Dauphine | PSL
UNIVERSITÉ PARIS

M1-I2D

Rapport Détection de spams

Projet Machine Learning

Par :
Suruthy Sekar
Mathilde Da Cruz

12 mai 2021

Table des matières

1 Introduction

Les premiers spams, ou courriers indésirables, sont apparus au milieu des années 1990. Avec l'essor d'internet et de l'utilisation des boîtes mails, ce problème est devenu de plus en plus important. On considère aujourd'hui que plus de la moitié du traffic des courriels sont des spams. Outre les ennuis causés aux personnes qui les reçoivent (pollution de la boîte de réception, virus, phishing, etc.), les spams constituent également un enjeu écologique. La lutte anti-spam a donc débuté dès la fin des années 1990, mais elle a depuis beaucoup évolué, puisqu'elle doit s'adapter aux nouvelles techniques des spammeurs pour la contourner.

La question de la classification d'emails nous a attirées par son côté classique et à la fois nouveau - pour nous -, puisque n'ayant jamais traité le NLP. Désireuses d'étendre nos connaissances en matière de données et curieuses des techniques utilisées dans le *Natural Language Processing* (aussi appelé "traitement automatique des langues"), nous avons pensé que ce sujet serait intéressant et nous apporterait de nouveaux savoirs très utiles pour la suite. N'ayant ni l'une ni l'autre d'expérience concernant le traitement des données textuelles, ce sujet nous a demandé beaucoup de recherches dont nous parlerons ici.

Suite aux conseils de notre chargé de TP, nous avons écrit notre code dans le paradigme de l'orienté objet. Nous avons taché au mieux de commenter notre code afin de laisser les explications liées à ce dernier dans le document Jupyter, et se concentrer ici sur l'histoire de notre projet.

Tout au long de ce rapport, nous nommerons les courriels indésirables "spam" et les courriels légitimes "ham".

Nous avons réalisé ce projet sur Jupyter, et utilisé github pour gérer nos versions.

2 Choix du dataset

Nous avons eu le choix et hésité entre différentes bases de données pour ce travail.

- Spambase , une base de 4600 emails. Cette base de données semblait très bien pour une première approche, mais les données étant pré-traitées, nous avons préféré ne pas la choisir afin de procéder nous même au traitement au nettoyage et à la mise en forme de nos données.
- SMS Spam est également une base de données de UCI intéressante. Néanmoins, un SMS est bien plus simple qu'un email qui peut comporter de très nombreuses informations autres que le contenu du texte. Encore une fois, nous préférions avoir des messages dans un format brut.
- Enron Spam dataset est une base de données qui contient les mails pré-procéssés mais aussi bruts. Nous avons particulièrement hésité entre celle-ci et la suivante.
- SpamAssassin est celle que nous avons finalement choisie.

L'idée de créer notre propre dataset nous a traversé l'esprit ; en effet, nous ne manquions pas de spam dans nos propres boîtes mails. Cependant, la confidentialité des emails, notamment les emails "non spam", que nous allions traiter s'est avérée être un obstacle de taille. En effet, nous aurions besoin de faire des tests sur les emails et parfois, d'en lire le contenu. L'idée de faire notre propre base de données écartée, notre choix est donc resté sur le dataset de SpamAssassin.

Cette base de données très complète comporte non seulement un plutôt grand nombre d'emails, mais aussi de nombreux détails concernant ceux-ci : l'adresse de l'expéditeur, la date d'envoi, l'heure d'envoi, pour ne citer qu'eux. Elle a également l'avantage de proposer plusieurs niveaux de difficulté concernant les hams : Une catégorie *easy*, et une catégorie *hard*. De plus, cette base de données semblant être un incontournable dans le domaine du filtrage de spams, nous avons pensé que ce serait intéressant de la connaître.

3 Description de la tâche

3.1 Objectif du projet

Nous disposons d'une base d'emails déjà étiquetés. Le but de ce projet sera de parvenir à construire, à partir de ces derniers, plusieurs classifieurs, qui, pour un email donné, nous retournerons une prédiction quant à la nature désirable ou indésirable de ce dernier. Pour chacun des algorithmes, nous l'entraînerons sur une partie de la base de données, et le testeront sur une autre partie qui n'a pas été utilisée pour l'entraînement.

Il s'agit donc d'un exercice d'apprentissage supervisé pour de la classification.

L'une des subtilités dans la notion de spam est qu'elle peut varier d'un individu à l'autre : le spam de l'un peut être l'email légitime d'un autre. Il était donc important de nous attarder sur la construction d'un classfieur qui s'adapterait à l'individu et pourrait évoluer avec lui. Par exemple, si l'utilisateur du classifieur se prénomme X, alors les mails ayant son prénom dans leur corps seront à priori des emails légitimes. Il en va de même pour les expéditeurs : s'il s'agit de quelqu'un avec qui X a déjà correspondu, nous serons sûrs qu'il ne s'agit pas d'un email frauduleux.

Pour ce faire, la première étape est de préparer les données. En effet, les données que nous avons sont brutes : il y a un certain nombre d'information dans notre dataset qui ne nous serviront pas nécessairement dans la construction d'un classifieur efficace. Après cette étape, nous nous attendons aussi à devoir convertir les emails en un format qui soit simple et rapide à traiter. Enfin, l'idée est est de tirer les informations pertinentes de ces données textes avant de les faire passer dans les algorithmes de notre choix.

3.2 Méthodologie

Lors de l'études de textes, si chaque ligne représente un texte et chaque colonne un mot du vocabulaire total (c'est à dire l'ensemble de tous les mots utilisés dans l'union de tous les textes),

il est commun de se retrouver face à des matrices immenses mais aussi très creuses (un seul texte ne va contenir que relativement peu de mots par rapport à l'ensemble du vocabulaire), c'est à dire des matrices avec beaucoup de zéros. Nous avons donc pensé à plusieurs solutions que nous pourrons utiliser pour faire face à ce problème :

- L'utilisation de l'analyse en composantes principales (PCA). Cela devrait nous permettre de réduire le nombre de variables.
- L'utilisation de *stopwords*. Les *stopwords* sont une liste de mots très couramment utilisés et qui apportent peu de sens au message. Par exemple, pour des emails en fançais, une liste de *stopwords* pourraient contenir des pronoms, des verbes très utilisés comme être et avoir, etc. En enlevant les *stopwords* de la phrase "Le chat est dans le maison" on pourrait avoir "chat dans maison" voire "chat maison" selon la liste utilisée. Cela devrait également permettre de réduire le nombre de variables.
- La lemmatisation. Il s'agit d'un technique qui sert à transformer des mots en leur forme canonique. Un verbe deviendra l'infinitif, un pluriel le singulier. Cette technique nous semble intéressante puisqu'effectivement il semble inutile de garder comme variable le même mot mais conjugué à différents temps. La phrase "Les chats sont dans la maison" deviendrait "Le chat être dans la maison.". On peut compléter cette technique en utilisant ensuite les *stopwords*.
- Le *stemming*, ou racinisation. Cette technique consiste à remplacer chaque mot par sa racine. Par exemple, les mots "trouver" et "trouvaille" seront tout deux transformés en "trouv". Cette technique peut également être complétée par l'utilisation de *stopwords*.
- Enfin nous avons pensé qu'il serait peut être possible de classer les emails avec uniquement les objets des emails, et non pas leur contenu. En effet, les objets des emails contiennent beaucoup moins de mots que les textes. Travailler sur les objets devrait permettre de réduire considérablement le nombre de mots de notre vocabulaire, et ainsi de réduire la taille de nos tableaux.

Puisque nous voulons représenter nos données par un tableau dont les lignes sont les emails et les mots les colonnes, il faudra séparer les différents éléments d'une phrase. Cette étape est celle de la *tokenisation* pendant laquelle une phrase est transformée en liste de mots (de*tokens*). Nous avons choisi de ne garder que des mots (aucun caractère spécial, aucun chiffre). Il existe plusieurs méthodes pour cela, mais nous pensons utiliser les expressions régulières.

Nous avons pensé que le meilleur outil pour traiter nos tableaux de données serait le *DataFrame*, de la librairie *pandas*.

Nous avons pris le parti de tester plusieurs algorithmes de classification que nous trouvons particulièrement adaptés pour notre problème et d'en comparer les résultats. En effet, d'après le "No free lunch theorem" vu en cours, nous nous attendons à ce qu'il y ait des disparités entre les algorithmes. Nous nous demandons donc quel est celui qui sera le plus efficace d'un point de vue pratique et non seulement au niveau du pourcentage de classification correct. En effet, dans

le cadre très particulier de la classification des spams, il est aisé de voir comme un faux positif (un email désirable et légitime qui se place dans une les courriers indésirables) est infiniment plus problématique qu'un faux négatif (un email spam qui n'aurait pas été filtré). Cela est du au fait que les utilisateurs n'iront pas voir les spams de leur boite mail tandis qu'il leur est aisé de voir qu'un email est un spam.

C'est pour cela que nous avons décidé de nous baser sur les techniques de recherche d'information "bag of words" et "tf-idf". En effet, le dictionnaire légitime d'un individu lui est propre et peut évoluer au fil du temps. Travailler sur ces données nous permettrait d'avoir des performances plus intéressantes en matière de faux positifs (que nous cherchons donc à minimiser).

3.3 Résultats attendus

- Tout d'abord puisque nous avons décidé de tester chaque algorithme sur le contenu et sur l'objet, nous nous attendons évidemment à ce que les performances des algorithmes soient bien meilleures avec le contenu avec seulement l'objet. Neanmoins, n'utiliser que les objets promet un grand gain de temps et d'espace. Nous espérons donc que les performances sur les objets soient tout de même correctes.

- Nous mettons beaucoup d'espoirs sur la PCA. En effet, si cette technique fonctionne, cela devrait permettre également un gain de temps et d'espace mais en réduisant moins les performances qu'avec la technique ci-dessus. La PCA devant réduire la redondance d'information, il ne serait pas si étonnant qu'elle soit efficace sur notre problème, puisqu'il nous semble que le contenu de messages (qui sera donc nos variables) peuvent être redondants. Nous espérons notamment que la réduction de la dimension à 2 nous permette de voir que les données sont linéairement séparables, ce qui pourrait être très intéressant pour la suite.

- Concernant nos autres algorithmes, nous nous attendons à ce que Naïve Bayes soit particulièrement efficace sur le fait de minimiser les faux positifs. En effet, nous appuyons notre déduction sur le fait qu'un message frauduleux est bien plus caractéristique qu'un message qui ne l'est pas. Et Naïve Bayes est particulièrement efficace dans ce domaine.

- L'algorithme de SVM devrait, selon nos attentes, être très efficace à l'image de Naïve Bayes. En effet, d'après nos recherches, SVM est bien adapté pour le traitement de texte.

- Nous nous attendons également à ce que le random forest soit plus efficace que l'arbre de décision classique puisque ce dernier implique plusieurs arbres de décision pour classifier un email. De ce fait, nous pensons qu'il s'agira d'un processus coûteux et espérons que nos ordinateurs puissent le faire tourner en un temps raisonnable.

Pour résumer, nous nous attendons à de très bons résultats de tous les algorithmes. Nos attentes sont les plus élevées pour Naïve Bayes et SVM.

4 Description du dataset

Comme évoqué précédemment, les hams sont séparés en deux catégories "facile" et "difficile". Le fait que la base de données ait été réapprovisionnée quelques mois après sa création explique également le nombre de dossiers :

- easy_ham, qui comporte 2500 mails légitimes ;
- easy_ham_2, qui comporte 1400 mails légitimes ;
- hard_ham, qui comporte 250 mails légitimes ;
- spam, qui comporte 500 mails légitimes ;
- spam_2, qui comporte 1396 mails légitimes.

Notons qu'il y a donc un total de 6046 emails, dont 4150 hams et 1896 spams, soit environ 31% de messages indésirables, ce qui est probablement un peu inférieur au taux réel de nos jours.

Chaque email est du type 'email.message.EmailMessage' et est organisé en plusieurs sections, les plus communes étant 'From','Date','Subject', 'Received', 'To'.

Les catégories 'From', 'Date' et 'Subject' sont présentes dans tous les emails (sauf 6 pour subject mais il ne nous semble pas problématique de supprimer seulement 6 exemples), et nous semblent pertinentes à garder pour la classification d'emails en spam ou ham.

Il nous semblait intéressant de tester des algorithmes sur d'autres paramètres que le contenu ou le sujet. En effet l'heure d'envoi d'un message ou encore le domaine de l'adresse mail de l'expéditeur pourraient être des variables qui auraient permis de classifier les emails. Malheureusement nous n'avons pas eu le temps d'essayer cela.

5 Pré-traitement

Le pré-traitement est effectué grâce à la classe **Raw_data** de notre code.

5.1 Chargement des emails

Maintenant que nous avons récupéré les fichiers sur le site de SpamAssassin, il nous faut maintenant charger les emails sur Jupyter afin de les traiter par la suite. Dans un souci de généricité, nous avons tenu à écrire une fonction générique qui se chargerait elle-même de décider si un certain fichier est un ham ou un spam en fonction de s'il contient l'un de ces deux termes dans son titre (à la manière d'un humain donc). Ainsi, nous avons pu récupérer le nom des dossiers contenant les spams d'un côté et les hams de l'autre. Ce travail est effectué avec la méthode **expl()**.

Une fois ces dossiers classés, nous nous sommes occupées d'importer les emails et de les "parser" à l'aide d'un parser, BytesParser pour être précises. En effet, c'est le Parser qui a semblé être le plus adapté compte tenu du fait que nous nous attendions d'ores et déjà à être confrontées à plusieurs types d'emails différents. BytesParser nous a retournées une liste d'objets de la classe EmailMessage. Il s'agit un type assimilable à la classe dictionnaire qui dispose ainsi d'une liste de *keys* et de *values* associées. Ce travail est effectué par la méthode **load_email()**.

Pour finir cette première étape de pré-traitement, nous avons produit des histogrammes nous montrant la répartition des différents types présents dans notre dataset. Nous trouverons ces deux histogrammes ci-dessous.

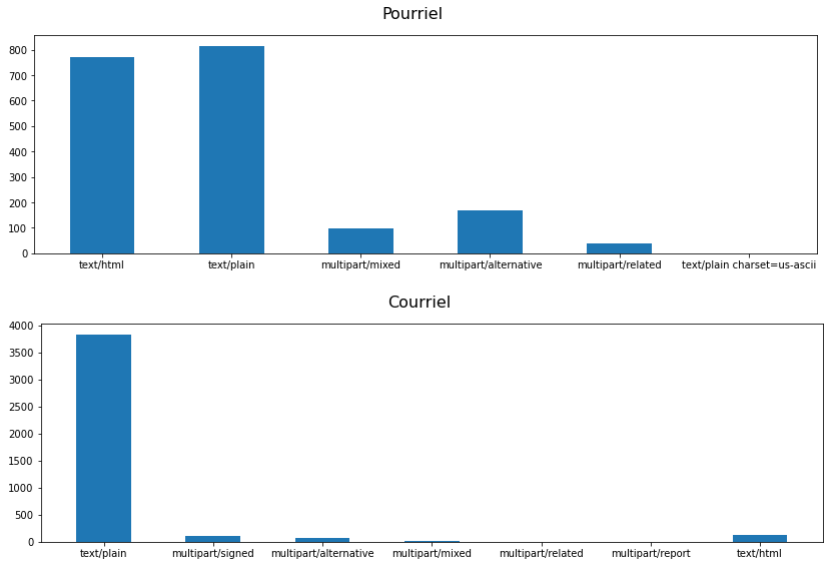

Figure 1: La répartition des différents types d'emails

C'est à cette occasion que nous remarquons qu'il y a trois grands types qui se dégagent :

- text/plain, il s'agit du type classique, qui ne contient que le texte brut
- html/plain est la même chose que text/plain mais est écrit en html
- multipart est le type généralement utilisé lorsque des images ou autres fichiers sont joints dans le corps de l'email. Ce type est hiérarchisé en plusieurs autres types notamment html/plain et/ou text/plain.

Ce travail est effectué par la méthode **show_types()**.

5.2 Conversion des emails

Nous avons ensuite créé la classe **Processed_data** qui nous servira à mettre les emails dans un format pratique pour utiliser nos algortihmes ensuite.

	Label	From	Date	Subject	Content
0	0	Robert Elz <kre@munnari.OZ.AU>	Thu, 22 Aug 2002 18:26:25 +0700	Re: New Sequences Window	Date: Wed, 21 Aug 2002 10:54:46 -05...
1	0	Steve Burt <Steve_Burt@cursor-system.com>	Thu, 22 Aug 2002 12:46:18 +0100	[zzzzteana] RE: Alexander	Martin A posted:\nTassos Papadopoulos, the Gre...
2	0	Tim Chapman <timc@2ubh.com>	Thu, 22 Aug 2002 13:52:38 +0100	[zzzzteana] Moscow bomber	Man Threatens Explosion In Moscow \n\nThursday...
3	0	Monty Solomon <monty@roscom.com>	Thu, 22 Aug 2002 09:15:25 -0400	[IRR] Klez: The Virus That Won't Die	Klez: The Virus That Won't Die\n \nAlready the...
4	0	Stewart Smith <Stewart.Smith@ee.ed.ac.uk>	Thu, 22 Aug 2002 14:38:22 +0100	Re: [zzzzteana] Nothing like mama used to make	> in adding cream to spaghetti carbonara, whi...

Figure 2: Dataframe représentant le corpus

Puisque les emails sont dans des types différents, nous avons écrit plusieurs méthodes pour que l'ensemble des emails soit dans un même et unique format : text/plain. Cette uniformisation rendra le traitement et l'analyse des données avec un classfieur bien plus simple.

Nous avons par exemple utilisé la librairie *BeauifulSoup* pour la conversion du html vers le texte brut. Pour le type multipart, nous avons simplement parcouru la structure en arbre de celui-ci jusqu'à trouver ou bien un type text/plain que nous utiliserions ou un type html/plain que nous pourrions donc convertir vers du text/plain.

6 Analyse

6.1 Nettoyage et préparation des données

Avec les étapes précédentes, nous avons pu construire le dataframe (dans l'attribut *corpus*) dont voici quelques lignes ci-dessous.

Afin de traiter nos textes, nous allons donc créer un nouveau dataframe, qui n'utilisera dans le corpus que la colonne qui nous intéresse (donc le contenu ou l'objet) et construira un vecteur de caractéristiques. C'est à dire que nous allons transformer nos données catégorielles en données numériques.

6.2 Construction de vecteurs de caractéristiques

6.2.1 Bag of word

Pour construire nos vecteurs de caractéristiques, nous utiliserons le modèle du *bag of words*. C'est à dire que pour chaque document, nous allons construire un vecteur qui compte le nombre d'occurences d'un mot dans un document. Plus précisément, nous construisons pour chaque email, un dictionnaire. Ensuite tous ces dictionnaires seront transformés en un unique DataFrame. Si une case du DataFrame est vide, cela signifie que le document à la ligne correspondante ne contenait pas le mot de la colonne correspondante et il suffira donc de remplacer ces cases vides par des 0.

6.2.2 Tokenization

Notre méthode qui construit le *bag of words* appelle la méthode **tokenize()**. Ainsi le DataFrame du *bag of words* est directement rempli avec les tokens. **tokenize()** utilise les expressions régulières pour ne trouver que les mots de plus de 2 lettres dans le document. Après avoir transformer un document en liste de mots, notre "tokenizer" va ensuite filtrer afin de garder que les mots lemmatisés et qui ne sont pas dans les *stopwords*. Nous avons préféré utiliser la lemmatisation plutot que le *stemming*, car nous avions peur que le *stemming* retire trop de sens aux mots.

A la suite de ces étapes, nous obtenons un DataFrame comme suit :

	date	wed	aug	chris	garrigues	cwg	deepeddy	com	message	tmda	...	aggrandize	composure	soothes	refreshes	hairricane	pkg	drops	adsub	k
0	2.0	1.0	1.0	1.0	1.0	1.0	2.0	4.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

Figure 3: Dataframe représentant le bag of words

Il sera stocké dans l'attribut "bow".

L'utilisation des différentes techniques évoquées plus haut pour diminuer le nombre de variables a permis de passer d'environ 70 000 variables (donc 70 000 mots dans le vocabulaire) à environ 60 000.

NB Notons que dans notre *bag of words*, chaque colonne représente un seul mot. Nous avons donc utilisé le modèle *1-gram*.Le modèle *n-grams* serait un bag of words dont les colonnes sont des éléménts de n mots. Par exemple la phrase "Les chats sont dans la maison", en suivant un modèle *2-grams* donnerait les colonnes "les chats", "chats sont", "sont dans", "dans la", et "la maison. Le choix du n dépend de l'utilisation souhaitée pour le *bag of words*. Certaines études suggèrent que pour n = 3 ou 4, on obtient de bonnes performances pour la detection de spams. Néanmoins, cela aurait énormément élargi notre vocabulaire (et ainsi le nombre de colonnes et donc la taille du dataframe), c'est pourquoi nous n'avons pas utilisé cette technique.

6.2.3 TF-IDF

Certains mots peuvent apparaître dans tous les documents mais sans apporter beaucoup de sens, ou tout d'un moins d'information très discriminante pour notre classification. La technique du tf-idf (Term frequency - inversed document frequency) est une méthode de fouilles de textes dont le but est de trouver les informations importantes d'un message. Un mot sera important s'il apparait beaucoup dans le texte (relativement au nombre de mots du texte) mais peu dans le corpus (relativement au nombre de documents). Le tf-idf est donc le produit de la fréquence d'un terme t par rapport a un document d, par l'inverse de la fréquence de ce terme t. La formule peut être définie comme suit :

$$\text{tf-idf}(t,d) = \text{tf (t,d)} \times \text{idf}(t)$$

Avec $tf(t,d)$ étant donc simplement le nombre d'occurences du mot t divisé par le nombre de mots de d. Et $idf(t)$ est donc l'idf normalisée qui peut être définie comme suit :

$$\text{idf}(t) = \log \frac{n_d}{1 + \text{df}(t)},$$

n_d étant le nombre de documents total du corpus. et $df(t)$ est le nombre de documents qui contiennent le terme t.

Ce score est donc calculé pour chaque mot par rapport à chaque document. Si le mot t n'apparait pas dans le document d, un score de 0 sera attribué. Néanmoins, un score de 0 peut aussi indiquer que le mot n'est pas du tout important.

Nous avons implémenté cette fonction dans la classe **Processed_data**. Cette dernière fonctionne bien sur de petites bases de données, mais elle n'est pas assez efficace et optimisée, et prend donc trop de temps. Nous avons donc préféré utiliser le TfidfTransformer de sklearn.

6.2.4 PCA

Nous avons commencé par utiliser la PCA pour réduire la dimension des données (plus de 60 000 au départ) à 2 pour voir si les données étaient linéairement séparables.

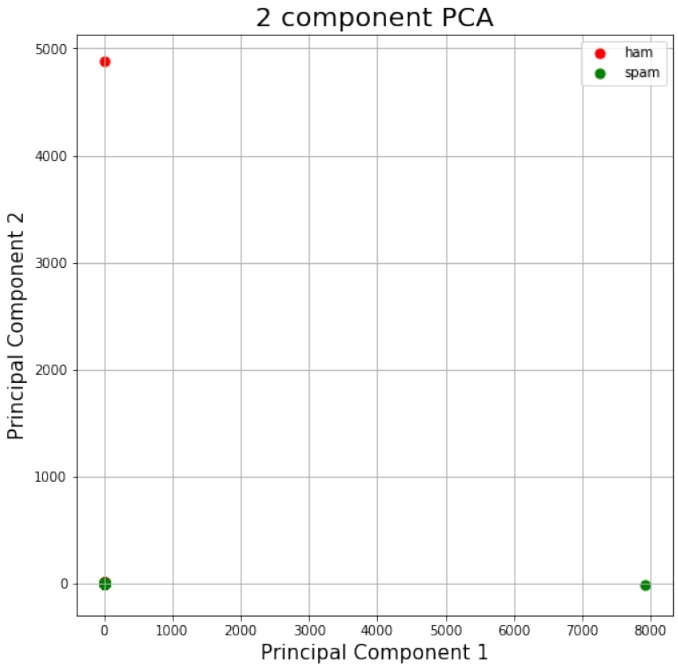

Figure 4: Résultat du PCA

Comme nous le remarquons, les résultats ne sont malheureusement pas concluants. En effets, les données se confondent beaucoup et ne sont pas du tout linéairement séparables, à l'exception de quelques *outliers*. Ainsi, ces données ne seront pas exploitables et nous passons aux autres algorithmes de classification.

6.3 Régression Logistique

Nous avons obtenu de très bons résultats avec la régression logistique. Néanmoins il est intéressant de remarquer que la précision lorsque qu'on utilise bag of words est quasiment la même que lorsqu'on utilise tfidf. On pourrait donc penser que c'est dommage de dépenser autant de mémoire et de temps pour calculer le tableau de tfidf s'il y a si peu de différence. Mais malgré une précision quasi identique il est important de remarquer que la régression logistique avec tfidf trouve moins de faux positifs, ce qui est avantageux.

6.4 Naïve Bayes

Naïve Bayes donne une bonne précision mais le taux de faux positifs n'est pas très bon, en particulier lorsque la recherche ne se fait que sur les objets (81% de faux positifs!).

6.5 SVM

Pour cet algorithme, nous avons utilisé un GridSearchCV issu de la librairie sklearn. Cette méthode permet de faire une recherche exhaustive parmi tous les paramètres proposés afin de trouver le meilleur set de paramètres pour un estimateur. Pour le paramètre C nous avons essayé [0.1,0.5,1,1.5,2,5,10,50], car après plusieurs tests manuels il nous avait semblé que le meilleur C se trouvait autour de 1. Et pour kernel nous avons essayé avec 'linear' et 'rbf'.

Cette méthode de recherche de paramètres est assez couteuse en temps. C'est pourquoi nous n'avons pas pu l'essayer avec le svm sur toute la base de données lorsqu'on se basait sur le contenu du message. Ce calcul aurait pris environ 6h et nous n'avons pas eu le temps de le tester jusqu'au bout. Nous avons donc appliqué cette méthode sur seulement 10% de la base de données. Les resultats étaient bons (95%). En essayant cette méthode sur les objets des messages, nous avons obtenu (avec le meilleur set de paramètres qui était 1.5 et 'linear') , nous avons obtenu 90% de précision. Etant donné qu'à chaque fois, le resultat sur le contenu est un peu meilleur d'au moins 5%, on peut supposer que le 95% trouvé avec une base de données très réduite n'est pas très loin de la réalité.

Notons également que le SVM (sur les objets) trouve environ autant de faux positifs que de faux négatifs, ce qui n'est pas très bon puisqu'on veut éviter les faux positifs.

6.6 Arbres de décision

6.6.1 Arbres de décision classiques

Pour cet algorithme, nous avons choisi nous même les paramètres. En ce qui concerne le calcul de l'impureté nous avions le choix entre la formule de Gini et celle de l'entropie. Pour une profondeur de 4 (dans les deux cas), la précision était de 87% dans les deux cas. Cela rejoint ce que nous avons pu voir en cours de machine learning, c'est à dire que l'un n'est pas drastiquement meilleur que l'autre. Nous obtenions de meilleurs résultats avec une profondeur maximale de 10 (92% donc toujours moins que nos meilleurs algorithmes) mais nous avons préféré garder 4 pour garder une bonne lisibilité de l'arbre crée, puisque un des gros points forts des arbres de décision est sa lisibilité et sa compréhension par un individu extérieur.

Un très bon point pour les arbres de décision et qu'il repère particulièrement bien les messages légitimes. Il est d'ailleurs assez étonnant de voir que, même si la précision de l'arbre de décision sur les objets est relativement mauvaise (environ 71%), ce dernier trouve presque tous les emails légitimes et n'aura donc placé que 3 emails désirés (avec entropy, et 8 avec Gini) dans les spams.

L'arbre de décision ci-dessous est le classifieur entrainé sur les objets des emails et utilisant Gini.

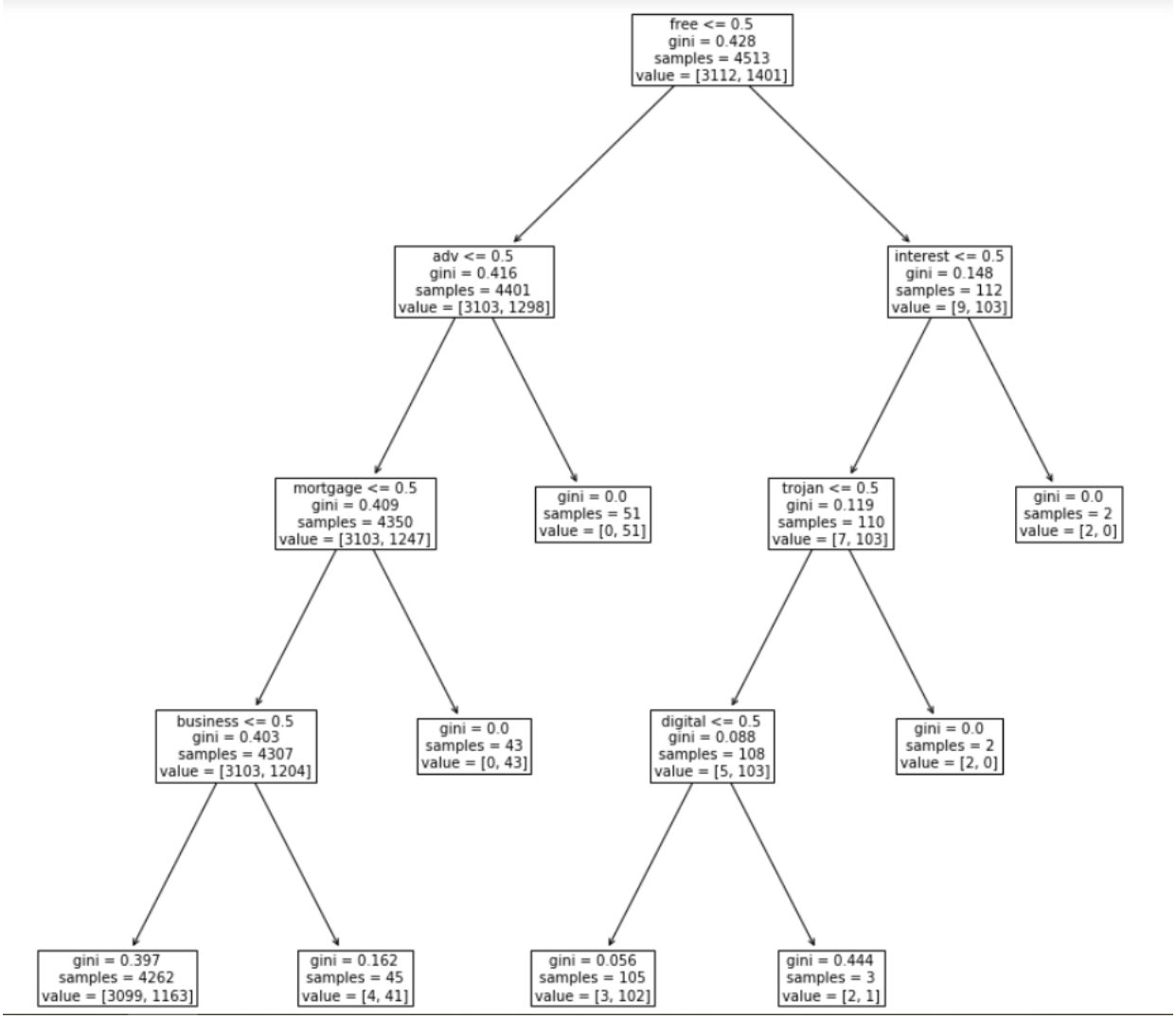

Figure 5: Classfieur entraîné en utilisant Gini

6.6.2 Random Forest

Nous pensions que cet algorithme fonctionnerait particulièrement bien donc pour lui donner toutes ses chances nous avons là aussi utilisé la méthode du grid search qui teste plusieurs combinaisons de paramètres pour trouver la meilleure. Le meilleur set de paramètres lorsque nous avons testé cette méthode sur les contenus des emails était : Gini pour le calcul de l'impureté ; 10 pour la profondeur maximale de l'arbre ; et 150 pour le nombre d'arbres aléatoires créés. Notons que ce set de paramètres diffère légèrement lorsqu'on applique cette méthode aux objets des mails : le meilleur nombre d'arbres aléatoires créés est 150. Comme on peut le voir dans le tableau en dernière partie, les resultats de ce classifieur sont bien moins bons que les autres classifieurs. En particulier, ils sont moins que ceux de l'arbre de décision classique, ce qui nous a plutot étonné.

Tout comme pour l'arbre de décision classique, même si la précision est relativement mauvaise, la foret aléatoire n'a trouvé qu'un seul faux positif lorsqu'elle utilisait les contenus, et aucun lorsqu'elle utilisait les objets ! Le Random forest qui filtre selon les objets des messages est donc le meilleur si on veut être certain de n'avoir aucun message légitime dans les spams, mais malheureusement, on risque d'avoir pas mal de spams dans la boite de réception.

6.7 Autres

Il existe d'autres algorithmes que nous n'avons pas pu choisir par manque de temps. Nous aurions beaucoup aimé essayer le méthode du "out-of-core" learning, qui est normalement utilisée pour des bases de données beaucoup plus grandes que la notre, mais que nous aurions aimé découvrir plus en profondeur néanmoins. Cette méthode permet de travailler sur de très grandes bases de données (ou bien simplement de gagner beaucoup de temps sur notre base de données pas si grande !) ou entrainant le classifieur au fur et à mesure avec de plus petits sets de données (minibatchs).

Enfin, il aurait également été possible d'utiliser tensorflow (en utilisant des Recurrent Neural Networks) mais n'ayant encore jamais vu ces techniques en cours cela nous a semblé un peu trop compliqué et un peu trop long pour se lancer dedans maintenant, mais ce seront probablement des techniques que nous pourrons utiliser l'an prochain !

7 Résultats finaux - Conclusion

Note : les résultats affichés sur Jupyter et ceux traités dans le rapport sont très légèrement différents. En effet, nous avons été obligées de relancer tous les calculs suite quelques petites corrections dans le code. Mais tout ce qui est dit ici s'applique toujours.

Après nos trois classes, nous avons finalement écrit une fonction qui nous permet de regrouper tous les résultats obtenus jusque là dans un tableau, que nous joignons ci-dessous.

	Content			Subject		
	Accuracy	Faux positif	Faux négatif	Accuracy	Faux positif	Faux négatif
Prédicteur						
Naïve Bayes	0.951495	16.0	57.0	0.872425	156.0	36.0
Régression logistique (TFIDF)	0.974086	5.0	34.0	0.890365	22.0	143.0
Régression logistique (BOW)	0.972757	25.0	16.0	0.889037	46.0	121.0
svm	0.947020	1.0	7.0	0.903654	71.0	74.0
Arbre de décision (Gini)	0.867774	35.0	164.0	0.709635	8.0	429.0
Arbre de décision (Entropy)	0.866445	40.0	161.0	0.720930	3.0	417.0
Random Forest	0.829236	1.0	256.0	0.677076	0.0	486.0

Figure 6: Résultats finaux

7.1 Application au contenu

Comme attendu, Naïve Bayes a eu une précision très bonne (95%). Cependant, il présente un taux relativement haut de faux positifs (21.9%), ce qui nous surprend un petit peu.

Celui qui tire son épingle du jeu en termes de faux positif est Random Forest qui en a un taux très faible - presque nul. Même son de cloche pour SVM qui a taux de faux positif très proche de 0 mais qui a aussi l'avantage d'avoir une précision très élevée avec 94%.

Nous pouvons aussi noter que les deux régressions logistiques (une avec TFIDF et l'autre avec bag of words), bien qu'ayant exactement la même précision, ont des résulats vastement différents en terme de faux positifs et faux négatifs. En effet, la régression logistique avec TF-IDF est bien plus performante pour la minimisation des faux positifs que TF-IDF avec bag of words. Ce dernier point ne faisait pas partie des résultats attendus ; nous n'étions pas même sûres que les deux auraient grande différence.

Enfin, sans grande surprise, nous notons que les arbres de décision, que ce soit avec Gini ou l'entropie ont des résultats très similaires entre eux mais ont aussi exactement la même précision que le random forest. Cependant, Random Forest est bien plus performant pour minimiser les faux positifs - ce qui fait partie des résultats attendus.

7.2 Application à l'objet

A notre grand plaisir, nous observons que les résultats des algorithmes appliqués à l'objet de l'email cette fois (et non au contenu) sont loin d'être décevants. En effets, les différences entre les algorithmes évoqués à la section précédente se retrouvent ici. La seule différence est la précision, qui est globalement moins bonne. Cependant, la complexité en temps (les algorithmes

qui n'utilisent que les objets prennent 10 fois moins de temps que ceux sur les contenus !) et en espace est bien meilleure, ce qui est un avantage non négligeable et pourrait, dans certains cas, justifier cette précision moins bonne. En effet, le vocabulaire sur tout le contenu fait 60 404 mots contre seulement 4963 sur les objets.

7.3 Conclusion

Ce projet a été extrêmement formateur pour pour notre binôme. Nous avons appris quantité de choses, notamment sur les librairies Pandas et scikit-learn, sur le langage Python en général, mais aussi sur le *Natural Language Processing*. Nous avons également eu plaisir à développer nos connaissances et appliquer les différents algorithmes vus en cours sur une base de données qui a été préparée par nos soins. Cela a permis une mise en situation réelle où nous avons du passer de données brutes, récupérées sur Internet, à un résultat satisfaisant.

En particulier, nous avons particulièrement pu approfondir nos capacités d'organisation, qui nous le pensons, ont été fondamentales à la réussite de ce projet. Le travail d'équipe et les capacités de communication ont également été décisives quant au bon déroulement de ce projet.