SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

3I005

Projet 1

You've got mail!

Encadré par Eloi Zablocki

Réalisé par :

AMROUCHE Sara (3523540)

RIABI Arij (3702151)

Sommaire

- **Introduction**
- **Classification d'emails et détection de spam**
  - **Classification à partir de la longueur d'un email**
  - **Classification à partir du contenu d'un email**
- **Visualisation**
- **Conclusion**

Introduction

Dans le cadre de l'UE 3I005, nous avons travaillé sur un premier projet traitant différentes implémentations de la classification bayésienne afin de classifier des emails (spam ou non-spam).

En effet, les spams ou les pourriels autrement dit, représentent une très importante partie du trafic mondial des courriels ce qui, bien évidemment, met la sécurité de nos données personnelles en danger de piratage..., d'où l'importance de trouver des algorithmes pour lutter ce genre de fléau efficacement.

Tout d'abord, il faut bien comprendre comment distinguer entre un courriel désirable et non désirable, comment pouvoir juger la pertinence du contenu d'un courriel, et c'est là qu'on touche au cœur du problème de classification. On cherche ici à être capable d'emmètre un jugement concernant l'email et cela en analysant plusieurs critères qui seront détaillés plus loin dans le rapport.

Nous allons utiliser des techniques se basant sur la longueur puis sur filtrage du contenu, ces techniques peuvent être considérés comme des opérations statiques consistant à regrouper les emails en deux groupes et d'identifier l'appartenance d'un nouvel email à un de ces deux groupes dont les caractéristiques distinctives seront connues grâce aux données d'entrainement du modèle.

Le défi est donc de tester plusieurs critères et de conclure le meilleur sur lequel se baser afin de détecter le spam rapidement et d'éviter de classer des courriers légitimes en tant que spam. Les étapes et les algorithmes ainsi que les critères utilisés seront détaillés tout au long du rapport.

1 Classification d'emails et détection de spam

Classification à partir de la longueur d'un email

On a la probabilité que l'email soit un spam sachant sa description **x** (la longueur : soit le nombre de caractères où bien le nombre de mots) qui est de :

P(y=+1|x) = [P (X = x | Y = +1) P (Y = +1)] / P (X = x)

Ce qui correspond au produit de la probabilité que la longueur de l'email soit la même (c'est-à-dire x) sachant qu'il est un spam (P (X = x | Y = +1)) et la probabilité que le mail est un spam quelques soit sa description (P (Y = +1)) divisée par la probabilité que le mail soit de longueur x (P (X = x)).

-on fait la comparaison de P(y=+1|X=x) et P(y=-1|X=x) et par l'application de Bayes on obtient :
Pour P (y=+1| X= x) on a : [P (X = x| Y = +1) P (Y = +1)] / P (X = x)
Et pour P (y=-1| X=x) on a : [P (X = x| Y = -1) P (Y = -1)] / P (X = x)
On peut ainsi comparer les numérateurs entre eux vu qu'on dispose du meme dénominateur (P (X = x)).
C'est-à-dire [P (X = x| Y = +1) P (Y = +1)] et [P (X = x| Y = -1) P (Y = -1)], et donc il n'est pas nécessaire de calculer P (X = x) ce qui nous arrange bien car on ne dispose pas de moyen pour calculer une telle probabilité (on ne peut pas juste prendre un mail et estimer la probabilité qu'il ait une longueur donnée). En revanche on a besoin de connaitre P(Y=+1).

-Afin d'estimer la distribution P (X = x| Y =+1) :
On compte le nombre de mails spams dont la longueur est x, c'est-à-dire on regarde le nombre de mails qu'on obtient pour une description donnée (description pour spam) sur un échantillon d'email, ou bien le nombre de mails dont la longueur se trouve dans un intervalle donné (bins de l'histogramme), et comme dans ce cas, on a besoin de calculer

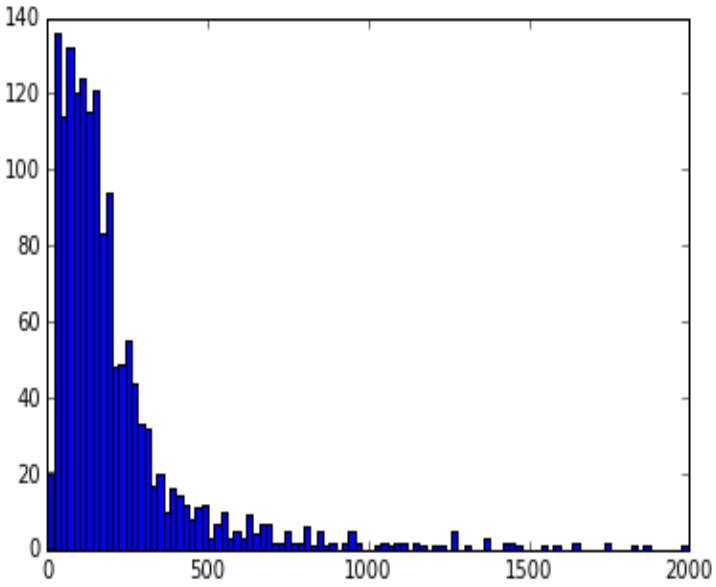
p(y=1) et p(y=-1), donc on va considérer une distribution uniforme telle que p(y=1) = p(y=-1) = ½ pour le reste de l'expérience. Ainsi notre priori est uniforme et P (X = x| Y =-1) = # (Y=1- inter Xi=xi) / #(Y=-1)

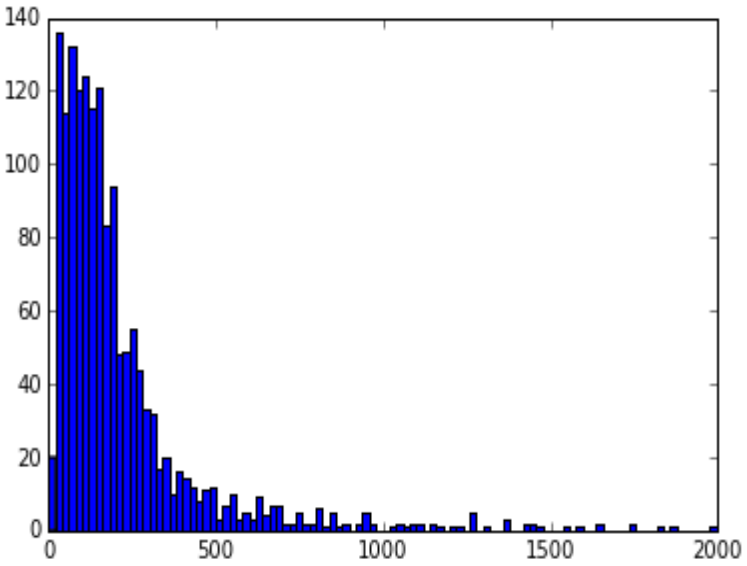
Et puisque on ne va pas prendre en considération p(X=x) et si l'on prend P(y=+1) =0.5 la comparaison entre P(y=+1|X=x) et P(y=-1|X=x) et revient à comparer P (X=x| Y=+1) et P (X=x| Y=-1).

Les Fonctions calculant la longueur de l'email :

Dans ce cas, on a considéré deux cas, la longueur de l'email en prenant en compte le nombre de caractères qu'il contient, mais aussi la longueur de l'email selon le nombre de mots qui le composent, et dans les deux cas on a obtenu les mêmes variations dans les histogrammes pour les spam et non spam. Et il n'est pas raisonnable de considérer tous les cas possibles, il serait donc plus raisonnable de regrouper les mails par intervalles (bins de l'histogramme) car peu de mails auront exactement la même longueur (que par chance) et comme la longueur des emails peut se rapprocher il serait préférable de regrouper dans un seul intervalle les emails qui ont des longueurs plus ou moins similaires.

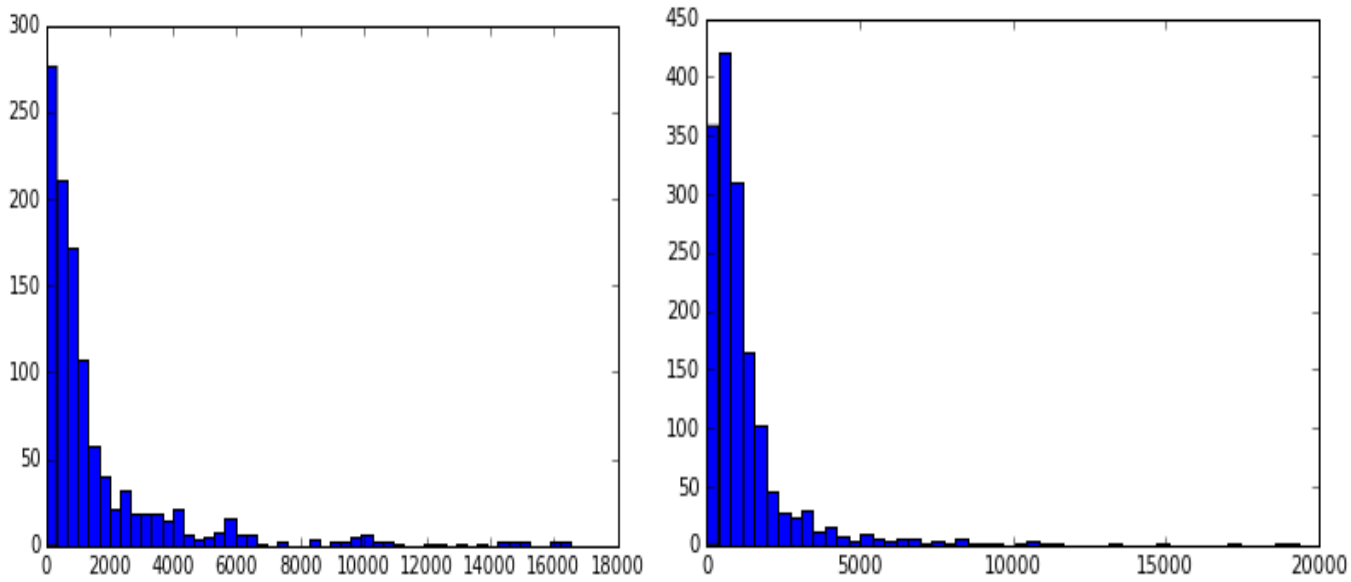
On a ci-dessous les deux histogrammes obtenus en ayant pris en compte le nombre de mot dans chaque email jusqu'à 2000 mots, (emails spam à gauche et nonSpam à droite).





On constate que pour les deux types d'emails, on a des histogrammes qui se ressemblent, avec une majorité d'email ayant des longueurs plutôt inferieurs à 500 mots par emails dans les deux cas, mais on a aussi des emails avec des un nombre de mots arrivant jusqu'à 2000 mots mais cela reste une minorité de nos deux ensembles.

On a aussi tracé les histogrammes en fonction du nombre de caractères (spam à gauche, non spam à droite) et on a obtenu les résultats suivants :



Mêmes observations qu'avec les premiers histogrammes.

On peut donc déjà voir avoir en tête que cette méthode est loin de donner des résultats satisfaisants dans notre classification à cause de la grande ressemblance de longueurs pour les deux types d'emails.

Les Fonctions apprend_modèle et predict_emails :

La fonction apprend_modele va considérer une distribution uniforme selon un intervalle donné, tout d'abord elle va trier les ensembles de mails (spam et nonSpam) selon leurs longueurs, et ensuite elle va associer à chaque intervalle de longueur, l'attribut ou le label qui lui correspond et cela en effectuant une comparaison entre les deux valeurs de P(X=x| Y=+1) et P(X=x| Y=-1) qui revient à comparer NbSpam/len(spam) et nbNonSpam/lenNonSpam dans notre cas pour voir quel type d'email est le plus dominant dans l'intervalle pris, cette fonction va renvoyer à la fin une liste d'intervalles avec le label attaché à chacun des intervalles.

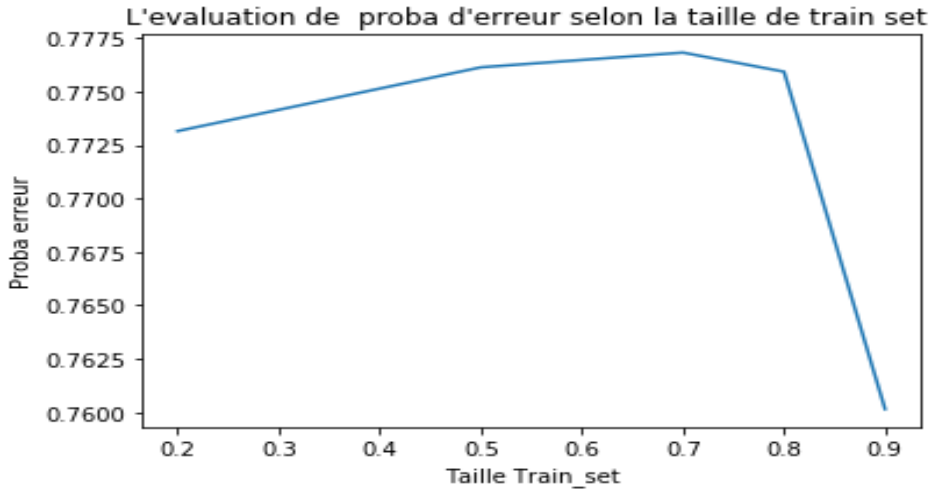
Quant à la fonction predit_email son rôle se résume à prendre un modèle correspondant à la distribution des spams selon les intervalles (retournée par apprend_modele) et un ensemble d'email, et à partir de ce modèle-là, elle doit prédire le label de chaque email de l'ensemble et cela en calculant sa longueur et regardant dans quelle classe d'intervalle se trouve-t-elle et en lui associant le label défini pour cet intervalle.

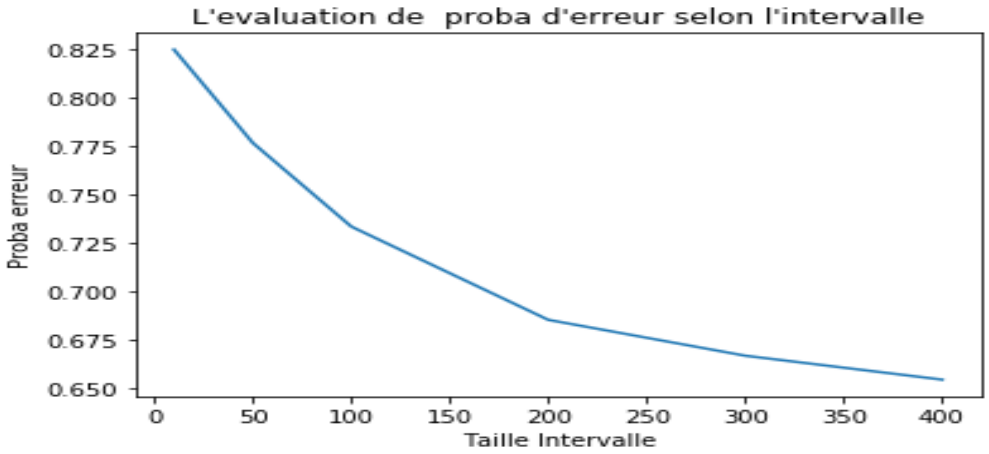
Fonctions Pour l'estimation de la probabilité d'erreur :

Pour que notre estimation d'erreur ait plus de sens, on a, à l'aide de notre fonction split, divisé notre ensemble de données en 2 sous-ensembles, un pour entraîner notre classifieur et l'autre pour le tester.

On a implémenté la fonction accuracy, qui fait appel à la fonction predit_email et qui renvoie le score de bonne prédiction sur l'ensemble de Test. Et une fonction Proba_err qui renvoie le taux d'erreurs (elle est simplement calculée en faisant 1-accuracy), et après avoir effectué plusieurs tests sur nos ensembles d'emails, On obtient des taux d'erreurs centré autour de la valeur 0 .79.

On va maintenant tracer les courbes des probabilités d'erreur en fonction des intervalles choisi et aussi en fonction de la taille de nos données d'apprentissage.





On constate à partir de ces courbes, que plusieurs facteurs peuvent influencer sur la probabilité d'erreurs, par exemple, plus on a de données d'apprentissage, moins on a d'erreur dans notre classification, et aussi avec les intervalles de longueurs on pourra dire que plus nos intervalles sont larges , moins on risque de faire des erreurs et cela est dû au fait que les spams sont répartis dans des intervalles définis donc plus on a d'intervalles(plusieurs petit intervalles) moins on risque de les reconnaître, on voit bien que le pourcentage d'erreur descend de 0 .82 jusqu'à 0.76.

Classification à partir du contenu d'un email

Après avoir constaté que la longueur d'un email était loin d'être une méthode efficace pour effectuer la classification d'email, nous allons alors, dans cette partie, programmer un classifieur, non pas basé sur la longueur mais sur le contenu d'un email (un email sera décrit par un vecteur binaire (x1, x2, ..., xd) $\in \{0,1\}^2$ selon la présence ou non d'un mot dans ce mail), et on utilisera un dictionnaire représentant l'association d'une dimension 1 ou 0 a chaque mot.

Mais dans ce cas aussi, calculer la distribution P (X = x |Y = +1) n'est toujours pas raisonnable, car la taille du dictionnaire correspondra à chaque fois au nombre de mots se trouvant dans tous les emails, et si l'on considère alors un dictionnaire de 1000 mots, il nous faudrait calculer 2^{1000} paramètres afin de caractériser cette distribution.

Dans notre cas, on a fait l'hypothèse de l'indépendance entre les mots, c'est-à-dire que les descripteurs sont deux à deux indépendants P(X = x| Y = +1) =$\Pi_{i=1}^{d}$

$P(X_i=x_i| Y= +1)$, mais cette hypothèse ne peut pas être toujours vérifiée, car en vrai il y'a une dépendance entre les mots, la présence ou l'absence d'un certain mot peut conduire à la présence ou non d'un autre, par exemple si l'on a un email administratif, la présence du mot « inscription » indique qu'il sera forcément suivi par le mot « administratif » ou « pédagogique » et c'est encore le cas pour le reste des mots.

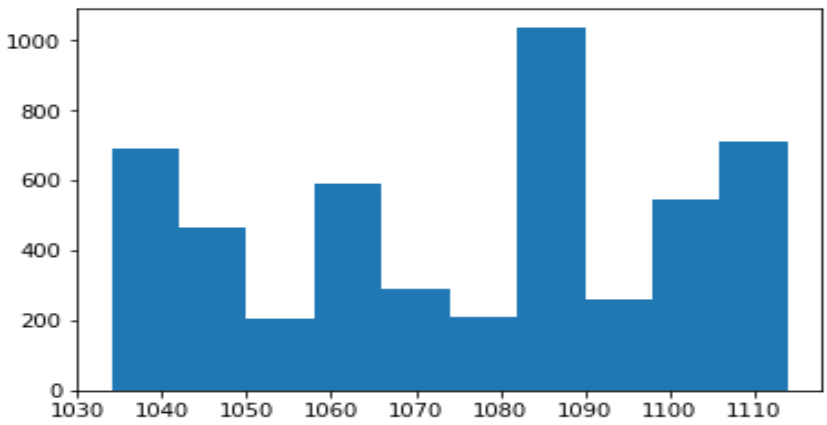
Mais si l'on garde cette hypothèse d'indépendance, il faudra calculer 2*1000 paramètres afin d'estimer la distribution P (X = x |Y = +1) ce qui est raisonnable en termes de complexité.

Fonction calculant la fréquence d'apparition des mots :

La toute première étape dans une telle classification, consiste à faire un « texte cleaning » c'est-à-dire enlever tous les mots qui ne sont pas susceptibles de contribuer à l'information qu'on essaie d'extraire de nos emails, ces derniers pouvant contenir un tas de charactères indésirables, telle que les pronoms (le, la, les…) les mots du genre (hi, hello, salut...) mais aussi des mots apparaissant au singulier mais encore au pluriel, des verbes au présent et au futur dans un même email…etc.

Dans notre cas, lorsqu'on a été apporté à calculer les fréquences d'apparition des mots on a décidé d'ignorer tous les caractères numériques ainsi que les ponctuations, les mots contenant moins de trois lettres et aussi ceux contenant plus de 27 lettres, et on a aussi transformer tous les mots en minuscules afin d'éviter d'avoir des doublons dans notre dictionnaire dû aux majuscules. Et on a décidé de prendre en considération que les 3000 mots plus fréquents pour limiter la taille de notre dictionnaire et ensuite on testera avec plusieurs autres tailles différentes pour voir si cela affecte le résultat.

On va maintenant calculer les fréquences de certains mots que l'on va tirer aléatoirement et dessiner l'histogramme correspondant pour voir les variations.



Cet histogramme représente le nombre de mots pour chaque fréquence, on peut bien constater les variations du nombre de mot dans chaque intervalle de fréquences, donc on en conclut que dans notre ensemble des mails, il y'aura des mots que l'on trouvera souvent soit dans les spam soit dans les autres, et d'autres mots qui seront les moins présents, et donc pour réduire la taille du dictionnaire, on va sélectionner les mots les plus fréquents dans les deux ensembles et on crée un dictionnaire à partir ces mot et que l'on va passer à notre classifieur pour espérer avoir une bonne classification.

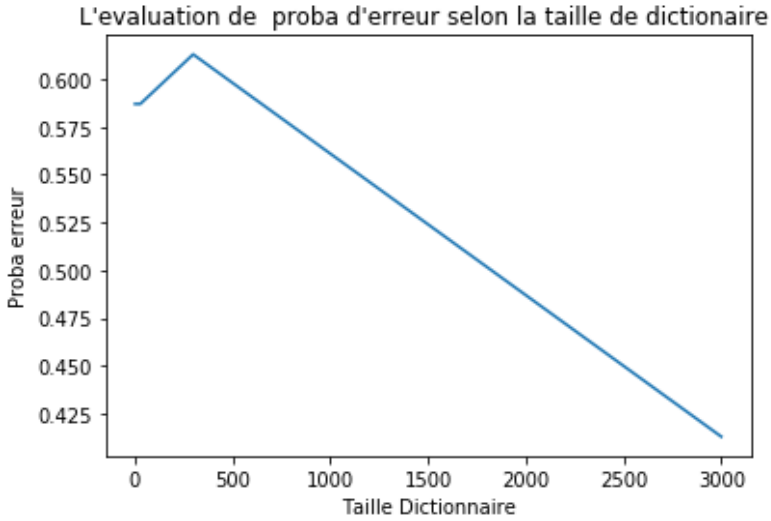
Classifieur bayésien naïf :

Le filtre anti-spam bayésien que l'on va utiliser dans cette partie repose sur le théorème de bayes afin de calculer la probabilité qu'un mot ne soit un pourriel en considérant tous ses mots, où plus précisément un sous-ensemble significatif de ses mots (après la sélection des mots).

Nous avons implémenté pour cela une fonction apprend_modele_m qui prend en paramètre un dictionnaire de mots construit par make_dic_init() selon les critères qu'on a définis et deux ensembles de mails ; un de spam et un autre de non spam après pour chaque mot i du dictionnaire elle calcule P (X=xi| Y=+1) et P (X=xi| Y=-1) et on retourne la distribution obtenue.

Après pour la fonction predict : on construit un dictionnaire pour le mail que l'on cherche à classifier , après on calcule à l'aide de distribution retourné par notre fonction apprend_modele_m $P(X = x| Y = +1)$ et $P(X = x| Y = -1)$ afin de les comparer et donner un label au mail , et grâce à notre hypothèse d'indépendance cela revient à calculer le produit suivant : $\prod_{i=1}^{d} P(X_i=x_i| Y= +1)$ et $\prod_{i=1}^{d} P(X_i=x_i| Y= -1)$; dans ce calcul on a pris une valeur lamda très petite à la place du 0 pour s'assurer que le résultat ne s'annule pas quand on a des mots qui n'existent pas, et de plus, sachant que l'implémentation de ce classifieur repose dans le cas général sur la multiplication à virgule flottante, on a décidé, pour éviter tous les problèmes potentiels liés à la multiplication de très petits nombres, d'effectuer un logarithme sur l'équation pour transformer toutes les multiplications en addition.

On va maintenant tester l'optimalité de notre classifieur en fonction des variations de la taille du dictionnaire



On peut constater clairement à partir de la courbe obtenue que la probabilité décroit de plus en plus que l'on augmente la taille du dictionnaire utilisé pour la classification car avec un dictionnaire plus grand, on prend en considération plusieurs mots pour faire la classification, et don il est très trivial que l'on fasse moins d'erreurs que dans le cas ou on considère un plus petit ensemble de mots.

On conclut donc à la fin de cette partie de classification, qu'en effet, prendre en compte le contenu de l'email s'avère être plus efficace que la classification par longueur simplement.

2 VISUALISATION

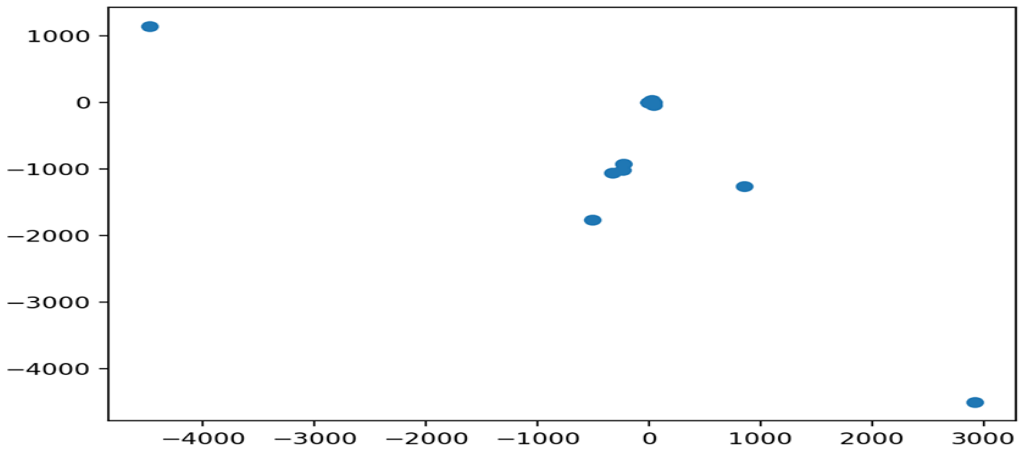
On cherche dans cette partie à faire une réduction de dimension non linéaire, dont l'objectif est d'assurer que des points proches dans l'espace de départ aient des positions proches dans l'espace (2D) projeté. Dit autrement, la mesure de distance entre points dans l'espace 2D doit refléter la mesure de distance dans l'espace initial. Et Pour cela on va implémenter une version simplifiée de SNE.

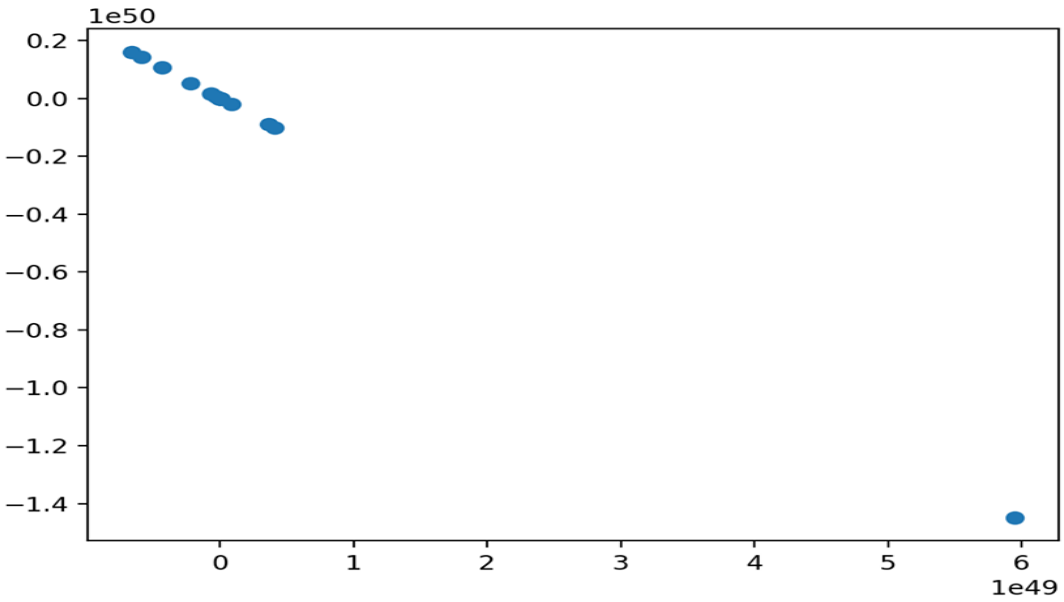
Afin d'effectuer cette implémentation (SNE), nous avons mis en œuvre un certain nombre de fonctions indépendantes qui calcule chacune l'une des formules fournies nécessaires pour l'algorithme.

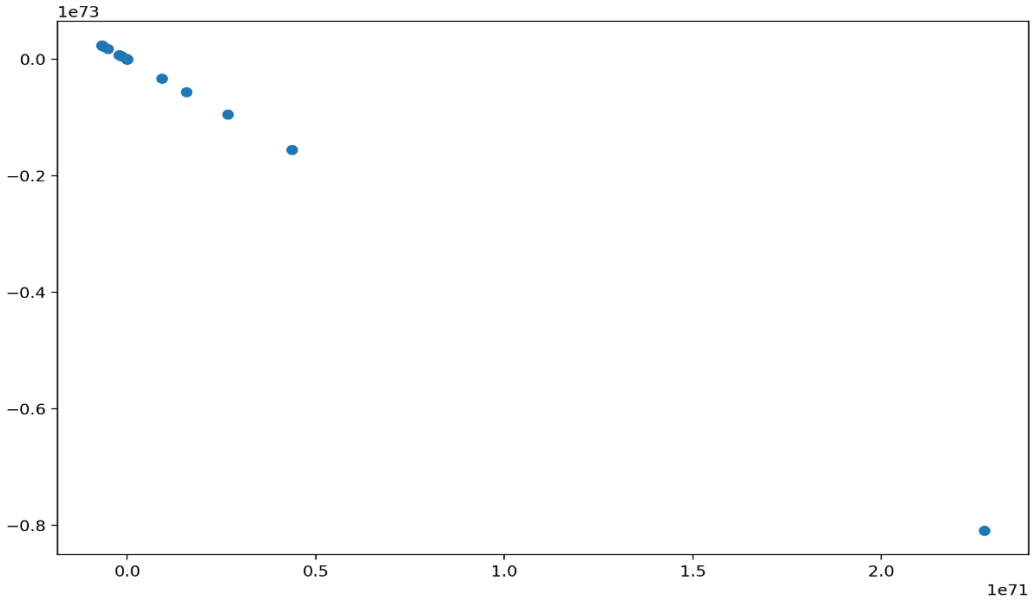
Comme condition de convergence possible, on a choisi de tester à chaque itération si la dernière valeur de C obtenu lors des itérations s'est annulée et cela pour être sûr de garder le rapprochement qu'avait les emails au début, et comme on effectue plusieurs itérations on a décider de définir une variable max_itération qui va s'assurer que l'on ne la dépasse jamais pour éviter de reboucler infiniment.

Pour tester notre algorithme, nous avons choisi de visualiser un petit ensemble d'emails avec une distribution uniforme pour les deux ensembles spam et non spam pour éviter que la convergence ne soit très lente et on a pris une distribution équitable pour les deux types.

Test : On prend un vecteur de dimension 50 et on représente 2% des emails de l'ensemble des spams et 2% dans l'ensemble des non spams et on applique l'algorithme SNE que l'on a effectué on répète les opérations et on obtient les représentations suivantes :

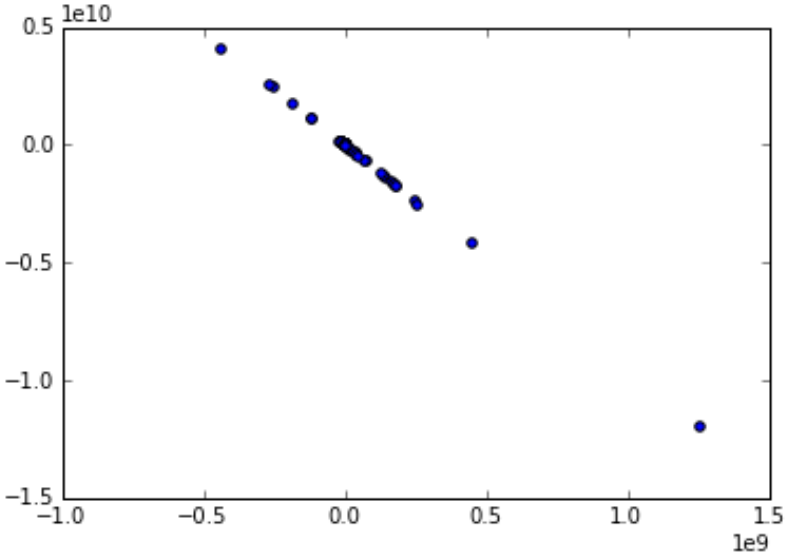


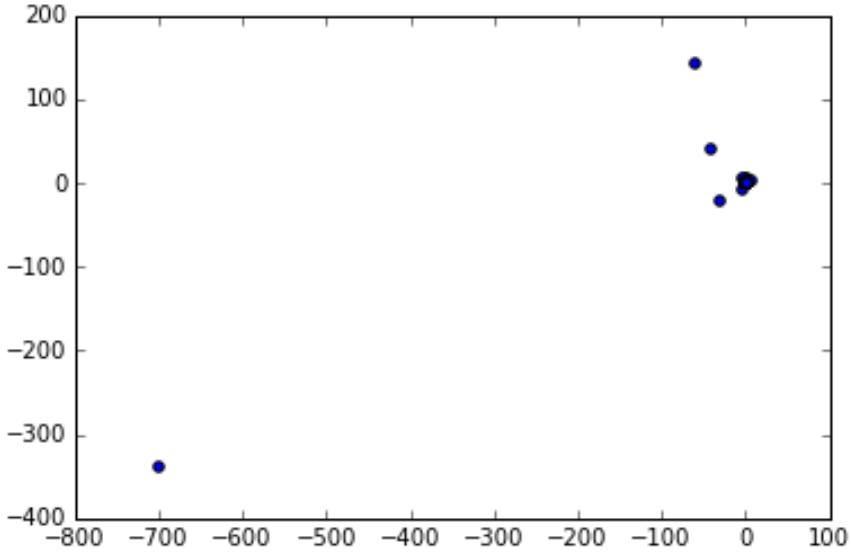


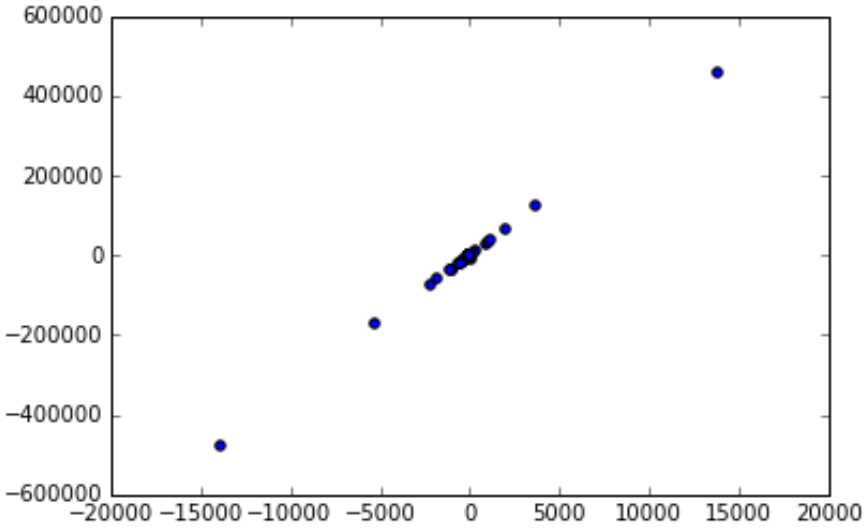


A partir des représentations graphique 2D obtenues, on peut bien voir un ensemble de points qui se rapprochent jusqu'à se superposer, en effet ces points là représentent un ensemble qui ont le plus de mots en commun et donc l'algorithme SNE a pu garder le rapprochement de ces points malgré le fait qu'on ait diminué le nombre de dimension à 2, comme on peut apercevoir des points qui s'éloignent sur le plans, et on peut déduire que ces points la sont forcément les points qui n'ont pas de mots en communs avec les autres, ou bien juste quelques-uns.

Quelques autres tests de visualisation :







Conclusion

En conclusion de ce projet, nous avons pu nous initier à quelques mécanismes d'apprentissage statique basées sur les probabilités. Les résultats obtenus au court des différentes étapes, ont pu nous confirmer que le choix des paramètres, et les critères de classification (longueur ou contenu) jouent un grand rôle dans la réalisation d'un bon classifieur sans oublier toutes les opérations de « nettoyage » autrement dit pré-traitement de données et filtrage de mots.

Il est aussi nécessaire d'optimiser les couts des opérations afin de pouvoir réaliser la classification en de brefs délais.

Autres critères possibles pour la réalisation de la classification de spams : Il est possible de se baser sur plein d'autres caractéristiques afin d'entrainer les classifieurs, voici quelques exemples qui pourraient être déterminant dans ce type de classification (spam filter) : nombre et type de pièces jointes, présence d'HTML, de scripts et de liens hypertextes, présence d'images, expéditeur : est-il connu par l'utilisateur ?, le domaine du service d'envoi est-il sur liste noire ? est-il autorisé à effectuer ce type d'envoi ?

Enfin, il ne faut pas oublier qu'il est possible de combiner les différentes techniques afin d'obtenir les performances escomptées et il est aussi envisageable de faire varier la distribution initiale, car en effet dans ce projet, nous avons pris en compte une distribution uniforme entre les deux classes mais il est évident que dans la vrai vie, la majorité des e-mails que l'on reçoit (voir 90% ou plus) ne sont pas des pourriels donc il serait envisageable de tester une classification avec de telles distribution, pour espérer d'avoir de plus bons résultats.