***Ministère de l’enseignement supérieur et de la recherche scientifique***



***Université des sciences et de la technologie Houari Boumediene***

***Faculté d’informatique***

***MASTER 01 : Système informatique intelligents***

***MODULE : AARN***

***PROJET De FIN DU SEMESTRE***

* Zait Fouad 181831072145

***Introduction :***

De nombreux services de messagerie fournissent aujourd'hui des filtres anti-spam capables de classer les e-mails dans les spams et les non-spams avec une grande précision. Pour ce projet, l’objectif est d’utiliser les notions apprises durant ce semestre pour résoudre un problème de sciences de données. Nous devrions implémenter un détecteur de Spam en apprenant sur un ensemble de données.

**Introduction au problème des spams :**

Le problème des spams connaît depuis ces 20 dernières années un essor considérable. En effet, le pollupostage pourrait représenter plus de 72% de l'ensemble du trafic de courrier électronique. Au-delà de l'aspect intrusif des spams, ceux-ci peuvent comporter des virus ou des scripts néfastes, d'où l'intérêt de les détecter afin de les supprimer. Le coût d'un envoi de courriels par un spammeur étant infime, ce dernier peut se permettre de transmettre le spam au plus d'adresse de messagerie électronique. Pour le spammeur qui arrive à récupérer même une petite partie d'utilisateurs, son opération devient commercialement viable. Imaginant un million de courriels envoyés et seul 0,1% de personnes qui se font appâtées, cela représente tout de même 1 millier de personnes, et ce chiffre est très réaliste. Nous voyons que derrière la protection de la vie privée et le maintien d'un environnement de travail sain se cachent également des enjeux économiques.

La détection des spams est une course constante entre la mise en place de nouvelles techniques de classification du courriel et le contournement de celles-ci par les spammeurs. Jusqu'alors, ces derniers avaient une avance dans cette lutte. Nous présentons dans ce rapport toutes les modèles de classification étudiés adaptés à ce problème.

***1-Importation des libraires nécessaires au travail :***

import re : pour utiliser les expressions regulieres   
import os : pour interagir avec le systeme d’exploitation (lecture des fichiers contenant les donnees ).  
from string import punctuation : pour utiliser la punctuation lors de la preparation des données.  
from nltk.stem.snowball import SnowballStemmer :utiliser pour la radicalization des mots lors de la preparation des données.  
import pandas as pd : pour la manipulation et l'[analyse des données](https://fr.wikipedia.org/wiki/Analyse_des_donn%C3%A9es) (on l’a utiliser pour créer notre DataSet )

import numpy as np : pour manipuler les matrices et tableaux multidimensionnels ainsi que les fonctions mathématiques opérant sur ces tableaux.  
from sklearn.linear\_model import LogisticRegression : Classifieur  de regression logistique  
from sklearn.svm import SVC : Classifieur  des vecteurs à machines de support  
from sklearn.naive\_bayes import MultinomialNB :Classifieur bayésien naïf (Naive Bayes)  
from sklearn.tree import DecisionTreeClassifier : Classifieur  par arbre de decision  
from sklearn.neighbors import KNeighborsClassifier : Classifieur K plus proches voisins  
from sklearn.ensemble import RandomForestClassifier: Classifieur de forets aléatoirs   
from sklearn.metrics import accuracy\_score : pour calculer la métrique de performance precision  
from sklearn.metrics import recall\_score : pour calculer la métrique de performance rappel  
import sklearn.metrics as metrics: pour utiliser les métriques de performances  
from sklearn.model\_selection import train\_test\_split : pour diviser le DataSet en sous-ensembles d'entraînement et de test aléatoires.  
from sklearn.metrics import confusion\_matrix : pour calculer la matrice de confusion pour évaluer la precision d'une classification.  
import matplotlib.pyplot as plt : pour la manipulation des graphes (2d,3d) dans notre cas pour afficher la matrice de confusion et la courbe roc  
from sklearn.metrics import roc\_curve,auc, roc\_auc\_score: pour calculer le score auc et pour utiliser la metrique de performance courbe roc.

***2-Lecture des fichiers de données à classifier:***

* Dans cette phase on va lire les fichiers de données à classifier à partir des fichiers trouvé sur le site [*https://spamassassin.apache.org/old/publiccorpus/*](https://spamassassin.apache.org/old/publiccorpus/)

Certains des spams et des non spams sont coder en latin on a générer une exception si ce cas se présenter on va stocker tous les spams dans une liste appelé listofspam et les non spams dans une liste nommé listofham pour pouvoir les manipuler dans les étapes suivantes.

**Code**:

listofspam=[]   
listofham=[]#list ou on va mettre tous les non spam  
spam=os.listdir("C:/Users/Vaio/Downloads/DATA/spam")#chemin ou on a mis les spams dans le fichier DATA

for i in spam :  
        try :  
            f=open(os.path.join("C:/Users/Vaio/Downloads/DATA/spam/",i),'r', encoding='utf-8')  
            content=f.read()  
        except UnicodeDecodeError:  
            open(os.path.join("C:/Users/Vaio/Downloads/DATA/spam/", i), "r", encoding="latin-1")  
            content = f.read()  
            open(os.path.join("C:/Users/Vaio/Downloads/DATA/spam/", i), "w", encoding="utf-8")  
            f.write(content)  
        listofspam.append(content)  
ham=os.listdir("C:/Users/Vaio/Downloads/DATA/ham")")#chemin ou on a mis les spams dans le fichier DATA  
for i in ham:  
    try :  
        f=open(os.path.join("C:/Users/Vaio/Downloads/DATA/ham/",i),'r', encoding='utf-8')  
        content=f.read()  
    except UnicodeDecodeError:  
        open(os.path.join("C:/Users/Vaio/Downloads/DATA/ham/", i), "r", encoding="latin-1")  
        content = f.read()  
        open(os.path.join("C:/Users/Vaio/Downloads/DATA/ham/", i), "w", encoding="utf-8")  
        f.write(content)  
    listofham.append(content)

***1-Préparation des données:***

Ensemble d’opérations de nettoyage et transformation qui doivent être appliqués aux données brutes avant leur traitement et analyse. Il s'agit d'une étape importante avant le traitement proprement dit, qui implique souvent de reformater et corriger les données et de combiner des datasets pour enrichir certaines données.

La préparation des données est généralement une opération de longue haleine pour les spécialistes des données ou les utilisateurs de l'entreprise, mais il est essentiel de mettre les données en contexte pour pouvoir les convertir en connaissances exploitables et éliminer les biais résultant d'une mauvaise qualité des données.

* **Nettoyage des emails:** en ce qui concerne les emails nous allons proceder comme suit :
* Minuscule : l'intégralité de l'e-mail devra être convertie en minuscules.
* Suppression de balises HTML : Toutes les balises HTML devront être supprimées des e-mails. De nombreux e-mails sont souvent accompagnés d'un formatage HTML ; toutes les Balises HTML devront être supprimées, de sorte que seul à garder uniquement le contenu de l’email.
* Normalisation des URL : Toutes les URL devront être remplacées par le texte « httpaddr ».
* Normalisation des adresses e-mail : toutes les adresses e-mail devront être remplacées avec le texte "emailaddr".
* Normalisation des nombres : Tous les nombres devront être remplacés par le texte "nombre".
* Normalisation des dollars : Tous les signes dollar ($)devront être remplacés par le texte "dollar".
* Radicalisation de mots : Les mots devront être réduits à leur forme radicale. Par exemple, "discount", "discounts", "discounted" et "discounting" devront être tous remplacé par " discount", et "include", "includes", "included", et "included" devront être tous remplacés par « includ ».
* Suppression des non-mots : les non-mots et la ponctuation devront être supprimés. Tous les espaces blancs (onglets, nouvelles lignes, espaces) devront être remplacés par un seul espace.

**Fonction:** Cette fonction prendra en entrée une liste ,elle aura pour rôle de parcourir tous éléments de cette liste et appliquer chaque étape de la préparation des données .Nous allons appeler cette fonction 2 fois pour l’appliquer a nos spams et non spams.

def datapreparation(l):

    for i in range(0,len(l)):

        l[i]=l[i].lower()#minuscule

        l[i] = re.sub('<[^<>]+>', ' ', l[i])#suppression des balises html

        l[i]= re.sub('(http|https)://[^\s]\*', 'httpaddr', l[i])#normalisation des url

        l[i]= re.sub('[^\s]+@[^\s]+', 'emailaddr', l[i]) #normalisation des adresses mail

        l[i]= re.sub(r'\d+', 'number', l[i])#normalisation des nombres

        l[i]= re.sub('[$]+', 'dollar', l[i])#normalisation des dollars

        l[i] = l[i].translate(str.maketrans('', '', punctuation)) #suppression de la ponctuation

        l[i]= re.sub("\n"," ", l[i])#suppression des sauts de lignes

        #radicalisation et suppression des espaces et des non mots

        l[i]=l[i].split()

        l2=""

        stemmer = SnowballStemmer("english")

        for j in range(0,len(l[i])):

            if l[i][j].isalpha():

                l[i][j] = stemmer.stem(l[i][j])

            l2=l2+l[i][j]+" "

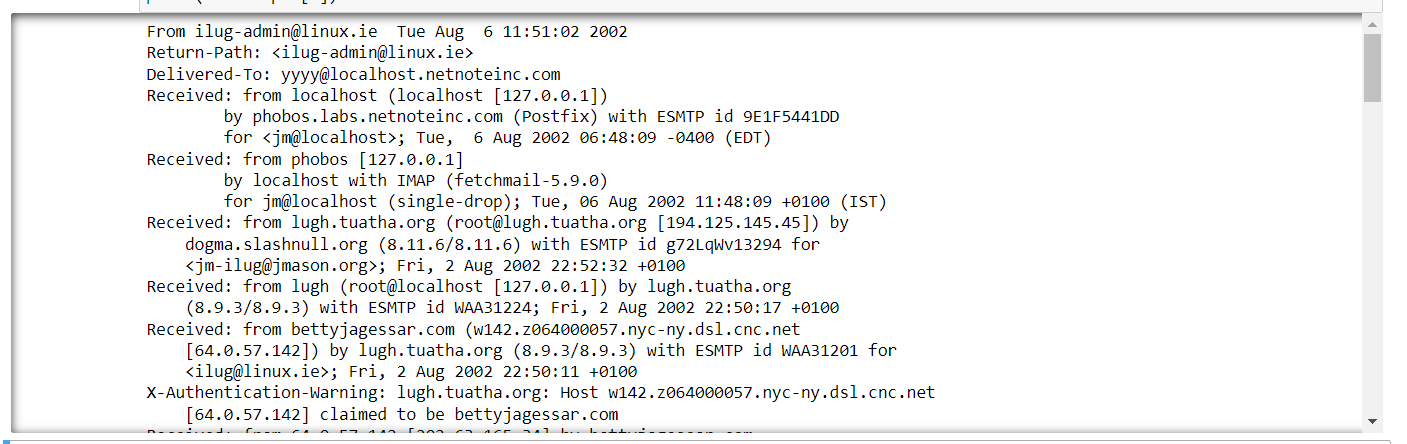
        l[i]=l2

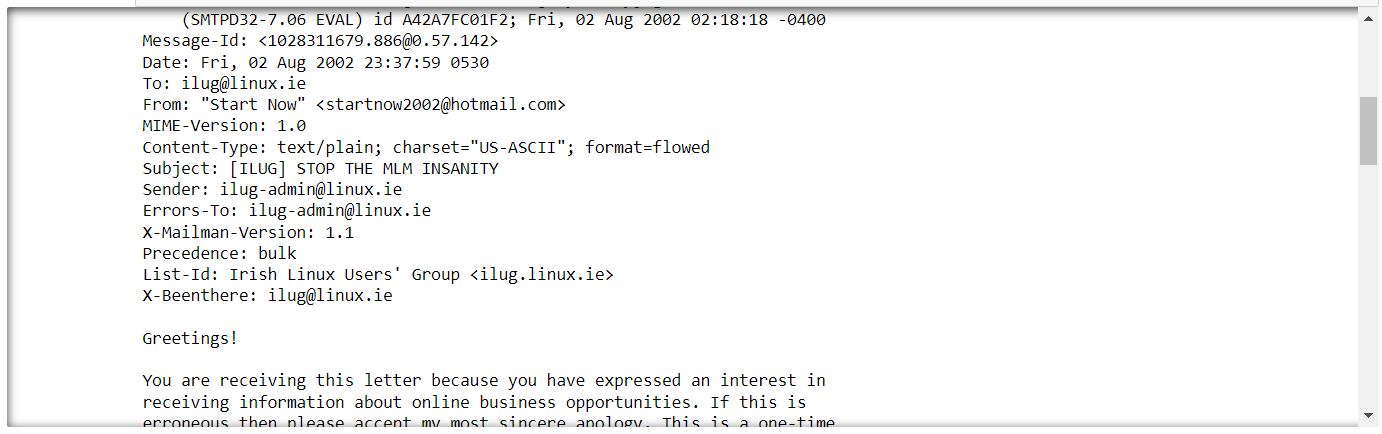
datapreparation(listofspam)#preparation des spam

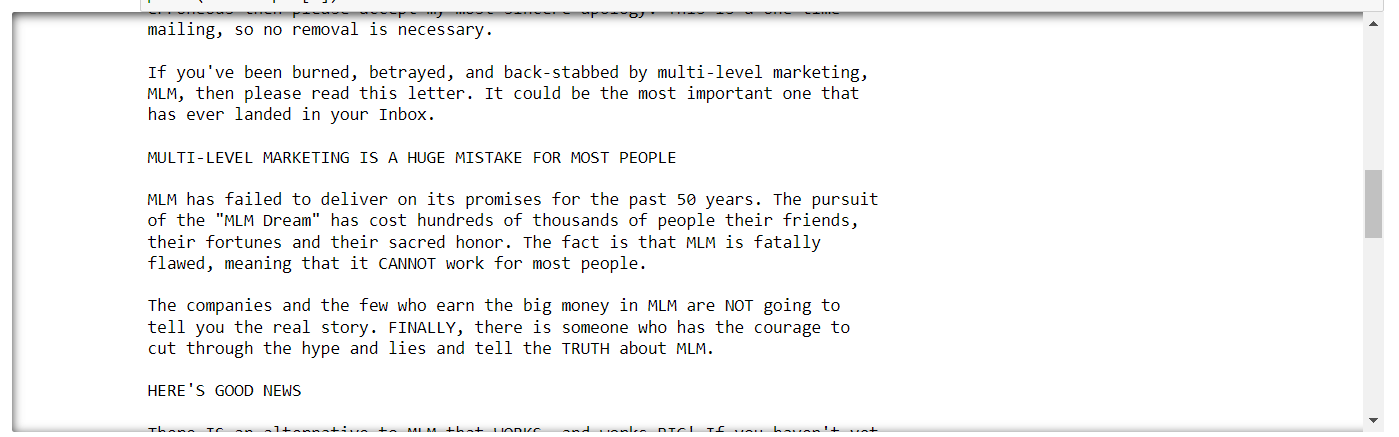
datapreparation(listofham)#preparation des non spam

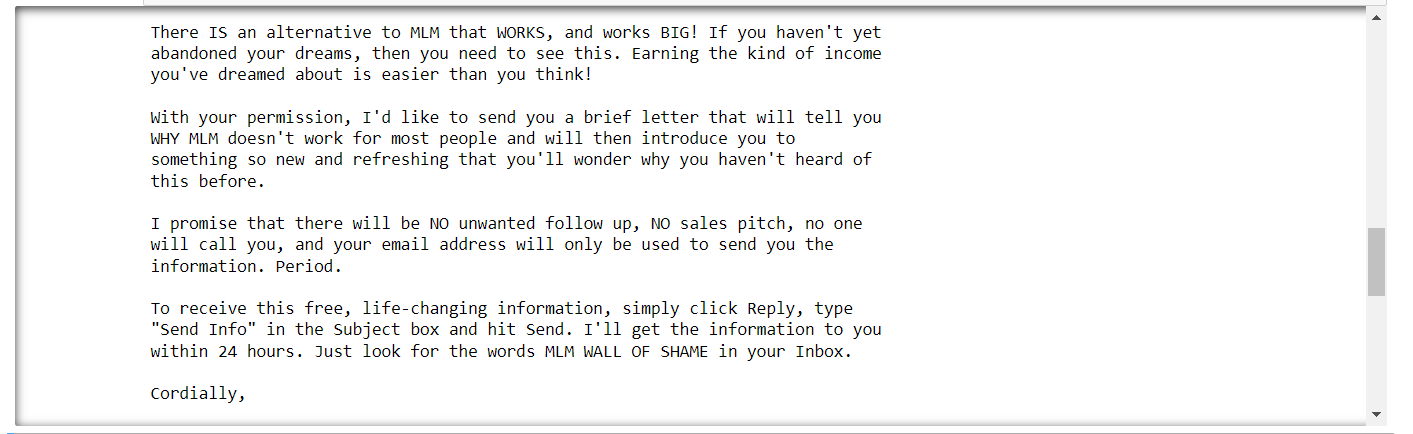
**Execution montrant un element spam avant et après préparation :**

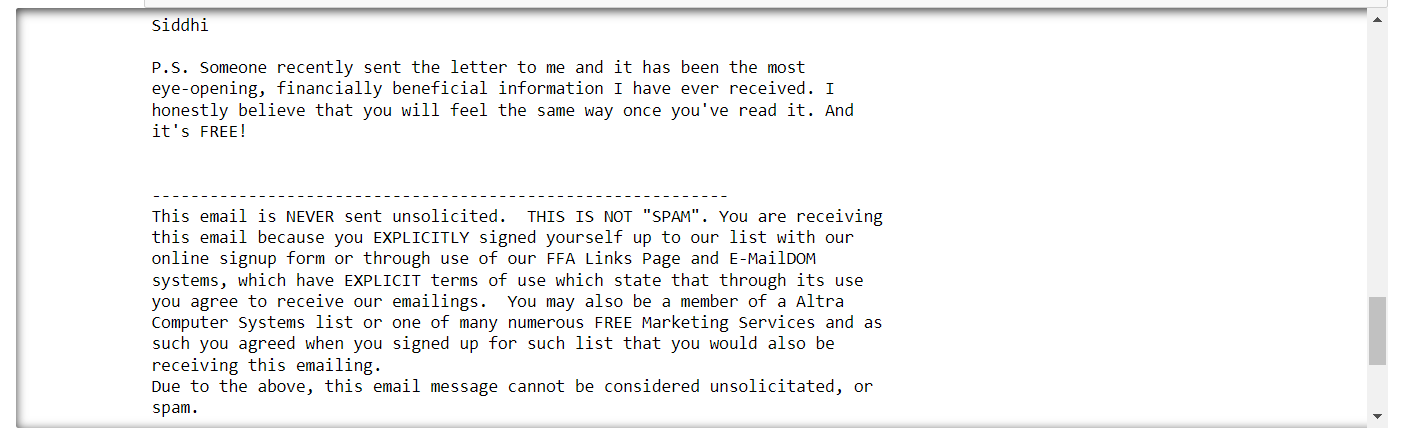
**Avant:**

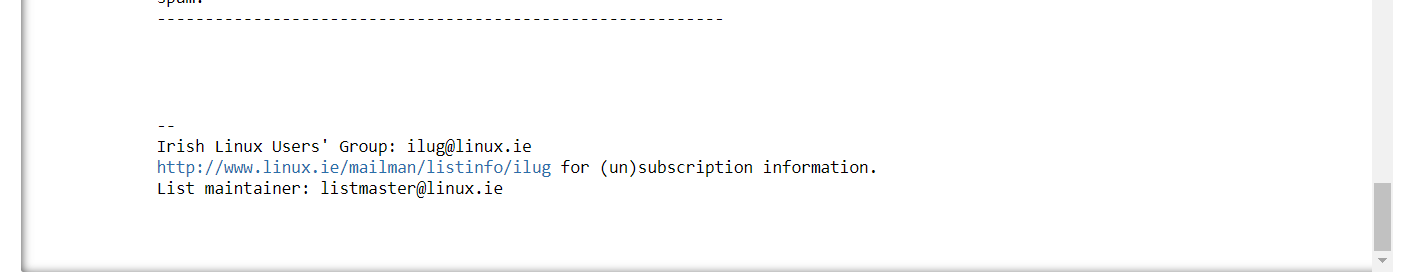




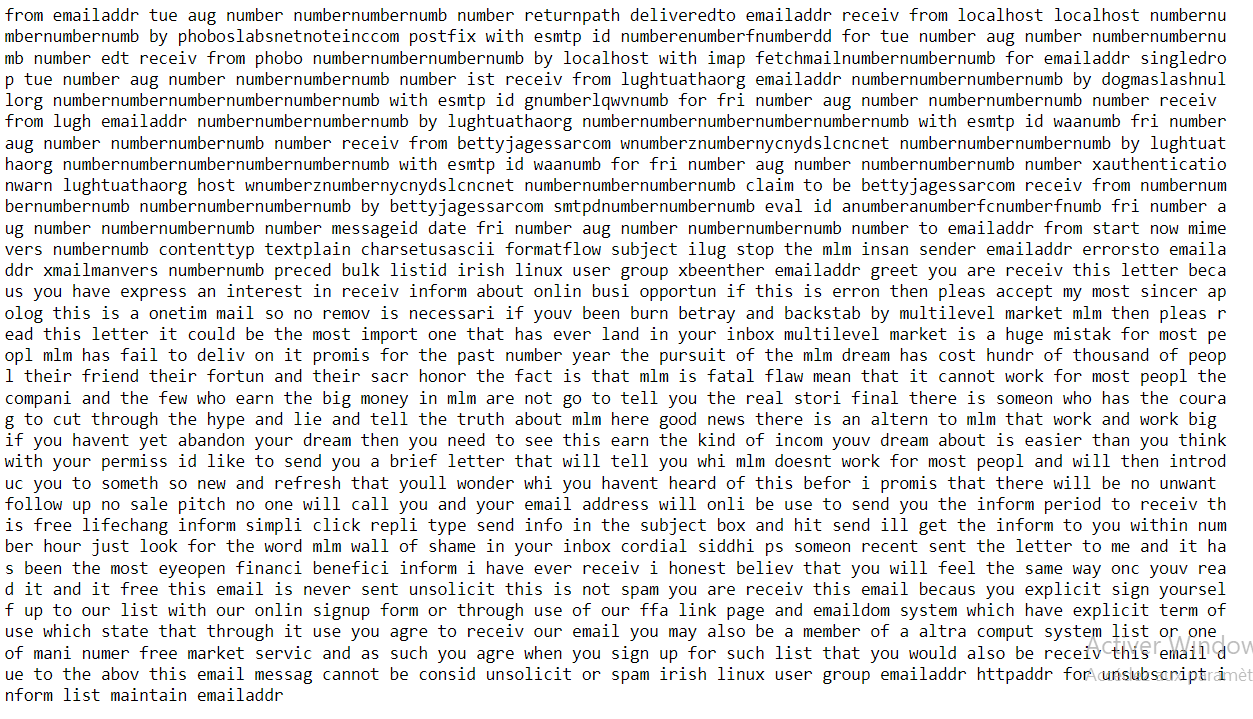








**Après:**



***2-Construction du vocabulaire:***

Après le prétraitement des e-mails, une liste de mots sera pour chaque e-mail. L'étape suivante consiste à choisir les mots que nous aimerions utiliser dans notre classificateur et que nous voudrions laisser de côté. Pour ce projet, il est possible de choisir uniquement les mots les plus fréquents (la liste de vocabulaire). Puisque les mots qui se produisent rarement dans l'ensemble de formation ne sont que dans quelques e-mails, ils peuvent provoquer un sur-apprentissage.

La liste complète du vocabulaire devra être sauvegardée dans un fichier, exemple vocab.txt. Dans cette liste de vocabulaire seulement les mots qui apparaissent au moins K fois dans le corpus de spam devront être gardés. K devra être choisi empiriquement. En pratique, une liste de vocabulaire avec environ 10 000 à 50 000 mots sont souvent utilisés.

Une fois ayant obtenu la liste de vocabulaire, il sera possible de mapper chaque mot dans l’email prétraité à son index dans une liste d'index de mots (qui contient l'index du mot dans la liste de vocabulaire).

Ceci est fait en cherchant le mot dans le vocabulaire liste vocabList et trouver si le mot existe. Si oui, il devra être ajouté dans la variable index des mots. Si le mot n'existe pas, et n'est donc pas dans le vocabulaire, le mot devra être ignoré.

**Fonction de creation du vocabulaire :**Dans cette fonction on a compter le nombre d’occurence de chaque mot des spams puis on a garder que les 10000 qui se repetent le plus qu’on a ensuite sauvegardé dans une liste vocab et dans un fichier vocab.txt puis on a construit le dictionnaire ou on a mapper chaque mot dans l’email a son indexe.

**Code :**

from collections import Counter

def createvocab(l):

fichier = open("vocab.txt", "w" ,encoding="utf-8")

cpt = Counter()

for i in range(len(l)):

for word in l[i].split(" "):

cpt[word] += 1 #compter le nombre d occurence de chaque mot

vocab = sorted(cpt, key=cpt.get, reverse=True)

vocab=vocab[:10000]#vocab sera une list des 10000 mots qui se repetent le plus

for i in vocab :#on remplit vocab.txt par les 10000 mots qui se repetent le plus

fichier.writelines(i)

fichier.writelines("\n")

fichier.close()

vocabdict = {}

lenvocab=len(vocab)

# mapper chaque mot dans l’email prétraité à son index

for i,word in enumerate(vocab):

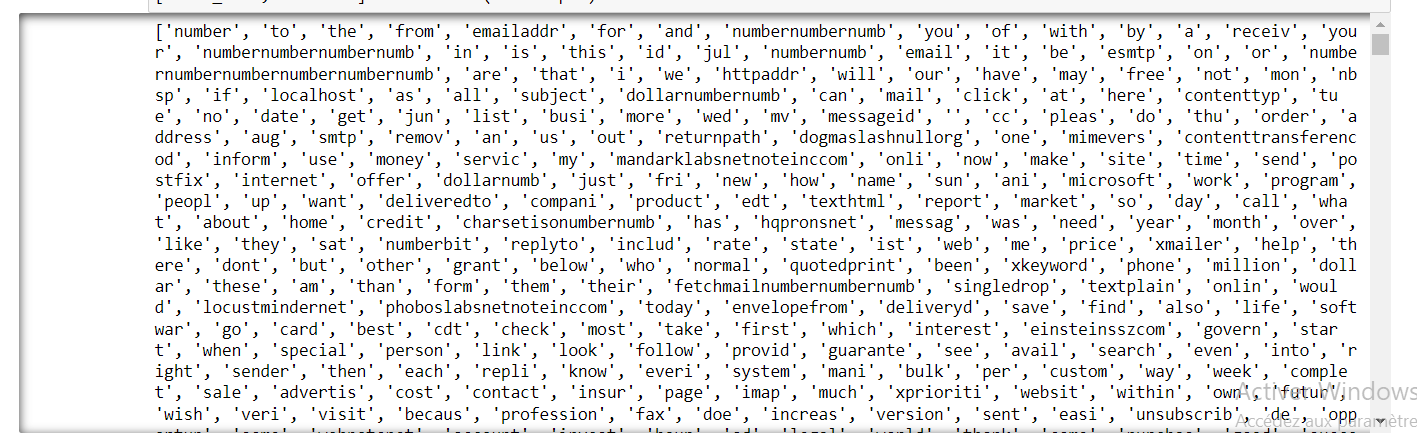
vocabdict[word]=i

return [lenvocab,vocabdict]

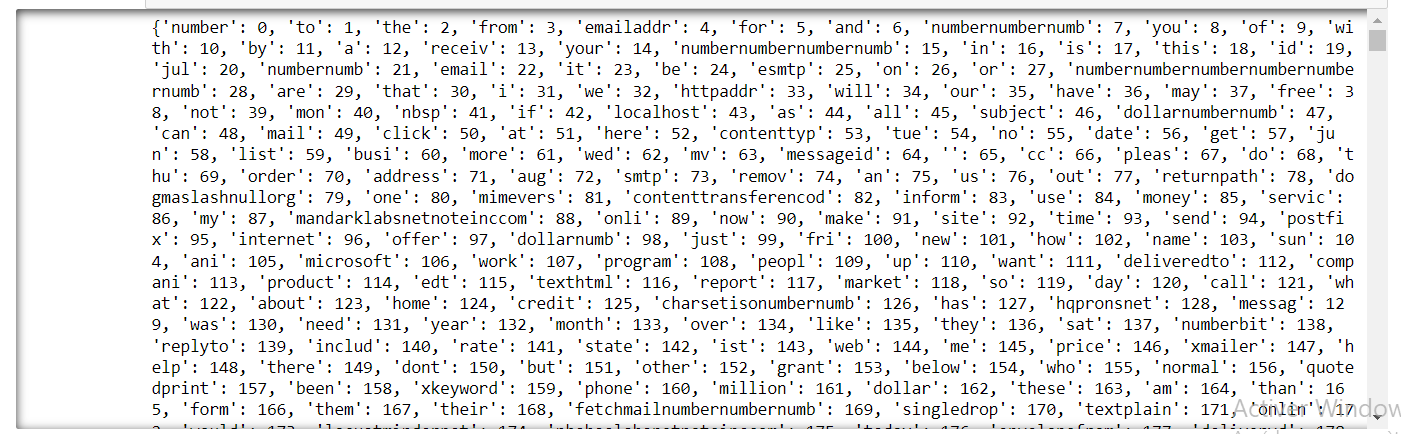
[vocab\_size,vocabdict]=createvocab(listofspam)

**Execution :**

la liste vocab contenant les mots qui se repetent le plus:



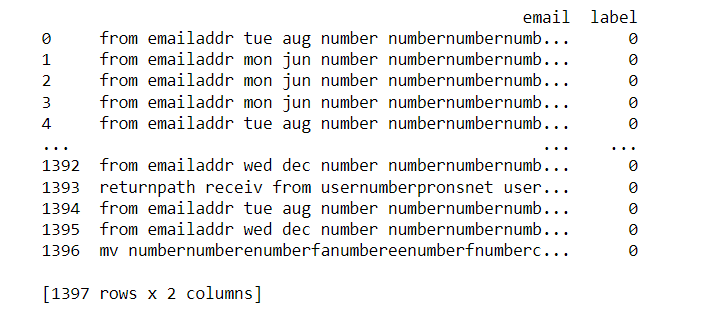
Le dictionnaire vocabdict qui contient chaque mot a son indexe :



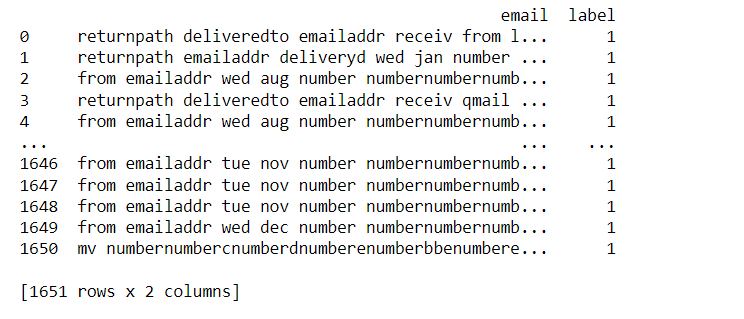
***-Création du DataFrame contenant les emails spams et non-spams avec label=1 si l’email est un spams sinon 0:***

**Code :**

df=pd.DataFrame(listofspam, columns = ['email'])  
df["label"]=1

**Execution** :Dataframe contenant les spams avec comme label 0.

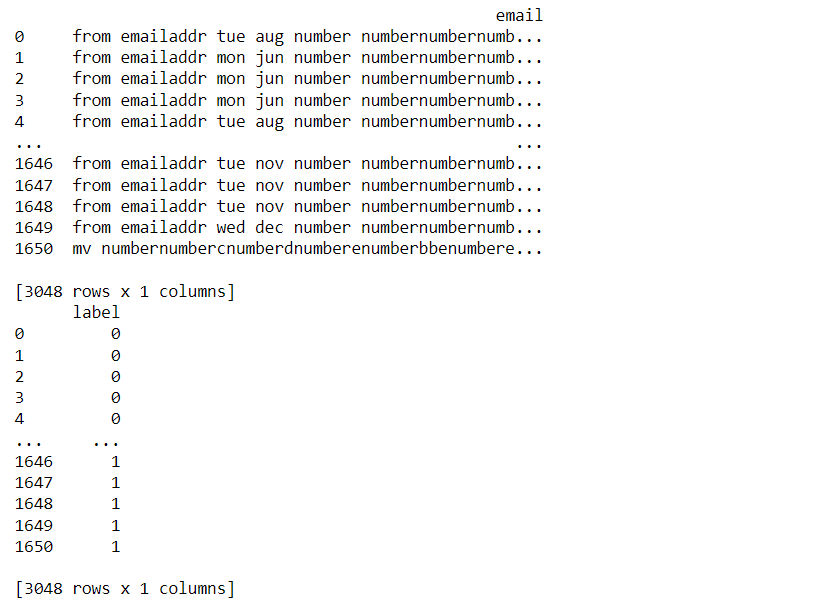
### Code : df2=pd.DataFrame(listofham, columns = ['email']) df2["label"]=1 Execution : Dataframe contenant les non spams avec comme label 1



### Concaténer les deux dataset crée pour pouvoir entrainer nos modèles : Code :

### data=pd.concat([df, df2]) email = pd.DataFrame(data['email']) label = pd.DataFrame(data['label'])

**Execution :**



***3-Extraction de caractéristiques :***

L'extraction de fonctionnalités devra convertir chaque e-mail en un vecteur dans Rn . Pour ce projet, nous utiliserons n = # mots de vocabulaire liste.

Nous avons choisi la représentation des caractéristiques par comptage :

#generation des caracteristiques pour chaque mail ca sera un vecteur de taille  du vocabulaire

def emailtovector(email):#cette fonction nous peremettera de generer le vecteur de caracteristiaue d'un email donnée

    vecteur\_mots = np.zeros(vocab\_size)# initialisation du vecteur de caracteristique

    for mot in email.split(" "):#pour chaque mot du mail si le mot existe dans le vocabulaire incrementer son nombre d'apparition

        if vocabdict.get(mot) is None:#si le mot n'existe pas dans le vocabulaire on va l'ignorer

            continue

        else:

            vecteur\_mots[vocabdict.get(mot)] += 1

    return np.array(vecteur\_mots)

#pour tous les email :

vecteurs\_mots = np.zeros((len(email), vocab\_size), dtype=np.int\_)

for i, (\_, text\_) in enumerate(email.iterrows()):

    vecteurs\_mots[i] = emailtovector(text\_[0])

vecteurs\_mots.shape

**Execution :**on a 3048 exemple et 10000 caractéristiques



***D*escription *des classificateurs utilis*ée*:***

Nous allons décrire tous les classifieurs qu’on a utilisé dans notre projet :

* **regression logistique :**

**Le fonctionnement de l’algorithme :** La régression logistique est une méthode statistique connue qui permet de déterminer la contribution de plusieurs facteurs à une paire de résultats. Le nom de régression logistique vient du fait que la courbe de données est compressée par une transformation logistique afin de réduire l'effet des valeurs extrêmes. En python c’est assez simple, on se sert de la classe LogisticRegression du module sklearn.linear\_model  comme un classificateur normal et que l’on entraîne sur des données déjà nettoyées et séparées en ensembles d’entraînement et de test.

-**Choix des paramètres** :Pour ce problème on a donner en entrée en paramètres comme solver =’liblinear’ vu qu’on a 3048 exemples liblinear est un bon choix pour ce nombre d’exemple si on avait un plus grand nombre de données on aurai plus penser a sag ou saga qui sont plus rapides pour un plus grand nombre , pour la régularisation cela se traduit par un rétrécissement vers zéro des coefficients des variables les moins contributives. on impose une pénalité au modèle logistique penalty=’l1’.

LogisticRegression(solver='liblinear', penalty='l1')

* Machine à vecteurs de support(SVM) :

Les SVM sont une famille d'algorithme de Machine Learning très populaire pour la résolution d'un problème de classification , Les SVM considère un ensemble de données d'entraînement qui contient par exemple des points spams et des points non spams. Ceux-ci occupent chacun une région différente d'un plan. L'objectif d'un algorithme SVM sera alors la résolution d'un problème particulier : prédire spam ou non spam d'un nouveau point dont on connaît la position dans le plan. Pour ce faire, le SVM doit trouver la frontière ou hyperplan entre ces deux catégories.

Pour que le SVM parvienne à trouver cet hyperplan, on le soumet à **des données d'entraînement**. On lui donne un ensemble de points de formes connues. À partir de ces données, l'algorithme SVM se chargera alors de trouver l'hyperplan le plus plausible. Le fonctionnement d'une machine à vecteurs de support repose sur une simple idée : trouver un hyperplan qui permet de diviser de façon optimale un jeu de données en deux classes différentes.

Dans un SVM, les données sont séparées en différentes classes en ayant recours à **la notion de marge maximale**. La marge représente la distance entre la ligne de séparation des données et le point le plus proche de l'un des ensembles. La frontière de séparation des classes est choisie de façon à maximiser la distance entre les groupes de données.

-**Choix des paramètres** :pour le paramètre de régularisation on a choisi de garder celui par default=1.0,le choix de cette valeur est très important pour éviter un sur apprentissage ou un sous apprentissage .On a choisi le noyau linéaire simple.

svm = SVC(kernel='linear', probability=True)

* **Naive Bayes :** Naive Bayes, couramment utilisé dans l’apprentissage automatique, est une collection d’algorithmes de classification basés sur le théorème de Bayes. Ce n’est pas un algorithme unique, mais une famille d’algorithmes. Tous ces algorithmes partagent tous un principe commun, à savoir que chaque caractéristique classée est indépendante de la valeur de toute autre caractéristique.

Un classificateur Naive Bayes considère chacune de ces caractéristiques de nos exemples comme des caractéristiques  indépendantes quelles que soient les corrélations existant entre les caractéristiques. Cependant, les caractéristiques ne sont pas toujours indépendantes, ce qui est souvent perçu comme un inconvénient de l’algorithme Naive Bayes et c’est pourquoi il est appelé «naïf».

Meme si c’est un concept relativement simple, Naive Bayes peut souvent surperformer les algorithmes les plus complexes et est extrêmement utile dans les applications courantes telles que la détection de spam .En gros, l’algorithme nous permet de prédire une classe, étant donné un ensemble de caractéristiques utilisant des probabilités.

-**choix des paramètres** :Les paramètres par default nous ont donnée une bonne précision pour notre problème .

nb = MultinomialNB()

* **Les arbres de decision :** Lorsque nous avons un problème à résoudre qui est soit un problème de classification soit un problème de régression, l'algorithme d'arbre de décision est l'un des algorithmes les plus populaires utilisés pour construire les modèles de classification et de régression. Ils entrent dans la catégorie de l'apprentissage supervisé, c'est-à-dire les données étiquetées.

De nombreuses étapes sont impliquées dans le fonctionnement d'un arbre de décision.

* **Fractionnement** - C'est le processus de partitionnement des données en sous-ensembles.
* **Élagage** - C'est le processus de raccourcissement des branches de l'arbre de décision, limitant ainsi la profondeur de l'arbre
* **Sélection d'arbre** - La troisième étape est le processus de recherche du plus petit arbre qui correspond aux données.

- **choix des paramètres** :On a choisi minimum 7 échantillons pour diviser un nœud interne ,le random state =111 pour toujours permuter de manière aléatoire à chaque fractionnement .

dt = DecisionTreeClassifier(min\_samples\_split=7, random\_state=111)

* **Méthode des k plus proches voisins :** L’algorithme des *k* plus proches voisins est un algorithme d’apprentissage automatique qui est qualifié de supervisé. Il s’agit de montrer à une machine un grand nombre d’exemples similaires afin de lui apprendre à résoudre certains problèmes.

L’algorithme des *k* plus proches voisins permet de classifier des données de manière **artificielle** : c’est le programme qui détermine à quelle groupe (famille) appartient une nouvelle donnée entrée, en s’appuyant sur des données déjà entrées qui ont déjà été classées par groupes (familles) en calculant les distances avec les tous les autres données ,dans le cas d’un très grand nombre d’exemple cet algorithme n’est pas efficace car il prendra beaucoup de temps pour calculer toutes les distances .

* On définit en entrée de cet algorithme un ensemble de données déjà classifiées ;**une distance *d*** et **un nombre entier *k***.
* L’algorithme des *k* plus proches voisins calcule la distance entre toutes les données déjà classifiées et la nouvelle donnée qui vient d’être entrée.
* L’algorithme extrait ensuite les *k* données déjà classifiées les plus « proches » de la nouvelle donnée entrée, c’est-à-dire les données déjà classifiées qui ont la distance *d* la plus petite avec la nouvelle donnée entrée.
* L’algorithme choisit enfin à quelle famille appartient la nouvelle donnée, en cherchant la famille majoritaire parmi les *k* données identifiées.

- **choix des paramètres** :

On a implémenter une fonction qui nous permettra d’avoir le meilleur paramètre (nombre de voisins) pour ce modèle :

n=neighbours()

kpp = KNeighborsClassifier(n)

on détaillera la fonction neighbours() par la suite.

## Random Forest : est une technique de Machine Learning très populaire auprès des Data Scientists.C’est une technique facile à interpréter, stable, qui présente en général de bonnes accuracies et qui peut être utilisée pour des tâches de régression ou de classification. Elle couvre donc une grande partie des problèmes de ****Machine Learning****.Dans ****Random Forest**** il y a d’abord le mot “Forest” . On comprend donc que cet algorithme va reposer sur des arbres de décision .En plus du principe de bagging, les forêts aléatoires ajoutent de l’aléa au niveau des variables. Pour chaque arbre on sélectionne un échantillon bootstrat d’individus et à chaque étape, la construction d’un noeud de l’arbre se fait sur un sous-ensemble de variables tirées aléatoirement.

On se retrouve donc avec plusieurs arbres et donc des prédictions différentes pour chaque individu. Comment obtenir l’estimation finale

* Dans le cas d’une classification : on choisit la catégorie la plus fréquente
* Dans le cas d’une régression : on fait la moyenne des valeurs prédites

- **choix des paramètres** :on a choisi 40 comme nombre de forets aléatoires et le random state =111 pour contrôler à la fois le bootstrap des échantillons et l'échantillonnage des fonctionnalités à prendre en compte lors de la recherche de la meilleure répartition à chaque nœud lors de la construction des arbres.

rf = RandomForestClassifier(n\_estimators=40, random\_state=111)

* **Réseaux de neuronnes :** Les réseaux de neurones, communément appelés des réseaux de neurones artificiels sont des **imitations simples des fonctions d’un neurone dans le cerveau humain** pour résoudre des problématiques d’apprentissage de la machine (Machine Learning).Pour commencer il va falloir choisir le nombre de couches cachées et de neurones que l’on souhaite utiliser pour l’algorithme. C’est simple : plus on ajoute de couches, plus les calculs sont longs, plus la prédiction lors de la phase d’apprentissage sera performant et plus le risque de sur-apprentissage est élevé.

Il faut donc trouver un juste milieu et choisir le bon nombre de neurones.

* Les réseaux ou algorithmes neuronaux artificiels sont généralement difficiles à configurer et lents à former, mais une fois préparés, leur application est très rapide. Ils sont généralement utilisés pour les domaines de problèmes basés sur l’approximation des fonctions et appréciés pour leurs capacités de généralisation et de tolérance au bruit. Ils sont connus pour être une boite noire, ce qui signifie qu’il est difficile d’expliquer les décisions prises par le réseau.

- **choix des paramètres** :pour notre problème on veut classifier des spams donc on a choisi 2 output (2 neurones de sorties ) et 200 d’entrées pour bien entrainer le réseau pour nous donner une bonne précision .

rn=MLPClassifier(solver='lbfgs', alpha=1e-5,hidden\_layer\_sizes=(200, 2), random\_state=111)

**4-Classification :**

Dans cette etape nous avons utiliser les classifieurs suivant :

**1-regression logistique**

**2-Machine à vecteurs de support(SVM)**

**3-Naive Bayes**

**4-Les arbres de decision**

**5-Méthode des k plus proches voisins**

**6-Random Forest**

**7- Reseaux de neuronnes**

**4-1 . Jeux d’entrainement**:division de notre data set en un ensemble test et un ensemble train on a pris 1 / 3 des données pour tester .

**Code**:

futures=vecteurs\_mots

X\_train, X\_test, y\_train, y\_test = train\_test\_split(futures, data['label'], test\_size=0.33, random\_state=100)

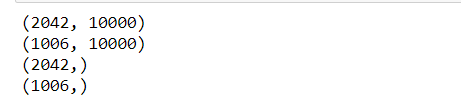
print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

**Execution :** taille de Xtrain(2042, 10000) ,taille de Xtest(2042, 10000) ,taille de Ytrain (2042,) ,taille de Ytest(1006,) .



**4-2.Modèles avec lesquels on va entrainer nos données :**

**Code :**

rl = LogisticRegression(solver='liblinear', penalty='l1')

svm = SVC(kernel='linear', probability=True)

nb = MultinomialNB()

dt = DecisionTreeClassifier(min\_samples\_split=7, random\_state=111)

n=neighbours()

kpp = KNeighborsClassifier(n)

rf = RandomForestClassifier(n\_estimators=40, random\_state=111)

rn=MLPClassifier(solver='lbfgs', alpha=1e-5,hidden\_layer\_sizes=(200, 2), random\_state=111)

On les a stocker dans un dictionnaire .

clfs = {'SVM' : svm,'k plus proches voisins' : kpp, 'Naive Bayes': nb, 'Les arbres de decision': dt, 'regression logistique': rl, 'Random Forest': rf,"Reseaux de neuronnes":rn}

**Fonction** **train et predict** :les fonction train nous permettra d’entrainer nos modèles et predict nous nous donnera les prédictions du modèle entrainer .

def train(clf, features, targets):

    clf.fit(features, targets)

def predict(clf, features):

    return (clf.predict(features))

**Fonction neighbours :** choisir le meilleur nombre de voisin pour knn on va faire une boucle qui a chaque fois nous teste l accuracy pour chaque nombre de voisins entre 1et 20 et nous retourne le meilleurhyperparameter (nombre de voisins avec lequel on obtient la meilleur precision adaptees a nos données)

def neighbours():

    bestneighbour=0

    prec=0

    for i in range(1,20):

        model = KNeighborsClassifier(n\_neighbors=i)

        train(model, X\_train, y\_train)

        pred = predict(model, X\_test)

        p=accuracy\_score(y\_test , pred)\*100

        if p>prec:

            prec=p

            bestneighbour=i

        #print("precision pour nombre de voisins",i,"voisins=",p)

    return bestneighbour

***4-3.Entrainement et mesures de performances :***

Dans cette étape on a parcouru le dictionnaire de modèles qu’on a définit puis on a entrainer chaque modele .on a afficher sa matrice de confusion et sa courbe roc ,pour les precisions,rappel,taux de faux positifs,taux de vrai negatives on les a stocker dans des listes qu’on a ensuit afficher.

precision = [ ]

recall = [ ]

fpr= [ ]

tnr= [ ]

def trainandmetrics():

    for k,v in clfs.items():

        train(v, X\_train, y\_train)

        pred = predict(v, X\_test)

        probs = v.predict\_proba(X\_test)

        preds = probs[:,1]

        fp, tn, threshold = roc\_curve(y\_test, preds)

        roc\_auc = auc(fp, tn)

        precision.append((k, [accuracy\_score(y\_test , pred)\*100]))

        recall.append((k, [recall\_score(y\_test , pred)\*100]))

        print("Confusion Matrix:")

        cm = confusion\_matrix(y\_test,pred)

        f, ax = plt.subplots(figsize =(2,2))

        sns.heatmap(cm,annot = True,linewidths=0.5,linecolor="blue",fmt = ".0f",ax=ax)

        plt.title(k)

        plt.xlabel("y pred")

        plt.ylabel("y true")

        plt.show()

        print("ROC CURVE:")

        plt.title(k)

        plt.plot(fp, tn, 'b', label = 'AUC = %0.2f' % roc\_auc)

        plt.legend(loc = 'lower right')

        plt.plot([0, 1], [0, 1],'r--')

        plt.xlim([0, 1])

        plt.ylim([0, 1])

        plt.ylabel('True Positive Rate')

        plt.xlabel('False Positive Rate')

        plt.show()

        fp=fp.mean()

        fpr.append((k,fp\*100))

        tn=tn.mean()

        tnr.append((k,tn\*100))

    print("Precision pour chaque classifieur")

    for i in precision:

        print(i)

    print("\n")

    print("Rappel pour chaque classifieur")

    for i in recall:

        print(i)

    print("\n")

    print("fpr pour chaque classifieur")

    for i in fpr:

        print(i)

    print("\n")

    print("tnr pour chaque classifieur")

    for i in tnr:

        print(i)

trainandmetrics()

**Execution :**

* **SVM**

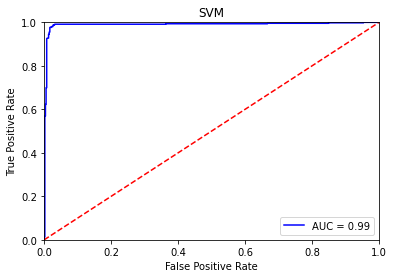
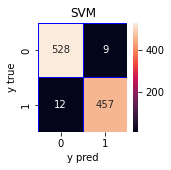


Figure 1: confusion matrix Figure 2:ROC CURVE1

* **K plus proches voisins :**

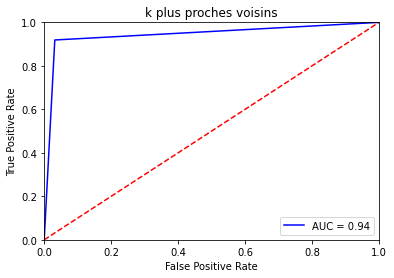
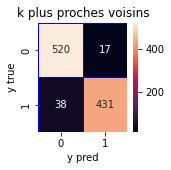


Figure 3: confusion matrix2 Figure 4: ROC CURVE2

* **Naive bayes :**

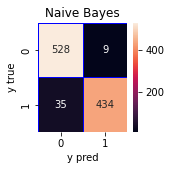
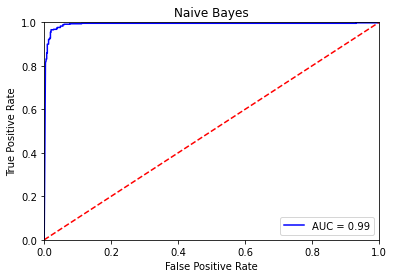
******

Figure 5:Confusion Matrix3Figure 6:ROC CURVE3

* **Les arbres de décision :**

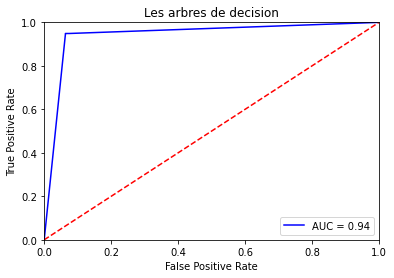
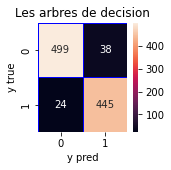


Figure 7:Confusion matrix4Figure 8: ROC CURVE4

* **Regression logistique :**

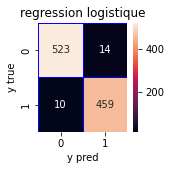
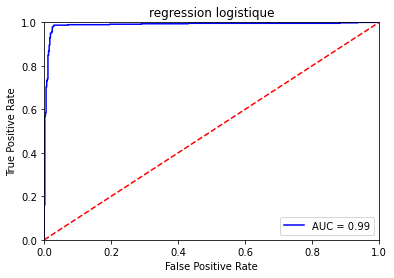
******

Figure 9: confusion matrix5Figure 10:ROC CURVE5

* **Random forest :**

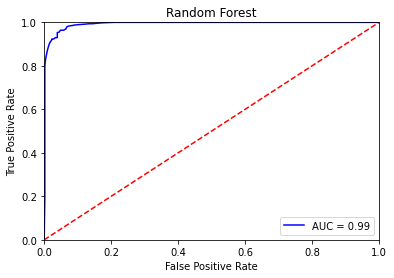
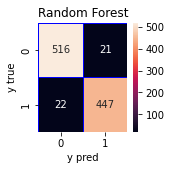
******

Figure 11: confusion matrix6Figure 12: ROC CURVE6

* **Réseaux de neuronnes :**

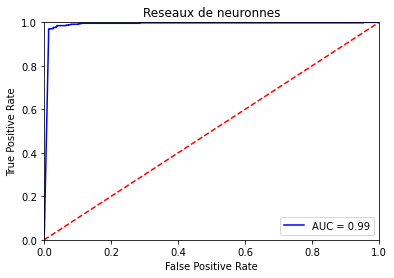
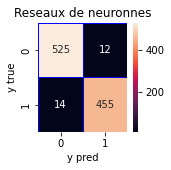
******

Figure 13:Confucion matrix7 Figure 14:ROC CURVE7

* **Les precision pour chaque classifieur :**

('SVM', [97.91252485089463])

('k plus proches voisins', [94.53280318091451])

('Naive Bayes', [95.62624254473161])

('Les arbres de decision', [93.83697813121272])

('regression logistique', [97.61431411530815])

('Random Forest', [95.72564612326043])

('Reseaux de neuronnes', [97.4155069582505])

* **Rappel pour chaque classifieur**

('SVM', [97.44136460554371])

('k plus proches voisins', [91.89765458422174])

('Naive Bayes', [92.53731343283582])

('Les arbres de decision', [94.88272921108742])

('regression logistique', [97.86780383795309])

('Random Forest', [95.3091684434968])

('Reseaux de neuronnes', [97.01492537313433])

* **Taux de faux positives pour chaque classifieur**

('SVM', 16.82572936064556)

('k plus proches voisins', 34.38857852265673)

('Naive Bayes', 12.490689013035382)

('Les arbres de decision', 21.198013656114213)

('regression logistique', 12.729449321628092)

('Random Forest', 12.104283054003725)

('Reseaux de neuronnes', 17.894553072625698)

* **Taux de vrai negatives pour chaque classifieur**

('SVM', 74.27594171997157)

('k plus proches voisins', 63.965884861407254)

('Naive Bayes', 91.40191897654584)

('Les arbres de decision', 79.8507462686567)

('regression logistique', 75.8033810539141)

('Random Forest', 86.78038379530916)

('Reseaux de neuronnes', 95.18256929637526)

**Conclusion :**

En comparant les classifieurs utilisée on obtient les meilleurs performances avec les SVM , Régression logistique et réseaux de neurones avec 97% de précision ensuite vient Naive Bayes avec une précision de 95%. Ceci confirme la fiabilité des méthodes d'apprentissage et les réseaux de neurones pour la détection de spam.