Assignment 3
Malware classification with API call traces

T-710-MLCS, Machine Learning in Cyber Security

Reykjavik University - School of Computer Science, Menntavegi 1, IS-101 Reykjavík, Iceland

Damiano Pasquini
damiano23@ru.is

3. Octobrer 2023

Contents

1 Introduction

This third assignment asked to do the best preprocessing steps for the given dataset containing *API call traces*, then to execute some classifiers and to compare them in order to find the best one that is able to classify this kind of data. In this case, the *API call traces* were classified using the following classifiers as requested in the requirements:

- Multinomial Naive Bayes Classifier;
- Key Nearest Neighbors Classifier;
- Logistic Regression Classifier;
- Decision Tree Classifier;
- Random Forest Classifier.

Then, two more classifiers as additional experiments were trained:

- Support Vector Machine Classifier
- Gradient Boosting Classifier

2 Data Pre-Processing and Splitting

Initially, the API traces dataset is preprocessed with different steps. First of all the duplicated lines were dropped, the infinite values (in case they are present) were replaced with the maximum value of the *float32* type and the *NaN* values were replaced with 0. The dataset is then split into training and testing, respectively in 70% and 30%.
Then with the Count Vectorizer is obtained the sparse matrix, is needed to create a matrix containing rows of the same length since in the initial dataset all the rows contained a different number of features. The dataset is also scaled but without any improvement in the results.
Lastly, in the preprocessing step are saved the vectorizer and the scaler as **joblib** files since they are used again in the file *classify-trace.py*[1] to vectorize and scale the input file basing on the previously trained classifier.

3 Training and Testing

Once the data was preprocessed as said before, seven different classifiers were trained. About these classifiers listed in the introduction, one in particular performed better than the others and it is the *Random Forest Classifier* that gained a **F1 Score** of 0.61 while the worst one was the Logistic Regression Classifier with a F1 score of 0.29, probably given by the non-convergence.
About this last classifier, trying to solve the non-convergence different tests are done using different maximum number of iterations without any positive result.

	NB	**KNN**	**LR**	**DT**	**RF**	**SV**	**GB**
F1-score	0.3039	0.4139	0.2939	0.5052	**0.6107**	0.3070	0.5727

Table 1: F1 scores of the trained classifiers.

[1]The code is available through this Github Repository

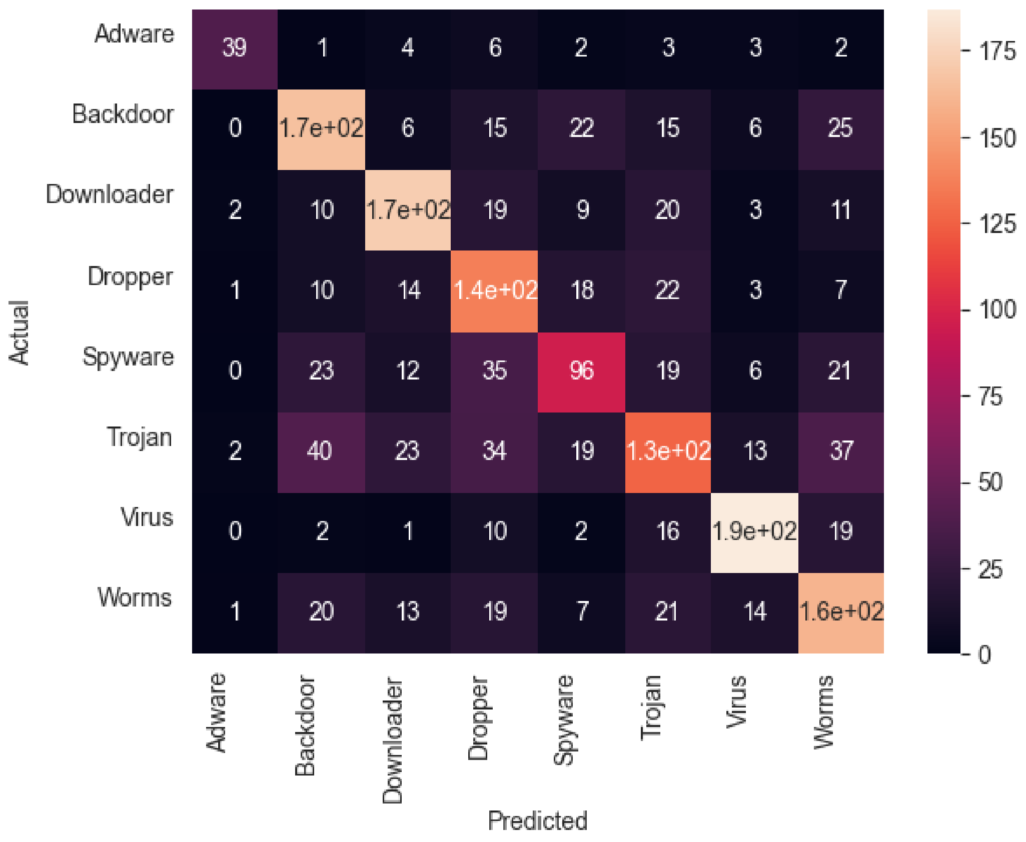

Figure 1: Heatmap of the Random Forest Classifier.

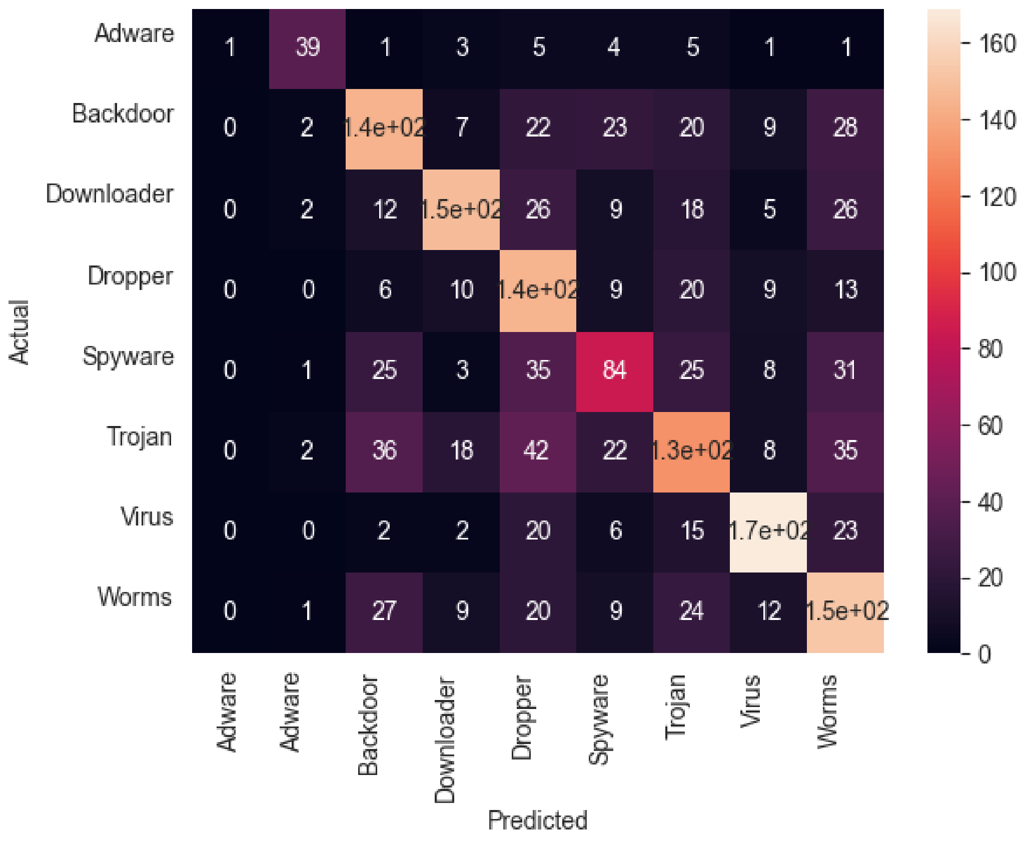

Figure 2: Heatmap of the Gradient Boosting Classifier.

These two classifiers are the two that performed better but not as well as expected since the F1 scores are not so high, even if these are good results since the diagonal on the heatmaps is distinguishable in both models.

4 Exporting the model

A second task asked to write a Python file that takes in input a dataset as the one given for the previous task, using the most performing model to predict the labels of this last dataset. In order to do this, the best model (Random Forest Classifier), the count vectorizer and the encoder were exported in the Jupyter file previously. In this second file named *classify-trace.py* the basic preprocessing steps are then repeated and the output is printed through the console.

Listing 1: Python function to import the model and classify each line singularly.

```
def main():
    if len(sys.argv) != 2:
        print("Usage: python classify-trace.py <filename>")
        sys.exit(1)
    clf = load('classifier.joblib')
    df = read_csv(sys.argv[1])
    df = preprocess(df)
    for line in df:
        label = clf.predict(line.reshape(1, -1))
        label = (str(label).replace('[', '')
                  .replace(']', '')
                  .replace('\'', '')
                  .replace('\\n', ''))
        print(label)
```

5 Conclusions

In conclusion, this assignment focused on preprocessing a dataset containing API call traces and training various classifiers to identify the most effective one for classifying this type of data. The preprocessing steps involved handling duplicates, followed by splitting the dataset into training and testing sets. Count Vectorization was used to ensure uniform feature lengths, and scaling was applied to the data. The preprocessing also included saving the vectorizer and scaler to be used in *classify-trace.py*.
Seven different classifiers, including **Support Vector** and **Gradient Boosting** classifiers were trained and evaluated. Among these, the **Random Forest** Classifier outperformed the others, achieving the highest F1 score of 0.6107. Notably, the Logistic Regression Classifier faced convergence issues and produced the lowest F1 score of 0.2939.
Despite some challenges, the results from both the Random Forest and Gradient Boosting Classifiers are promising. While the F1 scores may not be exceptionally high, the clear diagonal patterns in their respective confusion matrix heatmaps indicate reliable performance. Further optimization and experimentation may lead to improved results in future iterations of this analysis.