SPAMBASE

Creating an API to classify spam emails using Machine Learning techniques

Ulysse BERTHET

Tristan DARRIGOL

THE DATASET

Origin : archive.ics.uci.edu/ml/datasets/Spambase

Task : Classification : Spam / Non Spam

Number of instances : 4601

Attributes : 57 floats, 1 class

ATTRIBUTS

48 continuous real [0,100] attributes of type word_freq_WORD in %

For words in :

```
['make', 'address', 'all', '3d',
'our', 'over', 'remove', 'internet',
'order', 'mail', 'receive', 'will',
'people', 'report', 'addresses',
'free', 'business', 'email', 'you',
'credit', 'your', 'font', '000',
'money', 'hp', 'hpl', 'george', '650',
'lab', 'labs', 'telnet', '857',
'data', '415', '85', 'technology',
'1999', 'parts', 'pm', 'direct', 'cs',
'meeting', 'original', 'project',
're', 'edu', 'table', 'conference’]
```

6 continuous real [0,100] attributes of type char_freq_CHAR in %

For char in :

```
[';', '(', '[', '!', '$', '#’]
```

3 continuous real [1,...] attribute :

- Average length of uninterrupted sequences of capital letters
- Length of longest uninterrupted sequence of capital letters
- Total number of capital letters in the e-mail

1 nominal {0,1} class attribute of type spam :

1 = Spam
0 = Non Spam

DATA INFORMATIONS

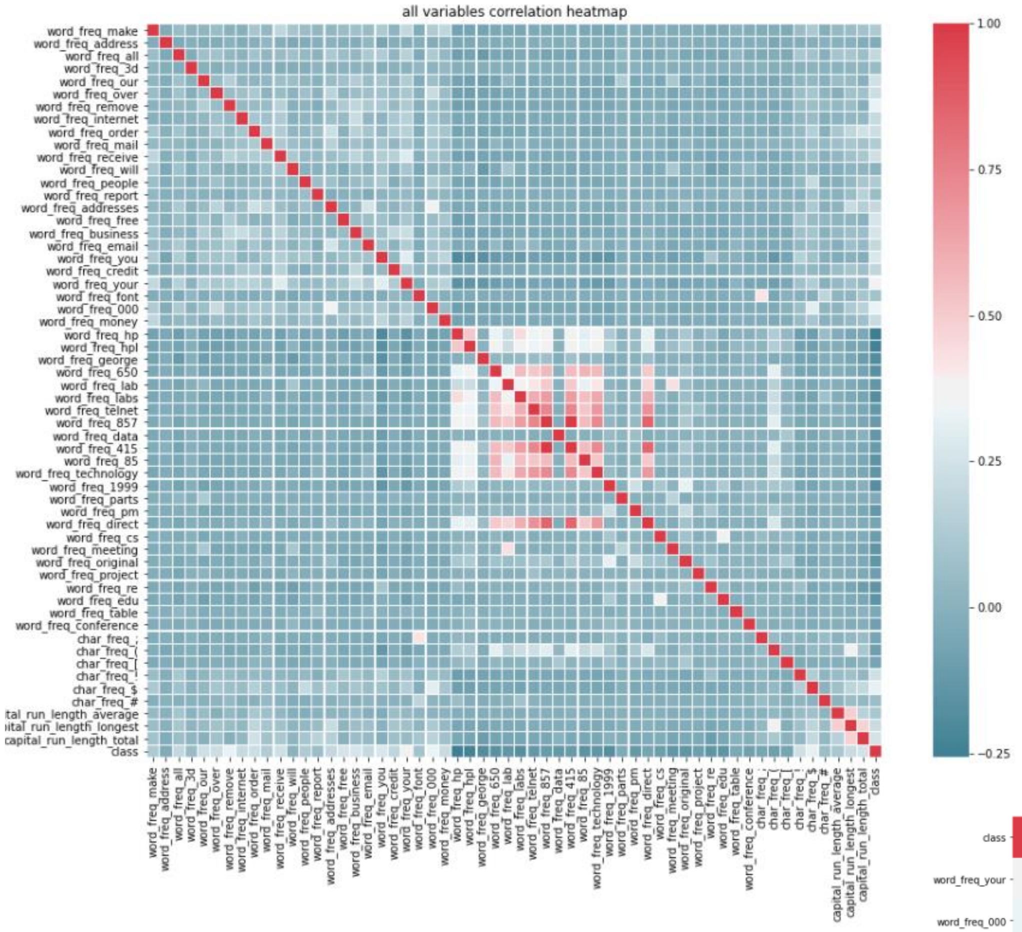
39% of the emails are spams

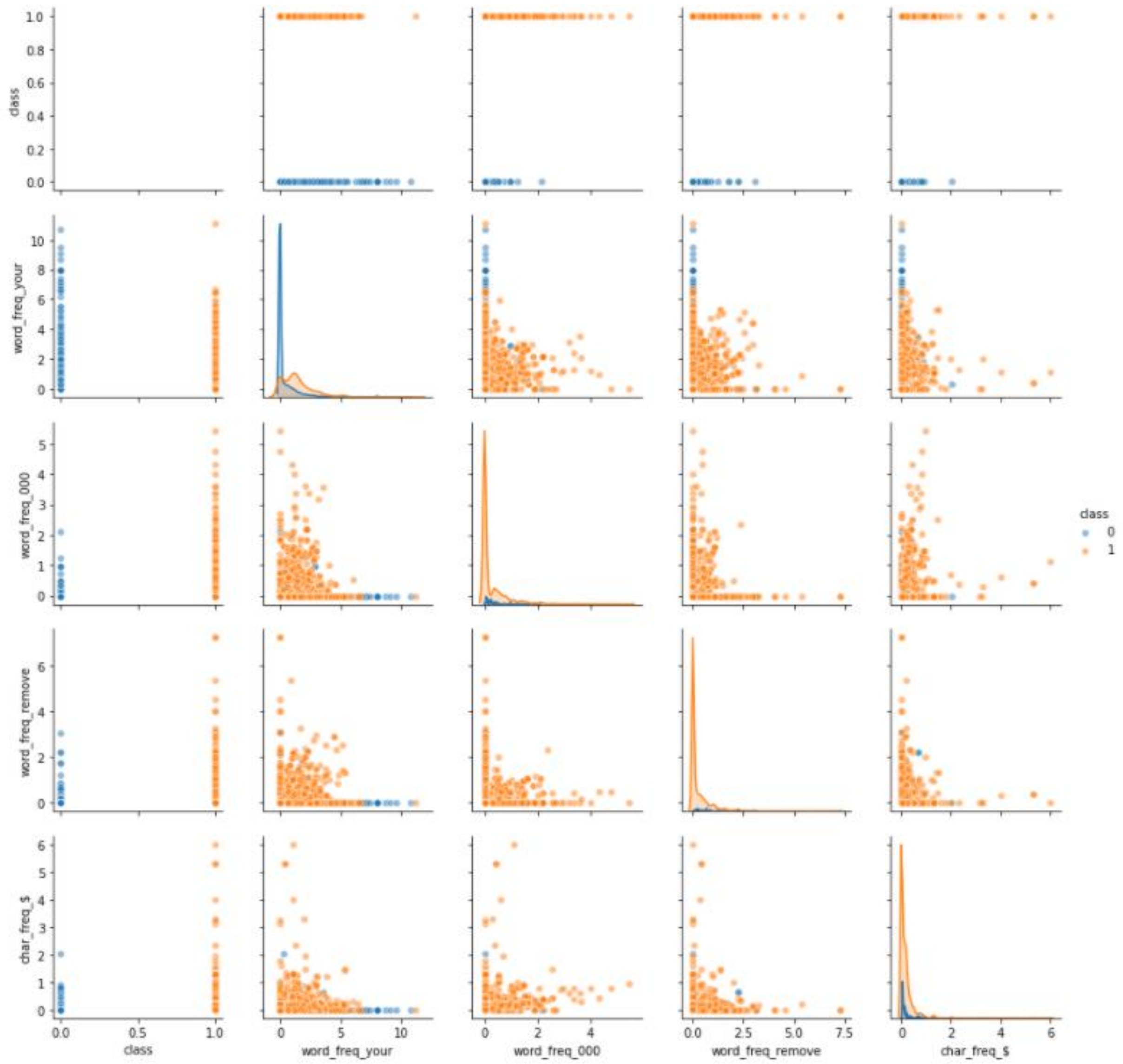
No outliers

No missing values

VARIABLES CORRELATION

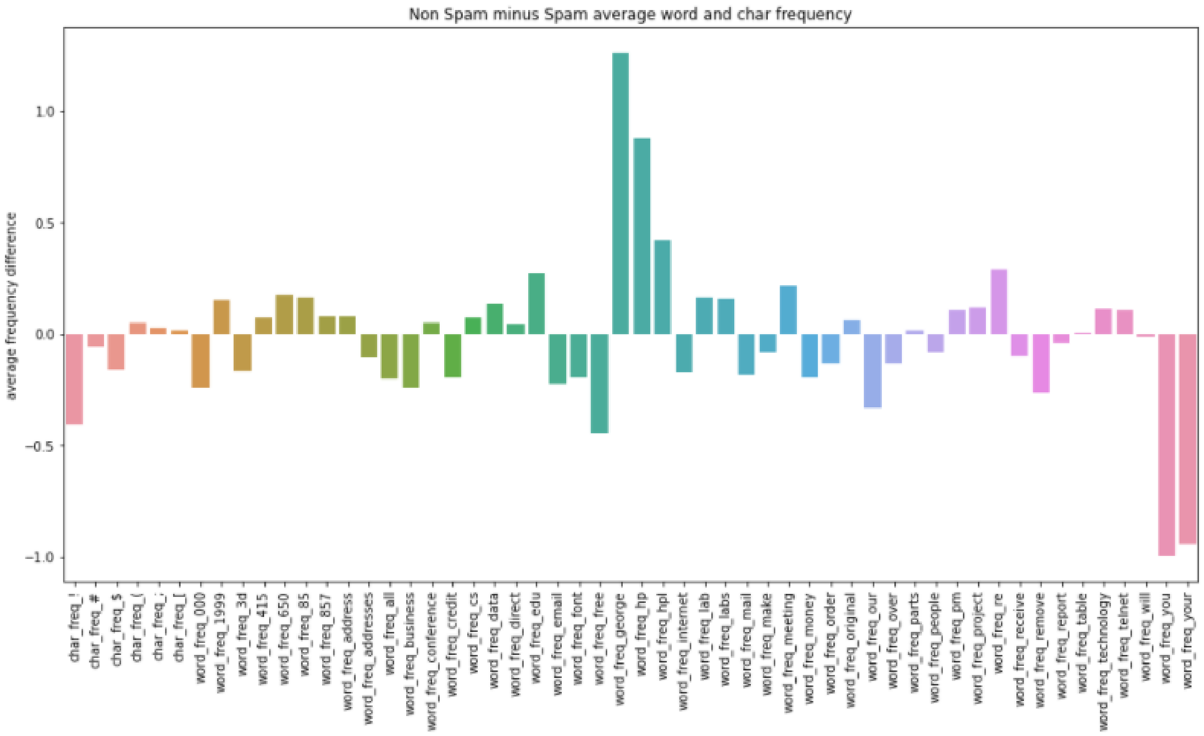
- « your », « 000 », « remove », « $ », « you », « free », « business » and total number of capital letters have the highest positive correlation with the email being a spam
- « hp » has the highest correlation with the email not being a spam
- Their is no strong correlation between one variable and the email being a spam or not (all <0.38)

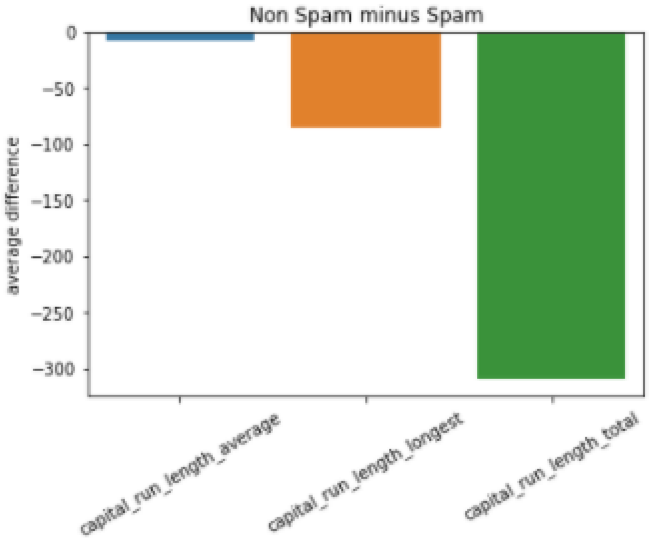




PAIR PLOT TOP 4 MOST CORRELATED VARIABLES

- “your” is more present in non spam at lower frequency. It often appears once or less. In spam it appears more than once.
- “000”, “remove” and “$” appear more in spam at all frequencies.





SPAM VS NON SPAM AVERAGE WORD FREQUENCY

- Word and Char average frequency difference :
  - The words "you" and "your" are fare more frequent in spam emails than in non spam.
  - The words "george" and "hp" are fare more frequent in non spam emails than in spam.
  - Interpretation : spamer don't know your name and call you "you" instead of your name (George ?)
- Capital average difference:
  - Capital letters are far more used in spam emails

CHOICE OF METRICS

- In our situation, we can not afford to get an important mail classified as « spam », so, we need our « positive » prediction to be as good as possible.
- Thus, we choose to select our model according to it's « precision » : $\frac{TP}{TP+FP}$ as it evaluates the quality of our « spam » prediction.
- Accuracy is still taken into account, as it's a global information on how good the model is.

True Class

Predicted Class	Positive	Negative
Positive	True Positive Spam categorized as spam	False Positive Non-spam categorized as spam
Negative	False Negative Spam categorized as non-spam	True Negative Non-spam categorized as non-spam

COMPARING MODELS AND PRE-PROCESSING

	Model	Fitting time	Scoring time	Accuracy	Precision	Recall	F1_score	AUC_ROC
5	Random Forest	0.992928	0.058778	0.953106	0.953631	0.948216	0.952955	0.984245
8	Gradient Boosting	2.295435	0.018560	0.948447	0.948254	0.943648	0.948299	0.985290
9	Adaptative Boosting	0.507737	0.041629	0.946584	0.944953	0.942963	0.946536	0.983374
0	Logistic Regression	1.786466	0.022088	0.931677	0.931505	0.925262	0.931458	0.973868
1	Decision Tree	0.149801	0.018211	0.912422	0.908309	0.907807	0.912349	0.907807
3	Linear Discriminant Analysis	0.071625	0.019741	0.885404	0.894320	0.865712	0.883186	0.950616
7	Bayes	0.016464	0.021062	0.827950	0.836301	0.851723	0.829793	0.946415
4	Quadratic Discriminant Analysis	0.025401	0.021484	0.815528	0.828287	0.841833	0.817289	0.952514
6	K-Nearest Neighbors	0.089474	0.050587	0.795031	0.785475	0.783277	0.794643	0.858521
2	Support Vector Machine	5.663136	0.104894	0.708385	0.707495	0.658313	0.688702	0.802672

Raw data

After deletion of low-correlation data

	Model	Fitting time	Scoring time	Accuracy	Precision	Recall	F1_score	AUC_ROC
5	Random Forest	0.845678	0.049564	0.943789	0.943731	0.938565	0.943637	0.978861
8	Gradient Boosting	1.263818	0.015826	0.933851	0.934042	0.927080	0.933581	0.977883
9	Adaptative Boosting	0.503322	0.055388	0.922981	0.921339	0.917040	0.922754	0.974733
1	Decision Tree	0.047631	0.014026	0.909006	0.904522	0.906178	0.909150	0.906178
0	Logistic Regression	0.960666	0.017877	0.901863	0.904057	0.889748	0.900990	0.964057
7	Bayes	0.008050	0.013555	0.895652	0.893258	0.887402	0.895242	0.946283
4	Quadratic Discriminant Analysis	0.013998	0.022049	0.896273	0.892297	0.890369	0.896182	0.947979
3	Linear Discriminant Analysis	0.036270	0.019619	0.863975	0.878131	0.838212	0.860117	0.939924
6	K-Nearest Neighbors	0.043591	0.034901	0.786025	0.776864	0.770305	0.784782	0.850468
2	Support Vector Machine	3.366793	0.058789	0.708696	0.707808	0.658710	0.689064	0.801941

	Model	Fitting time	Scoring time	Accuracy	Precision	Recall	F1_score	AUC_ROC
5	Random Forest	1.013617	0.055453	0.953727	0.954471	0.948716	0.953560	0.984479
8	Gradient Boosting	2.213982	0.015958	0.948447	0.948254	0.943648	0.948299	0.985461
9	Adaptative Boosting	0.620388	0.051448	0.946584	0.944953	0.942963	0.946536	0.983374
2	Support Vector Machine	3.094253	0.059947	0.929503	0.931217	0.920794	0.929051	0.972686
1	Decision Tree	0.148654	0.020052	0.914907	0.910905	0.910846	0.914895	0.910846
3	Linear Discriminant Analysis	0.063063	0.019119	0.885404	0.894320	0.865712	0.883186	0.950616
6	K-Nearest Neighbors	0.093030	0.176170	0.894720	0.892108	0.886501	0.894376	0.945892
0	Logistic Regression	0.056436	0.020386	0.884783	0.891703	0.866227	0.882880	0.948401
7	Bayes	0.011316	0.014878	0.821429	0.831772	0.846235	0.823234	0.880763
4	Quadratic Discriminant Analysis	0.025968	0.022449	0.809938	0.825056	0.837556	0.811597	0.952433

After Scaling

- First, we decided to compare a lot of un-tuned models, and a few preprocessing method, in order to choose which models and which methods to tune later on.
- We quickly saw that suppressing correlated variables affects the results in a bad way, scaling is sufficient (and necessary)

COMPARING MODELS AND PRE-PROCESSING

	W/out reduction	Precision	Linear+SFM	Precision_sfm	Linear+RFECV	Precision_RFECV	Extra trees	Precision_trees
5	Random Forest	0.954471	Random Forest	0.950532	Random Forest	0.944874	Random Forest	0.943177
8	Gradient Boosting	0.948254	Gradient Boosting	0.941985	Gradient Boosting	0.942689	Gradient Boosting	0.935497
9	Adaptative Boosting	0.944953	Adaptative Boosting	0.933841	Adaptative Boosting	0.930341	Adaptative Boosting	0.927845
2	Support Vector Machine	0.931217	Support Vector Machine	0.926756	Support Vector Machine	0.923352	Support Vector Machine	0.911398
1	Decision Tree	0.910905	Decision Tree	0.910179	Decision Tree	0.909567	Decision Tree	0.904961
3	Linear Discriminant Analysis	0.894320	Linear Discriminant Analysis	0.894001	Linear Discriminant Analysis	0.880715	Linear Discriminant Analysis	0.872241

Best results for every model

We then tried different features selections methods :

- Linear Support Vector Machine + Select From Model
- Linear Support Vector Machine + Recurrent Features Selection
- Linear Support Vector + Tree-based Selection

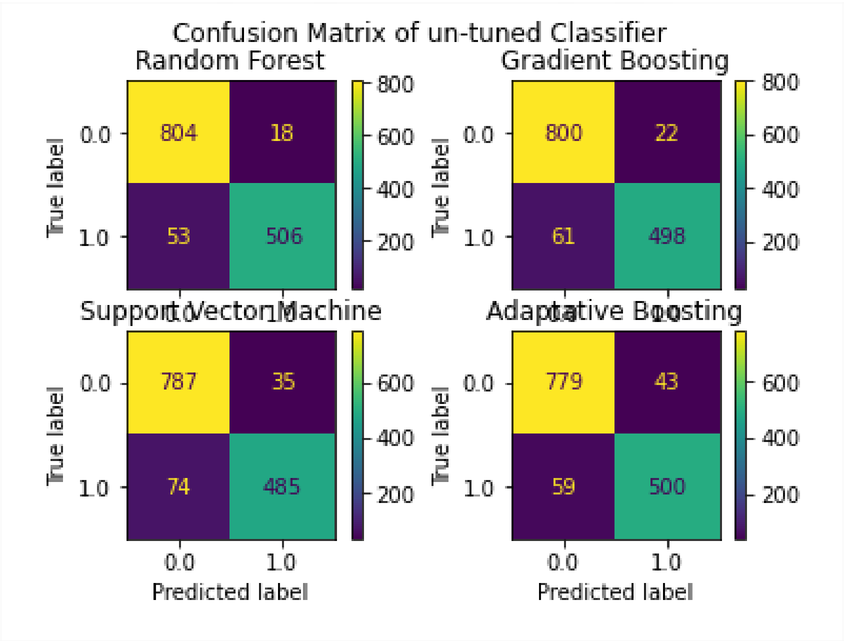
but they ended up decreasing the precision of our models.

TUNING THE BEST MODELS

We selected the 4 best models in order to tune their hyper parameters :

- Random Forest Classifier
- Gradient Boosting Classifier
- Adaptative Gradient Boosting
- Support Vector Machine

There we can see their confusion matrixes and accuracy/precison score



Top 4 models confusion matrix before tuning

Top 4 models scores before tuning

	Model	Accuracy	Precision
2	Random Forest	0.948588	0.965649
1	Gradient Boosting	0.939899	0.957692
3	Support Vector Machine	0.921072	0.932692
0	Adaptative Boosting	0.926140	0.920810

TUNING THE BEST MODELS

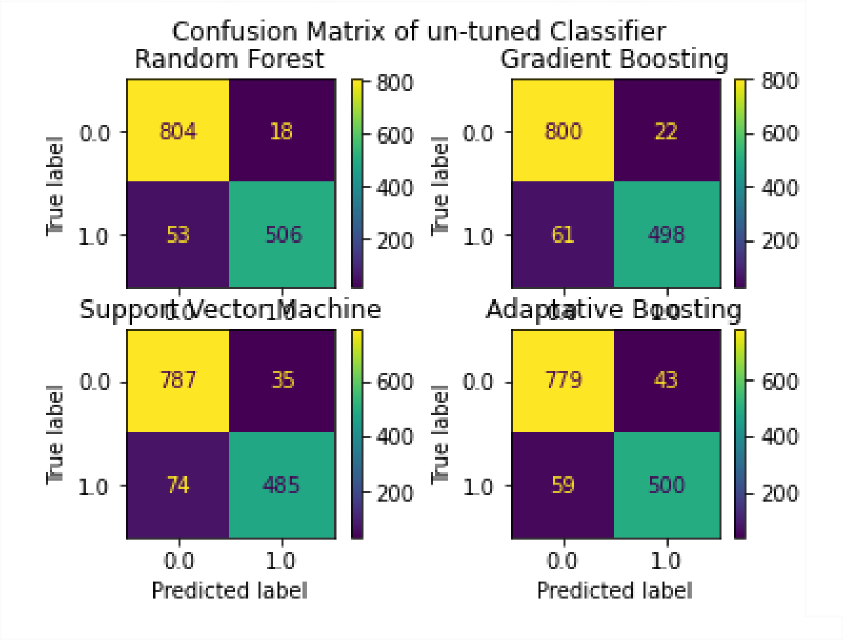
We then worked on those 4 models in order to improve them as much as possible

We used randomized grid search and classic grid search, here are the final results :

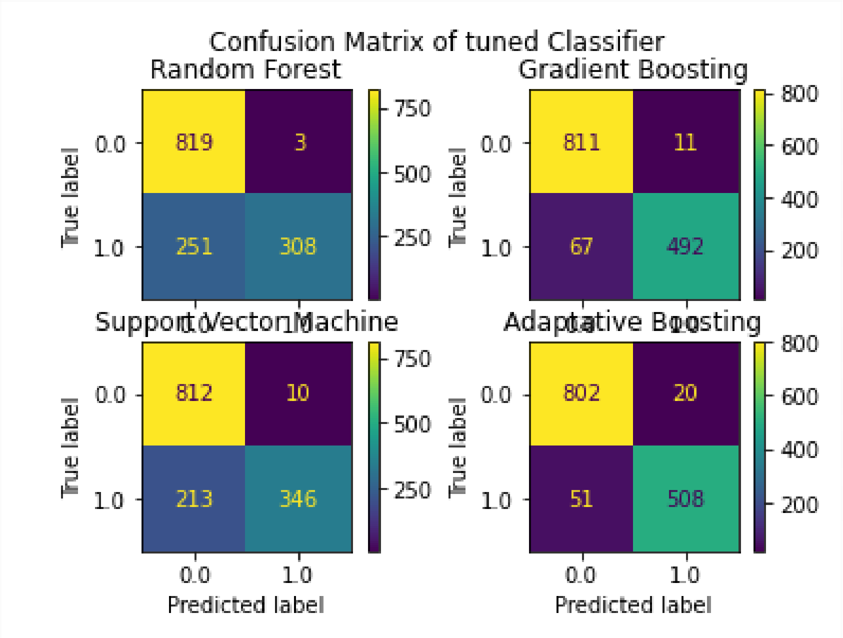
As we really don't want to see any important mail classified as a spam, we still choose **Random Forest Classifier**(precision = 0.99, accuracy = 0.816) over **Gradient Boosting** (precision = 0,978, accuracy = 0,944)

Even though its accuracy is way lower. We have only 3 non-spam emails classified as spam on our test set (1381 mails).

Top 4 models before tuning



Top 4 models after tuning



Scores

	Model	Accuracy	Precision
2	Random Forest	0.948588	0.965649
1	Gradient Boosting	0.939899	0.957692
3	Support Vector Machine	0.921072	0.932692
0	Adaptative Boosting	0.926140	0.920810

	Model	Accuracy	Precision
2	Random Forest	0.816075	0.990354
1	Gradient Boosting	0.943519	0.978131
3	Support Vector Machine	0.838523	0.971910
0	Adaptative Boosting	0.948588	0.962121

Raw email :

«You have won 1 MILLION $, please send you ADDRESS !! »

Email vectoriser

Vectorized email :

```
[0.0, 11.11111111111111, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 22.22222222222222, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 11.11111111111111, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
22.22222222222222, 11.11111111111111, 0.0, 5.333333333333333, 11, 16]
```

Classifier model

Classiffied email :

[1] : Spam

EMAIL TO VECTOR CONVERTER

To test raw emails with our model

- Takes a raw email (string) in input
- Calculates the word and char frequencies using regex
- Calculates the variables about capital letters using regex
- Output a vector that can be interpreted by our model

API :

Created using Flask

Input :

Request to the api must have the following Json format :

- For vectorized data :
  - ➢ data = `[1,[X1,X2,...] = [1,[[x1i]*57, [x2i]*57, ... [xni]*57]`

    with every xni corresponding to a columns of Xn
- For raw email data :
  - ➢ data = [0,["raw email text 1", "raw email text 2", ...]]

data[0] is a flag to know if the data is a row emails or a vector :

- ➢ data[0]=0 if it is a list of raw emails, data[0]=1 if it is a list of vector compatible with the model

Output :

The api will return data in the following Json format :

- [p1,p2,...]

  with pn the prediction for the n th element

FINAL WORDS

Further tuning of boosting techniques could be a solution in order to avoid the accuracy/precision trade-off of the Random Forest Classifier

Few adjustments could be made to mail vectorization in order to adapt the model to more recent informations :

- Date: the current year would be better than '1999
- Name : using the receiver's name instead of 'Georges'

Adding more variables (words and chars) could help refine the algorithm.

Such as badly encoded characters ("Ã©" instead of "é"), which is often found in spam email

Taking into account the mail adress of both the sender and the receiver could be insightful