

Naive Base

TP BDM



2019/2020

***Réalisé par :***

*REFFAD Sonia*

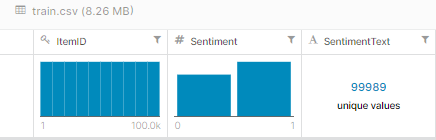
*REKKAS Imene*

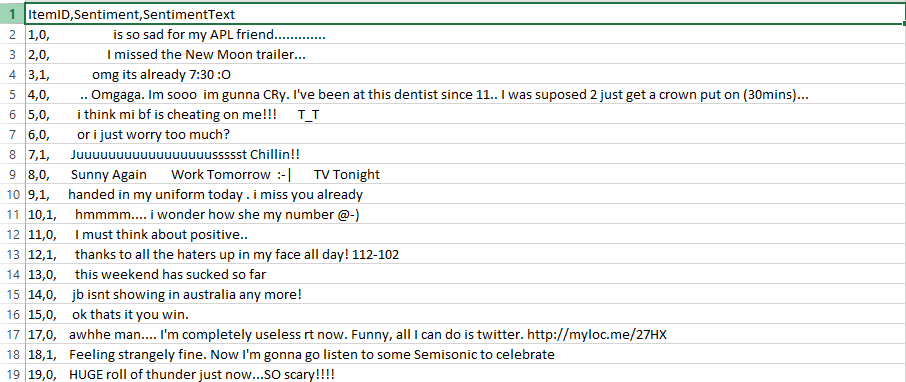
# Introduction :

Afin de nous familiariser avec le domaine de Machine Learning, nous a été demandé de faire exercice 3 de la série Apprentissage supervisé qui consiste à effectuer à l'aide de python l'algorithme naive bayes sur un dataset d’analyse des sentiments. Pour cela nous avons choisis le dataset « tweets » d’après le site : <https://www.kaggle.com/imrandude/twitter-sentiment-analysis>.

Ce dataset détermine la coloration émotionnelle des tweets, il contient les champs de données suivant :

* ItemID : id de tweet
* Sentiment : sentiment 🡪 0 – négatif 1 - positif
* SentimentText : texte du tweet





# Algorithme de naive base :

Le processus d'analyse du langage naturel et de son sens relève du domaine du traitement du langage naturel (NLP). L'analyse des sentiments est une tâche courante de la NLP, qui implique de classer des textes ou des parties de textes en un sentiment prédéfini. Nous allons utiliser le Natural Language Toolkit (NLTK), une bibliothèque NLP couramment utilisée en Python, pour analyser les données textuelles.

Nous allons préparer un ensemble de données d'exemples de tweets à partir du package NLTK pour NLP avec différentes méthodes de nettoyage des données. Une fois l'ensemble de données prêt à être traité, vous formerez un modèle sur les tweets pré classifiés et utiliserez le modèle pour classer les exemples de tweets en sentiments négatifs et positifs.

## Étape 1 - Installation de NLTK et téléchargement des données

Dans cette étape, nous allons installer NLTK et télécharger les exemples de tweets que nous utiliserons pour former et tester votre modèle.

pip install nltk==3.3

python3

Ensuite, importez le module nltk dans l'interpréteur python :

import nltk

Téléchargez les exemples de tweets du pack NLTK :

nltk.download('twitter\_samples')

## Étape 2 - Tokenisation des données

La langue dans sa forme d'origine ne peut pas être traitée avec précision par une machine, nous devons donc traiter la langue pour la rendre plus facile à comprendre pour la machine. La première partie pour donner un sens aux données passe par un processus appelé tokenisation, ou la division des chaînes en parties plus petites appelées tokens.

Un token est une séquence de caractères dans le texte qui sert d'unité. Selon la façon dont on crée les tokens, ils peuvent être constitués de mots, d'émoticônes, de hashtags, de liens ou même de caractères individuels. Un moyen de base de décomposer le langage en tokens consiste à fractionner le texte en fonction des espaces et de la ponctuation.

Pour commencer, on crée un nouveau fichier main.py pour contenir notre script.

from nltk.corpus import twitter\_samples

Cela importera trois jeux de données de NLTK qui contiennent divers tweets pour entraîner et tester le modèle :

* negative\_tweets.json : 5000 tweets avec sentiments negatives
* positive\_tweets.json : 5000 tweets avec sentiments positives

La méthode strings () de twitter\_samples affichera tous les tweets d'un ensemble de données sous forme de chaînes. La définition des différentes collections de tweets en tant que variable facilitera le traitement et les tests.

Avant d'utiliser un tokenizer dans NLTK, on doit télécharger une ressource supplémentaire, punkt. Le module punkt est un modèle pré-formé qui nous aide à symboliser les mots et les phrases.

import nltk

nltk.download('punkt')

Une fois le téléchargement terminé, on est prêt à utiliser les tokenizers de NLTK. NLTK fournit un tokenizer par défaut pour les tweets avec la méthode .tokenized ().

## Étape 3 - Normalisation les données

Les mots ont des formes différentes - par exemple, “ran”, “runs” et “running” sont diverses formes du même verbe, “run”. Selon les besoins de votre analyse, toutes ces versions peuvent devoir être converties sous la même forme, “run” La normalisation en PNL est le processus de conversion d'un mot dans sa forme canonique.

import nltk

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

« wordnet » est une base de données lexicale pour la langue anglaise qui aide le script à déterminer le mot de base. On a besoin de la ressource « averaged\_perceptron\_tagger » pour déterminer le contexte d'un mot dans une phrase.

Voici la liste des éléments les plus courants et leur signification :

* NNP: Noun, proper, singular
* NN: Noun, common, singular or mass
* IN: Preposition or conjunction, subordinating
* VBG: Verb, gerund or present participle
* VBN: Verb, past participle

...

from nltk.tag import pos\_tag

from nltk.stem.wordnet import WordNetLemmatizer

def lemmatize\_sentence(tokens):

lemmatizer = WordNetLemmatizer()

lemmatized\_sentence = []

for word, tag in pos\_tag(tokens):

if tag.startswith('NN'):

pos = 'n'

elif tag.startswith('VB'):

pos = 'v'

else:

pos = 'a'

lemmatized\_sentence.append(lemmatizer.lemmatize(word, pos))

return lemmatized\_sentence

## Étape 4 - Suppression du bruit des données

Dans cette étape, on supprime le bruit de l'ensemble de données. Le bruit est une partie du texte qui n'ajoute pas de sens ou d'information aux données.

On va utiliser des expressions régulières en Python pour rechercher et supprimer ces éléments :

**Hyperliens** - Tous les hyperliens de Twitter sont convertis en raccourcisse d'URL t.co. Par conséquent, les conserver dans le traitement de texte n'ajouterait aucune valeur à l'analyse.

**Twitter gère les réponses** - Ces noms d'utilisateur Twitter sont précédés d'un symbole @, qui ne donne aucun sens.

**Ponctuation et caractères spéciaux** - Bien que ceux-ci fournissent souvent un contexte aux données textuelles, ce contexte est souvent difficile à traiter. Par souci de simplicité, vous supprimerez tous les signes de ponctuation et caractères spéciaux des tweets

...

import re, string

def remove\_noise(tweet\_tokens, stop\_words = ()):

cleaned\_tokens = []

for token, tag in pos\_tag(tweet\_tokens):

token = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+#]|[!\*\(\),]|'\

'(?:%[0-9a-fA-F][0-9a-fA-F]))+','', token)

token = re.sub("(@[A-Za-z0-9\_]+)","", token)

if tag.startswith("NN"):

pos = 'n'

elif tag.startswith('VB'):

pos = 'v'

else:

pos = 'a'

lemmatizer = WordNetLemmatizer()

token = lemmatizer.lemmatize(token, pos)

if len(token) > 0 and token not in string.punctuation and token.lower() not in stop\_words:

cleaned\_tokens.append(token.lower())

return cleaned\_tokens

Ce code crée une fonction remove\_noise () qui supprime le bruit et incorpore la normalisation et la lemmatisation mentionnées dans la section précédente. Le code prend deux arguments : les tokens de tweet et le tuple de mots vides.

En plus de cela, on supprime également les mots vides à l'aide d'un ensemble intégré de mots vides en NLTK, qui doit être téléchargé séparément.

nltk.download('stopwords')

On peut utiliser la méthode .words () pour obtenir une liste de mots vides en anglais.

...

from nltk.corpus import stopwords

stop\_words = stopwords.words('english')

Avant de passer à l'étape suivante, on utilise la fonction remove\_noise () pour nettoyer les tweets positifs et négatifs.

positive\_tweet\_tokens = twitter\_samples.tokenized('positive\_tweets.json')

negative\_tweet\_tokens = twitter\_samples.tokenized('negative\_tweets.json')

positive\_cleaned\_tokens\_list = []

negative\_cleaned\_tokens\_list = []

for tokens in positive\_tweet\_tokens:

positive\_cleaned\_tokens\_list.append(remove\_noise(tokens, stop\_words))

for tokens in negative\_tweet\_tokens:

negative\_cleaned\_tokens\_list.append(remove\_noise(tokens, stop\_words))

## Étape 5 - Préparation des données pour le modèle

Nous allons créer un ensemble de données d'entraînement pour former un modèle. Il s'agit d'un processus d'apprentissage machine d'apprentissage supervisé, qui nous oblige à associer chaque ensemble de données à un «sentiment» pour la formation. Dans ce didacticiel, votre modèle utilisera les sentiments «positifs» et «négatifs».

Dans l'étape de préparation des données, on prépare les données pour l'analyse des sentiments en convertissant les tokens au format dictionnaire, puis en divisant les données à des fins de formation et de test.

Tout d'abord, on prépare les données à introduire dans le modèle. on utilise le classifieur Naive Bayes dans NLTK pour effectuer l'exercice de modélisation. Notons que le modèle nécessite non seulement une liste de mots dans un tweet, mais un dictionnaire Python avec des mots comme clés et True comme valeurs. La fonction suivante crée une fonction de générateur pour changer le format des données nettoyées.

...

def get\_tweets\_for\_model(cleaned\_tokens\_list):

for tweet\_tokens in cleaned\_tokens\_list:

yield dict([token, True] for token in tweet\_tokens)

positive\_tokens\_for\_model = get\_tweets\_for\_model(positive\_cleaned\_tokens\_list)

negative\_tokens\_for\_model = get\_tweets\_for\_model(negative\_cleaned\_tokens\_list)

Ensuite, on doit préparer les données pour la formation de la classe NaiveBayesClassifier.

...

import random

## si la donnée est positive, on retourne un 1 sinon un 0

positive\_dataset = [(tweet\_dict, "1")

for tweet\_dict in positive\_tokens\_for\_model]

negative\_dataset = [(tweet\_dict, "0")

for tweet\_dict in negative\_tokens\_for\_model]

dataset = positive\_dataset + negative\_dataset

random.shuffle(dataset)

train\_data = dataset[:7000]

test\_data = dataset[7000:]

Ce code attache une étiquette positive=1 ou négative=0 à chaque tweet. Il crée ensuite un ensemble de données en joignant les tweets positifs et négatifs.

Étant donné que le nombre de tweets est de 10000, vous pouvez utiliser les 7000 premiers tweets de l'ensemble de données mélangé pour entraîner le modèle et les 3000 derniers pour tester le modèle.

## Étape 6 - Construction et test du modèle

Enfin, on peut utiliser la classe NaiveBayesClassifier pour construire le modèle. Utilisez la méthode .train () pour entraîner le modèle et la méthode .accuracy () pour tester le modèle sur les données de test.

...

from nltk import classify

from nltk import NaiveBayesClassifier

classifier = NaiveBayesClassifier.train(train\_data)

print("Accuracy is:", classify.accuracy(classifier, test\_data))

print(classifier.show\_most\_informative\_features(10))

Output

Accuracy is: 0.9956666666666667

Most Informative Features

:( = True Negati : Positi = 2085.6 : 1.0

:) = True Positi : Negati = 986.0 : 1.0

welcome = True Positi : Negati = 37.2 : 1.0

arrive = True Positi : Negati = 31.3 : 1.0

sad = True Negati : Positi = 25.9 : 1.0

follower = True Positi : Negati = 21.1 : 1.0

bam = True Positi : Negati = 20.7 : 1.0

glad = True Positi : Negati = 18.1 : 1.0

x15 = True Negati : Positi = 15.9 : 1.0

community = True Positi : Negati = 14.1 : 1.0

La précision est définie comme le pourcentage de tweets dans l'ensemble de données de test pour lesquels le modèle a correctement pu prédire le sentiment. Une précision de 99,5% sur l'ensemble de test est assez bonne.

Ensuite, nous allons vérifier les performances du modèle sur les tweets aléatoires de Twitter

## Étape 7 - Vérification des performances du modèle

Pour cette partie nous allons utiliser le dataset choisis depuis le site « Kaggle ».

Ce code nous permettra de tester des tweets personnalisés en mettant à jour la chaîne associée à la variable custom\_tweet depuis SentimentText de dataset (row[2]), ensuite on remplis le résultats dans un tableau de prédiction « arrP ».

On remplit aussi un autre tableau « arrR » par les sentiments réels « row[1] » pour calculer après l’erreur .

...

from nltk.tokenize import word\_tokenize

import csv

arrR = []

arrP = []

with open('testdata.csv', encoding="utf8") as file:

reader = csv.reader(file)

for row in reader:

custom\_tweet = row[2]

custom\_tokens = remove\_noise(word\_tokenize(custom\_tweet))

print(row[1] +' '+classifier.classify(dict([token, True] for token in custom\_tokens)))

arrP.append(classifier.classify(dict([token, True] for token in custom\_tokens)))

arrR.append(row[1])

## Étape 8 – Matrice de confusion et calcule de l’erreur

from sklearn.metrics import confusion\_matrix

tableDeConfusion = confusion\_matrix(arrR,arrP)

print(tableDeConfusion)



**Erreur :**

Err = (15917 + 19626) / (15917 + 19626+8897+9399)

= 30%

# Exo4 :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Outlook** | **Temperature** | **Humidiy** | Wind | playTenis |
| Dl | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| Dl0 | Rain | Mild | Normal | Weak | Yes |
| Dl1 | Sunny | Mild | Normal | Strong | Yes |
| Dl2 | Overcast | Mild | High | Strong | Yes |
| Dl3 | Overcast | Hot | Normal | Weak | Yes |
| Dl4 | Rain | Mild | High | Strong | No |

1. L’entropie de la sortie (14=9OUI+5NON) :

E(S)= −9/14 log2(9/14) –5/14 log2(5/14)=0.940

E(Snn, S) = −3/5 log2( 3/5 ) – 2/5 log2( 2/5 ) = 0.970

E(Ran, S) =−3/5 log2( 3/5 ) – 2/5 log2( 2/5 ) = 0.970

E(Ovr, S) =−4/4 log2( 1 )=0

Alors le gain de l’attribut Outlook est :

G(O/S) = 0.940 – (5/14)\*0.970 – (5/14)\*0.970= 0.247

E(Hot) = −2/4 log2(2/4) –2/4 log2(2/4) = 1

E(Mild) = −4/6 log2(4/6) – 2/6 log2(2/6) = 0.918

E(Cool)= −3/4 log2(3/4) – 1/4 log2(1/4)=0.811

Alors le gain de l’attribut Temperature est

G(T/S) = 0.940 −4/14 \*1−6/14 \*0.918−4/14 \*0.811 = 0.029

E(High )= −3/7 log2(3/7) – 4/7 log2(4/7)= 0.985

E(Normal )= −6/7 log2(6/7) – 1/7 log2(1/7)=0.591

Alors le gain de l’attribut Humidity est

G(H/S) = 0.940−7/14 \* 0.985−7/14 \*0.591= 0.152

E(Weak) = −2/8 log2(2/8) −6/8 log2(6/8) = 0.811

E(Strong) = −3/6 log2(3/6) −3/6 log2(3/6) = 1

Alors le gain de l’attribut Wind est

G(W/S) = 0.940 −8/14 \* 0.811−6/14 \*1= 0.048

La racine de l’arbre est l’attribut Humidity.

Outlook

OverCast

Sunny

Rain

Oui

1. L’entropie de la sortie Sunny (5=3OUI+3NON) :

E1(S)= −3/5 log2(3/5) −2/5log2(2/5)= 0.970

E1(Hot) = −1log2(1) = 0

E1(Mild) = −1/2 log2(1/2) – 1/2 log2(1/2) = 1

E1(Cool)= 0

Alors le gain de l’attribut Temperature est

G1(T/S) = 0.970– (2/5)\*1= 0.57

E1(High )= 0

E1(Normal )= 0

Alors le gain de l’attribut Humidity est

G1(H/S) = 0.970

E1(Weak) = −2/3 log2(2/3) −1/3 log2(1/3) = 0.918

E1(Strong) = −1/2 log2(1/2) –1/2 log2(1/2) = 1

Alors le gain de l’attribut Wind est

G1(W/S) = 0.970 –3/5 \* 0.918−2/5 \*1= 0.019

prochain noeud pour Sunny est le Humidity.

Outlook

OverCast

Sunny

Rain

Oui

Humidity

Hight

Normal

Non

Oui

1. L’entropie de la sortie Rain (5=3OUI+3NON) :

E2(S)= −3/5 log2(3/5) −2/5log2(2/5)= 0.970

E2(Mild) = −2/3log2(2/3) −1/3log2(1/3) = 0.918

E2(Cool)= −1/2 log2(1/2) – 1/2 log2(1/2)=1

Alors le gain de l’attribut Temperature est

G2(T/S) = 0.970– (2/5)\*1– (3/5)\* 0.918= 0.020

E2(High )= −1/2 log2(1/2) – 1/2 log2(1/2)=1

E2(Normal )= −2/3log2(2/3) −1/3log2(1/3) = 0.918

Alors le gain de l’attribut Humidity est

G2(H/S) = 0.020

E2(Weak) = 0

E2(Strong) = 0

Alors le gain de l’attribut Wind est

G2(W/S) = 0.970

prochain noeud pour Rain est le Wind. On aura donc l’arbre de décision suivant :

Outlook

OverCast

Sunny

Rain

Oui

Humidity

Humidity

Hight

Normal

Non

Oui

Strong

Weak

Non

Oui