École d'ingénieurs
Télécom Physique Strasbourg

Université
de Strasbourg

Data Protection Project
"Detect malware at runtime"

Projet réalisé par:

BENYAHYA Oumayma
ELAMINE Mohamed
JANATI Siham

Sommaire

I. Introduction sommaire

The existence of obfuscated malware that hides to avoid detection prompts us to do analytical studies of data representing such situations. The data set we have was created to represent a situation as close to reality as possible using malware prevalent in the real world. Made up of Spyware, Ransomware and Trojan Horse malware, it provides a balanced dataset that can be used to test obfuscated malware detection systems.

The objective of this project is to develop a python module for intrusion detection, for the scenario of **“ Detect malware at runtime”.**
First, we started with the pre-processing of the data, then by creating the models and then integrating everything into a model.

II. Exploratory Data Analysis chain

1. Data Cleaning

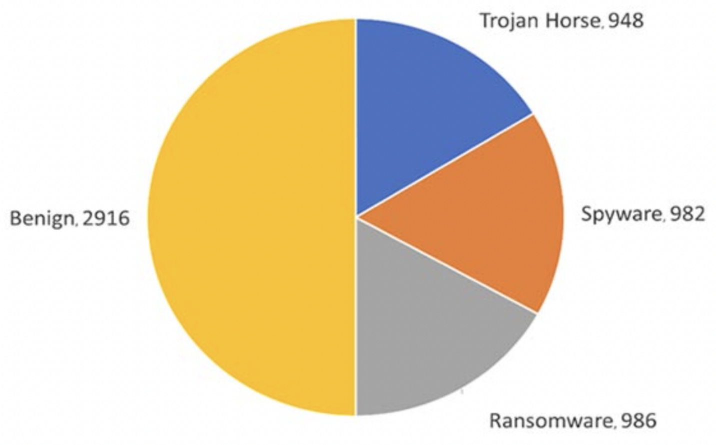
1.1 Useless Columns :

After a deep visualization of the data, we noticed that some columns always take the same values. Therefore these columns don’t bring any additional information to our dataset. Hence, it is better to delete them.

“pslist.nprocs64bit”, ”handles.nport” & svcscan.interactive_process_services”
Always take the same value (0)

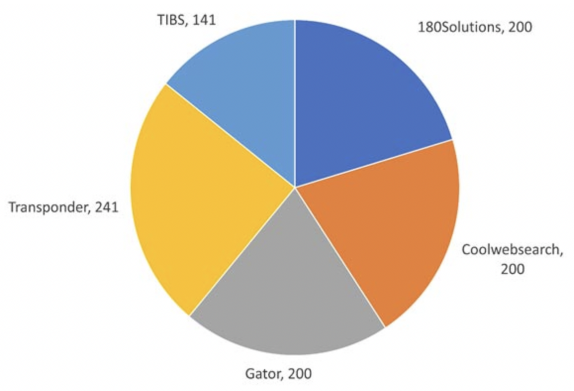
1.2 Creation of new features :

- **“matware_cat” :** contain malware categories

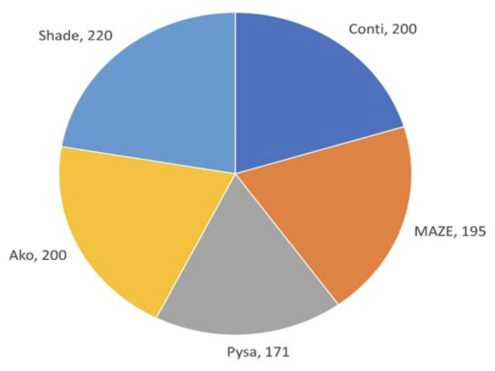


- ***" malware_fam" :*** contain malware family

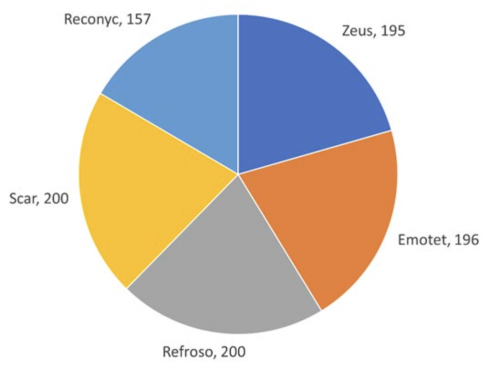
Spyware



Ransomware



Trojan



1.3 Data Coherence :

we checked the consistency of our data as follows:

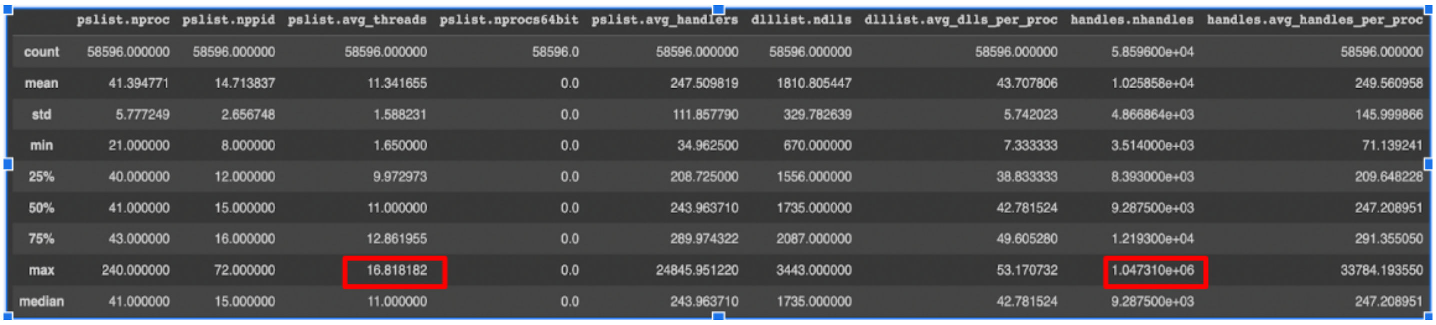
- All malwares are either Ransomeware, Trojan or Spyware
- if a malware category is not benign, it is surely a malware
- All benign data, their malware category is benign
- if a malware category is benign, it surely belongs to the class Benign

⇒ All malware data belongs to the class "benign" doesn't have a malware category and vice-versa

2. Data Standardization (Scaling, PCA, Feature Extraction)

2.1 Statistics :

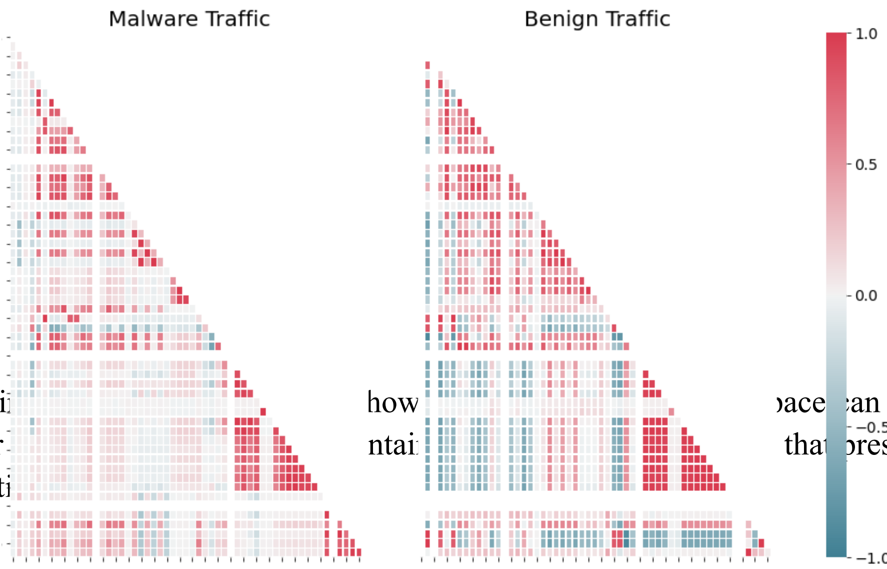
According to the figure below, we noticed that the columns have different size scales. Hence, our data needs to be scaled first before being given to the models for fitting.



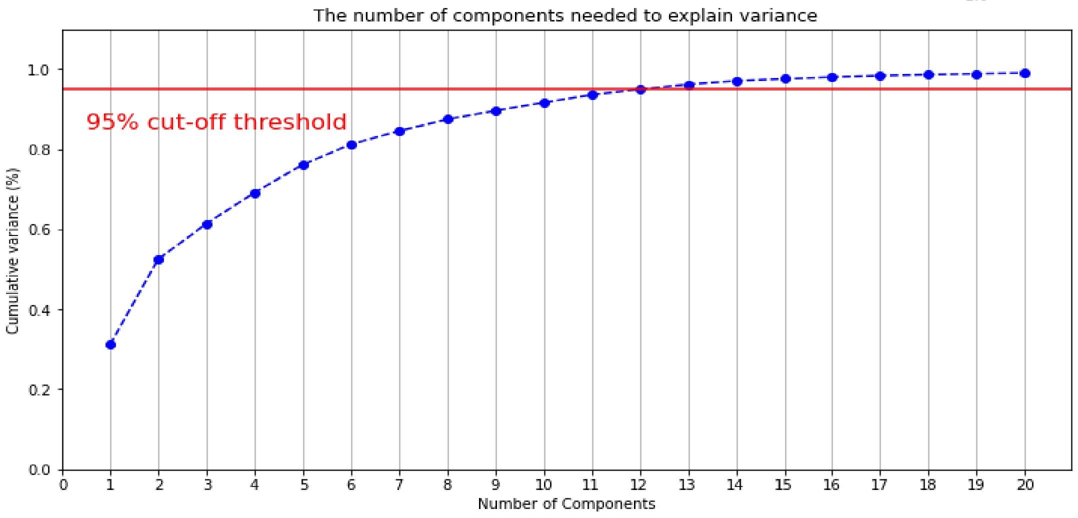
	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc	handles.nhandles	handles.avg_handles_per_proc
count	58596.000000	58596.000000	58596.000000	58596.0	58596.000000	58596.000000	58596.000000	5.859600e+04	58596.000000
mean	41.394771	14.713837	11.341655	0.0	247.509819	1810.805447	43.707806	1.025858e+04	249.560958
std	5.777249	2.656748	1.588231	0.0	111.857790	329.782639	5.742023	4.866864e+03	145.999866
min	21.000000	8.000000	1.650000	0.0	34.962500	670.000000	7.333333	3.514000e+03	71.139241
25%	40.000000	12.000000	9.972973	0.0	208.725000	1556.000000	38.833333	8.393000e+03	209.648228
50%	41.000000	15.000000	11.000000	0.0	243.963710	1735.000000	42.781524	9.287500e+03	247.208951
75%	43.000000	16.000000	12.861955	0.0	289.974322	2087.000000	49.605280	1.219300e+04	291.355050
max	240.000000	72.000000	16.818182	0.0	24845.951220	3443.000000	53.170732	1.047310e+06	33784.193550
median	41.000000	15.000000	11.000000	0.0	243.963710	1735.000000	42.781524	9.287500e+03	247.208951

2.2 Correlations :

The figure (x) below represents the correlation between different columns and we notice that the correlation of some columns is zero and these columns are indeed the useless columns “pslist.nprocs64bit”, ”handles.nport” & “svcscan.interactive_process_services” .



After applyi how ace an be projected into another ntai that preserve 95% of the informati



III. Pipeline/Meta Model

1. Model structure

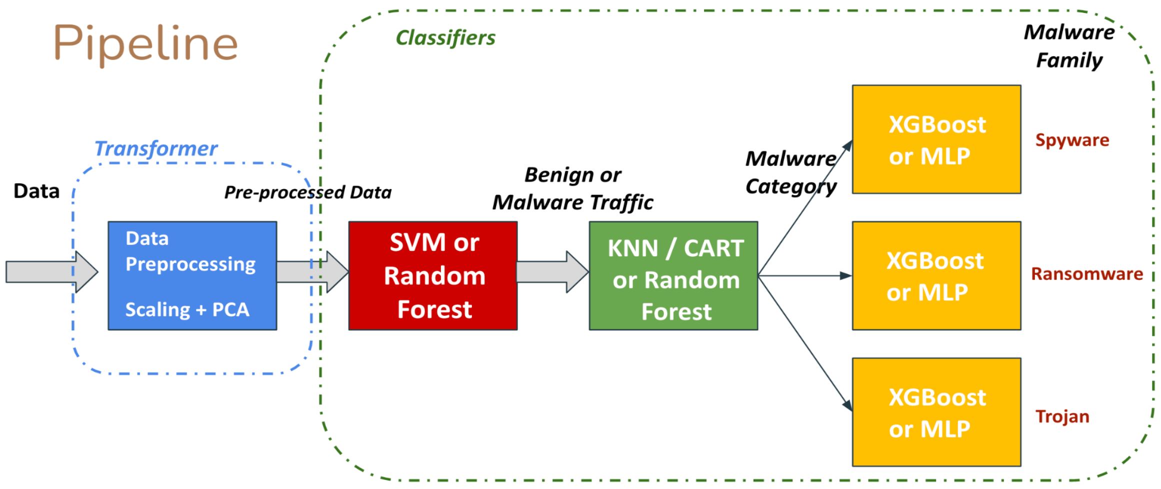
Since the malware traffic can be classified to multiple categories and families, we decided to split the classification process to multiple small processes in order to improve the classification performance. In fact, our “Meta Model” uses multiple classification models in a pipeline where every model uses the prediction of the previous one in order to predict the next class. We believe that this architecture can improve the performance of our final model because every sub-model focuses on a specific type of malware traffic and a more simple classification problem, which also makes the dataset more balanced.

Below we represent the general structure of the proposed pipeline. It gathers a “Transformer” and a “Classifier”.
The first one applies some preprocessing transformations:

- **Scaling:** because machine learning classifiers perform better when data is scaled.
- **PCA (Principal Component Analysis):** to select non-correlated features that encodes 95% of the total information held by our data. Hence projecting our data in a smaller space helps the models to focus on fewer features and learn better the information encoded in them.

The second one gathers multiple classifiers where each one is used for a specific classification type:

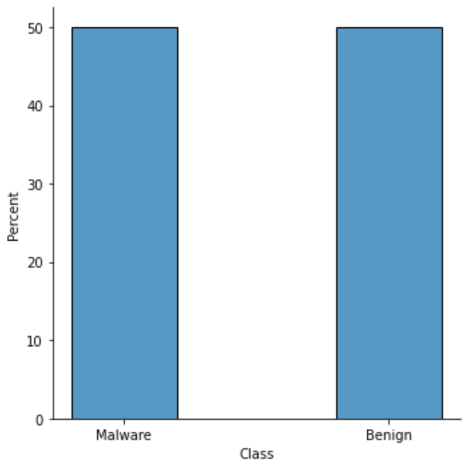
- **Benign/Malware: Binary Classification**
  Decide whether the traffic is malware or benign. We advise using SVM or Random Forest.
- **Malware Category: Multi-Class Classification**
  Used when the traffic is malware in order to predict the category it belongs to (Trojan, Spyware or Ransomware). We advise using KNN, CART or Random Forest for this type of classification.
- **Malware Family: Multi-Class Classification**
  This type of classification uses 3 different models, each for a unique malware category. Hence it is used to predict the family of the malware traffic knowing its category. We advise using XGBoost or MLP for this type of classification.



2. Tested algorithms

a. Benign vs Malware

Class repartition in percentage

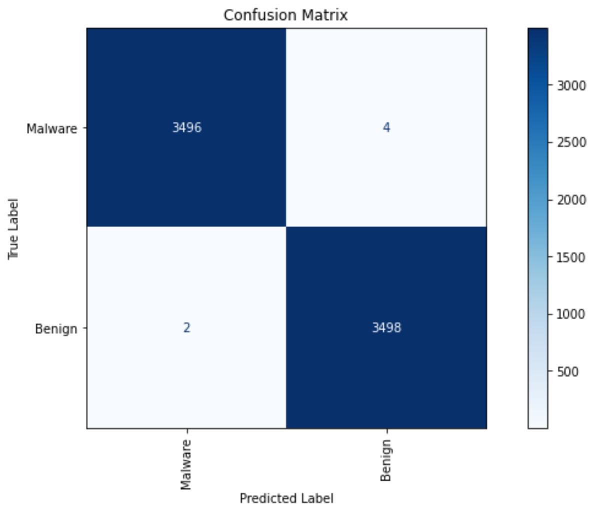


For this first part of our model, we tested two algorithms: SVM and Random Forest. Both algorithms had a good performance on the binary classification task.
We tested with different sizes of our Dataset (between 10^2 and 10^4), and it had no significant impact on the results.

❖ ***SVM***

Accuracy Score: 0.99
Balanced Accuracy: 0.99
Matthews Correlation Coefficient: 0.99

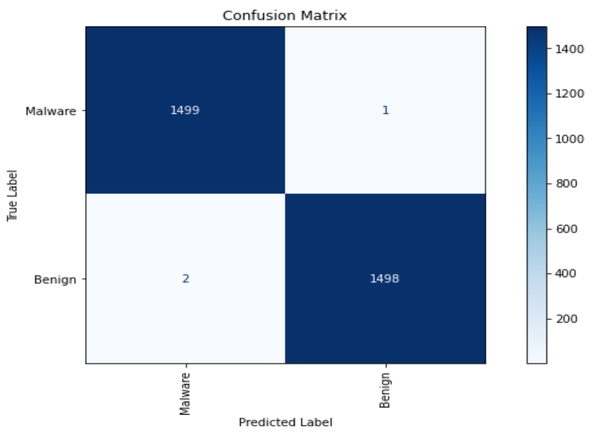
	precision	recall	f1-score	support
Malware	1.00	1.00	1.00	1500
Benign	1.00	1.00	1.00	1500
accuracy			1.00	3000
macro avg	1.00	1.00	1.00	3000
weighted avg	1.00	1.00	1.00	3000



❖ ***Random Forest***

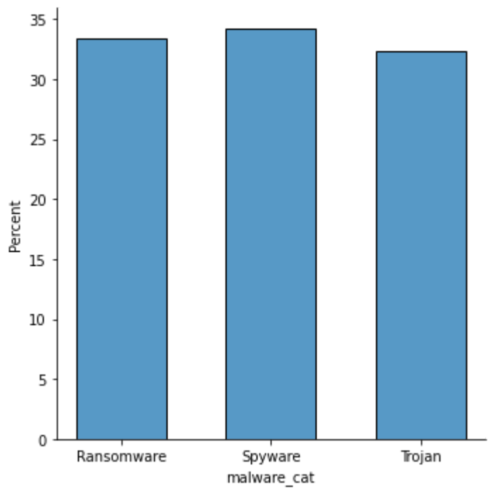
Accuracy Score: 0.99
Balanced Accuracy: 0.99
Matthews Correlation Coefficient: 0.99

	precision	**recall**	**f1-score**	**support**
Malware	1.00	1.00	1.00	1500
Benign	1.00	1.00	1.00	1500
accuracy			1.00	3000
macro avg	1.00	1.00	1.00	3000
weighted avg	1.00	1.00	1.00	3000



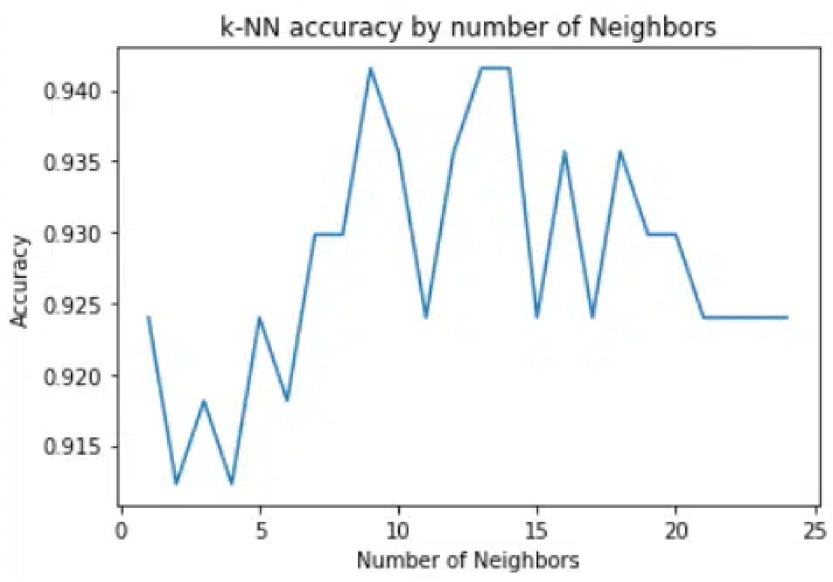
b. Malware Categories

Class repartition in percentage



i. KNN

<u>Choice of k :</u>



Evaluation on test data for k= 9 :

Accuracy Score: 0.95
Balanced Accuracy: 0.92

ii. CART

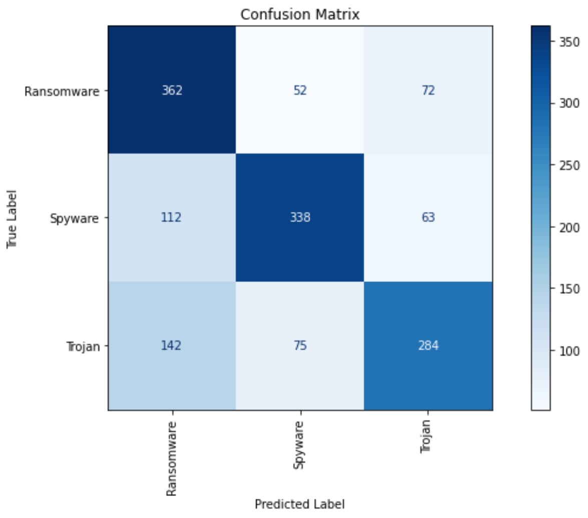
Accuracy Score: 0.93
Balanced Accuracy: 0.90
Matthews Correlation Coefficient: 0.90

	precision	recall	f1-score	support
Trojan	0.90	0.98	0.94	441
Spyware	0.88	0.99	0.93	472
Ransomware	0.87	0.98	0.93	471
accuracy			0.93	2266
macro avg	0.93	0.93	0.93	2266
weighted avg	0.93	0.98	0.93	2266

iii. Random Forest

Accuracy Score: 0.67
Balanced Accuracy: 0.67
Matthews Correlation Coefficient: 0.51

	precision	**recall**	**f1-score**	**support**
Ransomware	0.63	0.69	0.66	486
Spyware	0.71	0.71	0.71	513
Trojan	0.69	0.62	0.65	501
accuracy			0.67	1500
macro avg	0.68	0.67	0.67	1500
weighted avg	0.68	0.67	0.67	1500

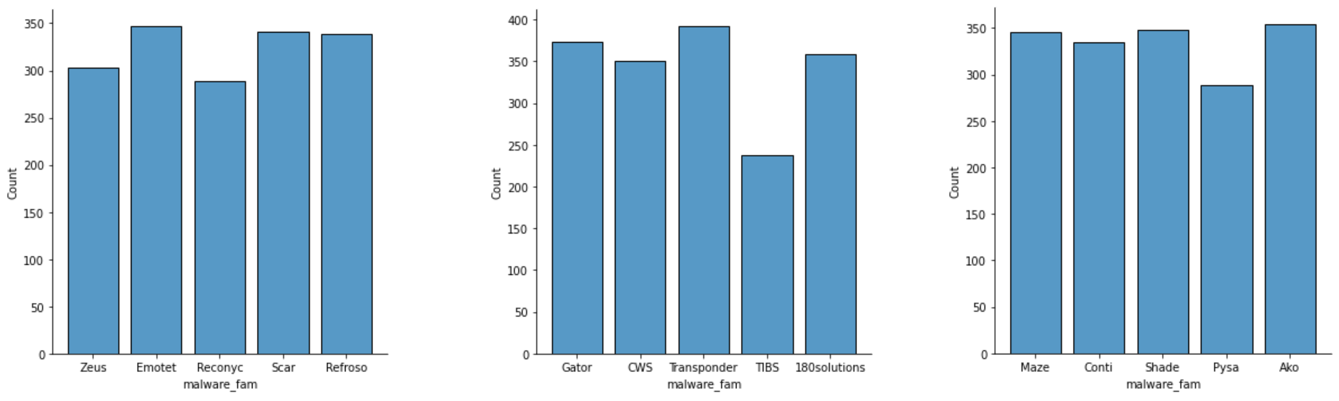


By reducing the size of the dataset below 10^4, we observed a decrease in performance with an accuracy not exceeding 55%.

c. Families

Family repartitions in percentage

Trojan **Spyware** **Ransomware**



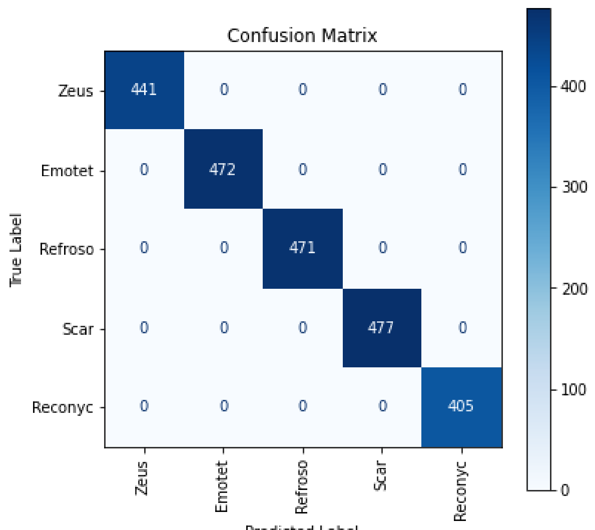
i. XGBoost

Trojan

<u>Evaluation on test data</u>

Accuracy Score: 1.0
Balanced Accuracy: 1.0
Matthews Correlation Coefficient: 1.0

	precision	recall	f1-score	support
zous	1.0	1.0	1.0	441
emotet	1.0	1.0	1.0	472
refroso	1.0	1.0	1.0	471
scar	1.0	1.0	1.0	477
accuracy			1.0	2266
macro avg	1.0	1.0	1.0	2266
weighted avg	1.0	1.0	1.0	2266

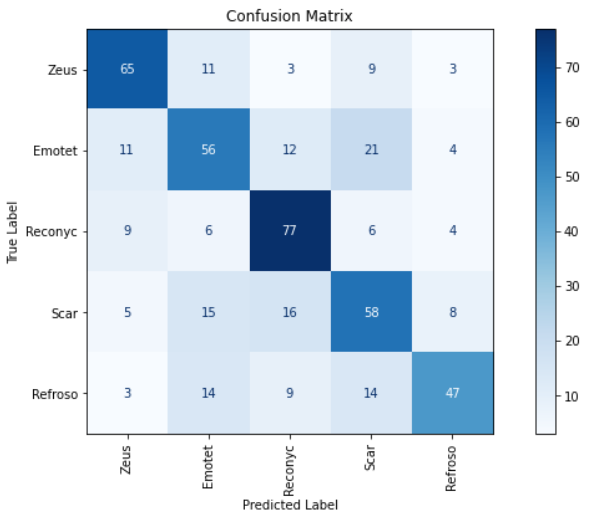


ii. MLP

Trojan

Accuracy Score: 0.8
Balanced Accuracy: 0.82
Matthews Correlation Coefficient: 0.75

	precision	**recall**	**f1-score**	**support**
Zeus	*0.85*	*0.79*	*0.81*	91
Emotet	*0.87*	*0.80*	*0.83*	*104*
Reconyx	0.79	0.88	0.83	102
Scar	0.9	0.73	0.8	102
Refroso	0.75	0.94	0.83	87
accuracy			*0.82*	*486*

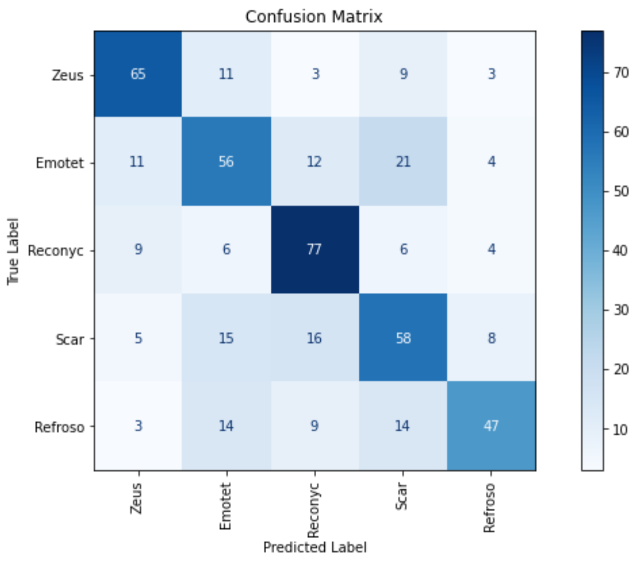


iii. Random Forest

- **Trojan**

Accuracy Score: 0.62
Balanced Accuracy: 0.62
Matthews Correlation Coefficient: 0.53

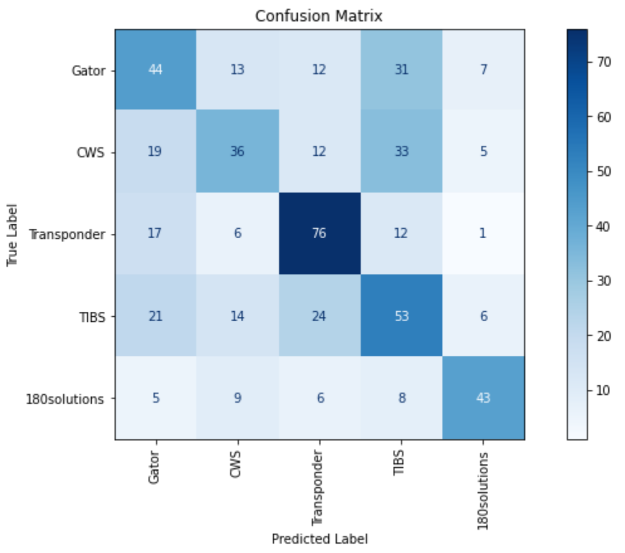
	precision	**recall**	**f1-score**	**support**
Zeus	0.70	0.71	0.71	91
Emotet	0.55	0.54	0.54	104
Reconyx	0.66	0.75	0.70	102
Scar	0.54	0.57	0.55	102
Refroso	0.71	0.54	0.61	87
accuracy			0.62	486
macro avg	0.63	0.62	0.62	486
weighted avg	0.63	0.62	0.62	486



- **Spyware**

Accuracy Score: 0.49
Balanced Accuracy: 0.49
Matthews Correlation Coefficient: 0.36

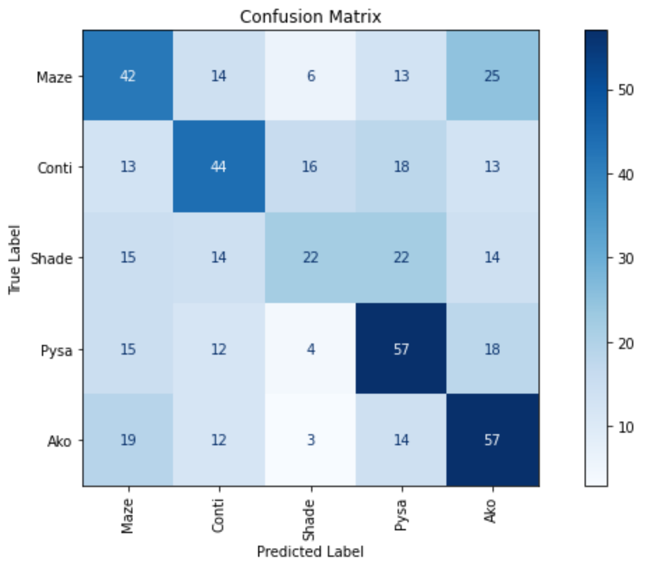
	precision	**recall**	**f1-score**	**support**
Gator	0.42	0.41	0.41	107
CWS	0.46	0.34	0.39	105
Transponder	0.58	0.68	0.63	112
TIBS	0.39	0.45	0.42	118
180solutions	0.69	0.61	0.65	71
accuracy			0.49	513
macro avg	0.51	0.50	0.50	513
weighted avg	0.49	0.49	0.49	513



- **Ransomware**

Accuracy Score: 0.44
Balanced Accuracy: 0.43
Matthews Correlation Coefficient: 0.30

	precision	recall	f1-score	support
Maze	0.40	0.42	0.41	100
Conti	0.46	0.42	0.44	104
Shade	0.43	0.25	0.32	87
Pysa	0.46	0.54	0.50	106
Ako	0.45	0.54	0.49	105
accuracy			0.44	502
macro avg	0.44	0.44	0.43	502
weighted avg	0.44	0.44	0.44	502



3. Final Architecture ⬆

After comparing the performance of the different tested models for each classification type, we suggest using the following architecture because it has the best performance:

- ❖ Transformer: Scaling and PCA
- ❖ Classifier:
  - Benign/Malware: SVM
  - Malware Category: Random Forest
  - Malware Family:
    - ➔ Trojan: Random Forest
    - ➔ Spyware: Random Forest
    - ➔ Ransomware: Random Forest

4. Advantages

❖ Integrated to “scikit-learn” environment:

Apart from the architecture of the model, we made sure that its implementation respects the constraints of defining models in “scikit-learn”. In fact, we implemented two classes:

- One for the preprocessing part, called **MalwareTransformer**. It inherits from the two classes **TransformerMixin** and **BaseEstimator** and implements two methods "***fit()***" and "***transform()***".
- The other one is used for defining the models that will be used for the classification. This class is called **MalwareClassifier**, inherits from the two classes **ClassifierMixin** and **BaseEstimator** and implements two methods "***fit()***" and "***predict()***".

Hence these classes can be called as any other preprocessing or classification algorithm of the "scikit-learn" library.

These two classes are then used by a superclass that we called **MalwareDetector** that handles the instantiation of the "transformer" and the "classifier" and the preparation of the data.

❖ Generalized to any classification situation

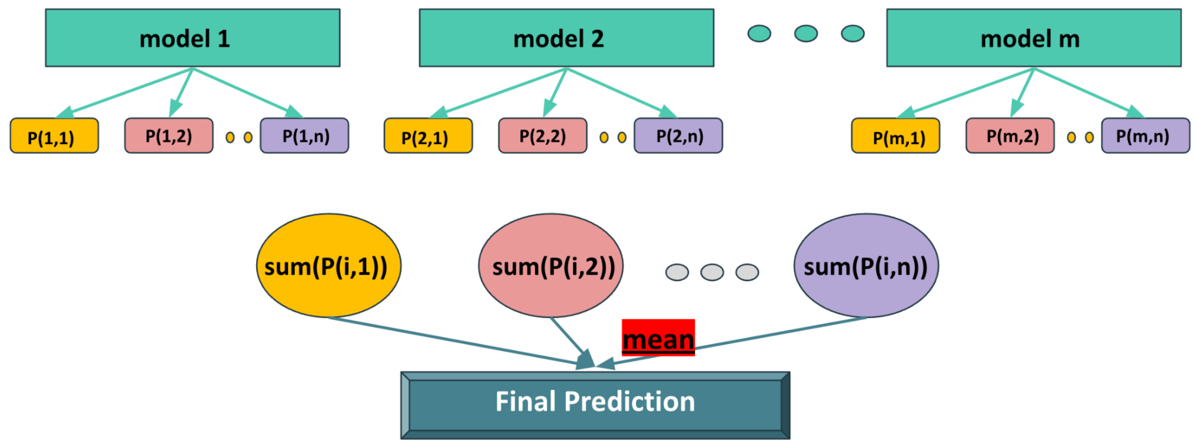
The proposed meta-model can handle any type of classification problem with any architecture (for example: Benign/Malware -> Malware Categories -> Malware Families for our situation). This architecture can be explained and specified in the `target_classes` parameter. For instance, the user can specify the models they want to use for every classification level which gives them the ability to create an infinite number of models in order to find the adequate model architecture for the classification problem in question. Hence, the meta-model handles the data preparation, the models creation, the fitting, the prediction, the report generation…

❖ Collaborative Prediction

The model offers the user the ability to specify multiple models to use for a specific classification subproblem. In this situation, all the models fit on the data related to the specified subproblem and predict the classes' probabilities. Once the probabilities are generated, two types of vote are available:

(Let P(i,j) be the predicted probability of the model 'i' that the sample belongs to the class 'j')

- **mean:** $max_j(\sum_{i=0}^{n} P(i,j)/n)$
- **max:** $max_{i,j} P(i,j) \quad \forall\, 0 \preccurlyeq i \prec n, 0 \preccurlyeq j \prec m$



IV. Meta-Model Performance Evaluation

1. Metrics

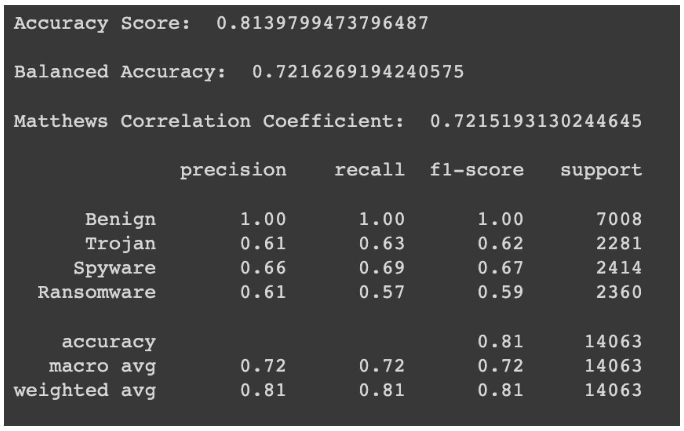
We selected multiple metrics in order to efficiently evaluate the model. In fact, we used some well known metrics such as:

- accuracy
- precision & recall
- confusion matrix
- f1-score

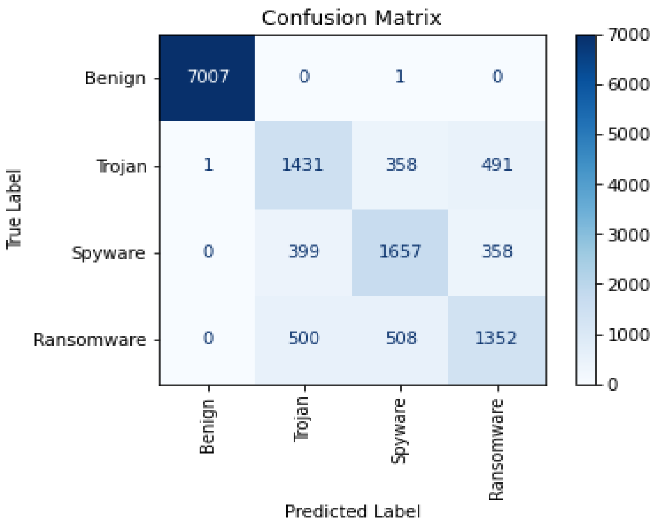
But also some other metrics that gives a better idea of the performance of our model when data is not scaled and well distributed between the classes, such as:

- Balanced Accuracy
- Matthews Correlation Coefficient
- Precision Recall Curve
- Receiver Operating Characteristic Curve

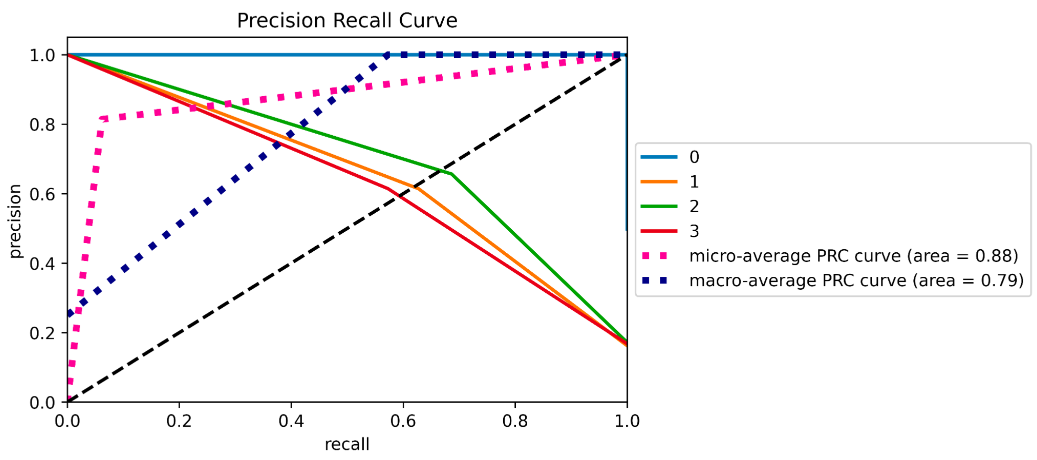
All of these metrics are generated and measured by the class 'Outils'. It offers the user a great tool to easily display multiple plots and different evaluation metrics.
Let's take a look at the final report:

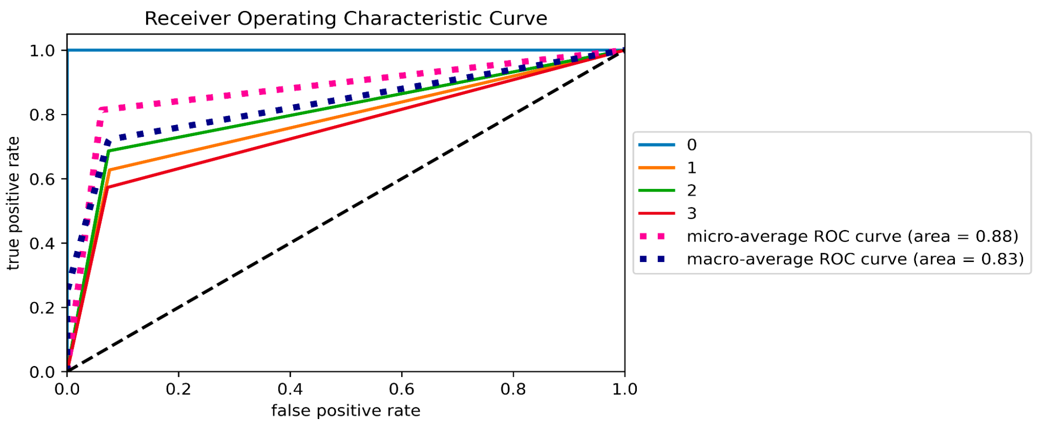


```
Accuracy Score:  0.8139799473796487

Balanced Accuracy:  0.7216269194240575

Matthews Correlation Coefficient:  0.7215193130244645

              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00      7008
      Trojan       0.61      0.63      0.62      2281
     Spyware       0.66      0.69      0.67      2414
  Ransomware       0.61      0.57      0.59      2360

    accuracy                           0.81     14063
   macro avg       0.72      0.72      0.72     14063
weighted avg       0.81      0.81      0.81     14063
```



True Label \ Predicted Label	Benign	Trojan	Spyware	Ransomware
Benign	7007	0	1	0
Trojan	1	1431	358	491
Spyware	0	399	1657	358
Ransomware	0	500	508	1352





2. Model scalability ⬆

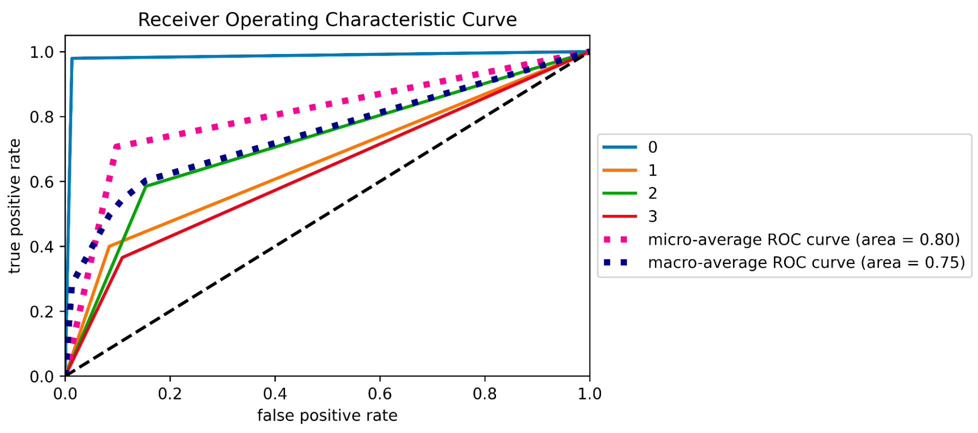
★ ***Size:*** *10^3*

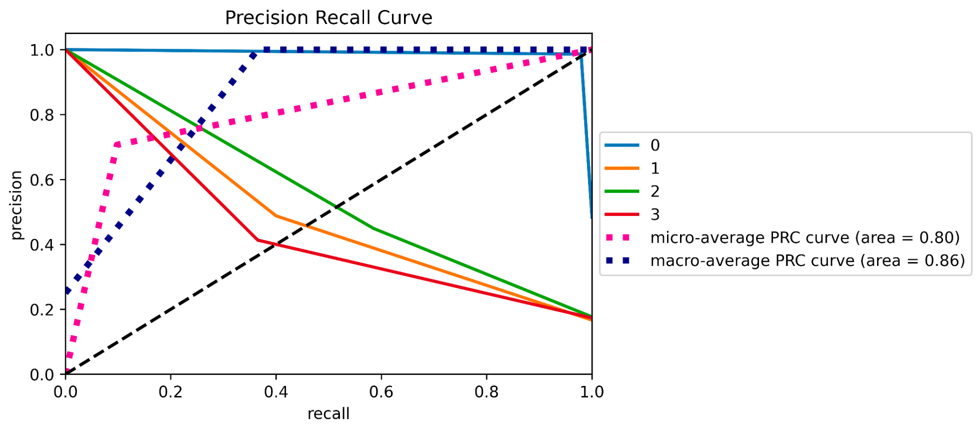
Accuracy Score: *0.71*

Balanced Accuracy: *0.58*

Matthews Correlation Coefficient: *0.57*

	precision	recall	f1-score	support
Bening	0.99	0.98	0.98	145
Trojan	0.49	0.40	0.44	50
Spyware	0.45	0.58	0.51	53
Ransomware	0.41	0.37	0.39	52
Accuracy			0.71	300
Macro Avg	0.58	0.58	0.58	300
Weighted Avg	0.71	0.71	0.71	300





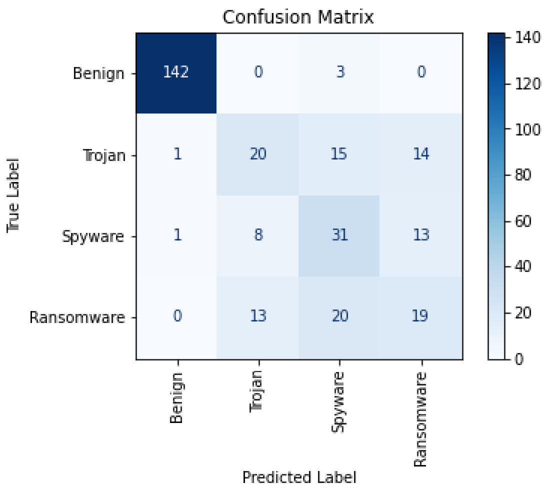
★ ***Size:*** *10^4*

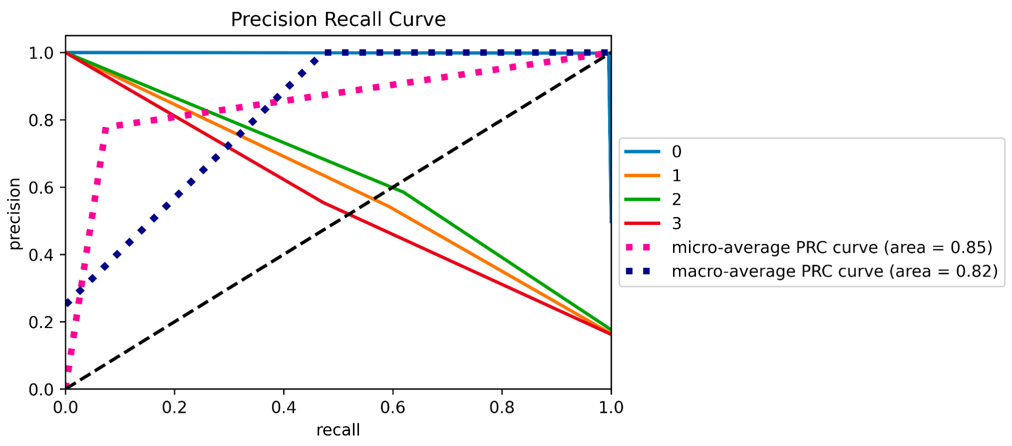
Accuracy Score: *0.78*

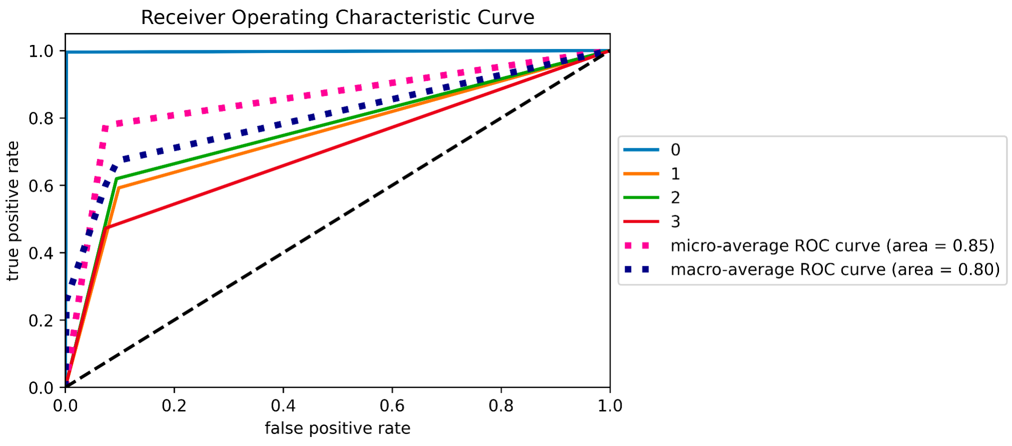
Balanced Accuracy: *0.67*

Matthews Correlation Coefficient: *0.67*

	precision	*recall*	*f1-score*	*support*
Benign	*1.00*	*1.00*	*1.00*	*1492*
Trojan	*0.54*	*0.59*	*0.57*	*493*
Spyware	*0.58*	*0.62*	*0.60*	*528*
Ransomware	*0.55*	*0.47*	*0.51*	*487*
accuracy			*0.78*	*3000*
macro avg	*0.67*	*0.67*	*0.67*	*3000*
weighted avg	*0.78*	*0.78*	*0.78*	*3000*

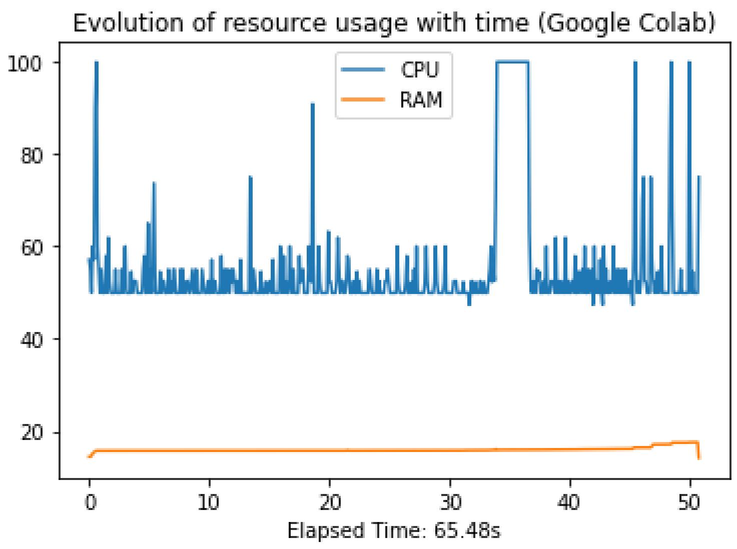




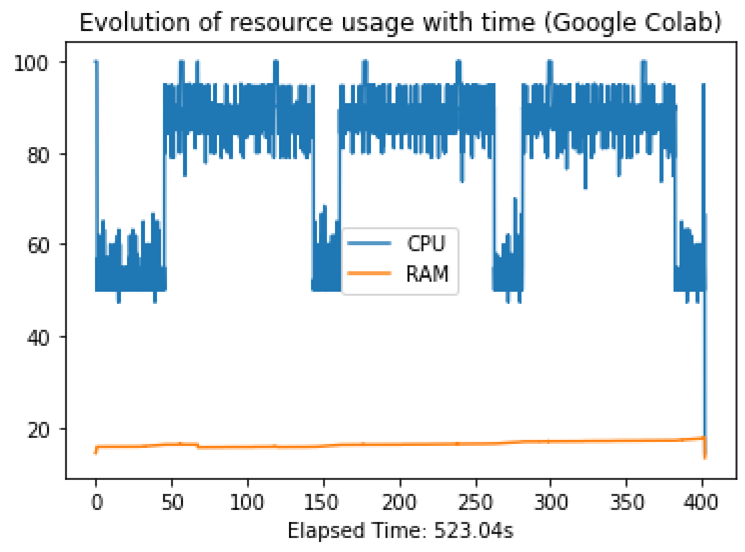


3. Resource consumption ⬆

i. Malware Category



ii. Malware Family



V. Conclusion

After implementing multiple scikit-learn algorithms for the malware detection problem, we decided to make them in a unique pipeline. This end-to-end tool offers the users the ability of specifying their own architecture based on the classification problem they work in. Hence, the pipeline is generalized for any classification problem and for any purpose.

The evaluation of this pipeline showed that it performs well in detecting the malware traffic (accuracy equals to 99%). It showed that it is also able to classify the malware traffic into its three categories (Trojan, Spyware and ransomware) with an accuracy that achieves 85%. However, the drawback of the meta-model is the classification of the malware families. This can be explained by the small amount of data that every family has (16% of the global size of our dataset). However, we believe that adding some advanced preprocessing algorithms and a different vote method in the prediction can improve the performance of our model.

Finally, the meta-model performs better than the first approach based on using one model to predict the families. In fact, the distribution of the problem into multiple small subproblems helps better detect the malware traffic and classify it to its corresponding family.