X_train.shape[1],))
x = Dense(64, activation='relu')(inputs)
#x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
#x = Dropout(0.3)(x)
predictions = Dense(y_train.shape[1], activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()

model.fit(X_train, y_train, epochs=100, batch_size=128, verbose=0)
y_pred = to_categorical(np.argmax(model.predict(X_test),axis=-1))
print("Accuracy", model.evaluate(X_test, y_test, batch_size=128, verbose=0)[1], "\n\n")
print(classification_report(y_test, y_pred, target_names=y_temp), "\n\n")
```

```
Using TensorFlow backend.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 137648)            0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8809536
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 24)                1560
=================================================================
Total params: 8,815,256
Trainable params: 8,815,256
Non-trainable params: 0
_________________________________________________________________
Accuracy 0.967721518535


                   precision    recall  f1-score   support

          Plankton       1.00      0.97      0.98        30
       DroidKungFu       0.94      0.94      0.94       114
          Plankton       1.00      0.13      0.24        15
         GinMaster       0.89      0.93      0.91        27
           FakeDoc       0.98      0.99      0.98       206
         GinMaster       1.00      0.81      0.89        21
     FakeInstaller       1.00      0.98      0.99        46
            Opfake       0.98      0.98      0.98       330
     FakeInstaller       1.00      1.00      1.00        13
            Opfake       0.78      0.95      0.86        19
        BaseBridge       1.00      0.95      0.97        39
        BaseBridge       0.99      0.98      0.99       115
            Opfake       0.94      0.89      0.91        18
     FakeInstaller       1.00      1.00      1.00        16
              Adrd       0.98      0.98      0.98        52
              Kmin       1.00      1.00      1.00        12
         GinMaster       1.00      1.00      1.00         6
              Adrd       1.00      0.98      0.99        43
              Kmin       1.00      1.00      1.00        20
           Geinimi       0.94      0.99      0.97       198
        DroidDream       0.99      1.00      1.00       194
     FakeInstaller       0.73      0.73      0.73        11
        BaseBridge       0.95      1.00      0.97        18
             Imlog       0.85      1.00      0.92        17

       avg / total       0.97      0.97      0.96      1580



CPU times: user 33min 22s, sys: 10min 28s, total: 43min 50s
Wall time: 10min 25s

/home/isenilov/miniconda3/lib/python3.6/site-packages/sklearn/metrics/clas
sification.py:1428: UserWarning: labels size, 24, does not match size of t
arget_names, 4785
  .format(len(labels), len(target_names))
```

Simple Neural Network with 3 layers gives us 96.8% accuracy, surpassing other classififcators we examined in this work so far.

Concluding our experiments, we showed how to extract and preprocess features from the dataset and tested three classifiers on them. The best results were showed by the Neural Network. However, there may be place for improvement, for example by applying Convolutional Neural Network or experimenting with deeper architectures.

References:

1. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014, February). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS.
2. Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., & Hoffmann, J. (2013, March). Mobile-sandbox: having a deeper look into android applications. In Proceedings of the 28th Annual ACM Symposium on Applied Computing (pp. 1808-1815). ACM.