Master's degree in Artificial Intelligence and Data Engineering
Data Mining and Machine Learning project presentation

PDF MALWARE DETECTION

Domenico D'Orsi - 665495

Github repository:
https://github.com/dominicofthebears/PDFMalwareDetector

A.Y. 2022/2023

Contents

1. Introduction……………………………………………………………….……..3
2. Dataset……………………………………………………………………...4
3. Preprocessing……………………………………………………………..…6
4. Classifiers comparison………………………………………………….……..9
   4.1 Results……………………………………………………………….…...10
   4.2 Observations……………………………………………………….……..13
5. Real-world application………………………………………………….…14
   5.1 Application pipeline……………………………………………….……..15
6. Conclusions……………………………………………………………….…...18

1. Introduction

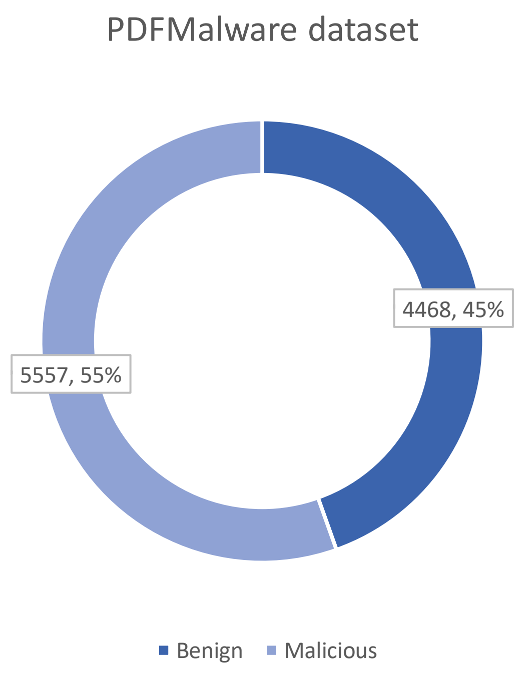
PDF Malware Detection is a simple security application, which allows a user to upload a PDF file which will be then classified as a Malicious or Benign file, exploiting one of the classifier available, and selected by the user himself.
The key idea behind this project is to focus on the explainability of the classification phase so the user can understand why the uploaded document may contain a Malware, and in this way several classifiers may be compared based not only on the performances obtained, but also comparing the rules produced.
In fact, the application doesn't show only the classification label for the document, but also the classification path (not for all the classifiers) and the rules produced by the classifier.

In order to realize these two parts, two kinds of models were realized:

- **Complete models**: these models were trained using a specific subset of features, which were extracted by a Feature Selection process, and they have been used for the comparison part.
- **Surrogated models:** these models were trained using a subset of features representing the ones that could be extracted from the PDF file using Python code, and they were used for the real-world application

2. Dataset

The dataset used for this application comes from the Canadian Institute for Cybersecurity, and it's named CIC-Evasive_PDFMAL2022: it is a collection of 10025 records consisting of various PDF files, labelled as Benign or Malicious (binary classification), collected from various sources such as Contagio and VirusTotal. The examples distribution is the following:



The dataset appears quite balanced, the examples are almost divided in half by the labels and because of this no sampling or other dataset rebalancing methods were applied.

The dataset in constituted by a total of 37 static features extracted from the PDF files, and divided in two main types:

1. **General features:** These features generally describe the PDF file, such as its size, whether it contains text or images, number of pages, and the title.

There are 12 features from this category.

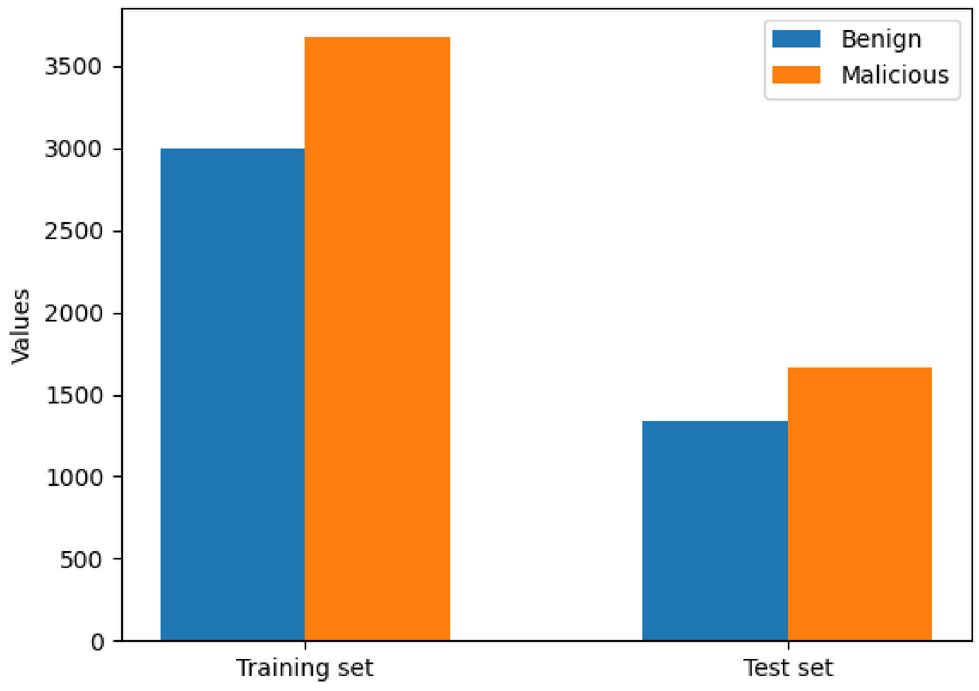
2. **Structural features:** These features describe the PDF file in terms of the structure, which requires a deeper parsing and provide an insight into the overall skeleton of the PDF. There are 25 features related to the PDF structure.

General features	Structural features
▪ PDF size ▪ title characters ▪ encryption ▪ metadata size ▪ page number ▪ header ▪ image number ▪ text ▪ object number ▪ font objects ▪ number of embedded files ▪ average size of all the embedded media	▪ No. of keywords "streams" ▪ No. of keywords "endstreams" ▪ Average stream size ▪ No. of Xref entries ▪ No. of name obfuscations ▪ Total number of filters used ▪ No. of objects with nested filters ▪ No. of stream objects (ObjStm) ▪ No. of keywords "/JS", No. of keywords "/JavaScript" ▪ No. of keywords "/URI", No. of keywords "/Action" ▪ No. of keywords "/AA", No. of keywords "/OpenAction" ▪ No. of keywords "/launch", No. of keywords "/submitForm" ▪ No. of keywords "/Acroform", No. of keywords "/XFA" ▪ No. of keywords "/JBig2Decode", No. of keywords "/Colors" ▪ No. of keywords "/Richmedia", No. of keywords "/Trailer" ▪ No. of keywords "/Xref", No. of keywords "/Startxref"

3. Preprocessing

The preprocessing phase that involved the dataset is really similar for the model comparison task and the classification task: it is made up of several steps that are resumed in the following points:

1. **Train-test split:** after dropping the null values, the first step was to split the dataset in training and test, in order to avoid any form of information leakage from the first to the second, because of the type of the operations that followed in the preprocessing phase. This split was made selecting **0.3** as test set size (30% of the entire dataset), and using the **Stratify** flag to preserve the classes distribution. After this operation, both training and test set distribution appeared like this:



2. **Cleaning and filling data:** the next operation performed on the split datasets was to clean data from all the noisy characters present in both continuous and categorical features, and encoding some non-significant values as missing ones: this was obtained by using regular expressions in order to overwrite those symbols. The filling operation was then obtained by using the ”SimpleImputer” class from the sklearn library, filling the mssing values with the median for the continuous features and with the most frequent for categorical ones
3. **Feature selection:** this part is the only difference between the preprocessing applied on the datasets used for the complete and surrogated models, because in the last ones this phase was not conducted. To perform this operation, the *mutual information* was chosen as feature selection method by using the “mutual_info_classif” method

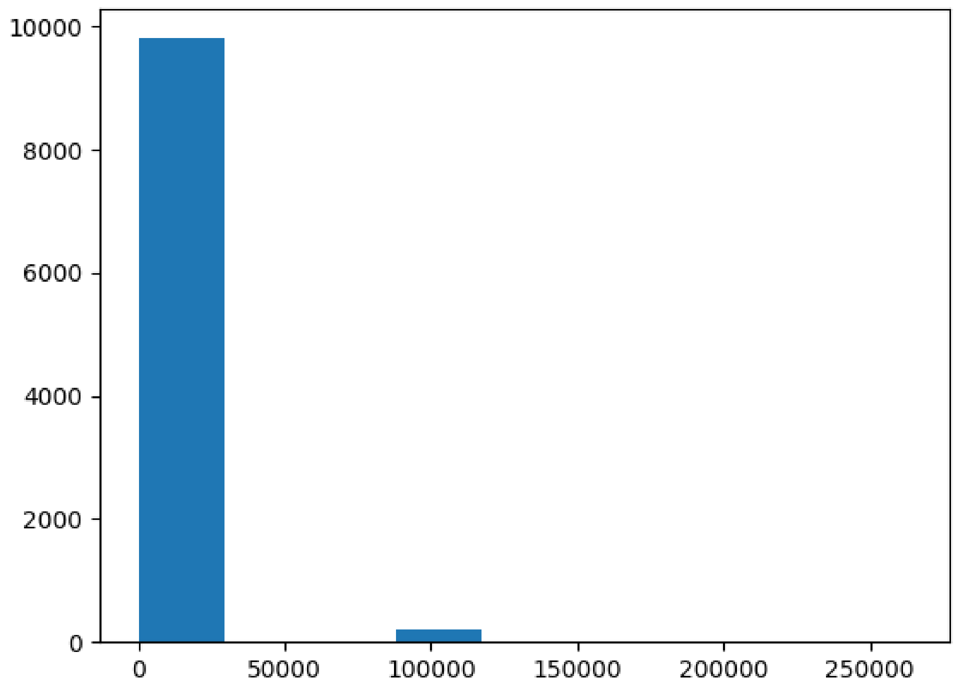
$$I(C,fi) = \sum_{c \in CC} \sum_{f_i \in FF} p(c,fi) \cdot \log \frac{p(c,fi)}{p(c)p(fi)}$$

$$NI(f_i,f_s) = \frac{I(f_i;f_s)}{min\{H(f_i),H(f_s)\}}$$

$$G = I(C,f_i) - \frac{1}{S} \sum_{f_s \in S} NI(f_i;f_s)$$

The top 10 features having the highest gain were then selected and used to form the final dataset

4. **Outlier detection:** the following step was an outlier detection performed by using histograms plot of some features, which showed how we had a very low density on some values, such as the following example for the feature “xref length”.



This step has a great importance because the discretization algorithms used in the next phase are quite sensitive to outliers, so this operation had to be carried out

5. **Discretization:** at last, a discretization phase was performed on the dataset, since the classifiers used lately required categorical values. The two tested algorithms are the equal frequency approach and the equal width approach, but in the end the first one was selected., dividing each feature into 5 bins and encoding each value into a string representing the corresponding bin. After this operation, both a continuous and a categorical cleaned dataset are produced and saved in the *dataset* folder

4. Classifiers comparison

As told before, one of the main goals of this project was to compare several classifiers in order to verify which one had had best performances on the dataset, but as we know metrics are not always enough. So, in order to give more insight about the classifiers and also to make the labelling process explainable to the user the choice for which models should be used fell on the ones capable to produce rules: *AdaBoost (with 50 base trees having 2 as max depth), DecisionTree (with max depth = 5), CN2Learner* and *Association Rules (apriori based).* For the first two, the treeplot is available as well in the folder "*models/plots*".

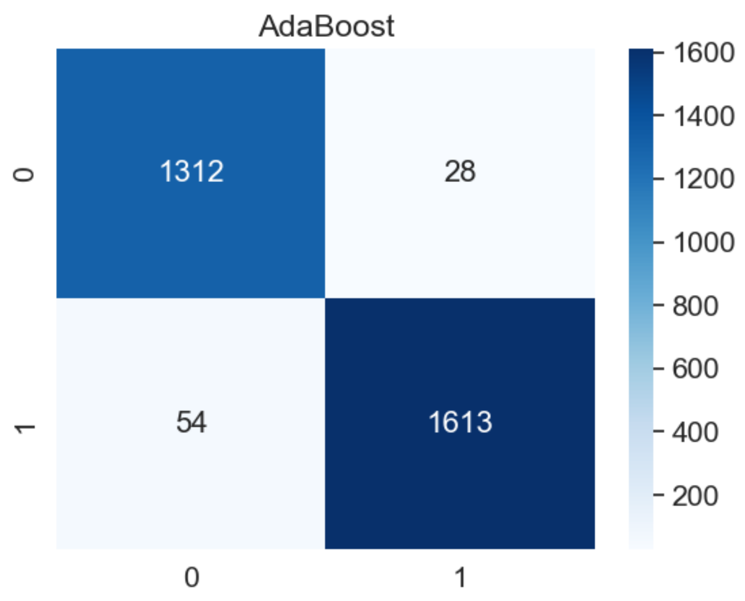
All of them are well-known classifiers, except for association rules which were used only for the rule comparison part.

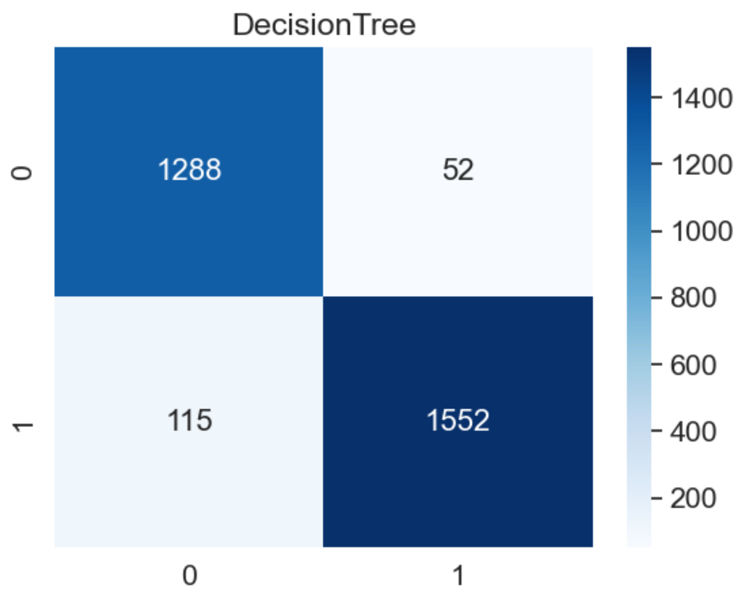
Moreover, is necessary to underline that before the training phase of the classifiers, a *OneHotEncoder* was applied to training and test set, inserting it into a *sklearn Pipeline* for the AdaBoost and for the DecisionTree. While training these two classifiers, a *K-fold cross validation* was used, with k = 5, and after this process the best model was chosen and re-trained on the entire dataset.

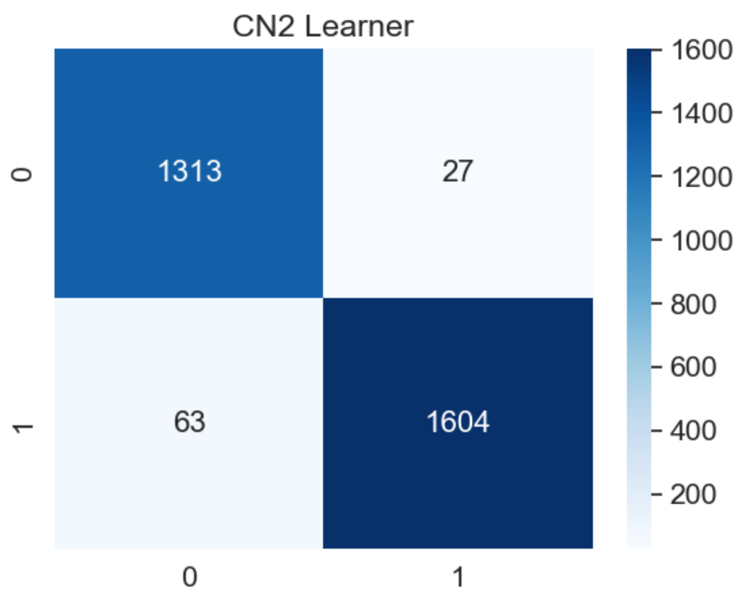
In the end, the validation phase was not performed for the CN2 Learner because of the way the classifier works, also because of the specific implementation used in this project.

4.1 Results

- **Confusion matrixes:** showing only test set results for brevity, validation set results are available in the project at "*models/results/validation*"







- **Metrics:** all the results for the classifiers comparison are available in the folder "*models/results/test*"

	Benign			Malicious			Macro avg			Weighted avg		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
AdaBoost	0.960	0.979	0.969	0.982	0.967	0.975	0.971	0.973	0.972	0.972	0.972	0.972
Decision Tree	0.918	0.961	0.939	0.967	0.931	0.948	0.942	0.946	0.944	0.945	0.944	0.944
CN2 Learner	0.954	0.979	0.966	0.983	0.962	0.972	0.968	0.971	0.969	0.970	0.970	0.970

- **Rules set:** the complete rules produced by the classifiers are available in the folder "*models/rules*". For the AdaBoost classifier, the rules refer to the last tree of the boosting method.

AdaBoost

```
if (endobj in range [0.0 - 9.0[ <= 0.5) and (xref Length in range
[0.0 - 110.0[ <= 0.5) then class: Benign

if (endobj in range [0.0 - 9.0[ <= 0.5) and (xref Length in range
[0.0 - 110.0[ > 0.5) then class: Malicious

if (endobj in range [0.0 - 9.0[ > 0.5) and (JS in range [1.0 -
149.0] > 0.5) then class: Malicious

if (endobj in range [0.0 - 9.0[ > 0.5) and (JS in range [1.0 -
149.0] <= 0.5) then class: Malicious
```

CN2 Learner

```
IF startxref in range [2.0 - 68.0] = 1.0 AND JS in range [1.0 -
149.0] = 1.0 THEN Class=Benign

IF obj in range [25.0 - 60.0[ = 1.0 AND metadata size in range
[2830.0 - 3430.0[ = 1.0 THEN Class=Benign

IF JS in range [1.0 - 149.0] = 1.0 AND pdfsize in range [250.0 -
580.0[ = 1.0 THEN Class=Malicious

IF JS in range [1.0 - 149.0] = 1.0 AND metadata size in range
[2830.0 - 3430.0[ = 1.0 THEN Class=Malicious

IF endobj in range [9.0 - 13.0[ = 1.0 AND xref Length in range
[280.0 - 850.0[ = 1.0 THEN Class=Benign

IF obj in range [60.0 - 2608.0] = 1.0 AND pdfsize in range [580.0 -
860.0[ = 1.0 THEN Class=Benign
```

Decision Tree

```
if (startxref in range [2.0 - 68.0] <= 0.5) and (Javascript in range
[0.0 - 1.0[ > 0.5) and (obj in range [14.0 - 25.0[ > 0.5) and (xref
Length in range [170.0 - 280.0[ > 0.5) and (pdfsize in range [860.0
- 9520.0] <= 0.5) then class: Malicious

if (startxref in range [2.0 - 68.0] > 0.5) and (xref Length in range
[0.0 - 110.0[ > 0.5) and (metadata size in range [2830.0 - 3430.0[
<= 0.5) then class: Malicious

if (startxref in range [2.0 - 68.0] > 0.5) and (xref Length in range
[0.0 - 110.0[ <= 0.5) and (pdfsize in range [0.0 - 90.0[ <= 0.5) and
```

```
(JS in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3430.0
- 7460.0] <= 0.5) then class: Malicious

if (startxref in range [2.0 - 68.0] <= 0.5) and (Javascript in range
[0.0 - 1.0[ <= 0.5) and (endobj in range [60.0 - 2608.0] > 0.5) and
(metadata size in range [3430.0 - 7460.0] > 0.5) and (Javascript in
range [2.0 - 149.0] > 0.5) then class: Benign
```

Association rules

```
IF JS in range [0.0 - 1.0[ AND Javascript in range [0.0 - 1.0[ THEN
Class = Benign

IF metadata size in range [1800.0 - 2560.0[ AND JS in range [1.0 -
149.0] THEN Class = Malicious

IF trailer in range [10.0 - 20.0[ AND JS in range [1.0 - 149.0] THEN
Class = Malicious

IF startxref in range [2.0 - 68.0] AND Javascript in range [0.0 -
1.0[ THEN Class = Benign
```

4.2 Observations

From a performance point of view, we can observe how the AdaBoost classifier obtained the best results, even if also the other two classifiers produced good performances on the test set, so all of them are valid for this task.
For what concerns the rules produced, they often make use of the “javscript” and “JS” features in the antecedents: this gives an interesting insight on how potentially dangerous PDF files are containing code scripts, of course because of the capability of these scripts to convey malicious operations, such as stealing data from the computer.
Also, xref tables and xref entries are quite common in Malicious PDF files, as we can observe these two features in all the rules produced by the Decision Tree, for example, and in several rules produced by the other classifiers.
Generally speaking, the rules obtained by the models give a useful insight on how malicious files usually are composed.

5. Real-world application

The second goal of this project is to simulate the way a *rule-based Intrusion detection system* works, remaining in the malware detection field but focusing on a different way to convey malware to a machine: for this reason, a real-world application was developed using the same three classifiers previously compared.

This application allows the user, through a dedicated GUI developed using *pysimplegui*, to upload a custom pdf file and to select a classifier of his choice between AdaBoost, Decision Tree and CN2 Learner, then the PDF file will be processed and another window will open, giving as a result of this operation the class prediction (Benign or Malicious) for the uploaded file along with the rules produced by the specific model of the classifier and the decision path if the selected one is AdaBoost or Decision Tree.

The proper way to install the requirements and run the application are specified in the *README.md* file of the project repository.

5.1 Application pipeline

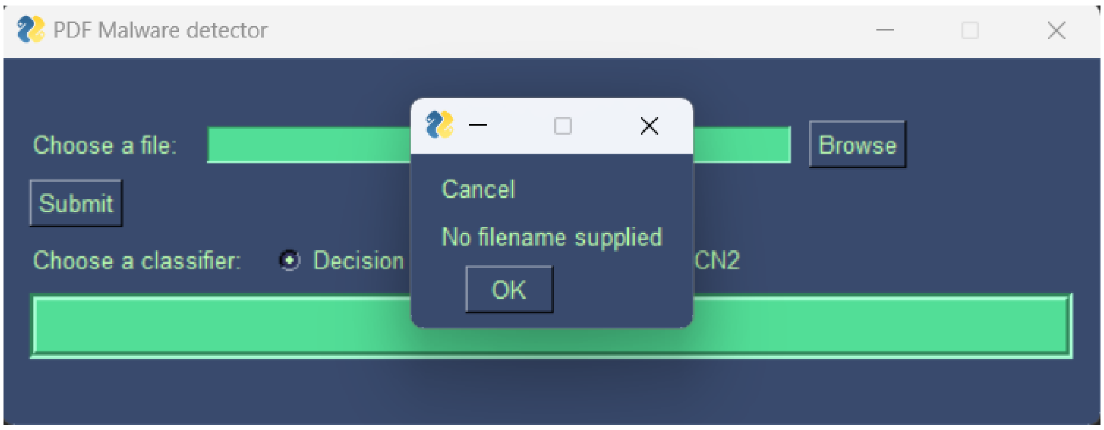
In order to make the application work in a proper way, several steps were required in addition to what was already done for the classifiers comparison part.
The operations performed in advance with respect to the interaction between the application and the user are:

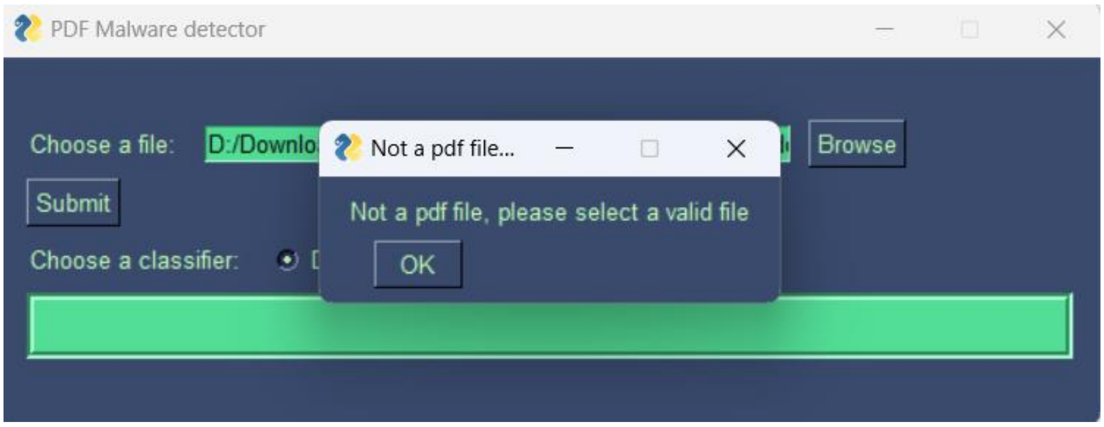
1. **Creation of the new dataset:** the first step was, of course, the creation of a new training set for the surrogated models, since only a specific subset of features could be used for this part of the project. There is no difference between the preprocessing phase for the dataset used in the first part and this second one, but no feature selection process was performed: instead, the selected features are the ones that could be extracted using another part of code from the PDF file. Those features are:
    - **PDF size**: size of the file in KiloBytes
    - **Metadata size**: size of the metadata part of the file
    - **Pages**: number of pages of the file
    - **Title characters**: number of characters composing the title
    - **isEncrypted**: whether the file is encrypted or not
    - **Images**: number of images
    - **Text**: presence of text or not
    - **Javascript**: count of Javascript keywords

2. **Training of the surrogated models:** after creating the new dataset, the three classifiers were trained on this newly produced examples (with the same configuration both for the K-fold cross validation and for the models themselves) in order to be saved and used to classify the user-inputted PDF. After the training phase, the models, as well as the rules produced by them, were stored in the folder *models/surrogated_models* with the *.pkl* extension, obtained by using the *pickle* library to perform this operation.

These initial operations are needed as a setup for the user's usage of the application. During this interaction, other steps are made in order to produce the required output:

1. **Feature extraction from the PDF file**: when the user start the application, he is asked to upload a PDF file and to select a classifier. An error message will be displayed in case no file is provided or the file extension is not .pdf

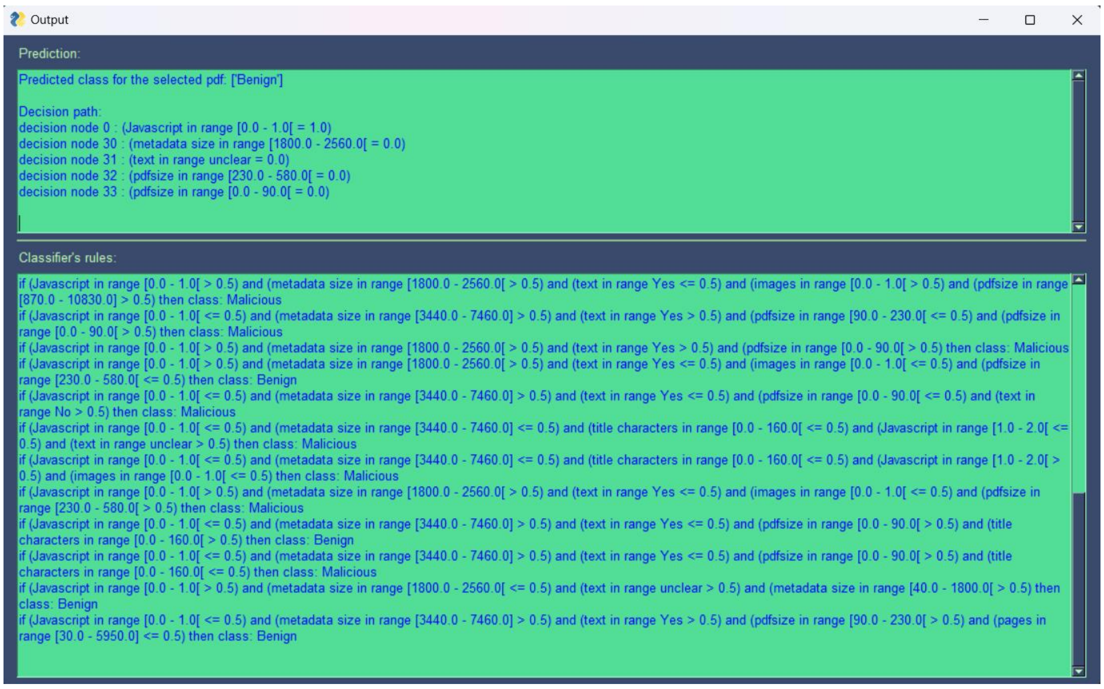




Wrong input examples

After selecting a valid .pdf file, the features extraction process will start: this step is performed by using the *PyPDF2* and *asposewords* libraries. All of the necessary features are obtain by using built-in function from PyPDF2, while the Javascript feature is obtained by transforming the PDF file into an HTML file and counting the number of the <*script*> tags. The example will then be encoded and discretized in the same way of the training set used for surrogated models

2. **Classification:** the last step, of course, is providing the example to the chosen classifier in order to output the predicted label and other information, which are the model's rules and decision path for AdaBoost classifier and Decision Tree classifier

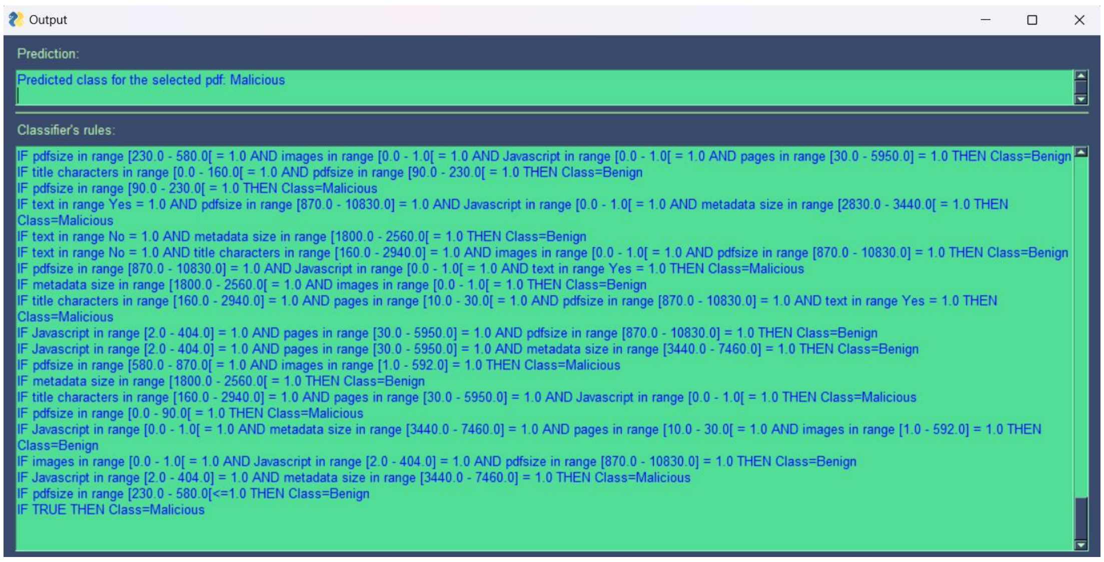


```
Prediction:
Predicted class for the selected pdf: ['Benign']

Decision path:
decision node 0 : (Javascript in range [0.0 - 1.0[ = 1.0)
decision node 30 : (metadata size in range [1800.0 - 2560.0[ = 0.0)
decision node 31 : (text in range unclear = 0.0)
decision node 32 : (pdfsize in range [230.0 - 580.0[ = 0.0)
decision node 33 : (pdfsize in range [0.0 - 90.0[ = 0.0)

Classifier's rules:
if (Javascript in range [0.0 - 1.0[ > 0.5) and (metadata size in range [1800.0 - 2560.0[ > 0.5) and (text in range Yes <= 0.5) and (images in range [0.0 - 1.0[ > 0.5) and (pdfsize in range [870.0 - 10830.0] > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] > 0.5) and (text in range Yes > 0.5) and (pdfsize in range [90.0 - 230.0[ <= 0.5) and (pdfsize in range [0.0 - 90.0[ > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ > 0.5) and (metadata size in range [1800.0 - 2560.0[ > 0.5) and (text in range Yes > 0.5) and (pdfsize in range [0.0 - 90.0[ > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ > 0.5) and (metadata size in range [1800.0 - 2560.0[ > 0.5) and (text in range Yes <= 0.5) and (images in range [0.0 - 1.0[ <= 0.5) and (pdfsize in range [230.0 - 580.0[ <= 0.5) then class: Benign
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] > 0.5) and (text in range Yes <= 0.5) and (pdfsize in range [0.0 - 90.0[ <= 0.5) and (text in range No > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] <= 0.5) and (title characters in range [0.0 - 160.0[ <= 0.5) and (Javascript in range [1.0 - 2.0[ <= 0.5) and (text in range unclear > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] <= 0.5) and (title characters in range [0.0 - 160.0[ <= 0.5) and (Javascript in range [1.0 - 2.0[ > 0.5) and (images in range [0.0 - 1.0[ <= 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ > 0.5) and (metadata size in range [1800.0 - 2560.0[ > 0.5) and (text in range Yes <= 0.5) and (images in range [0.0 - 1.0[ <= 0.5) and (pdfsize in range [230.0 - 580.0[ > 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] > 0.5) and (text in range Yes <= 0.5) and (pdfsize in range [0.0 - 90.0[ > 0.5) and (title characters in range [0.0 - 160.0[ > 0.5) then class: Benign
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] > 0.5) and (text in range Yes <= 0.5) and (pdfsize in range [0.0 - 90.0[ > 0.5) and (title characters in range [0.0 - 160.0[ <= 0.5) then class: Malicious
if (Javascript in range [0.0 - 1.0[ > 0.5) and (metadata size in range [1800.0 - 2560.0[ <= 0.5) and (text in range unclear > 0.5) and (metadata size in range [40.0 - 1800.0[ > 0.5) then class: Benign
if (Javascript in range [0.0 - 1.0[ <= 0.5) and (metadata size in range [3440.0 - 7460.0] > 0.5) and (text in range Yes > 0.5) and (pdfsize in range [90.0 - 230.0[ > 0.5) and (pages in range [30.0 - 5950.0] <= 0.5) then class: Benign
```

Output example for Decision Tree



```
Prediction:
Predicted class for the selected pdf: Malicious

Classifier's rules:
IF pdfsize in range [230.0 - 580.0[ = 1.0 AND images in range [0.0 - 1.0[ = 1.0 AND Javascript in range [0.0 - 1.0[ = 1.0 AND pages in range [30.0 - 5950.0] = 1.0 THEN Class=Benign
IF title characters in range [0.0 - 160.0[ = 1.0 AND pdfsize in range [90.0 - 230.0[ = 1.0 THEN Class=Benign
IF pdfsize in range [90.0 - 230.0[ = 1.0 THEN Class=Malicious
IF text in range Yes = 1.0 AND pdfsize in range [870.0 - 10830.0] = 1.0 AND Javascript in range [0.0 - 1.0[ = 1.0 AND metadata size in range [2830.0 - 3440.0[ = 1.0 THEN Class=Malicious
IF text in range No = 1.0 AND metadata size in range [1800.0 - 2560.0[ = 1.0 THEN Class=Benign
IF text in range No = 1.0 AND title characters in range [160.0 - 2940.0] = 1.0 AND images in range [0.0 - 1.0[ = 1.0 AND pdfsize in range [870.0 - 10830.0] = 1.0 THEN Class=Benign
IF pdfsize in range [870.0 - 10830.0] = 1.0 AND Javascript in range [0.0 - 1.0[ = 1.0 AND text in range Yes = 1.0 THEN Class=Malicious
IF metadata size in range [1800.0 - 2560.0[ = 1.0 AND images in range [0.0 - 1.0[ = 1.0 THEN Class=Benign
IF title characters in range [160.0 - 2940.0] = 1.0 AND pages in range [10.0 - 30.0[ = 1.0 AND pdfsize in range [870.0 - 10830.0] = 1.0 AND text in range Yes = 1.0 THEN Class=Malicious
IF Javascript in range [2.0 - 404.0] = 1.0 AND pages in range [30.0 - 5950.0] = 1.0 AND pdfsize in range [870.0 - 10830.0] = 1.0 THEN Class=Benign
IF Javascript in range [2.0 - 404.0] = 1.0 AND pages in range [30.0 - 5950.0] = 1.0 AND metadata size in range [3440.0 - 7460.0] = 1.0 THEN Class=Benign
IF pdfsize in range [580.0 - 870.0[ = 1.0 AND images in range [1.0 - 592.0] = 1.0 THEN Class=Malicious
IF metadata size in range [1800.0 - 2560.0[ = 1.0 THEN Class=Benign
IF title characters in range [160.0 - 2940.0] = 1.0 AND pages in range [30.0 - 5950.0] = 1.0 AND Javascript in range [0.0 - 1.0[ = 1.0 THEN Class=Malicious
IF pdfsize in range [0.0 - 90.0[ = 1.0 THEN Class=Malicious
IF Javascript in range [0.0 - 1.0[ = 1.0 AND metadata size in range [3440.0 - 7460.0] = 1.0 AND pages in range [10.0 - 30.0[ = 1.0 AND images in range [1.0 - 592.0] = 1.0 THEN Class=Benign
IF images in range [0.0 - 1.0[ = 1.0 AND Javascript in range [2.0 - 404.0] = 1.0 AND pdfsize in range [870.0 - 10830.0] = 1.0 THEN Class=Benign
IF Javascript in range [2.0 - 404.0] = 1.0 AND metadata size in range [3440.0 - 7460.0] = 1.0 THEN Class=Malicious
IF pdfsize in range [230.0 - 580.0[<=1.0 THEN Class=Benign
IF TRUE THEN Class=Malicious
```

Output example for CN2 Learner

7. Conclusions

Concluding, we can assert that those classifiers are quite suitable for this kind of task, having also the capability to explain the decision taken through the rules produces. This is confirmed by the optimal performances obtained on the complete dataset, filtered with a proper feature selection method, where we also obtain an interesting insight about how dangerous PDF files containing Javascript tags and xref tables are, resulting malicious most of the times.

For what regards the classification part, the tree-based classifiers analyzed are preferable compared to the CN2 learner, because they can give even more information thanks to the possibility to extract the decision path used to classify the example. Moreover, the low significance of the features makes harder for the CN2 Learner to classify the given PDF in a proper way, resulting in a Malicious prediction for totally safe files.

In a future enhancement to the program, a better feature extractor in order to obtain the same features resulted the best with respect to the Mutual Information score from the user-inputted PDF file, so more accurate predictions can be performed by the classifiers.