Evasive PDF Sample

Gonçalo Costa, up202103336

João Correia, up202005015

Ricardo Vieira, up202005091

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence 23/24

The chosen dataset is a collection of evasive PDF samples, labeled as malicious (1) or benign (0). Since the dataset has an evasive nature, it can be used to test the robustness of trained PDF malware classifiers against evasion attacks. The dataset contains 500,000 generated evasive samples, including 450,000 malicious and 50,000 benign PDFs.

This resource aims to support researchers and cybersecurity professionals in developing more advanced and robust detection mechanisms for PDF-based malware.

It's now our job to use this dataset to do exactly that: create a detection mechanism for PDF-based malware.

- Trad, F.; Hussein, A.; Chehab, A. Leveraging Adversarial Samples for Enhanced Classification of Malicious and Evasive PDF Files. Appl. Sci. 2023, 13, 3472. https://doi.org/10.3390/app13063472
- https://www.kaggle.com/datasets/fouadtrad2/evasive-pdf-samples
- Maryam Issakhani, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari1, "PDF Malware Detection Based on Stacking Learning", The International Conference on Information Systems Security and Privacy, February 2022
- https://www.unb.ca/cic/datasets/pdfmal-2022.html

PROBLEM DEFINITION & REFERENCES

Programming Language

Tools

Algorithms used

We checked the accuracy and other stats of many algorithms and ended up using the following:

- XGBoost
- LGBoost
- AdaBoost
- ExtraTrees

Dataframes

	pdfsize	pages	title characters	images	obj	endobj	stream	endstream	xref	trailer	...	ObjStm	JS	OBS_JS	Javascript	OBS_Javascript	OpenAction	OBS_OpenAction	Acroform	OBS_Acroform	class
0	644.326	70	0	1	348	351	128	128	1	1	...	0	1	0	1	0	1	0	1	0	1
1	648.050	68	0	1	348	345	124	124	1	1	...	0	1	0	1	0	0	0	1	0	1
2	696.506	68	0	1	353	353	128	125	1	1	...	0	1	0	1	0	0	0	1	0	1
3	715.926	68	0	0	759	667	250	192	1	1	...	0	1	0	1	0	1	0	1	0	1
4	707.102	70	10	2	388	373	141	138	1	1	...	0	1	0	1	0	1	0	1	0	1

TOOLS AND ALGORITHMS

TREATING THE DATA

At this point we saw that cleaning the dataset was something needed to be done. We started by removing the columns that did not add any information as they were always zero.

```
columns_max_min_zero = [col for col in df.columns if df[col].max() == df[col].min()] #get the columns where max and min are 0

print(columns_max_min_zero) # print the name of the columns

if (columns_max_min_zero):
    df.drop(columns=columns_max_min_zero, inplace=True) #remove the columns from the dataset
```

```
['OBS_JS', 'OBS_Javascript', 'OBS_OpenAction', 'OBS_Acroform']
```

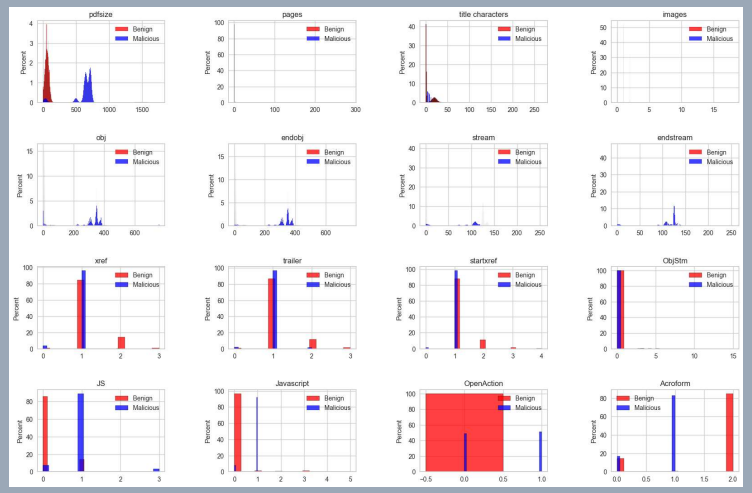
At this point we realized that there were rows with values zero where it did not make sense, like the number of pages, or its size. So we removed the rows where the values didn't make sense.

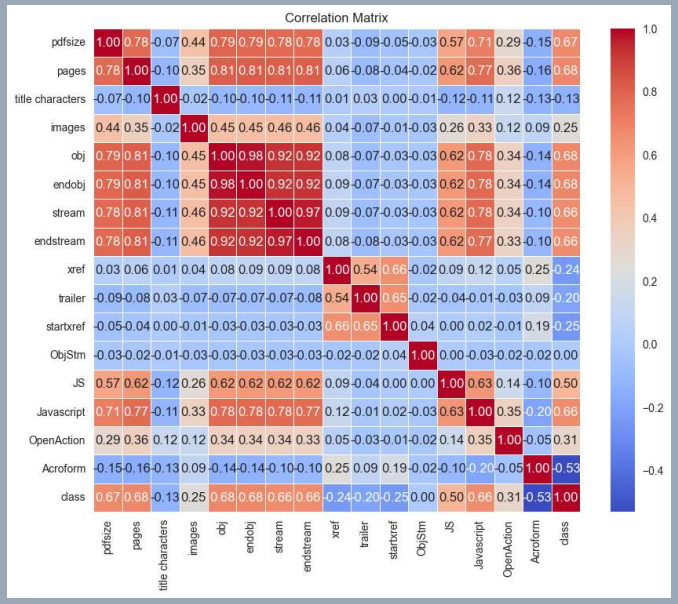
```
nonsenses = (df['pdfsize'] == 0) | (df['pages'] == 0) # get the rows where pages is zero or pdfsize is zero

print('Number of rows removed:', nonsenses.sum())

df = df.drop(df[nonsenses].index) # remove those rows from the dataset
```

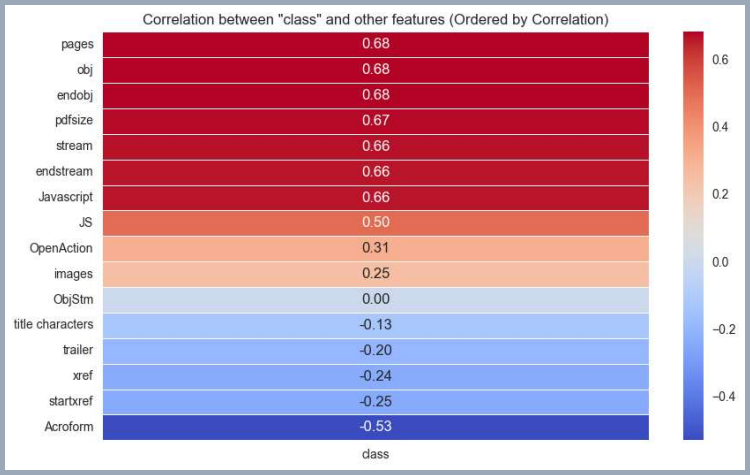
```
Number of rows removed: 25878
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 474122 entries, 0 to 499999
Data columns (total 17 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   pdfsize           474122 non-null  float64
 1   pages             474122 non-null  int64
 2   title characters  474122 non-null  int64
 3   images            474122 non-null  int64
 4   obj               474122 non-null  int64
 5   endobj            474122 non-null  int64
 6   stream            474122 non-null  int64
 7   endstream         474122 non-null  int64
 8   xref              474122 non-null  int64
 9   trailer           474122 non-null  int64
 10  startxref         474122 non-null  int64
 11  ObjStm            474122 non-null  int64
 12  JS                474122 non-null  int64
 13  Javascript        474122 non-null  int64
 14  OpenAction        474122 non-null  int64
 15  Acroform          474122 non-null  int64
 16  class             474122 non-null  int64
dtypes: float64(1), int64(16)
memory usage: 65.1 MB
```

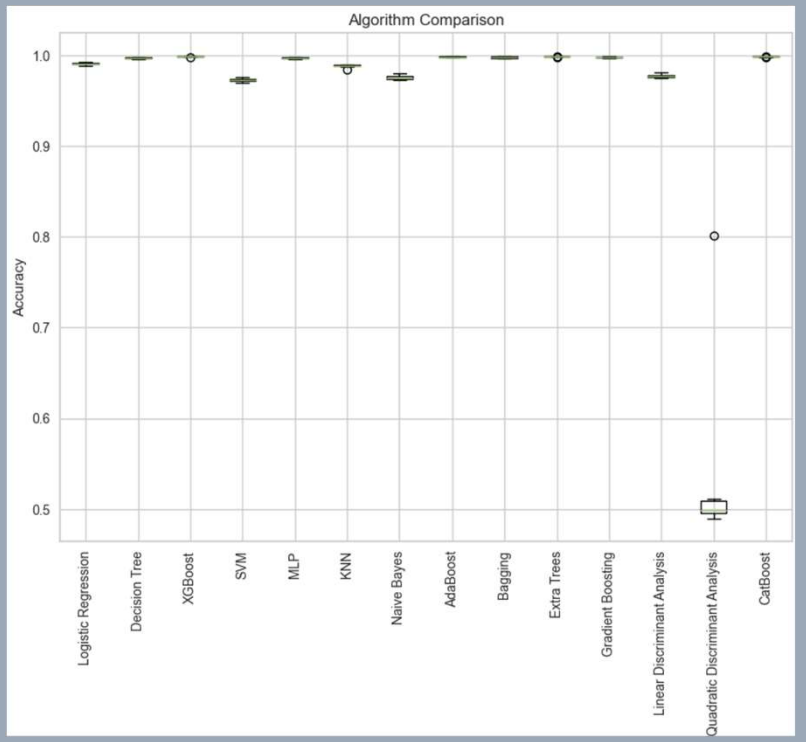




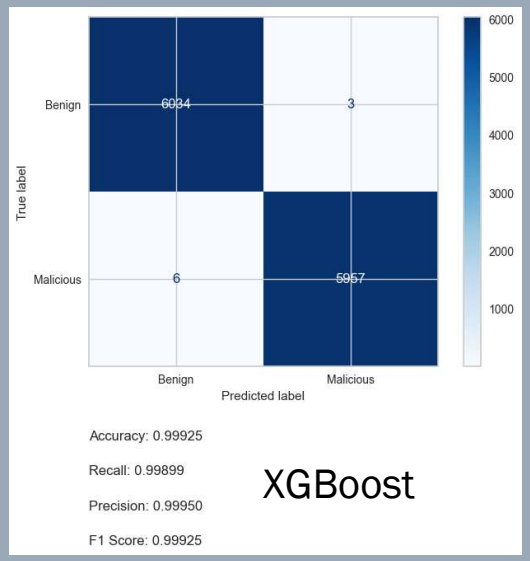


DATA ANALYSIS

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9991	1.0000	0.9988	0.9994	0.9991	0.9981	0.9981	0.604
lightgbm	Light Gradient Boosting Machine	0.9991	1.0000	0.9988	0.9994	0.9991	0.9982	0.9982	0.718
catboost	CatBoost Classifier	0.9990	1.0000	0.9986	0.9993	0.9990	0.9980	0.9980	8.394
rf	Random Forest Classifier	0.9989	1.0000	0.9983	0.9995	0.9989	0.9978	0.9978	1.421
et	Extra Trees Classifier	0.9989	1.0000	0.9985	0.9994	0.9989	0.9979	0.9979	1.013
ada	Ada Boost Classifier	0.9987	1.0000	0.9984	0.9989	0.9987	0.9973	0.9973	1.087
gbc	Gradient Boosting Classifier	0.9983	1.0000	0.9977	0.9989	0.9983	0.9966	0.9966	2.440
dt	Decision Tree Classifier	0.9973	0.9973	0.9970	0.9976	0.9973	0.9947	0.9947	0.569
knn	K Neighbors Classifier	0.9958	0.9990	0.9935	0.9980	0.9958	0.9916	0.9916	1.334
svm	SVM - Linear Kernel	0.9892	0.9982	0.9858	0.9926	0.9892	0.9785	0.9785	0.528
lr	Logistic Regression	0.9890	0.9993	0.9852	0.9928	0.9890	0.9780	0.9780	1.093
qda	Quadratic Discriminant Analysis	0.9803	0.9986	0.9619	0.9986	0.9799	0.9606	0.9612	0.504
ridge	Ridge Classifier	0.9779	0.9991	0.9562	0.9995	0.9774	0.9557	0.9566	0.470
lda	Linear Discriminant Analysis	0.9779	0.9991	0.9562	0.9995	0.9774	0.9557	0.9566	0.511
nb	Naive Bayes	0.9656	0.9880	0.9482	0.9824	0.9650	0.9312	0.9318	0.546
dummy	Dummy Classifier	0.5000	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.478



ALGORITHM STATISTIC COMPARISON



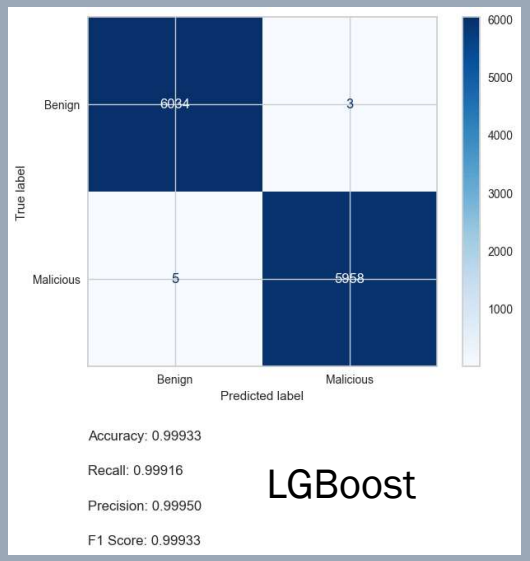
Accuracy: 0.99925

Recall: 0.99899

Precision: 0.99950

F1 Score: 0.99925

XGBoost



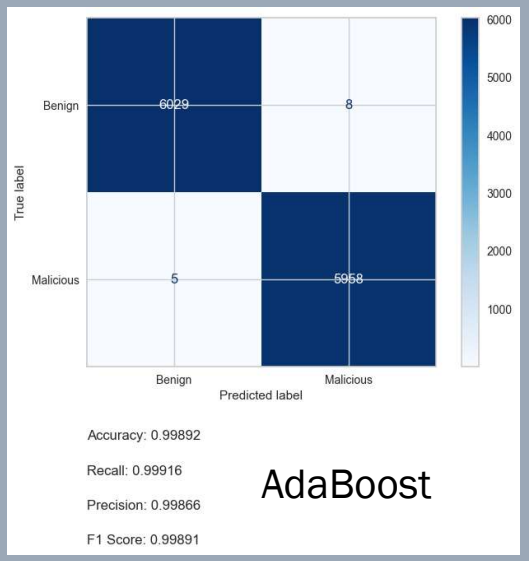
Accuracy: 0.99933

Recall: 0.99916

Precision: 0.99950

F1 Score: 0.99933

LGBoost



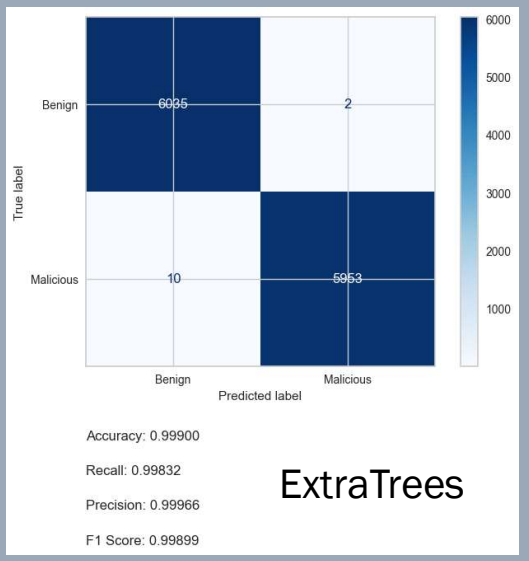
Accuracy: 0.99892

Recall: 0.99916

Precision: 0.99866

F1 Score: 0.99891

AdaBoost



Accuracy: 0.99900

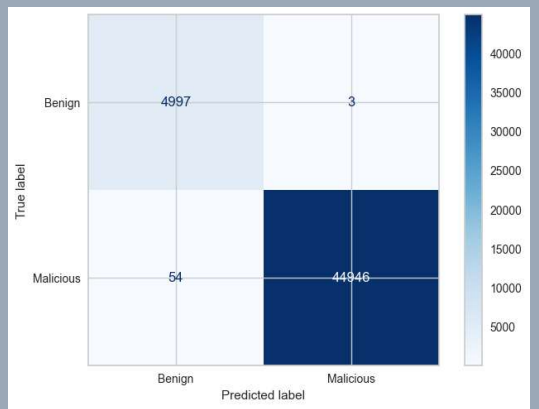
Recall: 0.99832

Precision: 0.99966

F1 Score: 0.99899

ExtraTrees

50/50 TRANING 50/50 TESTING



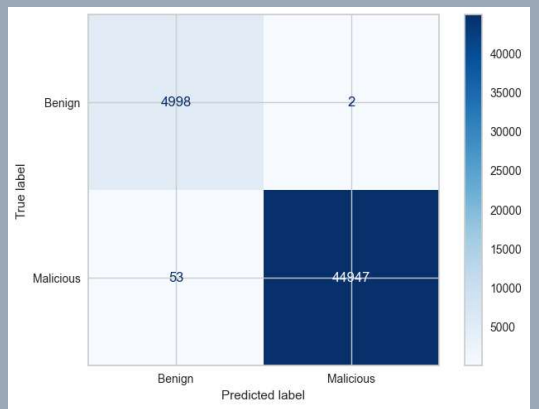
Accuracy: 0.99886

Recall: 0.99880

Precision: 0.99993

F1 Score: 0.99937

XGBoost



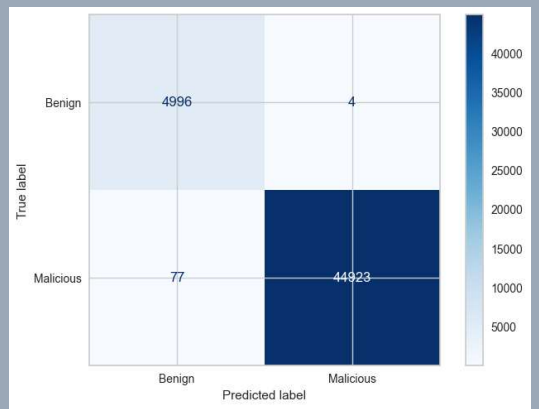
Accuracy: 0.99890

Recall: 0.99882

Precision: 0.99996

F1 Score: 0.99939

LGBoost



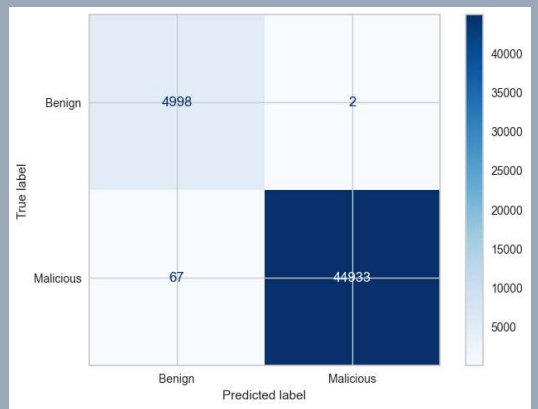
Accuracy: 0.99838

Recall: 0.99829

Precision: 0.99991

F1 Score: 0.99910

AdaBoost



Accuracy: 0.99862

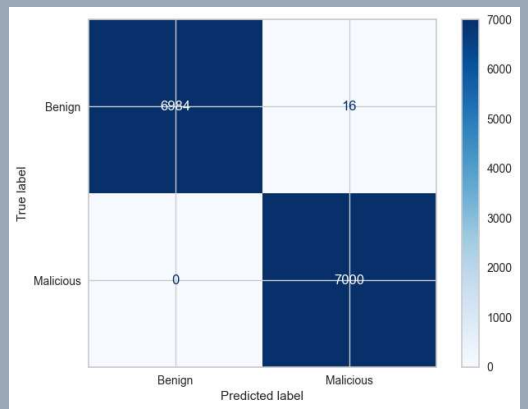
Recall: 0.99851

Precision: 0.99996

F1 Score: 0.99923

ExtraTrees

50/50 TRANING 90/10 TESTING



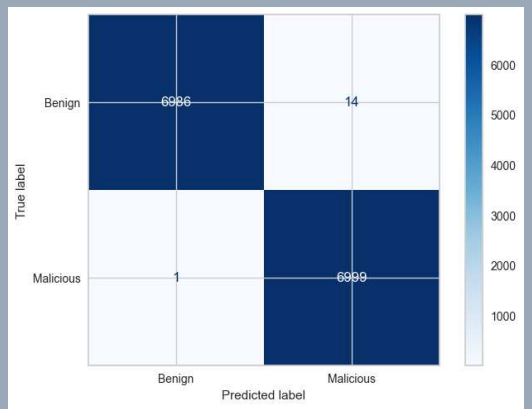
Accuracy: 0.99886

Recall: 1.00000

Precision: 0.99772

F1 Score: 0.99886

XGBoost



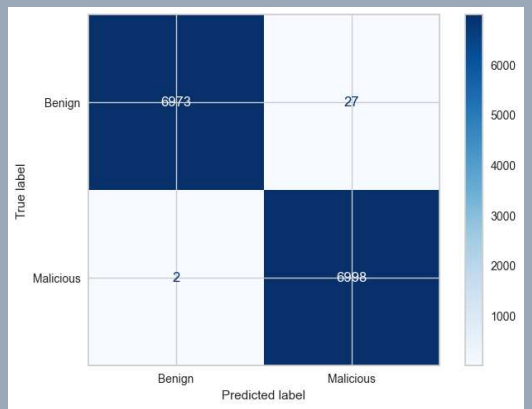
Accuracy: 0.99893

Recall: 0.99986

Precision: 0.99800

F1 Score: 0.99893

LGBoost



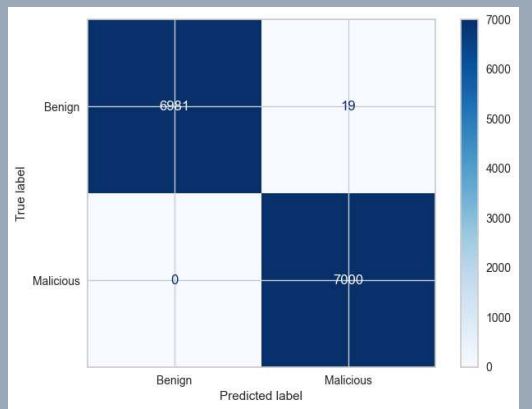
Accuracy: 0.99793

Recall: 0.99971

Precision: 0.99616

F1 Score: 0.99793

AdaBoost



Accuracy: 0.99864

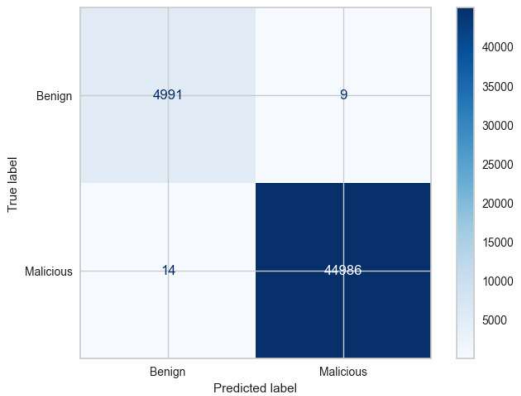
Recall: 1.00000

Precision: 0.99729

F1 Score: 0.99864

ExtraTrees

90/10 TRANING 50/50 TESTING



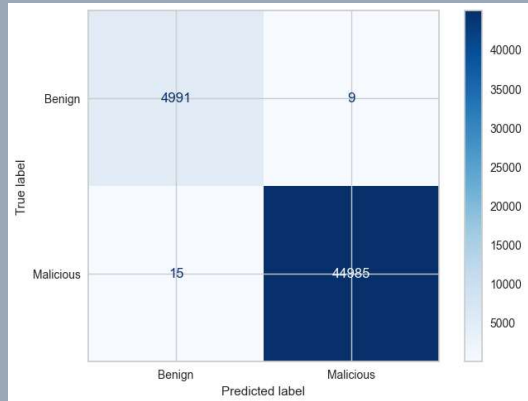
Accuracy: 0.99954

Recall: 0.99969

Precision: 0.99980

F1 Score: 0.99974

XGBoost



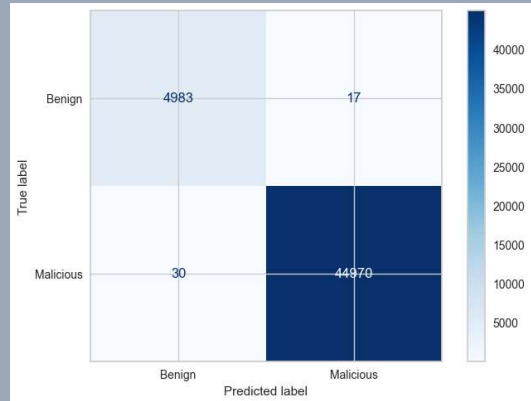
Accuracy: 0.99952

Recall: 0.99967

Precision: 0.99980

F1 Score: 0.99973

LGBoost



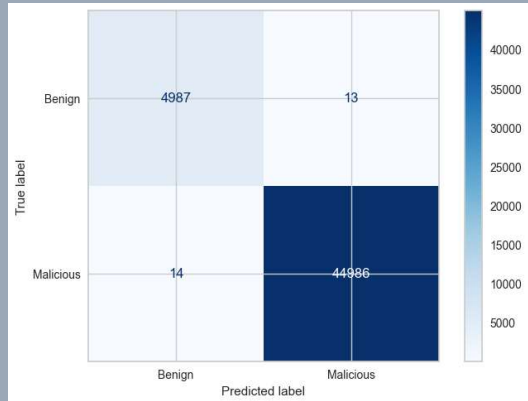
Accuracy: 0.99906

Recall: 0.99933

Precision: 0.99962

F1 Score: 0.99948

AdaBoost



Accuracy: 0.99946

Recall: 0.99969

Precision: 0.99971

F1 Score: 0.99970

ExtraTrees

90/10 TRANING 90/10 TESTING

As a conclusion, we were able to build a model to predict if a PDF contains some type of unwanted content. We were able to achieve a good accuracy, and the model was able to predict the malicious PDFs with a good precision.

The algorithms that presented the best results were Extreme Gradient Boosting, Light Gradient Boosting, AdaBoost and Extra Trees.
In each algorithm when we compared the results of the different approach when can see that the results are very similar.
We can conclude that cleaning the outsiders is not necessary, as the results are very similar.

Comparing the results when we give different percentages of benign and malign PDFs to train and test the model, even though the results are very similar, given the conclusions we want to take from this project, we assume that the most appropriate distribution is to give a train set with 90% benign and 10% malign PDFs and a test set with 50% benign and 50% malign PDFs.
This is the distribution where the most False Positives happen, because we consider to be best to say a given PDF is malign and it is not than to say it is benign and it is not.
When we trained with 50-50 and tested with 90-10 we can see the opposite condition, where the most False Negatives happens. In this case the algorithms consider that the PDF is benign when it is malign. We consider this situation dangerous because it gives a feeling of not danger when there is.

Finally, it is visible that our results were considerably good, with accuracies around 0.99 in every model and approach. This results can be explained with the amount of data that we had, making it possible to train the model with large and significant data, so that the model could really understand the difference between the data.
Besides, as we can see from our Feature Importance graph, two columns jump out, the "Acroform" and "pages". Connecting this information with our correlation matrix, we can see that the pages has a high positive correlation, while the Acroform has a high negative correlation. This means that the number of pages is a good predicator for a pdf to be malicious, while the Acroform is a good predicator for a pdf to be benign.

CONCLUSIONS