UNIVERSITE PARIS 8
UFR TERRITOIRES, ENVIRONNEMENT, SOCIETES
MATHEMATIQUES ET INFORMATIQUE APPLIQUEES AUX SHS

Compte rendu de Reconnaissance des Formes

SMS Spam Collection Data Set

Présenté par : ZIANE Feriel

TON Julienne

KAMYSHNIKOVA Veronika

Proposé par: M[r] Salvatore Anzalone

2018-2019

Introduction

SMS Spam Collection Data Set:

Le filtre anti-spam est un modèle d'apprentissage automatique utilisé pour détecter les messages non sollicités et non désirés et empêcher ces messages d'arriver dans la boîte de réception d'un utilisateur.

Nous utiliserons les données de SMS Spam Collection Data Set.

Algorithme :

Dans l'apprentissage automatique, la tâche la plus difficile consiste à appliquer un algorithme, car aucun algorithme aléatoire ne peut être appliqué à des données aléatoires.

Tout est une question de visualisation et de pré-perception de données. Il s'agit tout d'abord d'un modèle de classification. Nous avons deux colonnes : l'un contenant un texte ou un message, et l'autre une classe: spam ou ham. Il existe différentes techniques de classement, telles que knn, naïve bayes, la régression logistique, mais après avoir réfléchi de manière intuitive, nous utiliserons l'algorithme Naïve Bayes, car il utilise le théorème de Bayes pour déterminer la probabilité qu'un message soit du spam, en fonction des mots contenus dans ce message.

Si S est l'événement« un message est un spam » et si w est un mot du message, nous le classerons comme spam avec une probabilité :

$$P(S \mid w) = P(w \mid S) * P(S) / \{P(w \mid S) * P(S) + P(w \mid S\ ') * P(S')\}$$

Tout d'abord, nous devons prétraiter nos données textes et les convertir en vecteurs à l'aide de fonctions telles que BOW, TFIDF, Word2vec, Avg TFIDF Word2vec. Naïve utilise la probabilité de données pour que les valeurs des données soient positives dans le cas de Word2vec et Avg TFIDF, les vecteurs Word2vec peuvent être positifs ou négatifs, nous ne pouvons donc pas utiliser cette fonctionnalité.

Après le prétraitement, nous allons scinder nos données en training, puis les convertir en vecteurs à l'aide de BOW et TFIDF.

Nous utiliserons la bibliothèque Scikit de python pour implémenter Naïve Bayes en raison de sa lucidité.

Lissage Laplace :

Le lissage Laplace est utilisé pour lisser les données catégoriques et permet de prendre une observation.

x = (x1, x2, x3 ....., xd) à partir d'une distribution multinomiale à N essais, puis notre estimateur = (x + alpha) / (N + alpha * d).

Un biais élevé conduira à une sous-adaptation et une variance élevée à une sur-adaptation.

Nous devons donc déterminer notre meilleure valeur alpha de manière à maximiser notre surface sous la courbe ROC.

Ensuite, nous ajusterons nos données de training pour obtenir l'alpha optimal et prévoirons les données de test, puis résumerons cette information dans une matrice de confusion à l'aide de cartes thermiques.

Explication de code

Définition des librairies :

Dans notre projet on a utilisé plusieurs librairies et pour cela on va commencer par les définir :

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re

import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer


import pickle

from tqdm import tqdm
import os
```

Figure 1

***SQLite3* :** est une bibliothèque de python. Comme c'est un moteur de base de données légère sur disque ne nécessitant pas de processus serveur distinct et aussi

permet d’accéder à la base de données à l’aide d’une variante non standard. SQLite est autonome.

Pandas : est une librairie qui offre des outils efficaces pour lire et écrire des fichiers selon différents formats.

Numpy : est une bibliothèque qui permet d’effectuer des calculs numériques et d’introduit une gestion facilitée des tableaux de nombres.

Natural Language Toolkit (NLTK):est une boîte à outil qui nous a permis la création de programmes pour l'analyse de texte. (Comme la figure 12).

Matplotlib :c'est une librairie qui permet de tracer des graphes.

Seaborn : est une librairie qui vient s'ajouter à Matplotlib, et lui ajoute de nouvelles fonctionnalités. C'est une enveloppe autour de Matplotlib qui facilite la création de tracés statistiques communs.

sklearn.feature extraction: est un module peut être utilisé pour extraire des entités dans un format pris en charge par des algorithmes d’apprentissage automatique à partir d’ensembles de données.

Tfidftransformeret ***Tfidfvectorizer***de Scikit-learn visent la même chose: convertir une collection de documents bruts en une matrice de fonctionnalités TF-IDF. Cependant, cela aide à prendre en compte le fait que certains mots apparaissent plus fréquemment.

os *:* est le plus grand modules système de base du langage Python. Il contient tous les appels système. Ses appels traitent les répertoires, les processus, les variables Shell, etc.

pickle: est un module destiné à la sérialisation des objets afin qu'ils puissent être sauvegardés dans un fichier et chargés dans un programme ultérieurement.

Stopwords : ce sont des mots tels que «le», «un», «un», «dans» qu'un moteur de recherche a été programmé pour ignorer, à la fois lors de l'indexation d'entrées pour la recherche et lors de leur extraction. À la suite d'une requête de recherche.

Explication de code :

Affichage des targets de notre SMSSpam collection avec toutes les étiquettes ids.

```
import scipy as sp
target = sp.genfromtxt('SMSSpamCollection.csv', delimiter=",", usecols=(0), dtype=str, skip_header=1)
target_ids = np.unique(target)
#Afficher les étiquettes "ids"
print(target_ids)
```

```
['ham' 'ham""" 'spam']
```

Figure 2

Figure 3

On a commencé notre programme par la lecture des données de data set.

Puisque nous avons des ensembles de données plus volumineux, on a donc utilisé la méthode head () qui peut aussi être utile. On n'a pas fourni des arguments, donc il donne par défaut les 5 premières lignes (car notre base de donnée est très volumineuse elle est à 5572) et on ne pourra pas toutes les afficher.

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fna...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Figure 4

```
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)#retirer les données non nécessaires
#nomme label et message par v1 et v2 qu'on a rajouté à SMSSpamCollection
data = data.rename(columns={"v1":"label", "v2":"message"})
data.head()
```

Figure 5

data.drop : est utilisé pour retirer des données.

data.rename : est utilisé pour renommer les noms de colonne par label et message.

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Figure 6

Compter le nombre de messages spam et ham dans toute la base de données.

```
#Combien des messages sont ham et span
data['label'].value_counts()
```

Figure 7

Dans ce morceau de code, on a compté le nombre de messages spam et ham contenu dans notre base de données

Le résultat est le suivant :

```
1    4825
0     747
Name: label, dtype: int64
```

Figure 8

On a partitionné les messages de la base de donnée où :

0 représente spam

1 représente ham

```
#Changement les valeurs ham: 1 et spam : 0
def partition(x):
    if x=='ham':
        return 1
    else:
        return 0

actualScore=data['label']
hamSpam=actualScore.map(partition)
data['label']=hamSpam
data.head()
```

Figure 9

On obtient le résultat suivant :

	label	message
0	1	Go until jurong point, crazy.. Available only ...
1	1	Ok lar... Joking wif u oni...
2	0	Free entry in 2 a wkly comp to win FA Cup fina...
3	1	U dun say so early hor... U c already then say...
4	1	Nah I don't think he goes to usf, he lives aro...

Figure 10

Import re :est une librairie python, dont l'utilisation de re est de rechercher des modèles dans le texte (dans notre cas, on l'a utilisé pour chercher des chaînes de caractère) afin de les supprimer ensuite.

```python
#retirer "stop words" et Data Cleaning
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Figure 11

Dans l'image ci-dessous, on définit l'ensemble de la liste des « stop words » (mots vides) qui sera utilisée afin de filtrer les mots « inutiles » sur lesquels nous ne souhaitons pas que notre programme porte attention.

```python
# Tous les "stop words"
# On retire tous les mots de la liste de "stop words": 'no', 'nor', 'not'
# <br /><br /> ==> après les etapes suivantes on obtient "br br"
# On les ajoute dans La liste de "stop words"
# à la place de <br /> si on a <br/> ces tags auraient été retiré dans la 1ère etape


stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",\
            'won', "won't", 'wouldn', "wouldn't"])
```

Figure 12

La partie du script ci-dessous transforme les données brutes en données interprétables par le programme et affiche l'avancement du processus sous la forme d'une barre de complétion (fonction « tqdm »), qui affiche le pourcentage de

progression, le nombre de donnée transformé et le temps passé à réaliser cette tâche.

```
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_message = []

# tqdm c'est pour "print" le status bar

for sentance in tqdm(data['message'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_message.append(sentance.strip())
```

```
100%|██████████| 5572/5572 [00:03<00:00, 1409.75it/s]
```

Figure 13

Ci-dessous, les mille premiers messages sont traités et le dernier message traité est affiché.

```
preprocessed_message[1000]
```

```
'no heard abt tat'
```

Figure 14

Dans la partie du code ci-dessous, on importe la méthode Naïve Bayes avec la fonctionnalité « Bag of Word » qui va être utilisée afin de filtrer les messages.

Nous divisons ensuite les données entre training et test qui seront par la suite utilisées lors de nos futurs tests.

Les données que nous utilisons sont généralement divisées en données d’entraînement et en données de test. L'ensemble d'apprentissage contient une sortie connue et le modèle apprend sur ces données pour être généralisé ultérieurement à d'autres données.

Ici, nous allons importer les bibliothèques nécessaires :

Math - analyser les données.

Depuis Sklearn, nous avons importé le module des jeux de données afin de pouvoir charger un exemple de jeu de données.

Dans Sklearn, sous-bibliothèque model sélection, nous avons importé le train_test_splitafin de pouvoir le scinder en ensembles de formation et de test.

Nous pouvons maintenant utiliser la fonction train_test_split pour effectuer la scission. La taille de test = 0,3 à l'intérieur de la fonction indique le pourcentage de données qui doivent être conservées pour le test.

Nous mélangeons les données et les divisons en échantillons d'entraînement et d'essai. Une partie des données est utilisée pour la formation et une autre pour tester le modèle. Les données d'apprentissage et de test stockées dans des listes sont converties en tableaux numpy.

A la fin de cette partie nous obtenons la création de variable en training et en test.

```
import math
from sklearn.model_selection  import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection  import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
# Utilisation de "Multinomial Naive Bayes" pour "Spam Filtering"
# Fonctionnalité "BAG OF WORDS"

# Division des données entre train et test

X=preprocessed_message

y=np.array(data['label'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print(np.array(X_train).shape)
print(np.array(X_test).shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3900,)
(1672,)
(3900,)
(1672,)
```

Figure 15

Nous effectuons le « cross validation ». On cherche le valeur d'Alpha (le seul optimal). C'est très similaire à la division training / test, mais elle s'applique à un plus grand nombre de sous-ensembles. Cela signifie que nous divisons nos données en k sous-ensembles et formons sur k-1 l'un de ces sous-ensembles. Ce que nous faisons est de garder le dernier sous-ensemble pour le test. Nous sommes en mesure de le faire pour chacun des sous-ensembles.

```
count_vect=CountVectorizer()
X_train=count_vect.fit_transform(X_train)
X_test=count_vect.transform(X_test)

alpha_values = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]# Alpha de 10^-5 à 10^5
cv_scores = []

# Effectuer une "cross validation" 10 fois
for k in alpha_values:
    nnb = MultinomialNB(alpha = k)
    scores = cross_val_score(nnb, X_train, y_train, cv=10, scoring='accuracy', n_jobs=-1)
    cv_scores.append(scores.mean())

# Déterminer la meilleur valeur d'Alpha
optimal_alpha = alpha_values[cv_scores.index(max(cv_scores))]
print('\nLa valeur optimal de Alpha est %.3f.' % optimal_alpha)
```

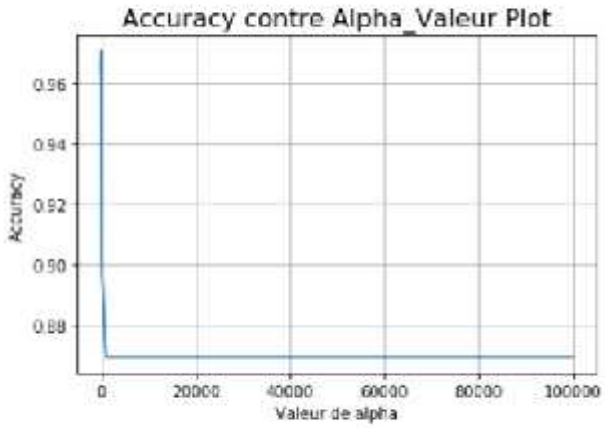
```
La valeur optimal de Alpha est 10.000.
```

Figure 16

On utilise « plt.plot » pour parceler « Accuracy » contre « Alpha » afin d'obtenir le graphique ci-dessous. De plus, on affiche les valeurs d'Alpha ainsi que les valeurs d'Accuracy pour chaque valeur d'Alpha.

```python
# On parcelle "Accuracy" contre "Alpha"
plt.plot(alpha_values, cv_scores)
plt.xlabel('Valeur de alpha',size=10)
plt.ylabel('Accuracy',size=10)
plt.title('Accuracy contre Alpha Valeur Plot',size=16)
plt.grid()
plt.show()

print("\n\nValeur de alpha :\n",alpha_values)
print("\nAccuracy pour chaque valeur de Alpha est :\n ", np.round(cv_scores,5))
```




```
Valeur de alpha :
 [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]

Accuracy pour chaque valeur de Alpha est :
  [0.96462 0.96513 0.9659  0.9659  0.96359 0.96257 0.97051 0.89898 0.86949
 0.86949 0.86949]
```

Figure 17

On a utilisé la méthode de MultinomialNB (Naïve Bayes) pour trouver la va leur d’alpha optimal. On utilise les données de training obtenues par les analyses pré cédants et on évalue la valeur d’alpha avec la valeur d’accuracy .

```python
# instancier le modèle d'apprentissage alpha = optimal_alpha
mnb = MultinomialNB(alpha = optimal_alpha)

# ajustement du modèle
mnb.fit(X_train, y_train)

# prédire la réponse
predictions = mnb.predict(X_test)

# Evaluer la précision
acc = accuracy_score(y_test, predictions) * 100
print('\nLe Teste Accuracy de Bernoulli Naive Bayes classifier pour alpha = %.1f is %f%%' % (optimal_alpha, acc))
```

```
Le Teste Accuracy de Bernoulli Naive Bayes classifier pour alpha = 10.000 is 96.852032%
```

Figure 18

On utilise la matrice de confusion, car elle est très utilisée dans les problèmes de multi-classe comme dans notre cas avec la fonction import confusion_matrix.On

affiche cette matrice avec des vrais labels et des labels prédictions à l'aide de la librairie seaborn (qu'on a déjà définie).

```
#Matrice de confusion utilisant "heatmap" pour les données de test
from sklearn.metrics import confusion_matrix
mnb=MultinomialNB(alpha=optimal_alpha)
mnb.fit(X_train,y_train)
predic=mnb.predict(X_test)
import seaborn as sns
conf_mat = confusion_matrix(y_test, predic)
class_label = ["ham", "spam"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Matrice de confusion")
plt.xlabel("Label predit")
plt.ylabel("Vrai Label")
plt.show()
```

Figure 19

On a pensé à utiliser la méthode TF-IDF, car c'est une méthode de pondération, elle est utilisée dans la recherche d'information et en particulier dans la fouille de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document.

On utilise la méthode tfidf.vectorizer à la base de X preprocessed message, Y array et les données de training et de test pour trouver la valeur d'Alpha optimal et effectuer un « cross validation ».

```python
# Essayons pour tfidf


X=preprocessed_message

y=np.array(data['label'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

tf_idf_vect = TfidfVectorizer(min_df=10)
X_train=tf_idf_vect.fit_transform(X_train)
X_test=tf_idf_vect.transform(X_test)

alpha_values = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]# Alpha de 10^-5 à 10^5
cv_scores = []

# Effectuer une "cross validation" 10 fois
for k in alpha_values:
    mnb = MultinomialNB(alpha = k)
    scores = cross_val_score(mnb, X_train, y_train, cv=10, scoring='accuracy', n_jobs=-1)
    cv_scores.append(scores.mean())

# Déterminer la meilleure valeur de l'Alpha
optimal_alpha = alpha_values[cv_scores.index(max(cv_scores))]
print('\nLe valeur optimal de Alpha est %.3f.' % optimal_alpha)
```

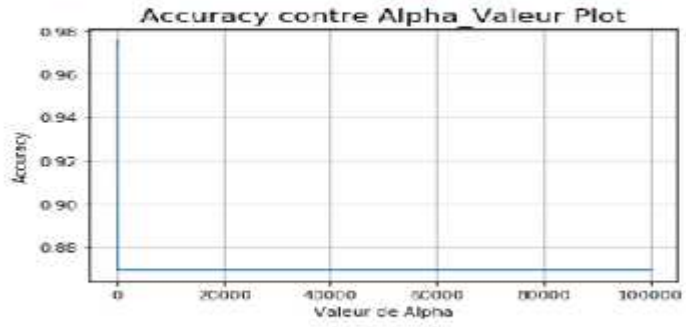
```
Le valeur optimal de Alpha est 0.100.
```

Figure 20

Méthode 2 « tfidf » : On utilise « plt.plot » pour parceler « Accuracy » contre « Alpha » afin d'obtenir le graphique ci-dessous. De plus, on affiche les valeurs d'Alpha ainsi que les valeurs d'Accuracy pour chaque valeur d'Alpha.

```python
# On parcelle "Accuracy" contre "Alpha"
plt.plot(alpha_values, cv_scores)
plt.xlabel('Valeur de Alpha',size=10)
plt.ylabel('Accuracy',size=10)
plt.title('Accuracy contre Alpha_Valeur Plot',size=15)
plt.grid()
plt.show()

print("\n\nValeur de Alpha :\n",alpha_values)
print("\nAccuracy pour chaque valeur de Alpha est :\n ", np.round(cv_scores,5))
```




```
Valeur de Alpha :
 [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]

Accuracy pour chaque valeur de Alpha est :
 [0.97411 0.97436 0.97462 0.97538 0.97564 0.97359 0.91803 0.86949 0.86949
 0.86949 0.86949]
```

Figure 21

Méthode 2 « tfidf » : On a utilisé la fonction de MultinomialNB (Naïve Bayes) pour trouver la valeur d'alpha optimal. On utilise les données de training obtenues par les analyses précédants et on évalue la valeur d'alpha avec la valeur d'accuracy.

```
# instancier le modèle d'apprentissage alpha = optimal_alpha
mnb = MultinomialNB(alpha = optimal_alpha)

# ajustement du modèle
mnb.fit(X_train, y_train)

# prédire la réponse
predictions = mnb.predict(X_test)

# évaluer la précision
acc = accuracy_score(y_test, predictions) * 100
print('\nLe Teste Accuracy de Bernoulli Naive Bayes classifier pour alpha = %.3f is %f%%' % (optimal_alpha, acc))
```

```
Le Teste Accuracy de Bernoulli Naive Bayes classifier pour alpha = 0.100 is 96.949761%
```

Figure 22

On continue la méthode 2 « tfidf »…

```
# Matrice de confusion utilisant "heatmap" pour les données de test
from sklearn.metrics import confusion_matrix
mnb=MultinomialNB(alpha=optimal_alpha)
mnb.fit(X_train,y_train)
predic=mnb.predict(X_test)
import seaborn as sns
cont_mat = confusion_matrix(y_test, predic)
class_label = ["ham", "spam"]
df = pd.DataFrame(cont_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Algorithme de clustering")
plt.xlabel("Label predit")
plt.ylabel("Vrai label")
plt.show()
```

Figure 23

Conclusion :

Dans ce rapport, nous avons expliqué d’une manière générale notre projet et son but ainsi que toutes les étapes de la création du code avec des explications.

Le thème de projet est très intéressant et nous estimons que le temps consacré au projet est peu suffisant pour l’améliorer.