CSEC 620 | Final Project Report

Team 4 - Nick Mangerian, Jacob Ruud, Hugo Tessier Due on 12-12-2021

Malicious PDF classification has been a problem faced by endpoint detection solutions for years, and is only continuing to get more popular as email filtering cracks down on link sharing. From 2019-2020, Palo Alto Networks noticed a 1,160% increase in malicious PDF files used in phishing attacks. From their Unit 42 threat intel team: “PDF files are an enticing phishing vector as they are cross-platform and allow attackers to engage with users, making their schemes more believable as opposed to a text-based email with just a plain link.”[2] This rapid growth makes being able to classify malicious PDF files even more important, making it the perfect subject for our term project.

Unfortunately, finding a good dataset of PDF’s to use for classification is hard since most research on the subject is being done by private companies who choose to keep their dataset confidential. Luckily, we were able to find a dataset that was compiled in 2013 containing all types of malicious files to use in signature analysis[1]. This dataset is composed of 19,979 samples with 10,979 malicious files and 9,000 benign pdfs. Once we extracted the pdf files from the rest of the dataset, we then set about the task of extracting meaningful features from the PDF files to use in a binary classifier.

1. Analysis of the dataset

- **Feature extraction with Exiftool**

Feature extraction was performed through the use of a command-line tool called “Exiftool”. This tool extracts the metadata from files. This tool also allowed for the creation of a csv table of the pdf files metadata organized by metadata tags as columns and a single sample for each row. This was performed with the command “`exiftool.exe -r -csv ./data/benign ./data/malware > metadata.csv`” , which created a csv of the metadata. The csv file was then imported into code as a pandas dataframe. The pandas array was then converted to a numpy array, therefore it could be used by the sklearn machine learning module. This was done by converting all non-numeric features by using one-hot encoding and conversion of boolean to integers to be used by the np.array.

- **Label extraction**

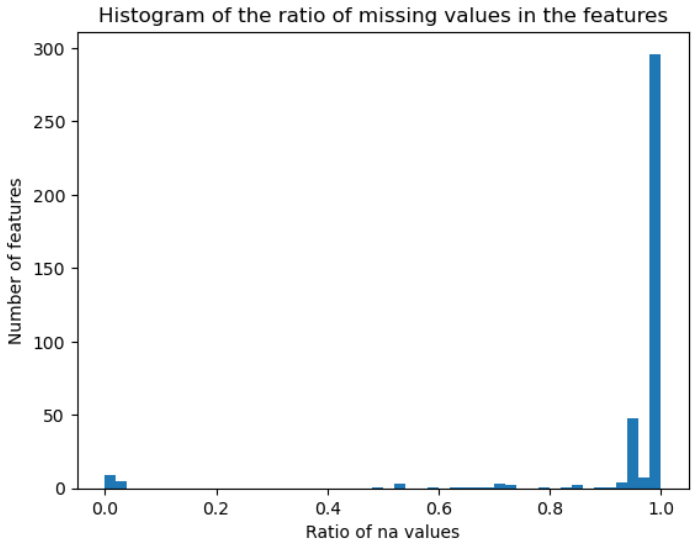
Label extraction was done through the use of paths of the samples. This is because samples were organized based upon whether the sample was malicious or benign. This was expedited due to the observation that malicious and benign samples followed their own naming schemes. This allowed for a simple check for the inclusion of either the string “malware” or “clean” in the path of samples to determine the samples’ appropriate labels.

- **Feature selection process**

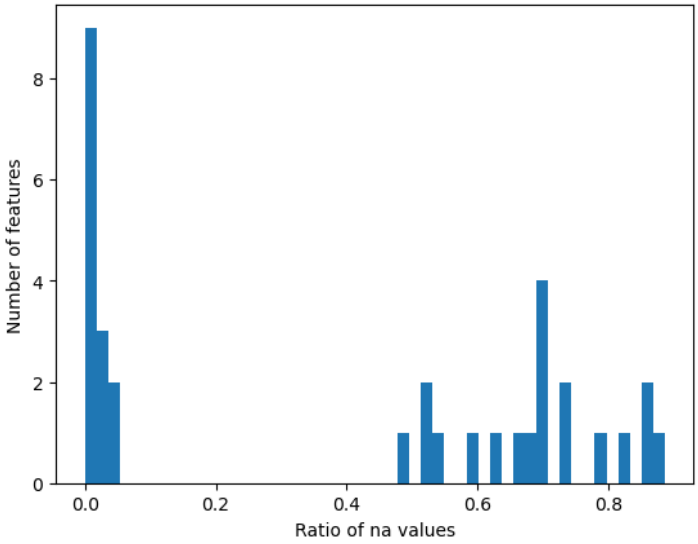
Before feature selection, there existed nearly 400 features. Although many issues were found with many of these features. One of the most apparent was that there were a lot of missing values for many of the features.

As it isn't possible to remove a pdf file from the dataset (we can't take the risk to discard a malicious pdf) all these values are considered empty values and not missing values. It means that there are two options: remove the entire attribute or replace the empty value.

The attributes with a too high rate of missing values are discarded. We estimate a threshold of removal by representing the histogram of missing values ratio.



As one can observe on this graph a lot of attributes only have empty values (around 290). By putting the threshold value to 0.9, we only keep 33 attributes and obtain the following histogram.



- **Data cleaning**

The kept variables are then cleaned one by one. All missing values had to be replaced as explained earlier. Moreover, some features can't be used in their raw form and can lead to the creation of new features. Some features also are in the wrong type and need to be imported correctly. Here are the main observations about the cleaning, one can find the entire process described in the `feature_selection` Ipython notebook. The strategy used for replacing empty values has been to create a new unknown category to the concerned features.

First we observed a strong correlation between `Author` and `Creator` features and decided to only keep one of those and create a new flag feature `isCreatorAuthor`.

For the `CreatorTool` feature, as there were a lot of categories some binning needed to be done. We first remove some information on the version, for example: *Adobe InDesign CS4 (6.0.6)* and *Adobe InDesign CS4 (6.1.2)* are combined into *Adobe InDesign CS4*. The empty values were replaced with "Unknow" and the categories with less than 20 samples are set to an "Other" category.

We created flags features if the file has a description, a title, keywords, etc. `hasDescription`, `hasKeywords`, `hasTitle`, etc.

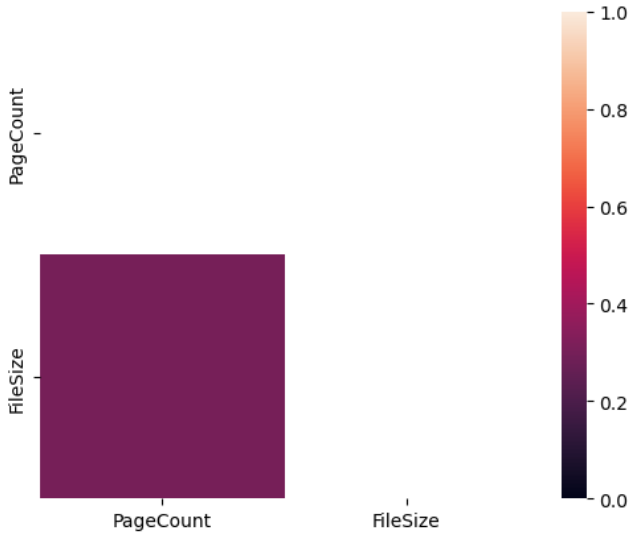
Moreover, the date had to be imported in the right format for this we used the python module `dateparser`. We extracted the creation and the modification date but we also created the modification time zone which can also give interesting geographic information about the file.

Here is a list of the features that are finally extracted after the cleaning:
`hasCreator`, `isAuthorCreator`, `ModificationTZ`, `CreatorTool`, `TaggedPDF`, `hasTitle`, `hasKeywords`, `hasSubject`, `CreationYear`, `ModificationYear`, `Linearized`, `PDFVersion`, `HasXFA` as categorical variables and `PageCount`, `FileSize` as continuous variables.

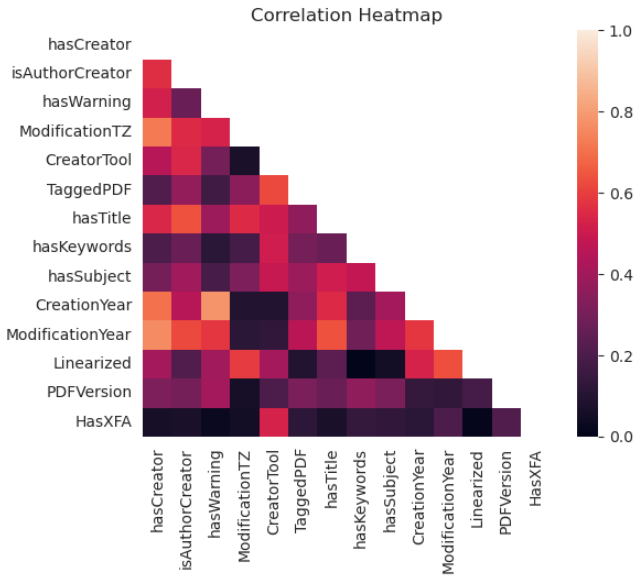
2. Data processing techniques

- Correlation study

The first step before preprocessing the data consisted of studying the correlation between the features. We don't want too much correlation as it would reduce the performance of our classifiers. We represent the correlation between the numerical features:



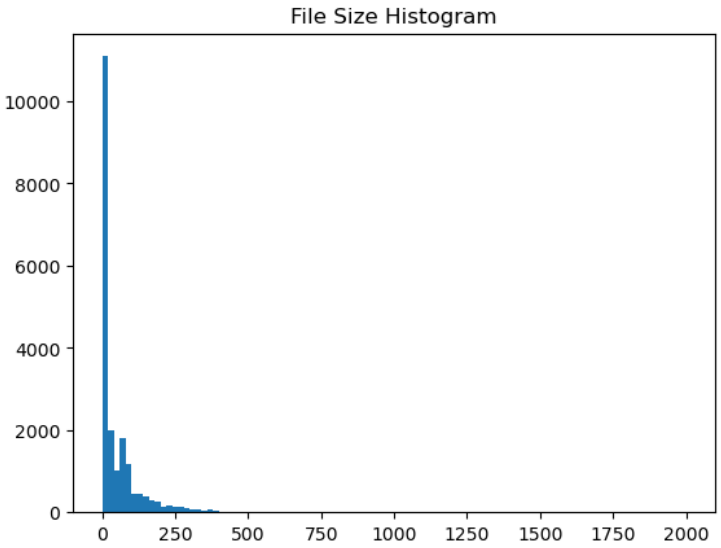
Correlation between categorical variables can also be calculated using Cramér's V which is based on chi-squared test. Here is the correlation matrix of the categorical features:

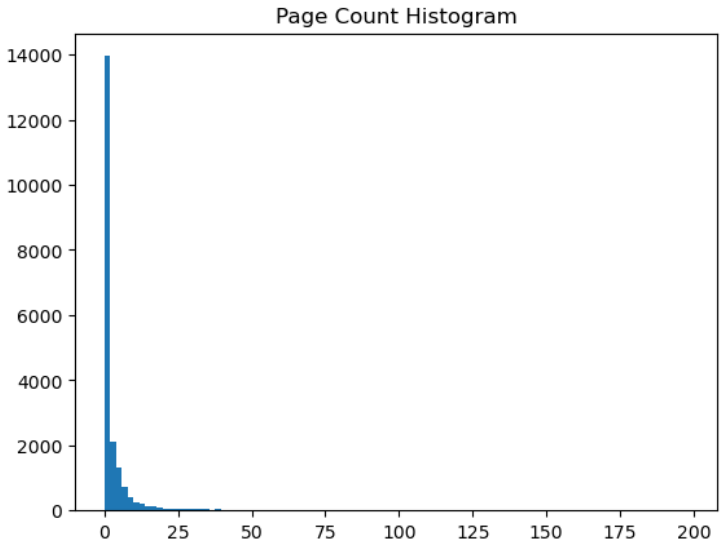


We observe that the chosen features are not correlated too much (the biggest correlation is 0.7 between `CreationYear` and `hasWarning`). Moreover, reducing the number of features isn't required and PCA would lead to a loss in explanatory power.

- **Normalization considerations**

As there is a mix between categorical and numerical variables it would be beneficial to normalize the data (to avoid scale imbalance). For categorical variables one-hot encoding will be performed. Reducing the number of categories as done in the previous step helps to not give too much importance to a feature. Concerning numerical variables, we need to look at their distribution with histograms:





We note that their values aren't normally distributed. They are strictly positive and look-zero inflated like Gamma distributions. Standardization can't be used as it would change the distribution. Therefore, we normalize these features by using the min-max scaler.

- **Data split for machine learning application**

Once these steps are done, the data need to be split into a training and testing set. This allows us to compare the performances of the classifiers without bias. As some classifiers need their hyperparameters to be tuned, we also split the training set into training and validation sets. We could have used cross validation for the tuning but as there are a lot of samples available using a separated validation set can be done. To summarize, the process for each classifier training consisted of: training the model on a small train set, tuning the hyperparameters on the validation set, once tuned training the model on the entire train set, and finally predicting the labels of the test samples. This method leads to no overfitting bias during the comparison of the algorithms.

3. Machine Learning techniques

The study consists of supervised learning -as the .pdf files are labeled- with two categories. The machine learning algorithms considered will then perform binary classification. The key characteristics of our dataset are that there is a large amount of samples and that their types are

mixed between numerical and categorical variables. This excludes models such as KNN which is made for numerical values. The ratio between the number of features and the number of samples is high so the data won't be sparse too much. The models we decided to compare are: a Logistic Regression classifier, a Random Forest classifier, SVM and Kernel SVM classifiers, and a Neural Network classifier.

The metrics used are dependent on our security application which is malware classification. As we considered it to be an on-machine classifier, the model has to be light in storage, testing fast (we don't want the user to wait too much time as the model is analysing the .pdf), thus testing time is important. For malware classification, the most important is to have a low false positives rate (not too many false alerts) and a low false negative rate (not misclassifying any malicious file). Thus, the metric we will use is the F1-score which takes those two into consideration. We also look at the confusion matrices.

Classifiers

- **Logistic regression**

We first choose the logistic regression model as it is a good basis of comparison and that it may work well with the type of dataset we have. It also has a great explainability capacity which is important for malicious file detection. This algorithm doesn't require tuning.

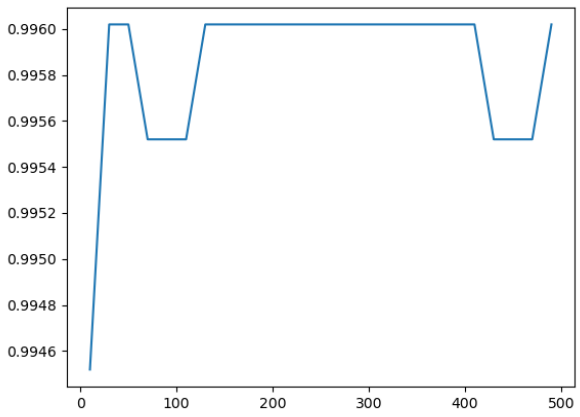
F1-score: 0.9902
Confusion matrix:

	Predicted Benign	Predicted Malicious
Actual Benign	0.4462	0.0052
Actual Malicious	0.0055	0.5430

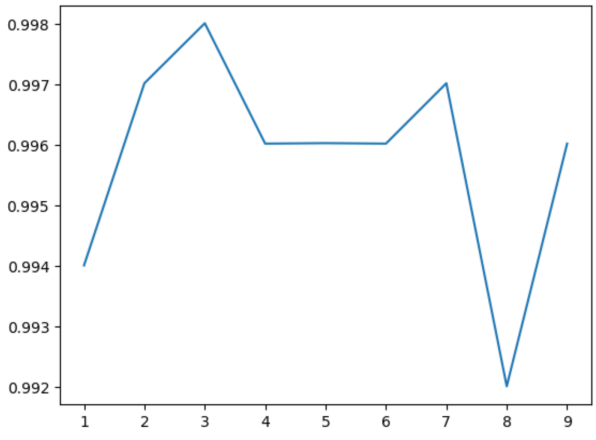
- **Random Forest**

Random Forest is known for its good performance for classification. We train this model and tune its four hyperparameters: the number of trees, the feature subcount, the maximum tree depth and the minimum number of samples per leaf node.

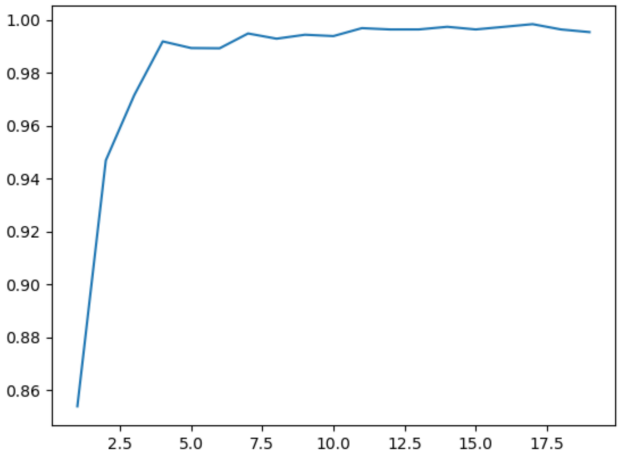
Number of trees:



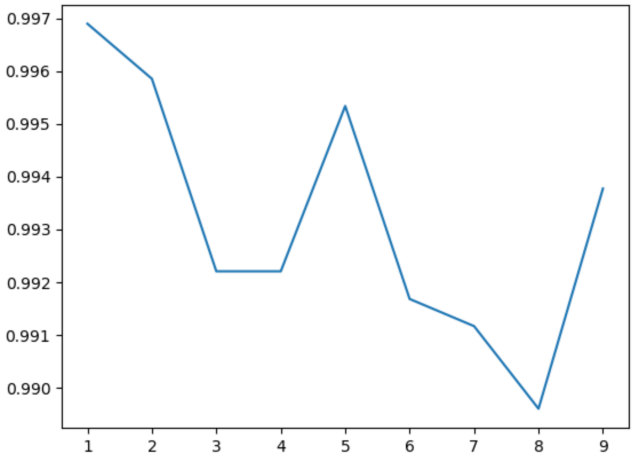
Number of features to consider at each split:



Maximum depth of the trees:



Minimum number of samples per leaf node:



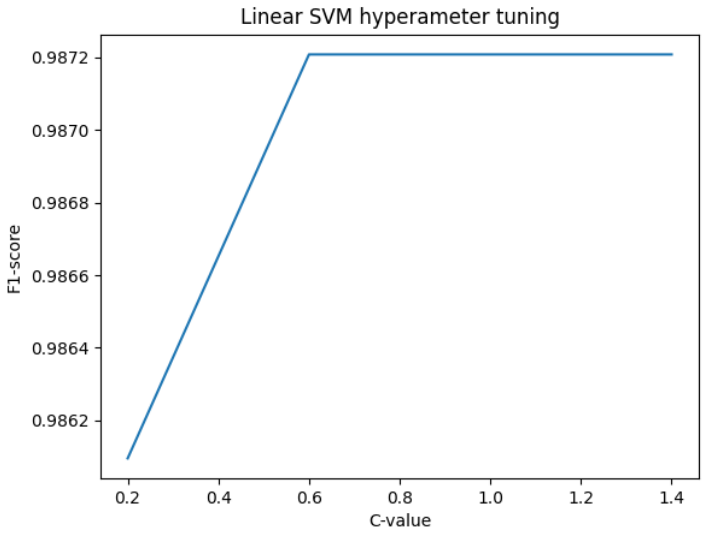
We choose n_trees=200, feature_subcount=3, max_depth=12, and min_node=1.

F1-score: 0.9971
Confusion matrix:

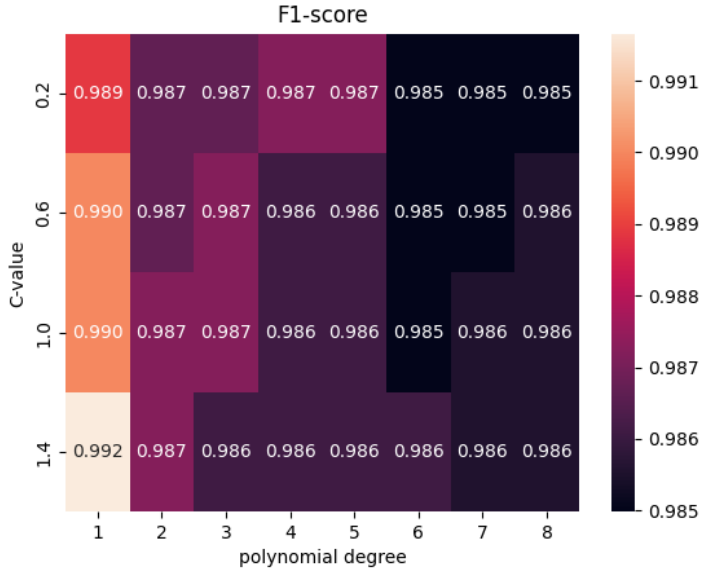
	Predicted Benign	Predicted Malicious
Actual Benign	0.4511	0.0004
Actual Malicious	0.0029	0.5457

- **SVM and Kernel SVM**

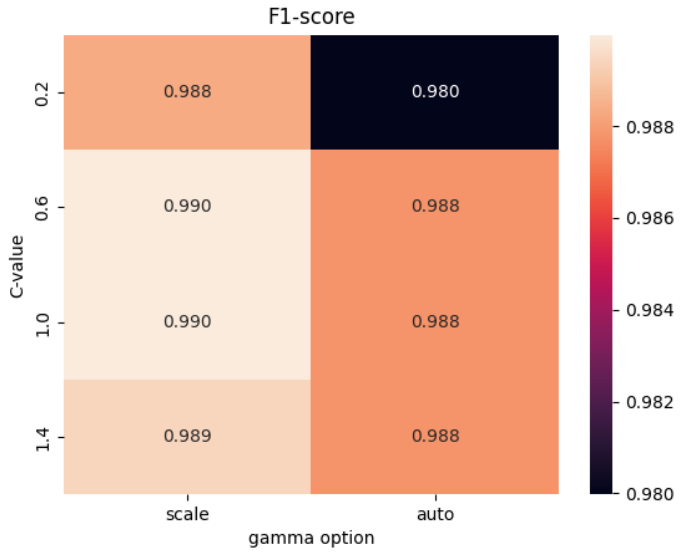
We chose to include an SVM as one of our models because of its notable performance as a binary classifier, as well as its ability to handle a large number of features with the help of the kernel trick. To ensure we were maximizing the performance of our SVM, we wanted to make sure all of our hyperparameters were tuned properly, and that we were using the right type of SVM. Shown below are graphical representations of our tuning results for four different types of SVM's: linear, polynomial, rbf, and sigmoid followed by the final classification results



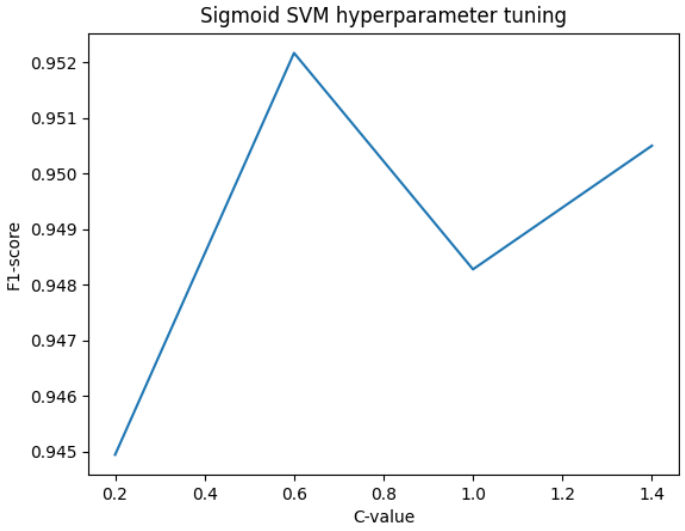
Shown here is the tuning graph for a linear SVM, we tested C values ranging from 0.2 to 1.4 and found the best value to be between 0.6 and 1.4.



When testing a polynomial kernel we tested degree values from 1 to 8 as well as C values from 0.2 to 1.4. We found that a polynomial kernel with degree 1, and C value of 1.4 was marginally the most accurate.



We tested the RBF kernel with two different options for gamma: “scale” which is calculated as “1 / (n_features * X.var())”, and “auto” which is calculated as “1 / n_features.” We also used C=0.2-1.4 as in previous tests. The best results we achieved were with a combination of C=1.0 with the “scale” gamma value.



Finally, we tuned a sigmoid SVM with the same C values of 0.2-1.4. In this case we found C=0.6 to be the most accurate.

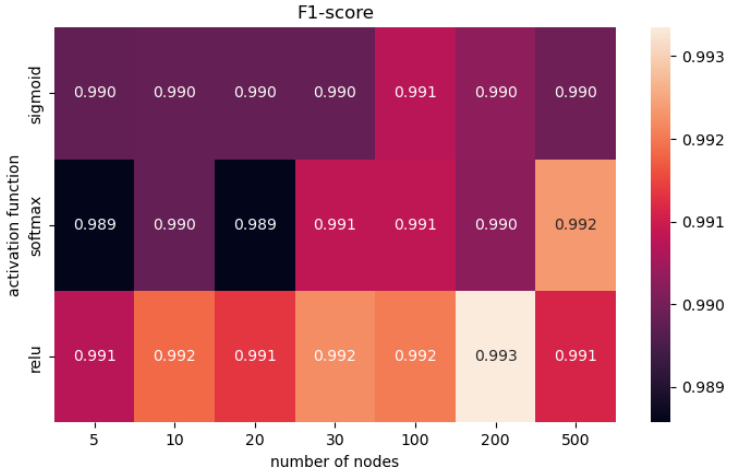
After experimenting with all the possible combinations of SVM that we could come up with. We decided to tune our final classifier as a RBF kernel SVM, C=1, and gamma set to "scale." This combination of hyperparameters yielded the most consistent results and allowed us to record an extremely high **F1-score of 0.9906**. The full confusion matrix for this final classifier is listed below:

F1-score: 0.9906
Confusion Matrix:

	Predicted Benign	Predicted Malicious
Actual Benign	0.4473	0.0041
Actual Malicious	0.0050	0.5435

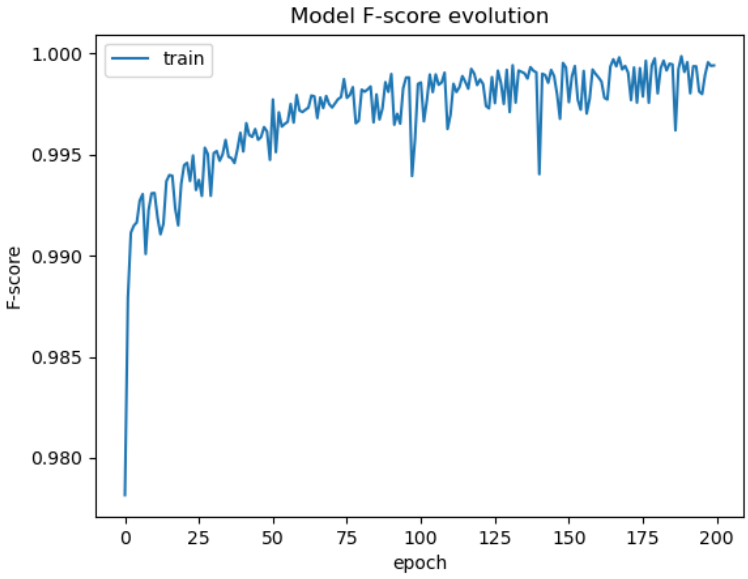
- **Neural Network**

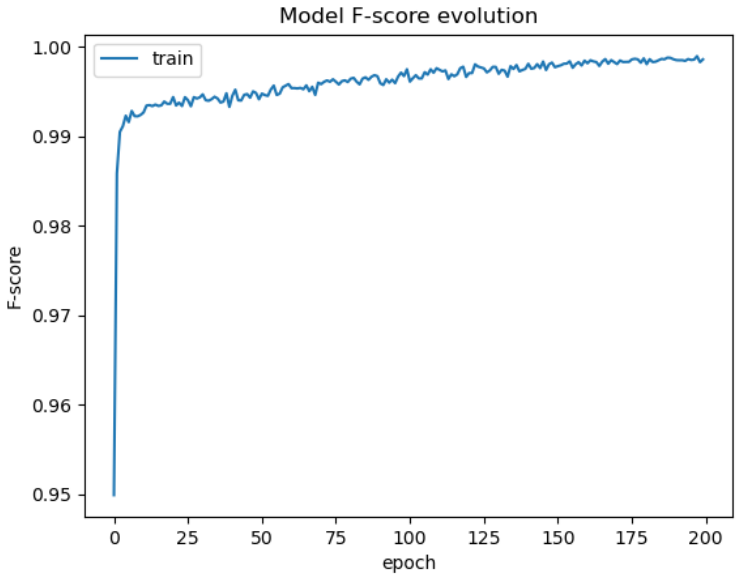
We choose to train a Neural Network because those are known for their good performances even if they tend to overfit. In this case, as we aren't working on image processing or text mining for example, we chose to train a Neural Network structure with only one hidden layer. The hyperparameters we are tuning are: the number of nodes in this layer, the activation function, the number of epochs, and the batch size. We represent the F1-score on the validation set for all the combinations of the first to hyperparameters as a heatmap:



As we can observe, the best performance is reached with the relu activation function and 200 nodes in the hidden layer.

However, we also wanted to check if this solution had converged. Neural networks use backpropagation. The number of epochs and the batch size determine how the score is evolving. We represent the evolution of the f1-score with the epochs for a batch size of 10 and 50.





As we can observe, the number of epochs have to be large enough to allow the weights to stabilize. A batch size of 50 seems to be more stable and lead to the same F1-score. We keep the number of epochs to 200 to not overfit too much and a batch size of 50.

F1-score: 0.9929
Confusion matrix:

	Predicted Benign	Predicted Malicious
Actual Benign	0.4479	0.0036
Actual Malicious	0.0039	0.5446

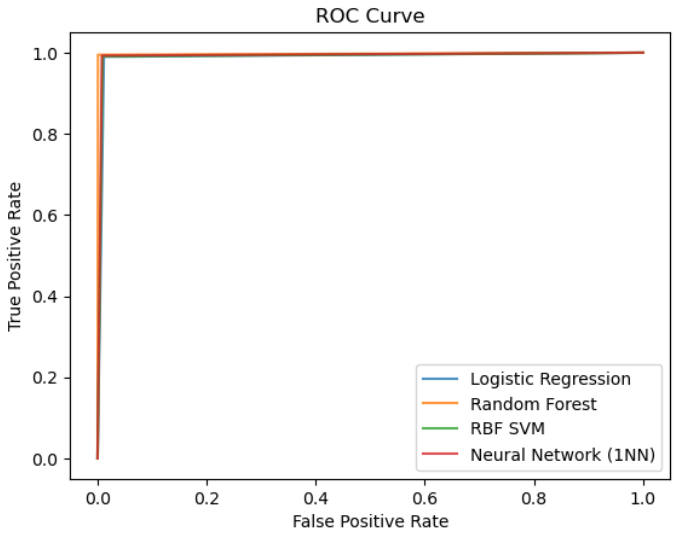
4. Algorithm comparison

To begin the model selection process, we represent the most important metrics of our study for all the best performing models in a table.

	False Negative Rate	**False Positive Rate**	**F1-score**
Logistic Regression	1.1562%	1.0036%	99.02%
Random Forest	0.0792%	0.5214%	99.71%
RBF Kernel SVM	0.9186%	0.9124%	99.06%
Neural Network	0.7919%	0.7169%	99.29%

We note that all the algorithms are performing very well with a F1-score above 99%. Random Forest has the best performance.

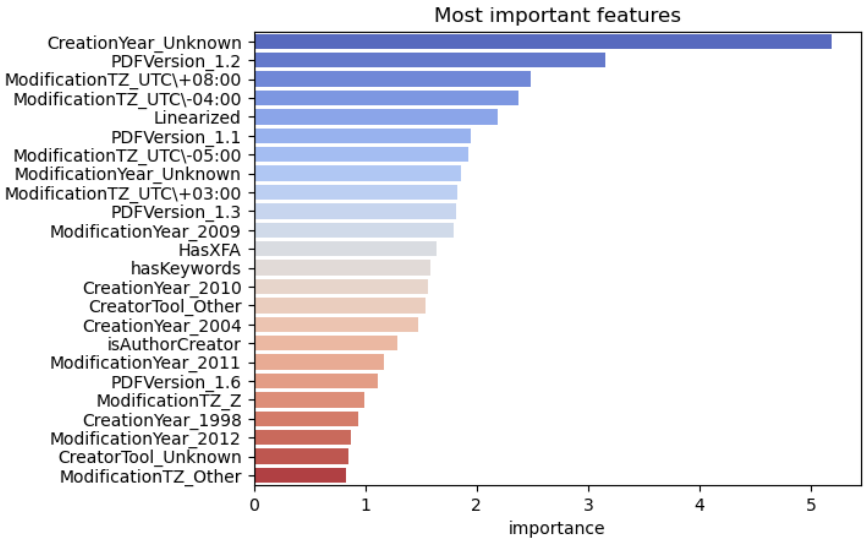
Even if it is not a comparison criteria, Neural Network is the algorithm that takes the most time to train. The testing times are low for all of these models so we can't select based on this criteria. We represent the ROC curve of the prediction made on the test set:

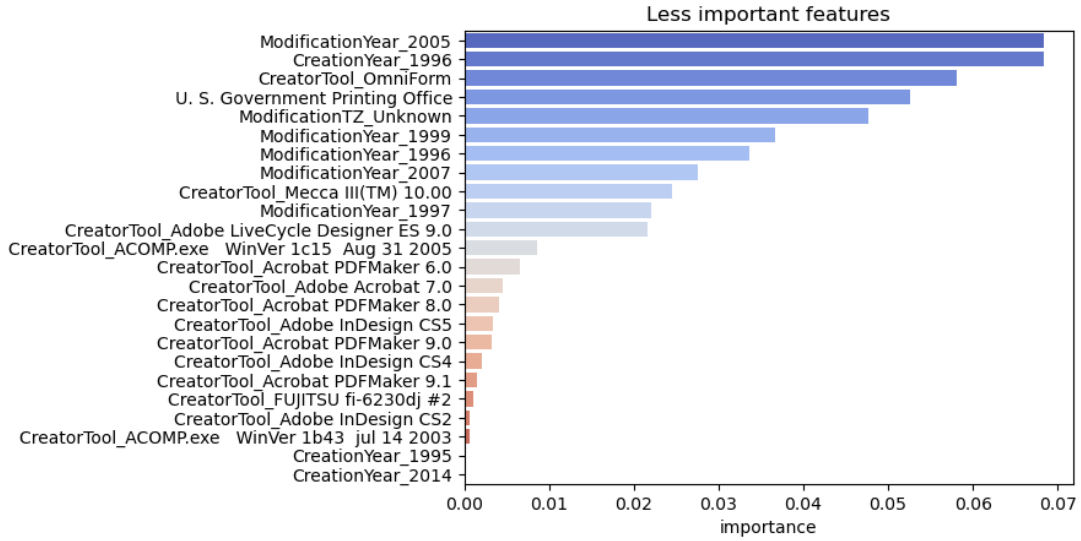


All the classifiers have similar AUC and the curves are very close. The classifier which seems to work the best is still the Random Forest classifier. However, the choice can't be made easily on the F1-score as it differs too slightly between the models.

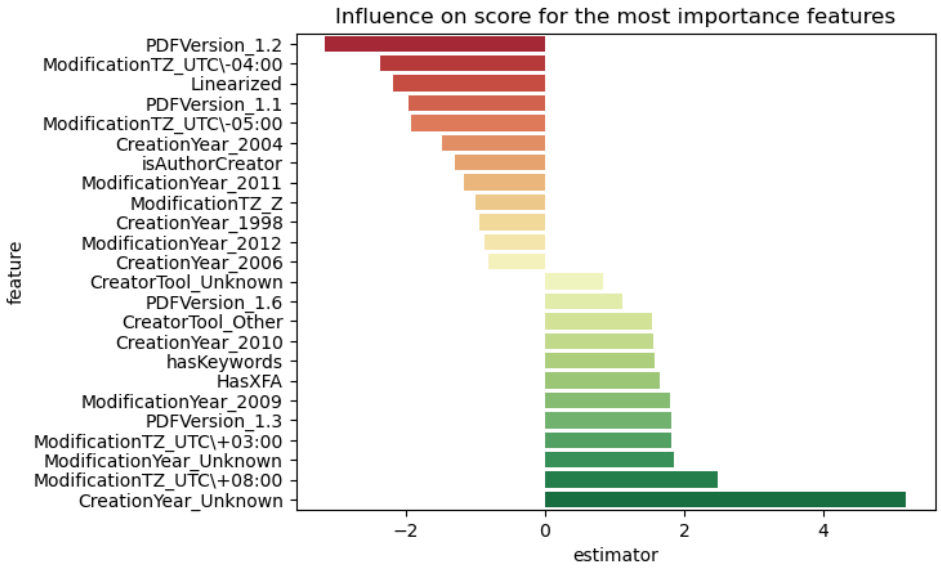
Analysing the storage size of the models -as we want a light model, the Random Forest is the heaviest (even if it isn't too big with only 200 trees) with the Neural Network classifier. SVM and Logistic Regression are light but in terms of explainability the Logistic Regression is better, SVM and especially Kernel SVM are very hard to interpret. We have the same concern with Neural Networks.

Regarding the performances, Random Forest is the best algorithm and is the one we would have chosen. However, even if permutation feature importance can be calculated based on the trees, if there is a need for explainability for the classification and how the categories influence the classification. We would choose Logistic Regression as our final model. We now depict a quick analysis of the features' importance with the SVM classifier.





These graphs can also be realized with Random Forest permutation feature importance. It shows the categories that contribute the most (positively or negatively) to the classification. As one can see the creationYear, the pdf version and the time zone are the features that contribute the most whereas creatorTool isn't very important. We now represent how those features impact the predicted class.



These representations can only be realized with Logistic Regression classifiers. We can see that the time zone is a good discriminator for maliciousness as well as the pdf version.

References

1. "2020 Phishing Trends With PDF Files", Unit 42,
   Ashkan Hosseini and Ashutosh Chitwadgi, April 2021,
   https://unit42.paloaltonetworks.com/phishing-trends-with-pdf-files/
2. "Clean and malicious files for signature testing and research", Contagio, March 2013,
   http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html