Homework I

Francesco Terenzi, 1702217

Fall 2018

Contents

1 Introduction

In this report it's described the solution implemented by the student (ID: 1702217) to solve the homework request. The programming language used for this project is Python, for its pseudocode-*like* syntax and for the number of useful libraries for Machine Learning: scikit-learn, numpy, pickle, tqdm and other are the main. Their functionality require an installation (via `pip` command or `conda` command if you use the Anaconda framework).
Each section describes a different file present in the working directory: in `dataset.py` are given tools for modelling the *drebin* dataset, in the `malware_classifier.py` is described the classifier and with `main.py` is illustrated a simply test. All the code described in this paper is available on Github platform: `https://github.com/fratere/Malware-Analysis`.

2 Dataset

The drebin dataset contains about 130.000 applications. 5560 of them are malicious samples (as reported in a useful csv file) and so, testing the whole dataset requires a huge memory space and a long execution time. Then, the drebin dataset is modified and "filtered" in accordance to the type of test you want to accomplish. In the `main.py` file, reported in Section 4, this idea will be clearer.
In this chapter are described the most important functions in `dataset.py` file.

2.1 Prefixes

```
def get_prefixes():
    all_prefixes = ['real_permission', 'feature',
        'api_call', 'call', 'permission', 'provider',
        'activity', 'intent', 'service_receiver',
        'activity', 'url']
    return all_prefixes[:6]
```

Every feature in a file is composed by a [*prefix*]::[*rest of the feature*] syntax. The `get_prefixes()` function returns all the prefixes categories, ordered following the non-decreasing number of frequency in the drebin applications, as shown in the figure below. How you can see, the last 5 categories are very frequently, so, probably they are insignificant in a malware analysis because they can represent features common both malicious and not malicious files: they are omitted from the output list. There is, obviously, a gain, computationally speaking.

Listing 1: Number of features ordered by prefixes

```
sizes: {'real_permission': 70,
        'feature': 72,
        'api_call': 315,
        'call': 733,
        'permission': 3812,
        'provider': 4513,
        'intent': 6379,
        'service_receiver': 33222
        'activity': 185729,
```

```
        'url': 310488,
}
```

2.2 Encoded Features

```
72 def get_encoded_features(prefixes):
73     files_features = pref_clean(get_files_features(),
           prefixes)
74     features = get_features(files_features)
75     print("number of features: %d " %  len(features))
76     res = {}
77     for file_name in tqdm(files_features, desc='creating
           encoded features file'):
78         file_features = files_features[file_name]
79         l = [1 if f in file_features else 0 for f in
               features ]
80         res[file_name] = l
81     return res
```

Every file a contains a set of features, that are strings. Our classifier, however, require a numerical representation so, a transformation is needed. In the line 73 all files with the respective features are taken and filtered from those features which prefix is not in the `prefixes` input list. With the cycle of line 77 a one-hot encoding operation is done: for every file is created an array with '0' values in those positions that represent features not present in the file, '1' values otherwise. Obviously, the length of the array depends on the number of the filtered features and represent the function cost input in term of space complexity.

2.3 Load Drebin

```
def load_drebin(prefixes):
    if os.path.exists('drebin.pickle'):
        with  open('drebin.pickle', 'rb') as all_data:
            data = pickle.load(all_data)
            return data[0].todense(), data[1]
    else:
        files = get_encoded_features(prefixes)
```

```
90        X = [v for v in files.values()]
91        hashes = malware_hashes()
92        y = []
93        for f in files:
94          if f in hashes:
95            y.append(1)
96          else:
97            y.append(0)
98        print('creating sparse matrix...')
99        X = csr_matrix(X)
100       print('done')
101       """
102       with open('drebin.pickle', 'wb') as all_data:
103           print('saving data in drabin.pickle...')
104           pickle.dump([X, y], all_data)
105           print('done')
106       """
107       return X, y
```

The `load_drebin` function returns the encoded drebin dataset, represented by 'X', a matrix and 'y', a vector. The 'X' represents the encoded files given by the `get_encoded_features()`, explained in the Section 2.2, and the 'y' labels a malicious file with 1 and a non-malicious file with 0. An important observation: the X matrix is a sparse matrix, because it's requires less space in memory than an array-*like* matrix. Anyway, the classifier accepts both types of matrix.
In the lines 102-105 thanks to the `pickle` library methods it's possible to save the dataset on disk (in a pickle file) to avoid repeating the encoding operation when the `load_drebin` is called subsequently. However, in the test reported in the `main.py` file of Section 4, different versions of the dataset are needed and so, in this context, the pickle operation is insignificant (and the 102-105 lines are commented).

3 Classifier

The `malware_classifier.py` is a Python Class which methods, following the Machine Learning libraries syntax, offer useful kits for a correct binary classification of a file (i.e. malicious or non-malicious).

```
classifiers = {
    'svm'       : SVC(kernel='linear'),
    'random'    : RandomForestClassifier(n_estimators=100),
    'multinomial' : MultinomialNB()
}
```

The three classifiers are taken from the scikit-learn library and are reported in a dictionary. As we'll see in the next section, the final user can select one of these simply instantiating a new `MalwareClassifier` object using 'svm', 'random' or 'multinomial' as parameter.

```
def __init__(self, classifier):
    self.classifier = classifiers[classifier]
```

3.1 fit and predict

The `fit` function simply trains the selected classifier according to the [X,y] input, the `predict` function return a predicted vector of labels given a test matrix X.

```
def fit(self, X, y):
    self.classifier.fit(X, y)
```

```
def predict(self, X):
    return self.classifier.predict(X)
```

3.2 scores

In the `score` method is returned a dictionary with three different evaluation metrics: the detection score (i.e. the accuracy score), the precision score and the false positive rate (given by the confusion matrix here not reported) calculated in the private auxiliary file `__false_positive_rate()`.

```
def scores(self, y_pred, y_true):
    d = {}
    d['detection_score'] = accuracy_score(y_true, y_pred)
    d['precision_score'] = precision_score(y_true, y_pred)
    d['false_positive_rate'] =
       self.__false_positive_rate(y_true, y_pred)
    return d
```

3.3 print_results

This method simply define provide a useful print function to standard output of the scores calculate in the previous function. For a better accuracy, the numbers are represented with the first three digits before the comma.

```
def print_results(self, all_scores):
    print("\n"+30*"-"+"\nResults:")
    for s in all_scores:
        print(s + ': %0.3f' % all_scores[s])
    print(30*"-"+"\n\n")
```

4 Main

Listing 2: `main.py` file

```
1 import numpy as np
2 import random
3 from malware_classifier import MalwareClassifier
4 from sklearn.model_selection import train_test_split
5 from dataset import load_drebin, get_prefixes
6
7 prefixes = get_prefixes()
8 random.shuffle(prefixes)
9 chunks = [prefixes[x:x+3] for x in  range(0,
    len(prefixes), 3)]
10
11 for p in chunks:
12     print("training with %s" % p)
13
14     X, y = load_drebin(p)
15
16     X_train, X_test, y_train, y_test = train_test_split(X,
        y, test_size=0.33)
17     md = MalwareClassifier('random')
18
19     print('training...')
20     md.fit(X_train, y_train)
21
22     print('evaluating...')
23     y_pred = md.predict(X_test)
24
25     res = md.scores(y_pred, y_test)
26     md.print_results(res)
```

That's the complete code of the main file of out test. It can be executed from terminal with the `python3 main.py` command.
In the line 7 the prefixes are taken and shuffled with the `random.shuffle` method to ensure a better impartiality of the final result. In fact, as already explained in 2.1, the output list of the `get_prefixes()` function is sorted is according to non decreasing number of the prefixes. In line 9 the prefixes list is separated in chunks (sub-lists with three elements each).

The core of the `main.py` file is represented by the cycle starting at line 11: for each sub-list the drebin-adapted dataset is given and used as the input for the Random-Forest classifier; every predicted label is compared with the real label using the `md.scores` method. The final results are reported in Section 5.

5 Results

Listing 3: `results.log` file

```
training with ['feature', 'api_call', 'provider']
4900 features
training...
evaluating...

------------------------------
Results:
detection_score: 0.989
precision_score: 0.930
false_positive_rate: 0.003
------------------------------


training with ['call', 'real_permission', 'permission']
4615 features
training...
evaluating...

------------------------------
Results:
detection_score: 0.993
precision_score: 0.974
false_positive_rate: 0.001
------------------------------
```

In this section are reported the results of our test. With the command `python3 main.py > results.log`, a new output file is created in the working directory. Using the Random Forest classifier and the selected prefixes as input, good result are obtained in an acceptable time.
It's very important to underline the test is executed on a PC with i7 processor and 8 GB memory and that in the worst case the program need a memory capacity higher than 6 GB. The random forest emerges as the fastest among the available methods of the `malware_classifier.py` class.