Assignment 4
Malware Classification

T-710-MLCS, Machine Learning in Cyber Security

Reykjavik University - School of Computer Science

Federico Maria Cruciani

federico24@ru.is

13. October 2024

1 Introduction

This report covers the work done for Assignment 4 of the Machine Learning in Cyber Security course. The objective of the assignment was to experiment and compare a number of models to classify different kinds of malware. This was done using a dataset containing API call traces recorded in a controlled sandbox environment.

2 Simple Data Preprocessing and Splitting

The preprocessing of the dataset takes place in the function `get_dataset()` of the file **preprocessing.py**, here a "bag-of-words" approach is employed by making use of the **CountVectorizer** class from the **scikit-learn** package. This class counts every occurrence of the API calls in the rows of the dataset and transforms them into vectors of counts. To improve its performance for this dataset the `token_pattern` parameter was modified from its default value, which is a regular expression that matches all words of length greater than 1. Since the dataset contains also single digit numbers, the expression was changed to include single characters as well and was also optimized to match only digits

Before being passed to the vectorizer, the data goes under one more step, which is to replace repeated API calls in a sequence with only one. The function then combines the data with its labels and returns the two train and test sets, split with a 70:30 ratio which was tested to yield the best results compared to other ratios.

In addition to this, another preprocessing technique is used: data scaling. This is applied separately, in the function `scale_data()`, because it's not needed by every tested model as explained further in section 3.

3 Train and Test Simple Classifiers

Training and testing of the classifiers is done inside the file **training.py**. The four chosen models are

- Naive Bayes
- Logistic Regression
- K-Nearest Neighbors
- Random Forest

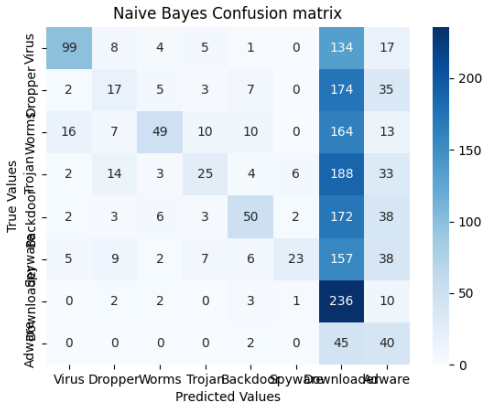
Their performance was compared using the f1-score and by plotting the confusion matrix.

The Naive Bayes classifier was chosen for its simplicity and its relation with the bag-of-words preprocessing technique, although it turned out to be the worst performing model, classifying most samples into a single class and getting an f1-score of just 0.2582.

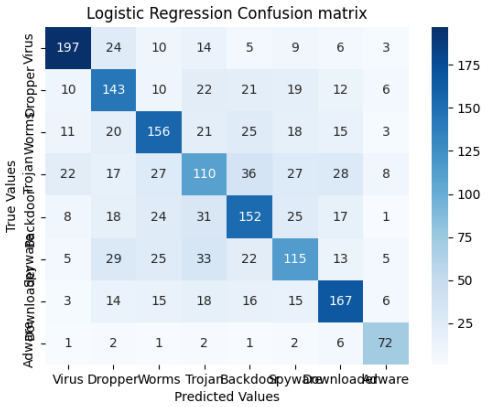
The choice of the Linear Regression was done because of its efficiency and potential to yield good results if the data is structured correctly. Its only downside compared to the others is that it needs the input data to be standardised. The f1-score of the Logistic Regression was 0.5925.

The K-Nearest Neighbors model was selected for this comparison because it uses a much different approach and could potentially identify patterns that the others can not see. This classifier also performed poorly with an f1-score of just 0.5098.

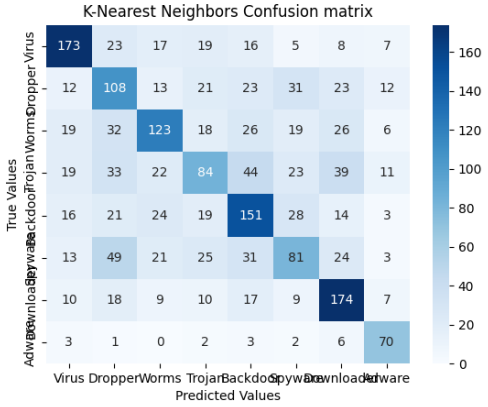
The main benefits of the Random Forest is that it doesn't need much data preprocessing to perform reasonably well and in fact it's the best model in this comparison with an f1-score of 0.6815, although that's still not very high. This model was trained with the `n_estimators` parameter set to 200.



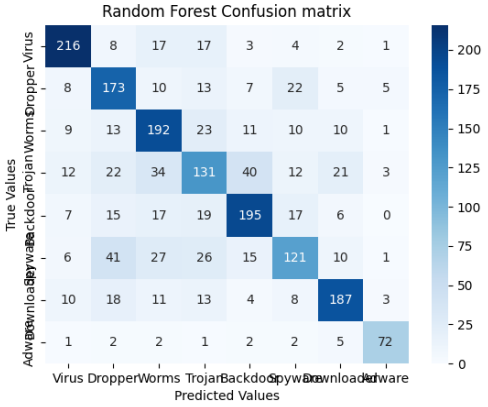
(a) Naive Bayes



(b) Logistic Regression



(a) KNN



(b) Random Forest

4 Export Model for an Independent Classifier

To export the best model from the experiments the library **skops.io** was used. When running **training.py** the best model is selected based on its f1-score and is saved to the file **model.skops**. The saved model can then be used by running

```
./classify-trace.py <filename>
```

where *filename* is the file containing the data to classify.

In addition to the model, the vectorizer used for preprocessing is persisted as well (in the file **vectorizer.skops**) because otherwise it would be impossible to use the model wince a new vectorizer wouldn't extract the same features from a different dataset.

5 Optional Enhancement - N-Grams

The additional approach chosen for the experiments was the use of n-grams. In the preprocessing, the vectorizer was initialized with the parameter `ngram_range` set to (1, 2) which improved the f1-score of the best model (the Random Forest) from 0.6742 to 0.6815. Higher n values where tested such as (1, 3) and (1, 4) but the performance improvement was negligible and the additional time taken to process the data didn't justify the change.