

Projet DATA731 – Modélisation Stochastique

Projet DATA731 – Modélisation Stochastique	1
Exploration de la donnée	2
Dataviz	2
Distribution variable de la cible (spam)	2
Matrice creuse	3
Preprocessing	4
Suppression de duplicatas	4
Suppression d'échantillons vides	4
Résolution du problème de distribution de la cible	4
Séparation du dataset	5
Normalisation des dataset	5
Feature extraction	6
F1 score	6
Méthode manuel – Variant Inflation Factor (VIF)	7
Méthode automatique – Recursive Feature Elimination (RFE)	8
Méthode automatique – Principal Component Analysis (PCA)	8
Modélisation prédictive	10
ROC-AUC score (Receiver Operating Characteristic – Area Under the Curve)	10
Cross validation (validation croisée)	11
Optimisation des hyperparamètres	11
Logistic Regression (LR)	12
Decision Tree Classifier (DT)	14
Random Forest Classifier (RF)	18
Naïve Bayes Classifier (NB)	20
Conclusion	21

Exploration de la donnée

On commence par s'intéresser au dataset.

Il contient 57 caractéristiques (features) (Il y aura donc surement de la [feature extraction](#) à faire) et 4601 échantillons (samples), la dernière colonne, spam, indique si le sample est un spam ou non. Cela nous permettra de séparer le dataset en test, train, et validation pour entrainer des modèles.

Parmi ces features, 47 sont des fréquences de mots (*word_freq_table*). 6 sont des fréquences de caractères (*char_freq_;*). 3 sont des autres métriques (*capital_run_length_average*, *capital_run_length_total*, *capital_run_length_longest*).

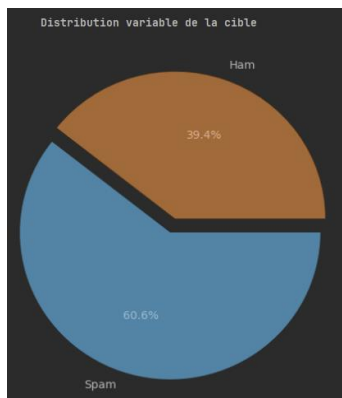
On sait par avance que la feature *word_freq_george* et *word_freq_650* sont des indicateurs de ham (non-spam), on s'intéressera donc à voir si notre modèle remarque cet indicateur.

Spam étant la seule caractéristique catégorique, toutes les autres sont des caractéristiques numérique.

Finalement, on vérifie les propriétés stochastiques (count, mean, std, min, 25% ou premier quartile, etc) des colonnes pour détecter une anomalie. On ne remarque rien de particulier. On passe donc à la visualisation.

Dataviz

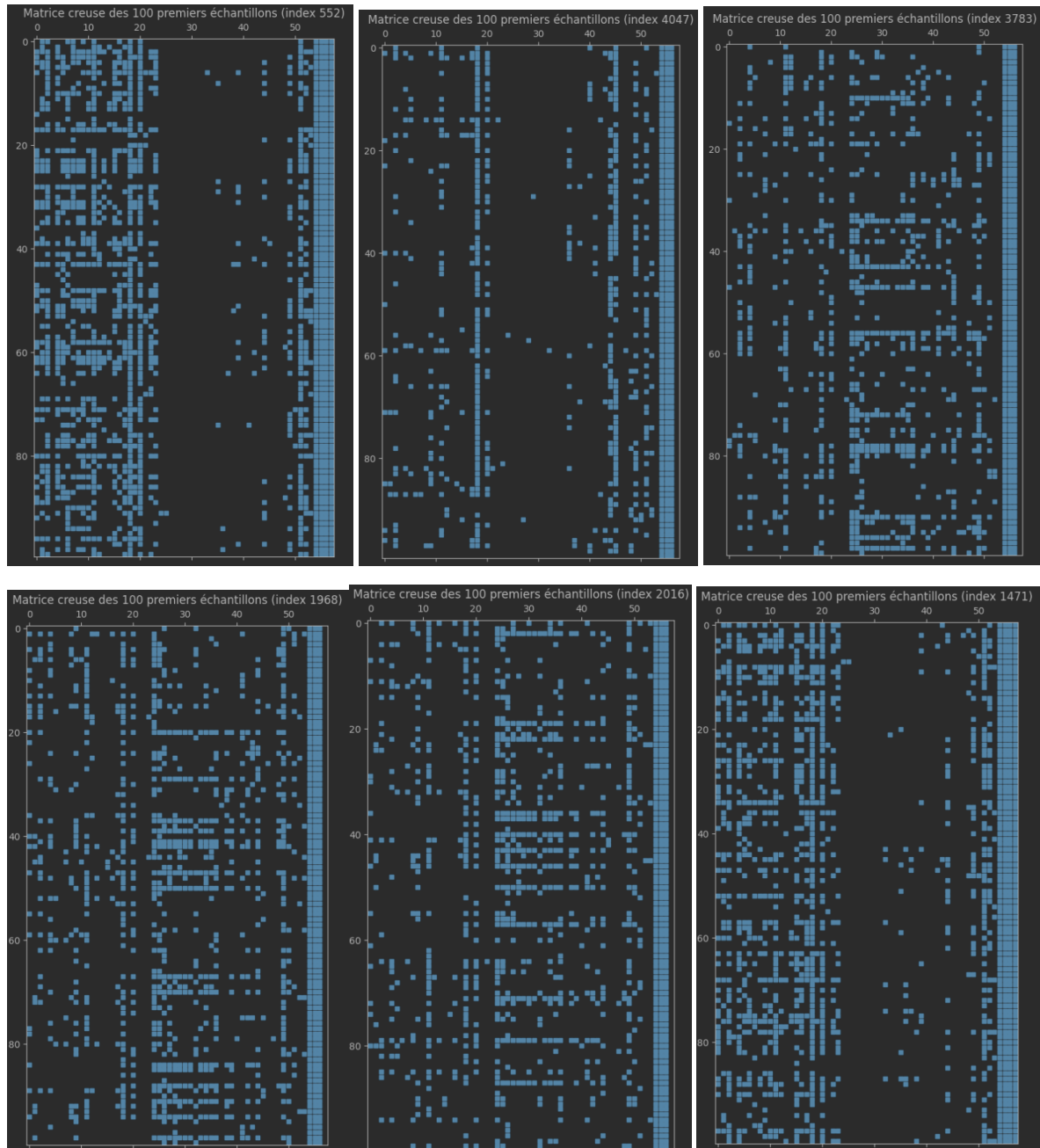
Distribution variable de la cible (spam)



La distribution de la cible semble inéquilibré. On va donc essayer de faire de la [data augmentation](#) afin de résoudre cela.

Matrice creuse

Quelques exemples de matrice creuse de 100 sample à partir d'index aléatoires



On observe un paterne où les features ~25 jusqu'à ~50 sont nulles, tandis que les précédentes sont plus actives que la moyenne (matrice 1 et 6 par exemple).

On observe aussi un paterne inverse où les features ~25 jusqu'à ~50 sont actives mais en contrepartie les feature précédentes sont moindre (matrice 3, 4 et 5).

Pour l'instant il ne s'agit que de spéculations / hypothèses à confirmer dans le futur (spoiler : elles seront confirmées). En attendant que ces pistes s'éclaircissent, on va faire du preprocessing pour supprimer les duplicatas et les échantillons vides, et utiliser SMOTE pour résoudre le problème de distribution inéquilibrée de la cible.

Preprocessing

Suppression de duplicatas

Inference: 391 duplicatas ont été supprimés

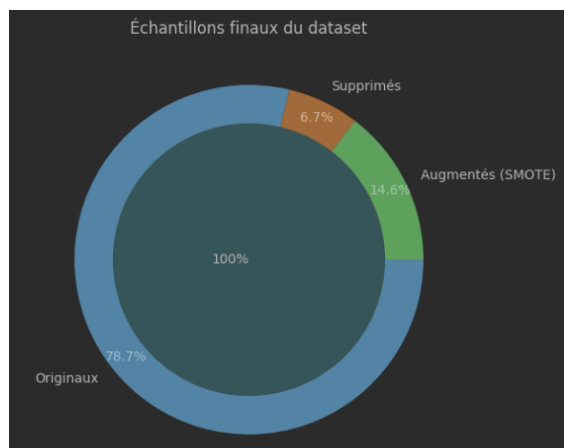
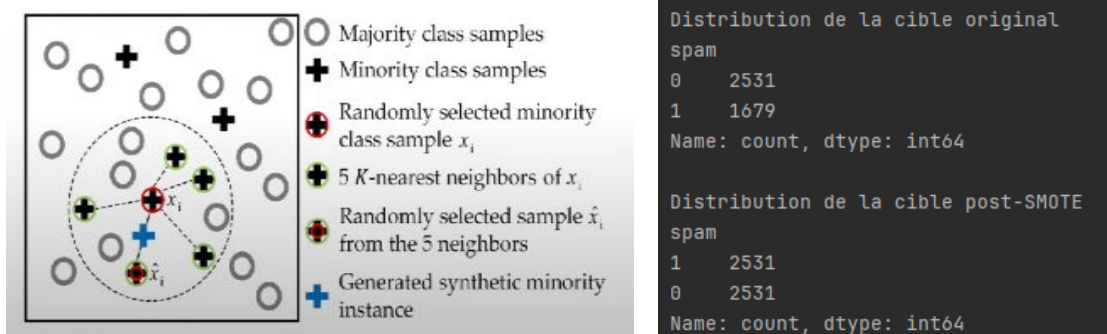
Suppression d'échantillons vides

Il n'y a aucun sample qui contient de valeurs null dans le dataset, cependant il y a des valeurs aberrants (outliers). On va essayer d'imputer ces valeurs aberrantes grâce à l'oversampling.

Résolution du problème de distribution de la cible

SMOTE est une technique d'oversampling qui permet de générer des samples similaires aux autres samples (à l'opposé de l'undersampling qui permet de supprimer des samples similaires).

Voici un schéma explicatif de l'algorithme SMOTE. Suivi du résultat de la technique sur notre dataset.



Le dataset final après nettoyage contient 5602 samples.

Maintenant que le dataset est nettoyé, il est temps de manipuler les données pour le séparer en test set et en train set.

Séparation du dataset

Pour séparer le dataset, on utilise le pourcentage typique 80%-20%, respectivement pour le train set et le test set.

La matrice X contient les features de chaque échantillon, la matrice Y contient le statut de la cible (0 pour ham, 1 pour spam), il s'agit donc d'un vecteur. Ainsi la répartition final des échantillons est la suivante :

```
Dataset original | X (5062, 57) | Y (5062,)  
Training dataset | X (4049, 57) | Y (4049,)  
Testing dataset | X (1013, 57) | Y (1013,)
```

Normalisation des dataset

Les valeurs brutes des features peuvent variées considérablement d'une feature à l'autre, on cherche à normaliser ces valeurs (càd obtenir une moyenne ou une variance similaire) afin que les features aient toutes la même importance lors de l'entraînement des modèles (étant donné que l'entraînement se fait très souvent en mesurant les distances euclidiennes, on cherche à ce qu'une feature n'ai pas le dessus sur les autres).

Pour cela on utilise la « standardization » qui utilise cette formule pour normalisé une feature x de variance σ et de moyenne \bar{x} :

$$x' = \frac{x - \bar{x}}{\sigma}$$

Résultat pour le training dataset

	word_freq_make :	word_freq_address :	word_freq_all :	word_freq_3d :	word_freq_our :	word_freq_over :	word_freq_remove :	word_freq_internet :	word_freq_order :	word_freq_mail :
count	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000	4049.000000
mean	0.111029	0.115259	0.308924	0.080421	0.360894	0.109238	0.154864	0.122288	0.105240	0.268916
std	0.290260	0.376185	0.497162	1.491291	0.693823	0.283104	0.447522	0.413365	0.294136	0.643653
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.066674	0.000000	0.490000	0.000000	0.499318	0.050000	0.040000	0.000000	0.000000	0.290000
max	4.540000	9.520000	5.100000	42.810000	10.000000	5.880000	7.270000	11.110000	5.260000	18.180000

Training set normalisée										
count	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03	4.049000e+03
mean	3.597463e-17	2.632290e-18	2.456804e-17	7.019439e-18	-6.054266e-17	-2.281318e-17	3.860692e-17	-5.089093e-17	6.668467e-17	1.228402e-17
std	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00	1.000124e+00

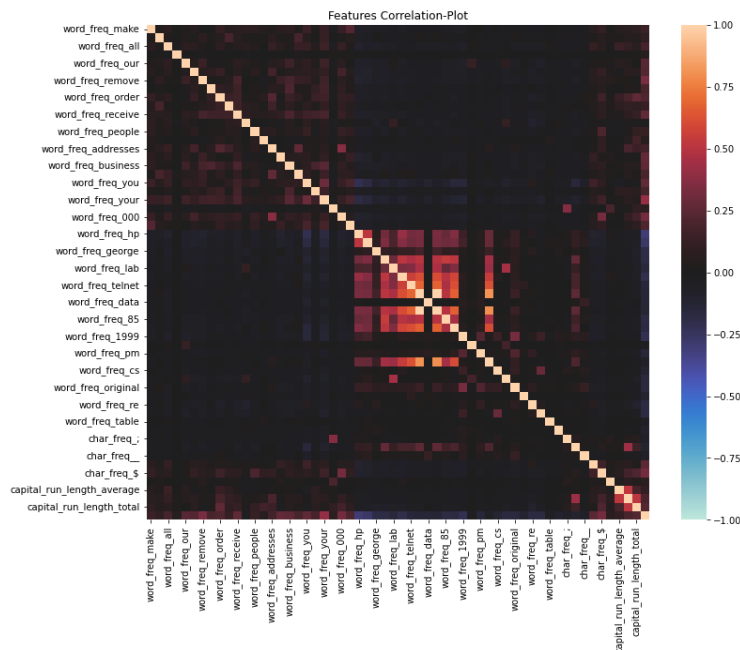
Résultat pour le test dataset (on ne divise pas par la variance pour celui-là)

Testing set original										
count	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000
mean	0.114609	0.139462	0.320613	0.095919	0.346332	0.113231	0.134312	0.116042	0.096698	0.2
std	0.296859	0.595629	0.521917	1.878640	0.598455	0.254161	0.377717	0.354878	0.250965	0.6
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	0.060000	0.000000	0.500000	0.000000	0.510000	0.090000	0.000000	0.000000	0.000000	0.2
max	2.770000	14.280000	3.700000	42.516123	6.250000	2.100000	3.120000	3.630000	1.630000	5.2

Testing set normalisée										
count	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000
mean	0.024261	0.061026	0.023753	0.019432	-0.024280	0.016573	-0.029028	-0.014169	-0.037130	0.0
std	1.027487	1.579306	1.051381	1.356286	0.850695	0.899529	0.897504	0.844816	0.833033	0.9
min	-0.372423	-0.308758	-0.622108	-0.049816	-0.516587	-0.384175	-0.348170	-0.290416	-0.358100	-0.4
25%	-0.372423	-0.308758	-0.622108	-0.049816	-0.516587	-0.384175	-0.348170	-0.290416	-0.358100	-0.4
50%	-0.372423	-0.308758	-0.622108	-0.049816	-0.516587	-0.384175	-0.348170	-0.290416	-0.358100	-0.4
75%	-0.164751	-0.308758	0.385121	-0.049816	0.208371	-0.065647	-0.348170	-0.290416	-0.358100	0.0
max	9.215108	37.554574	6.831389	30.644751	8.367707	7.048163	7.065360	8.351096	5.052392	7.6

Feature extraction

On a remarqué plus tôt que le nombre de feature était assez important. Avec la corrélation, on va pouvoir distinguer les features corrélés entre elles afin de les réduire.



Ce graphique complète les [observations faites plus tôt](#) sur la matrice creuse, effectivement on remarque que les features 0 jusqu'à ~24 semble (faiblement) corrélés, ainsi que les features 25 jusqu'à ~40. On observe donc bien deux « paternes ».

Il semble y avoir une forte multi colinéarité. On va donc tester si l'on peut améliorer les performances du modèle en utilisant la feature extraction et la feature selection.

Pour cela on va comparer différentes méthodes, la méthode manuel (VIF), méthode automatique (RFE) et une méthode par décomposition automatique (PCA). Avant de commencer, on précise que le score qui sera utilisé pour comparer les modèles est le F1 score.

F1 score

Le score F1 est une mesure de la précision d'un modèle de classification, prenant en compte à la fois la précision et le rappel (recall). C'est le score qui sera utilisé pour la suite. Voici sa formule et la formule généralisée avec $\beta \in \mathbb{N}^+$

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad F_\beta = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$$

On rappelle que la précision et le recall peuvent s'obtenir grâce à la matrice de confusion d'un modèle.

$$\text{precision} = \frac{\text{vrai positif}}{\text{vrai positif} + \text{faux positifs}} \quad \text{recall} = \frac{\text{vrai positif}}{\text{vrai positif} + \text{faux négatifs}}$$

Méthode manuel – Variant Inflation Factor (VIF)

La corrélation correspond à la colinéarité entre une variable et une autre, la VIF correspond à la colinéarité entre une variable et toutes les autres.

Imaginons qu'on associe un modèle régressif entre une variable X_1 et les autres X_2, X_3, X_4

$$X_1 = a_0 + a_1X_2 + a_2X_3 + a_3X_4 + \varepsilon$$

On calcule alors le coefficient de détermination R_1^2 de ce modèle régressif, que l'on insère dans cette formule pour trouver le VIF

$$VIF(X_1) = \frac{1}{1 - R_1^2}$$

Un VIF haut (> 5 par convention) signifie que la variable peut être expliquée grâce aux autres, donc dans notre cas d'utilisation, que la feature peut être extraite.

On applique cette opération pour toutes les variables (features) X_i séquentiellement, ainsi on obtient.

$$X_2 = a_0 + a_1X_1 + a_2X_3 + a_3X_4 + \varepsilon, VIF(X_2) = \frac{1}{1 - R_2^2}$$

$$X_3 = a_0 + a_1X_1 + a_2X_2 + a_3X_4 + \varepsilon, VIF(X_3) = \frac{1}{1 - R_3^2}$$

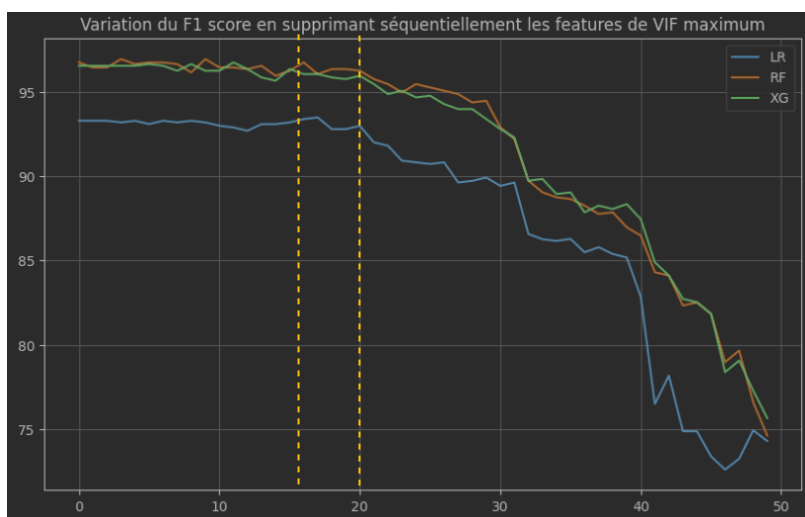
$$X_4 = a_0 + a_1X_1 + a_2X_2 + a_3X_3 + \varepsilon, VIF(X_4) = \frac{1}{1 - R_4^2}$$

A chaque fois on recalcule les coefficients de détermination R_i^2 et le VIF correspondant.

Remarque : Il existe une approche matricielle au VIF, mais elle est trop « complexe » pour qu'on y consacre du temps.

La feature avec le plus grand VIF est supprimée et on observe les performances des modèles avec ce nouveau dataset. On répète cette opération jusqu'à ce qu'il ne reste plus qu'une feature.

Remarque : L'algorithme peut s'arrêter lorsque $VIF_{max} \leq 1$



Voici la figure que l'on obtient.

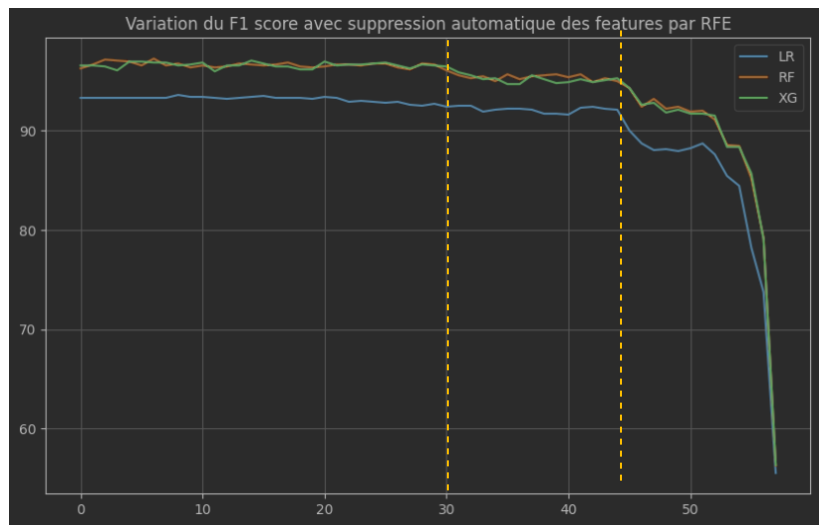
On remarque que le score est quasi-constant à 96% (94% pour la LR) jusqu'à la ~15^{ème} itération, et qu'il commence vraiment à décroître vers la 20^{ème}.

Méthode automatique – Recursive Feature Elimination (RFE)

Cette méthode est assez intéressante car elle utilise elle-même un modèle de régression, en l'occurrence on lui fournit un modèle de régression logistique.

Lors d'une itération, le modèle assigne un poids à chaque feature (dans notre cas il s'agit des coefficients du modèle régressif) puis il supprime la feature de plus petit poids (dans d'autre cas il peut même supprimé un pourcentage de features). Cette itération est répétée jusqu'à ce que le nombre de feature atteigne 1.

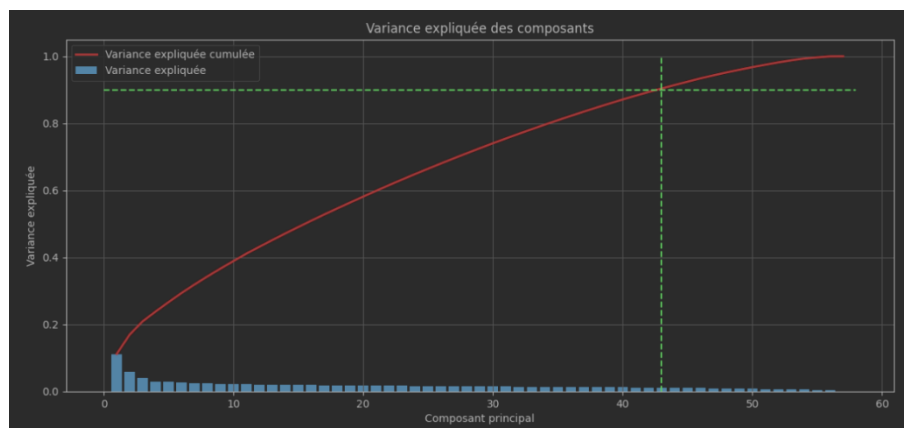
Avec cette méthode la perte d'information semble plutôt être exponentielle. Jusqu'à la 30^{ème} feature, le score semble quasi-constant (97% et 93% pour la LR). Puis il décroît progressivement jusqu'à la 43^{ème} itération, puis il chute.



Cette méthode a donc l'air plus efficace que la VIF, elle a un score similaire, mais il reste constant plus longtemps.

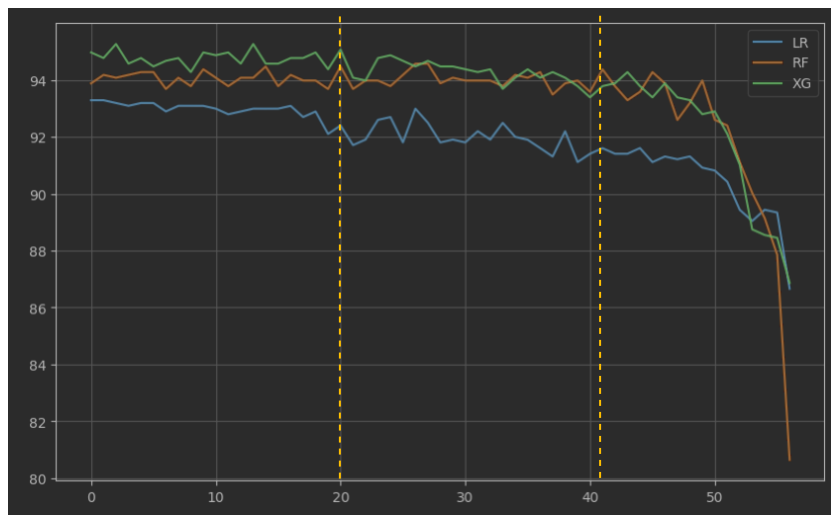
Méthode automatique – Principal Component Analysis (PCA)

La décomposition en composants principaux a énormément d'applications en physique, en biologie, en économie, en sociologie, en traitement et en apprentissage automatique. Elle a diverses utilités, mais celle qui nous intéresse est sa capacité à décorrélérer des variables dans une nouvelle base constituées de nouveaux axes (... de nouvelles features). Elle ne se contente donc pas de supprimer des features comme le VIF ou le RGE, mais elle en crée de nouvelles qui conserve au maximum l'information du dataset original. Pour connaître le nombre de composants principales suffisant, on observe à quel moment la variance cumulée des composants atteint un seuil clé, ici, il s'agit de 0,9.



La variance cumulée atteint 0.9 à partir de 43 composants.

Maintenant qu'on connaît la réduction possible avec PCA, on observe les scores des modèles pour chaque nombre de composant.



On observe que les résultats sont plus « chaotiques » qu'avec les deux dernières méthodes. Le score avoisine le 94% (inférieur à VIF et RFE). De plus la constance des résultats est « brisée » plus rapidement que la méthode RFE. Cette méthode n'est donc vraiment pas intéressante.

Finalement, pour toutes les méthodes, on n'observe pas une amélioration du score en supprimant des features multi colinéaires. Cependant méthode qui parait la plus avantageuse pour la feature extraction est la RFE, on peut l'appliquer pour réduire le nombre de features à 27 ($57 - 30$) ou même 14 ($57 - 43$).

On obtient une dimension finale du training dataset transformé : 4049×27

On obtient une dimension finale du test dataset transformé : 1013×27

Modélisation prédictive

On commence par créer une table qui stocke les résultats des différents modèles

	Accuracy	Precision	Recall	F1-score	AUC-ROC score
Logistic Regression (LR)	0.0	0.0	0.0	0.0	0.0
Decision Tree Classifier (DT)	0.0	0.0	0.0	0.0	0.0
Random Forest Classifier (RF)	0.0	0.0	0.0	0.0	0.0
Naïve Bayes Classifier (NB)	0.0	0.0	0.0	0.0	0.0
Support Vector Machine (SVM)	0.0	0.0	0.0	0.0	0.0
K Nearest Neighbours (KNN)	0.0	0.0	0.0	0.0	0.0
Gradient Boosting (GB)	0.0	0.0	0.0	0.0	0.0
Extreme Gradient Boosting (XGB)	0.0	0.0	0.0	0.0	0.0

On reconnait ici le *precision score*, *recall score* et *f1 score* précédemment cité.

L'*accuracy* est le rapport entre le nombre de prédictions correctes (vrai positifs et vrai négatifs) par rapport au nombre de sample.

Le *ROC-AUC score* est expliquée ci-dessous.

ROC-AUC score (Receiver Operating Characteristic – Area Under the Curve)

Le ROC-AUC évalue la capacité d'un modèle à discriminer entre les classes positives et négatives en fonction de différents seuils de classification.

$$recall = \frac{vrai\ positif}{vrai\ positif + faux\ négatifs} \qquad specificity = \frac{vrai\ négatif}{vrai\ négatif + faux\ positifs}$$

En appliquant ces formules à notre cas d'utilisation :

$$recall = \frac{vrai\ spam}{vrai\ spam + faux\ spam} \quad (\text{Taux de vrai spam})$$
$$specificity = \frac{vrai\ ham}{vrai\ ham + faux\ ham} \quad (\text{Taux de vrai ham})$$
$$1 - specificity \quad (\text{Taux de faux spam})$$

Le ROC est donc la courbe représentant le taux de vrai spam (*recall*) en fonction du taux de faux spam ($1 - specificity$) pour différents seuils de classification. Un point sur le graphe représente une matrice de confusion pour un certain seuil.

On a désormais une manière graphique pour comparer les seuils de classification pour un modèle donné. Cependant, c'est insuffisant pour comparer des modèles, il est complexe de comparer deux courbes ROC de deux modèles différents.

Pour cela, on utilise l'air sous la courbe ; le ROC-AUC mesure l'aire sous la courbe ROC. Plus l'AUC est élevée, meilleure est la capacité du modèle à discriminer entre les classes.

Pour la suite, on va commencer par expliquer les grands principes de chaque modèle, puis on montrera les scores *F1* et *accuracy*, la matrice de confusion, puis la courbe ROC avec le score *AUC-ROC*. On terminera sur une interprétation des résultats pour chaque modèle.

Cross validation (validation croisée)

Afin de mieux entraîner les modèles, on va utiliser la **validation croisée stratifiée** (Stratified K-Fold) avec la répétition des séparations. La validation croisée stratifiée est une méthode de validation croisée où les données sont divisées en k plis, mais de manière à ce que la distribution des cibles soit maintenue dans chaque pli.

On étend cette validation en introduisant la notion de répétition. Plutôt que de diviser les dataset une seule fois, on répète le processus de division des données et d'évaluation du modèle plusieurs fois. Cela permet d'obtenir des estimations plus robustes des performances du modèle, en particulier dans notre cas où le jeu de données est petit.

Les paramètres utilisés pour ce pli sont $n_split = 10$ et $n_repeats = 3$, on se retrouvera donc avec un ensemble de 30 split pour entraîner les modèles (Remarque : on gardera toujours le même *random state*).

Pour le train set, chaque pli contient 3644 samples, pour le test set, 405 samples.

Optimisation des hyperparamètres

Maintenant que l'on a des plis, on veut aussi trouver la meilleure combinaison d'hyperparamètres pour entraîner les modèles.

Pour cela on utilise RandomizedSearchCV. Cette fonction passe un ensemble aléatoire d'hyperparamètres à un modèle. Puis le fit avec les plis d'entraînement. Puis prédit les plis de test. Et finit par calculer le ROC-AUC score associé.

Après 50 ensembles testés (nombre choisit arbitrairement), il donne l'ensemble d'hyperparamètres qui donne le meilleur score.

En fonction du modèle choisit, les hyperparamètres sont différents, on essayera de les expliciter à chaque fois.

Logistic Regression (LR)

La régression logistique est utilisée pour modéliser la probabilité qu'un nouveau sample soit un spam ou non. Comparé à la régression linéaire, on cherche donc à attribuer une valeur entre 0 et 1 à un sample, plutôt qu'une valeur continue sur un ensemble non-fini.

Comme mentionné précédemment, on utilise un seuil pour déterminer à partir de quel probabilité le sample est un spam et on fait varier ce seuil en plaçant le point ROC correspondant à chaque fois.

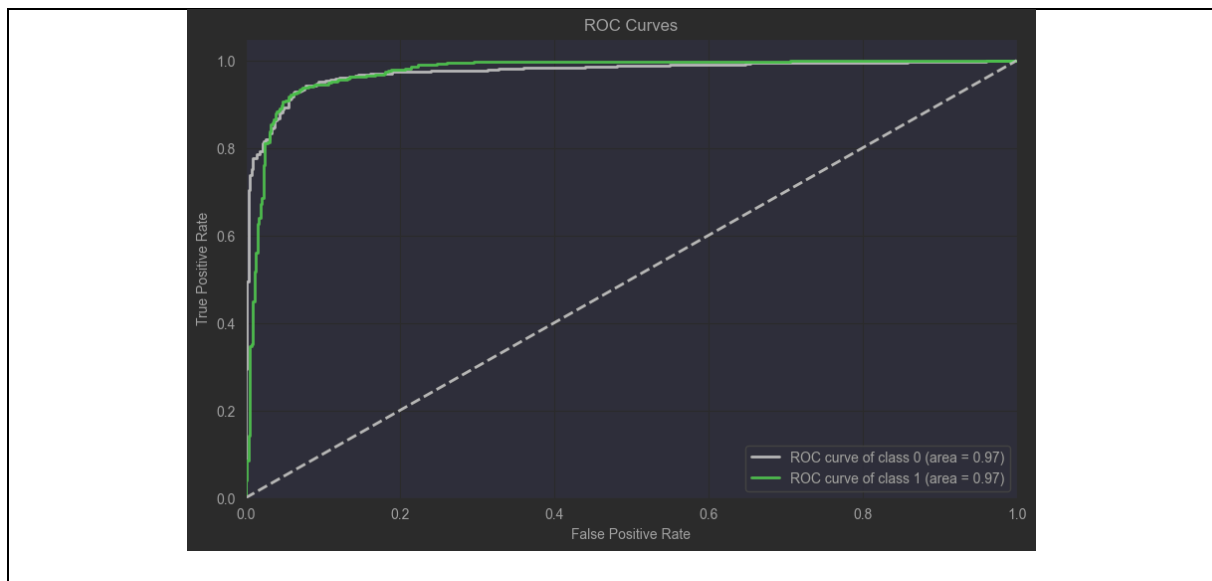
Hyperparamètres LR

Les hyperparamètres de la LR qui sont optimisés sont « solver » et « C ».

Paramètre	Valeurs possibles	Définition	Interprétation
C	Tiré de manière log uniforme entre 10^{-5} et 100	Corresponds à l'inverse de la force de régularisation. Impliqué dans la fonction de coût du modèle.	Un C élevé signifie que le modèle est autorisé à s'ajuster davantage aux données d'entraînement, tandis qu'un C faible favorise une régularisation plus forte.
solver	newton-cg liblinear	Algorithme à utiliser pour résoudre le problème d'optimisation qui se pose lors de l'ajustement du modèle.	Les deux algorithmes possibles sont très adaptés pour une petite taille de données, donc pour notre cas.

Résultat LR

Meilleur ensemble de paramètre	C \approx 5 solver = liblinear			9.955
Accuracy	93%			
F1 score	93%			
Matrice de confusion		Ham actuel	Spam actuel	
	Ham prédit	484	29	
	Spam prédit	42	458	
Rapport de classification		Precision	Recall	F1
	Ham	92%	94%	93%
	Spam	94%	92%	93%



Interprétation LR

$intercept \approx -3.653$

Voici les coefficients les plus importants :

	÷	coeff ÷
word_freq_george		-14.085269
word_freq_cs		-6.625853
word_freq_3d		3.508508
word_freq_hp		-2.636963
capital_run_length_longest		2.052879
char_freq_\$		1.813182
word_freq_meeting		-1.768253
word_freq_telnet		-1.546523
word_freq_remove		1.462489
word_freq_lab		-1.291230
char_freq_hash		1.221557

On remarque que La fréquence du mot George est un fort indicateur de ham.

Cela correspond à [ce qui a été annoncé lors de l'exploration](#)

On remarque aussi que la fréquence de « cs » est aussi un indicateur de ham, et que « 3d » est un léger indicateur de spam.

Decision Tree Classifier (DT)

Il s'agit d'une représentation sur forme d'arbre binaire de la prédiction du spam.

Pour prédire un nouveau sample, on parcourt l'arbre. A chaque nœud, on teste la condition sur une feature (puisque toutes nos features sont numériques, la condition est toujours une inégalité), si la condition est validée, on navigue dans le fils gauche, sinon celui de droite. On répète cette opération jusqu'à atteindre une feuille, qui portera la classe spam ou ham.

Pour créer cet arbre, l'algorithme commence par comparer une feature et la cible. Imaginons le trainset suivant

Feature 1	Feature 2	Spam
2.5	3.5	1
-1	-1	0
1.1	2.2	0
-0.2	0.1	1

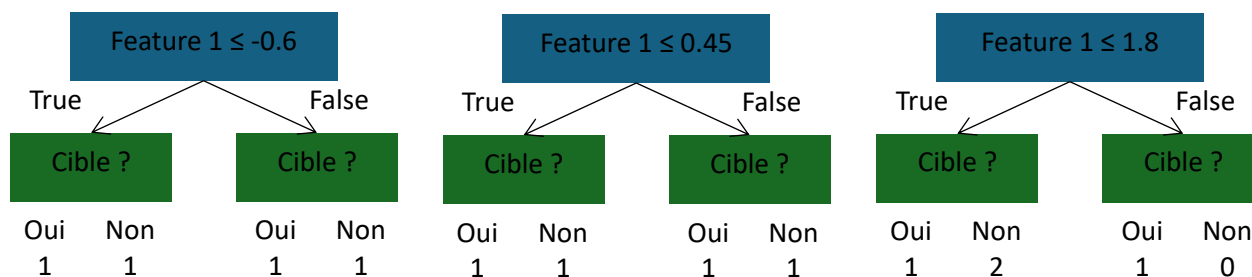
On trie par feature 1 croissante

Feature 1	Feature 2	Spam
-1	-1	0
-0.2	0.1	1
1.1	2.2	0
2.5	3.5	1

On calcule la moyenne d'un sample et de son voisin pour la feature 1

Moyenne sample un à un (feature 1)
$(-1 - 0.2) / 2 = -0.6$
$(-0.2 + 1.1) / 2 = 0.45$
$(1.1 + 2.5) / 2 = 1.8$

On crée alors un arbre pour chaque moyenne de la manière suivante :



On cherche à quantifier « l'incapacité d'un arbre à prédire que la cible soit un spam ou un ham ». Pour cela on a besoin d'un moyen de quantifier cette incapacité : c'est l'impureté.

Son calcul dépend de la fonction de quantification utilisée (c'est un hyperparamètre optimisable du modèle). On prendra pour exemple l'impureté Gini, dont la formule est

$$\text{Impureté Gini} = 1 - ((Tx \text{ de Oui})^2 + (Tx \text{ de Non})^2)$$

Prenons l'exemple du dernier arbre. On va calculer l'impureté de chaque feuille, puis de l'arbre :

Feuille		Impureté de la feuille	Impureté de l'arbre
1 Oui	2 Non	$1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = \frac{4}{9}$	$\frac{3}{4} \times \frac{4}{9} + \frac{1}{4} \times 0 = \frac{1}{3}$
1 Oui	0 Non	$1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0$	

(L'impureté de l'arbre est la somme pondérée des impuretés des feuilles)

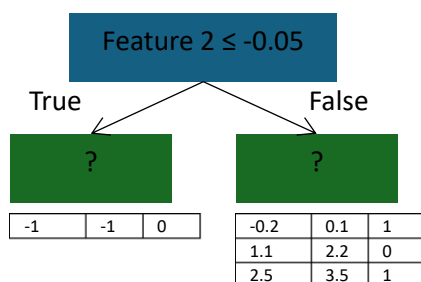
On répète cette opération pour les arbres 1 et 2, on trouve que leur impureté vaut 1.

En l'occurrence c'est donc le dernier arbre qui a la meilleure impureté (0.33 vs 1 vs 1) est qui est donc le mieux adapté pour prédire la cible, la meilleure condition est donc Feature 1 ≤ 1.8 .

C'est intéressant, mais on a fait que calculer l'impureté de la feature 1. Il nous reste à calculer celle de feature 2 pour savoir quel feature a la meilleure impureté, et quel sera donc la racine de l'arbre de décision.

Il ne reste plus qu'à recalculer l'impureté de feature 2. On trouve $impureté_{min} = \frac{1}{9}$, pour la condition Feature 2 ≤ -0.05 (...ou pour Feature 2 ≤ 2.85 d'ailleurs). L'impureté de la feature 2 (0.11) est donc inférieure à celle de la feature 1 (0.33).

C'est donc elle qui sera la racine.

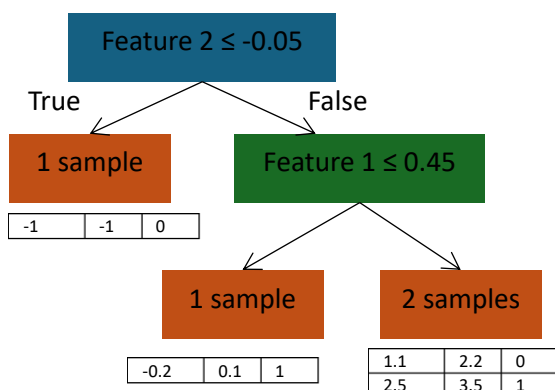


Maintenant que l'on a notre premier nœud, que doit-on faire ?

Et bien on répète l'opération entière aux deux nouvelles branches, sauf que cette fois, les samples seront différents.

De plus, **pour ces nœuds**, la condition ne portera pas sur la feature 2.

Lorsque qu'un nœud est pur (impureté = 0), ou qu'il ne contient plus qu'un sample, on ne répète pas l'algorithme. Car à ce moment, il n'y a plus besoin de séparer les samples, ils sont entièrement prédictibles.



Si mes calculs sont corrects, on finit par obtenir l'arbre suivant, les 2 premières feuilles ne contiennent qu'un sample, et la dernière est pure.

La dernière étape est d'attribuer une classe à chaque feuille, pour cela, on prend la classe majoritaire dans les samples.

Remarque : Quand des feuilles ne contiennent qu'un sample, c'est un signe que le modèle a été overfitted, pour éviter cela, on peut utiliser le pruning. Dans notre cas, puisqu'on utilise la cross validation, on peut fixer une limite de profondeur ou minimiser le nombre de sample par feuille. Cela peut entraîner des feuilles impures, mais une meilleure accuracy de la prédiction.

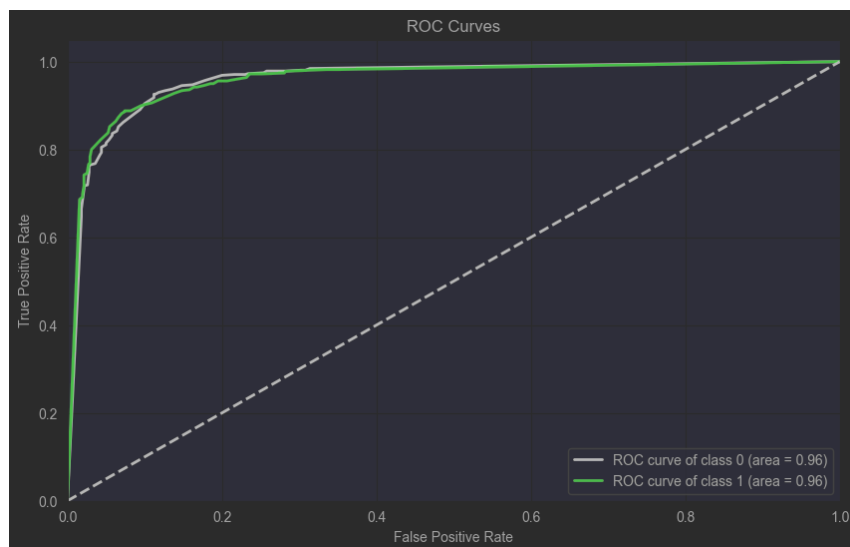
Ces contraintes correspondent à des hyperparamètres, dont voici les informations :

Hyperparamètre	Valeurs possibles	Définition	Interprétation
max_depth	3 None	Limite de profondeur de l'arbre	Permet d'éviter l'overfitting
max_features	Entier aléatoire entre 1 et 8	Nombre de features max à comparer lors d'un split	« Feature bagging », permet d'éviter l'overfitting.
min_samples_leaf	Entier aléatoire entre 1 et 8	Minimum de sample par feuille.	Permet d'éviter l'overfitting
criterion	gini entropy	Formule / fonction permettant de mesurer l'impureté des feuilles	Différents résultats à voir selon la fonction.

La complexité au pire de l'algorithme est $O(n \times m \times \log(n))$ avec n le nombre de samples et m le nombre de feature.

Résultat DT

Meilleur ensemble de paramètre	criterion=entropy max_depth=None max_features=8 min_samples_leaf=8			
Accuracy	90.3%			
F1 score	90.3%			
Matrice de confusion		Ham actuel	Spam actuel	
	Ham prédit	471	52	
	Spam prédit	56	444	
Rapport de classification		Precision	Recall	F1
	Ham	89%	92%	91%
	Spam	91%	89%	90%

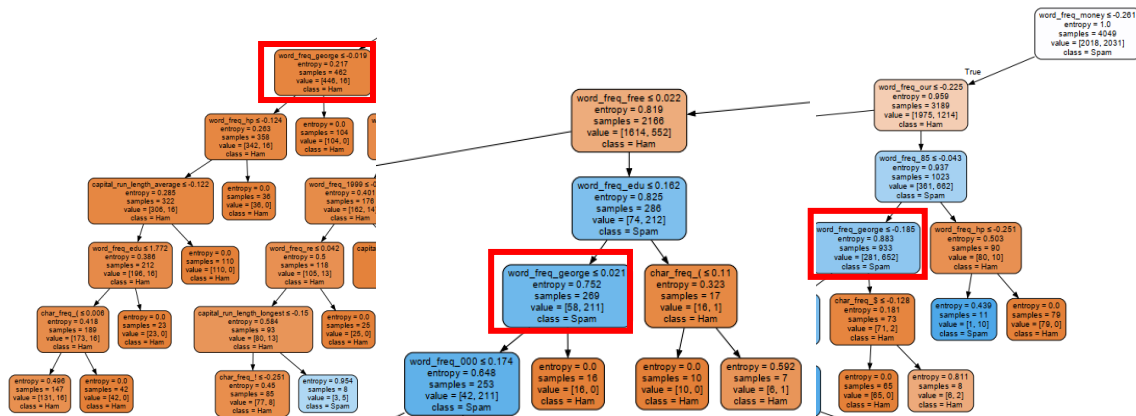


Interprétation DT

Pour une représentation de bonne qualité de l'arbre décisionnel : [DT.png](#)

Plus l'emplacement dans l'arbre est vers la gauche, plus la probabilité de ham augmente.

Si on s'intéresse aux occurrences de « george », on observe cela :

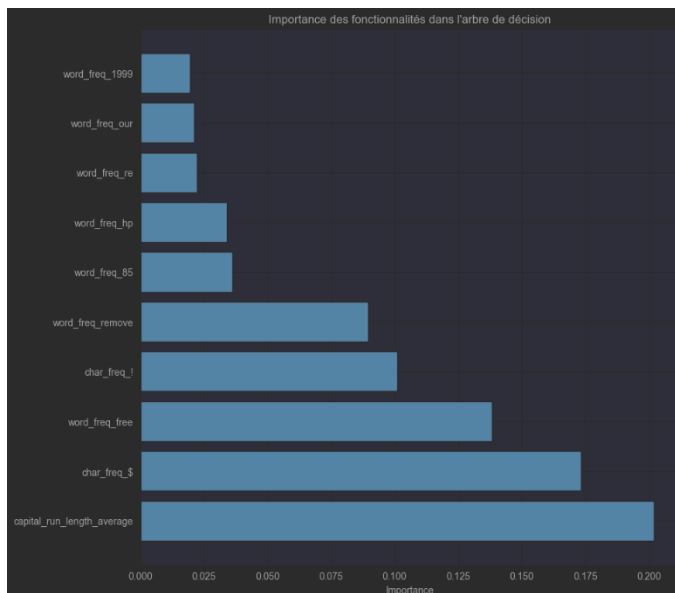


Un premier « george » est placé à l'extrême gauche de l'arbre (profondeur 10), là où la plupart des samples de ham sont présents. On remarque que tous ses fils mènent à un ham.

Un deuxième et troisième « george » sont placés plus proche de la racine (profondeur 4 et 3). Pour chacun, on remarque que leurs fils droit mène directement à un ham.

Il est difficile de savoir si ce qui a été annoncé lors de l'exploration est confirmé ici. Intéressons-nous plutôt à l'importance des features pour la détection de spam.

On remarque que les features suivantes sont indicateurs de spam d'après le DT :



La proportion de majuscule (~0.2)

La fréquence du caractère « \$ » (~0.175)

La fréquence du mot « free » (~0.14)

La fréquence du caractère « ! » (~0.1)

La fréquence du mot « remove » (~0.085)

On n'a pas d'informations sur la fréquence des mots « george » et « 650 », ils ne sont donc pas considérées ici comme indicateurs. On peut donc douter des performances du modèle.

Random Forest Classifier (RF)

D'après le livre *The Elements of Statistical Learning* ; « Il n'y a qu'un aspect qui empêche les arbres de décisions d'être un outil optimal pour la prédiction : l'imprécision ». Les DT sont bien pour leurs train-set, mais ne sont pas flexibles pour prédire de nouveaux échantillons. Les forêts aléatoires combinent la simplicité des DT et une flexibilité offerte par une vaste améliorations de l'imprécision.

Pour l'entraînement, on commence par créer un dataset « boostonné », la technique du bootstrap est une technique de rééchantillonnage où l'on sélectionne au hasard un sample et on le place dans un nouveau dataset. On peut donc trouver des duplicatas dans le nouveau dataset (avec cette technique, environ 1/3 des samples ne seront pas utilisés pour entraîner l'arbre, ce sont les out-of-bag samples).

A partir de ce dataset, on va créer un arbre de décision, mais les split se feront en comparant un nombre fixe de features (hyperparamètre *max_features*), typiquement on commence par la racine carrée du nombre total de feature. On répète la création d'arbre 50 ou 100 fois (hyperparamètre *n_estimators*), chaque arbre aura son propre bootstrapped dataset.

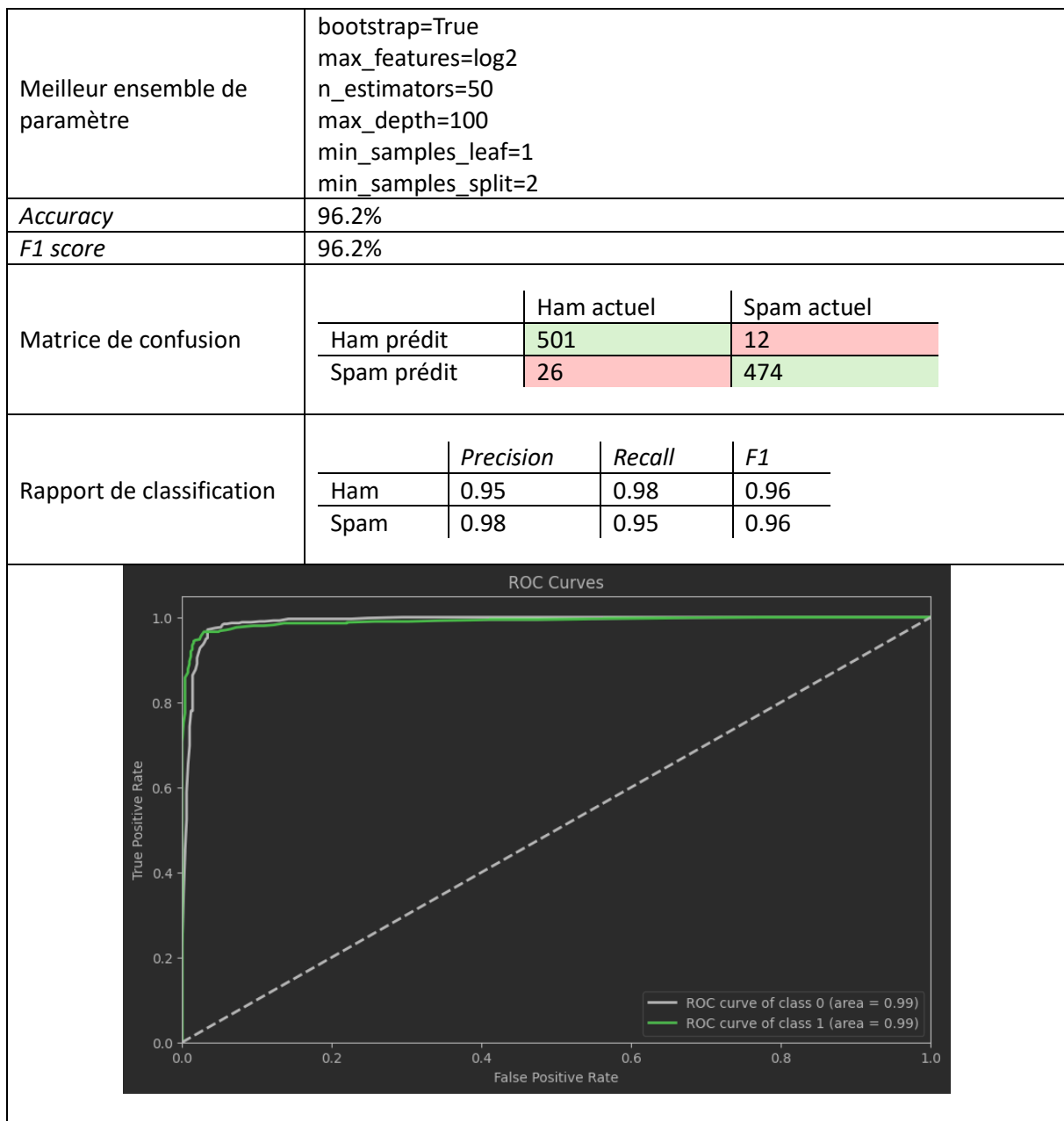
On obtient alors la forêt... Mais on ne s'arrête pas là, on va dupliquer cette forêt, supprimer les arbres, et les recréer avec un nouveau *max_features*. On compare alors l'erreur de prédiction de la nouvelle forêt et de l'ancienne forêt.

Pour évaluer l'erreur d'une forêt, on va prédire des samples. Mais pas n'importe lesquels, on va utiliser les out-of-bag samples ! On passe chaque sample dans chaque arbre. Après toute les prédictions faites, on garde le résultat avec le plus grand agrégat. Utiliser le bootstrapping et prédire grâce à l'agrégat des résultats s'appelle le « bagging ». L'erreur de la forêt est alors la proportion de out-of-bag samples qui ont été mal prédits, on l'appelle l'out-of-bag error.

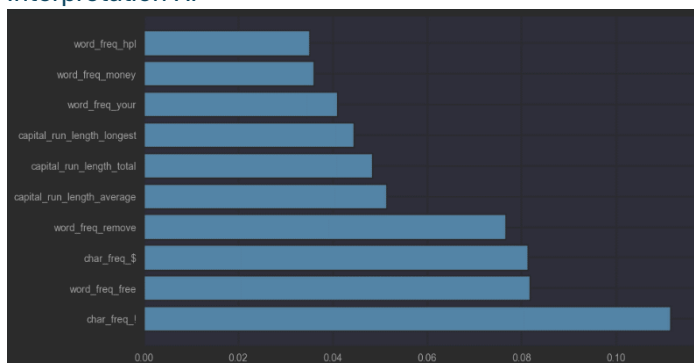
La complexité au pire de l'algorithme est $O(k \times n \times m \times \log(n))$, avec n le nombre de samples, m le nombre de feature, k le nombre d'arbres par forêt (ou nombre d'estimateurs).

Résultats RF

Hyperparamètre	Valeurs possibles	Définition	Interprétation
bootstrap	Booléen	Si on utilise le bootstrapping ou non	
max_features	None, sqrt, ou log2	Nombre de features aléatoires à comparer à chaque split.	Voir cette documentation pour plus d'info.
n_estimators	50 ou 100	Nombre d'arbre de la forêts	Permet d'éviter l'overfitting.
max_depth	10, 20, 50, 100 ou None	Paramètres utilisés par l'arbre de décision . Permettent d'éviter l'overfitting (et réduisent le temps d'exécution).	
min_samples_leaf	1, 2, 4, 10, 30 ou 100		
min_samples_split	2, 5, 10, 30, ou 100		



Interprétation RF

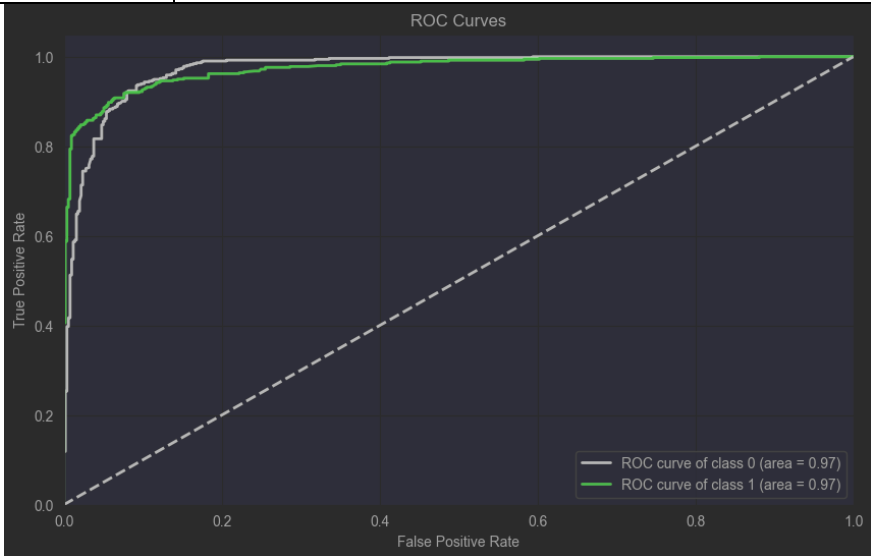


On obtient des résultats similaires au DT. Les indicateurs de spam semblent être la proportion de lettres en majuscule, le caractère « ! », « \$ », et le mot « free ». On n'a pas toujours pas d'informations sur la fréquence des mots « george » et « 650 ». On peut donc douter des performances du modèle.

Naïve Bayes Classifier (NB)

Hyperparamètre	Valeurs possibles	Définition		Interprétation
alpha	Flottant entre 0.01 et 10			
Meilleur ensemble de paramètre		alpha = 0.01		
Accuracy		91.3%		
F1 score		91.3%		
Matrice de confusion		Ham actuel	Spam actuel	
	Ham prédit	496	17	
	Spam prédit	71	429	
Rapport de classification		Precision	Recall	F1
	Ham	0.87	0.97	0.92
	Spam	0.96	0.86	0.91

ROC Curves



True Positive Rate

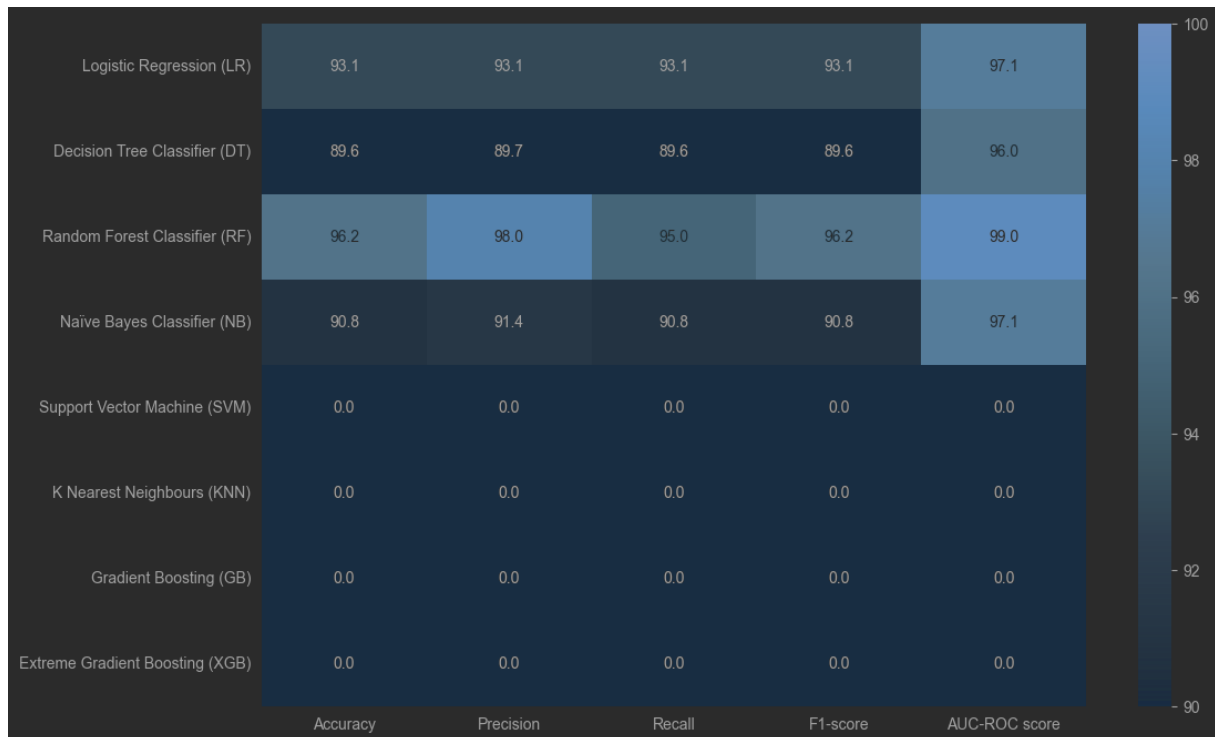
False Positive Rate

ROC curve of class 0 (area = 0.97)

ROC curve of class 1 (area = 0.97)

Conclusion

Voici la matrice de score des modèles, on peut voir que quatre modèles que l'on souhaitait tester n'ont pas été implémentés.



Pour notre objectif actuel, il est plus important de se concentrer sur le score de précision. Nous pouvons noter sur la heatmap que le modèle RF a obtenu de bons résultats

Voici quelques-uns des principaux résultats du projet :

- Le dataset était assez petit, comptant environ 4600 échantillons et après le prétraitement, 14,6 % des échantillons de données ont été supprimées.
- Les échantillons étant légèrement déséquilibrés, la technique SMOTE a été appliquée aux données pour équilibrer les classes, ce qui a permis d'ajouter 16,7 % d'échantillons.
- La visualisation de la distribution des données et de leurs relations nous a permis de mieux comprendre les relations entre les ensembles de caractéristiques.
- La sélection / élimination des caractéristiques a été effectuée et les caractéristiques appropriées ont été présélectionnées.
- Le test de plusieurs algorithmes avec des hyperparamètres de réglage fin nous a permis de comprendre les performances du modèle pour divers algorithmes sur cet ensemble de données spécifique.
- Le classificateur Random Forest et XG-Boost a obtenu des résultats exceptionnels sur l'ensemble de données actuel, en considérant le score de précision comme la mesure clé.
- Cependant, il est bien d'envisager également un modèle plus simple comme la régression logistique, car il est plus généralisable et moins coûteux en termes de calcul, mais il s'accompagne de légères erreurs de classification.