Assignment 2
Network Intrusion Detection and Decision Trees

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

19. September 2023

Contents

1 Introduction

This second assignment asks to use a slightly recent dataset of labelled network data[1]. Specifically, this experiment uses the .csv files in *MachienLearningCSV.zip*. This preprocessed dataset needed more preprocessing steps like editing each packet's labels and removing unuseful spaces from the column titles.
In the first part of the assignment, the entire dataset is pre-processed in two ways, then a Decision Tree Classifier is trained given these two pre-processed datasets.
The second part asks to replace the decision tree classifier with a random forest classifier, training the model over the two datasets previously created.
Results of all the experiments are shown using the confusion matrices and the metrics reported by the classification report (library given by sklearn.metrics).
The whole code is available at this GitHub repository

2 Data Pre-Processing and Splitting

Analyzing the given pre-processed dataset, the difficulties that we can encounter in using it to train a model are the following:

- column's names and values contain leading and trailing spaces that should be removed;
- the column "Labels" where each packet is classified has to be changed according to the assignment requirements;
- some values are "Infinity" and some others are "NaN";

These problems in pre-processing are solved as follows:

Listing 1: Data Pre-Processing

```
def pre_process_data(data_frame):
    data_frame.rename(columns=lambda x: x.strip(), inplace=True)
       # Remove leading/trailing spaces from column names
    data_frame["Label"] = data_frame["Label"].apply(lambda x: x.
       strip()) # Remove leading/trailing spaces from column
       values
    data_frame.replace([np.inf], np.finfo(np.float32).max, inplace
       =True) # Replace inf values with max of float 32
    data_frame.replace([-np.inf], -np.finfo(np.float32).min,
       inplace=True) # Replace inf values with min value of float
       32
    data_frame.fillna(0, inplace=True) # Replace all NaN values
       with 0
    data_frame["Label"] = data_frame["Label"].apply(lambda x :
       preprocess_label_column(x)) # Preprocess the column "Label"
    encoder = OneHotEncoder() # One hot encode the column "Label"
    encoded_data = encoder.fit_transform(data_frame[['Label']])
    encoded_df = pd.DataFrame(encoded_data.toarray(), columns=
       encoder.get_feature_names_out(['Label']))
    return encoded_df
```

This function is used to pre-process both the first kind of dataset (made of randomized training and testing sets of dimension 60% and 40% relatively) and the second kind (all data from Monday–Wednesday forms the training dataset, and all the data from Thursday/Friday forms the test dataset).

This second function instead is used to get the entire dataset split in train and test, according to **splitmode** parameter (if None it splits by days) and the directory to the files. The labels were also aggregated as asked in the requirements with the function **preprocess_label_column(label)** not reported here.

Listing 2: Function to return train and test data frames

```
def get_dataset(path_to_files, splitmode=None):
    if splitmode is not None and (splitmode < 0 or splitmode > 1):
        raise ValueError("splitmode should be between 0 and 1")

    d_train = pd.DataFrame()
    d_test = pd.DataFrame()
    if splitmode:
        df = merge_csv_files(path_to_files)
        pre_process_data(df)
        df = sklearn.utils.shuffle(df)
        split_index = int(len(df) * splitmode)
        d_train = pd.concat([d_train, df[:split_index]])
        d_test = pd.concat([d_test, df[split_index:]])
    else:
        d_train, d_test = train_test_by_days(path_to_files)
    return d_train, d_test
```

3 Decision Tree Classifier

This second point asked to write the Python code to run a Decision Tree Classifier (**DTC**) over the two pre-processed datasets created in the previous step. In the first case, the DTC must be trained with the **splitmode** parameter set to 0.6 (60% of training dataset and 40% of testing dataset) and a second time without **splitmode** and so using the dataset split by days of the week as said before. Experimental results are shown below with the confusion matrices.

This model gained an accuracy of 0.998682, and given this confusion matrix in Figure 1 we can say that it predicted quite well since the diagonal has the higher values, but we can also notice that the dataset is imbalanced since there are more true positive predictions on the "Benign" class than on the others.

The matrix in Figure 2 resulting from the second dataset shows that a lot of packets were predicted as *Benign* when they were *Scan*. This is caused by the absence of "Scan" in the training dataset since it was not shuffled. In fact "Scan" is present only in one of the three "Friday" files (part of the testing set) and the model was not able to recognize it since it is not trained in such a class. This model gained an accuracy of 0.817861.

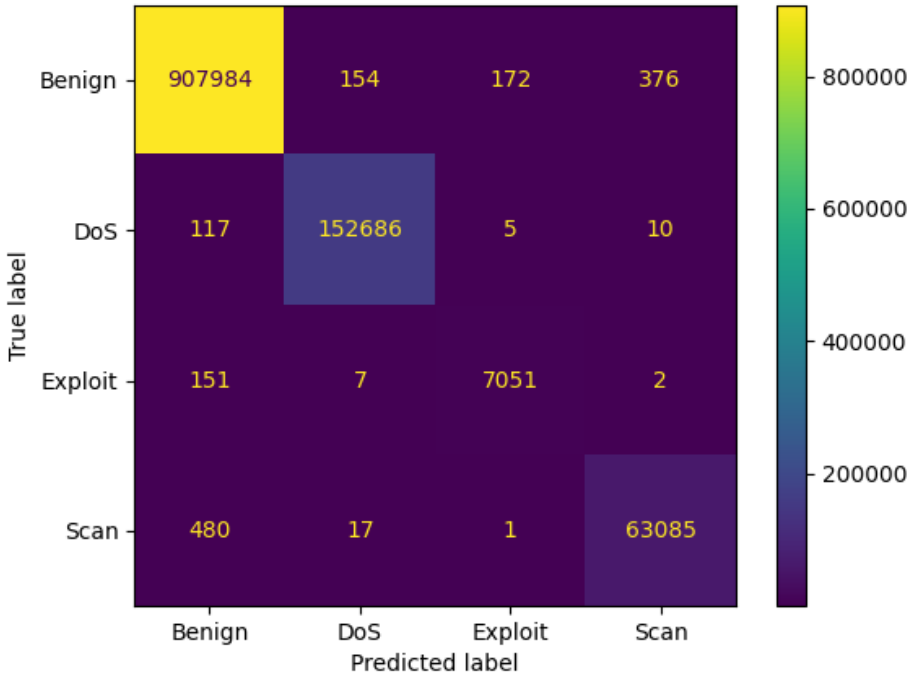

Figure 1: Confusion matrix resulted from DTC, using randomized subsets and **splitmode** parameter set to 0.6

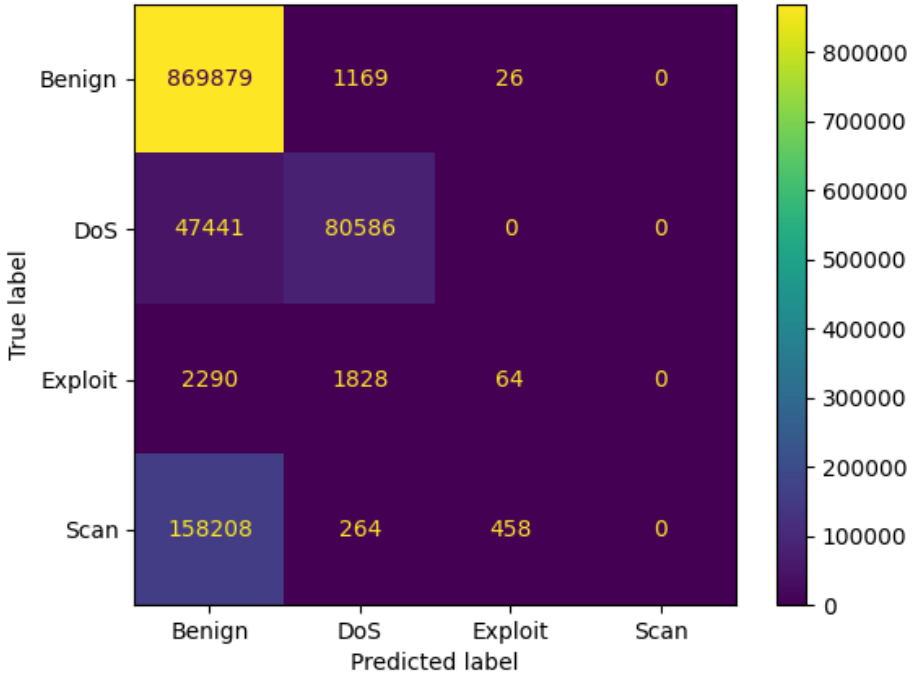

Figure 2: Confusion matrix resulted from DTC, using as training set Monday-Wednesday data and as testing set Thursday and Friday data.

4 Random Forest Classifier

The two datasets previously described are now used to train the Random Forest Classifier (**RFC**) and the resulting confusion matrices are analyzed and compared.

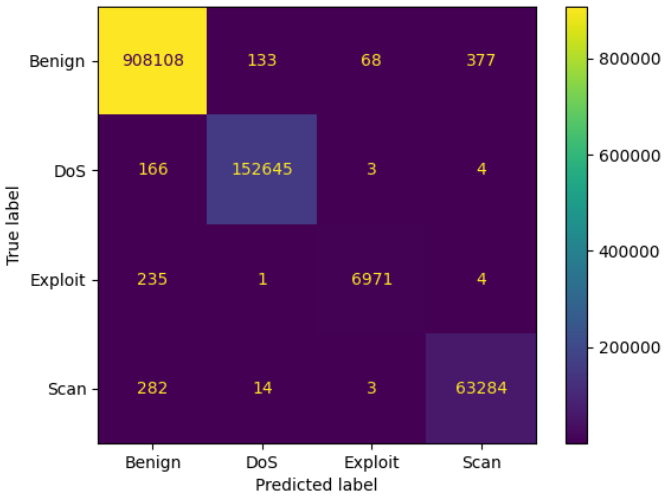

Figure 3: Confusion matrix resulted from RFC, using randomized subsets and **splitmode** parameter set to 0.6

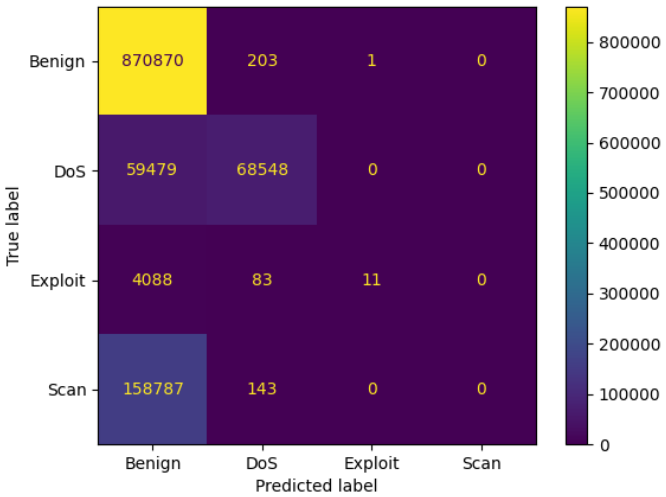

Figure 4: Confusion matrix resulted from RFC, using as training set Monday-Wednesday data and as testing set Thursday and Friday data.

Comparing these results from the previous ones, there aren't so many differences between the two approaches using the same datasets. The confusion matrices are quite the same as using DTC and one problem on the pre-trained dataset remains the high-class imbalance since also in the day-based approach *Class* values are not predicted at all (due to its absence in the training set).

5 Discriminating Node Labels

This last point asked to identify, looking at the plotted decision trees from both datasets, some main features that were used as discriminating node labels. In order to answer that question about the first dataset (with splitmode=0.6) we should consider the first two child nodes that are classified respectively as *Benign* and *DoS* by the Decision Tree Classifier. Their values are the following

Node	APS	Destination Port	gini	samples	value	class
True	≤ 7.694		0.223	1551928	[1114411, 4680, 7621, 32]	Benign
False		≤ 261.5	0.024	146517	[1681, 144771, 9, 56]	DoS

Table 1: Features referred to the first two child nodes of the tree, obtained by training a DTC with dataset split with splitmode = 0.6. **APS** refers to *Average Packet Size* feature.

Instead, watching the first two child nodes referred to the dataset pre-processed by days, we can notice that the features are the same as in the first dataset.

Node	PLM	Destination Port	gini	samples	value	class
True	≤ 5.882		0.131	1505323	[1399952, 91536, 13835]	Benign
False		≤ 261.5	0.025	163207	[2071, 161125, 11]	DoS

Table 2: Features referred to the first two child nodes of the tree, obtained by training a DTC with dataset split by days. **PLM** refers to *Packet Length Mean* feature.

Grouping the common features between the two datasets we can say that the most useful to detect an attack are:

- **Gini Index**: a lower Gini impurity indicates that the feature used for the split effectively separates the traffic into distinct classes
- **samples**: it represents the number of data points that have reached that particular node during the training process
- **value**: provides a breakdown of the class distribution among the data points that reach that node.

References

[1] Canadian Institute for Cybersecurity. Intrusion detection evaluation dataset (cic-ids2017).