République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'informatique

Département informatique

Projet de fin de semestre

Analyse de sentiments des tweets

Réalisées par:

- Hadjimi Lilia 202031075620
- Sebti asma 202031043740

Introduction:

Les réseaux sociaux, et en particulier Twitter, sont des sources riches et variées de données textuelles générées par les utilisateurs, reflétant un large éventail d'opinions et d'émotions. La classification des sentiments de tweets implique de catégoriser les tweets en classes prédéfinies, telles que positives, négatives ou neutres.

Dans ce projet, nous avons choisi d'utiliser l'algorithme des k-plus proches voisins (k-NN) pour construire notre classifieur de sentiments. L'algorithme k-NN est un algorithme de classification simple et intuitif qui attribue une classe à un échantillon en fonction des classes de ses k voisins les plus proches dans l'ensemble d'entraînement. En dépit de sa simplicité, le k-NN est capable de produire des résultats robustes, surtout lorsqu'il est correctement paramétré et optimisé.

La réalisation de ce projet implique plusieurs étapes clés, notamment la collecte et la préparation des données, la standardisation des caractéristiques, l'entraînement du modèle k-NN, et l'évaluation de ses performances à l'aide de métriques appropriées. L'objectif final est de développer un classifieur capable de prédire avec précision le sentiment exprimé dans de nouveaux tweets, démontrant ainsi l'application pratique des techniques de science des données apprises au cours du semestre.

Première étape:

1. préparation des données (preprocessing):

Le prétraitement des données est une étape essentielle, impliquant le nettoyage et la normalisation des tweets bruts pour une analyse précise. Ce processus comprend la suppression des éléments non pertinents, la tokenisation et le traitement des mots vides.

Dans cette étape nous avons procéder ainsi:

1.1. Choix des bibliothèques:

pour le prétraitement du Dataset des tweets nous avons utilisé les bibliothèques suivantes:

```
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
from nltk.stem import PorterStemmer
```

✓ 4.9s

- Pandas:

définition:

Pandas est une bibliothèque Python utilisée pour la manipulation et l'analyse des données. Elle offre des structures de données et des outils de manipulation de données puissants

Justification:

Nous avons utilisé la bibliothèque *pandas* par sa capacité à travailler avec des données tabulaires, ce qui a été le plus adapté pour charger et enregistrer nos données à partir du fichiers “training.1600000.processed.noemoticon.csv”.

- re :

définition:

Regular Expression, est une bibliothèque de Python qui offre des fonctionnalités pour le traitement des expressions régulières

Justification:

Elle est utilisée ici pour effectuer des opérations de recherche et de remplacement de motifs dans les tweets, comme la suppression des URL et des mentions.

- String :

Définition:

La bibliothèque string fournit des constantes et des fonctions utiles pour manipuler des chaînes de caractères.

Justification:

Dans ce cas, elle est utilisée pour vérifier et supprimer les caractères non alphabétiques et la ponctuation des tweets.

- NLTK (Natural Language Toolkit) :

Définition et justification:

NLTK est une bibliothèque puissante et largement utilisée pour le traitement du langage naturel en Python. Elle offre une variété d'outils et de ressources pour le prétraitement des données textuelles, y compris la tokenisation, le stemming, la lemmatisation, le traitement des mots vides.

  - StopWords : Dans le cadre de **NLTK**, les stopwords sont une liste prédéfinie de mots vides de la langue anglaise, extraits du **corpus** (qui est une collection de données textuelles), qui comprend des mots tels que "the", "is", "and", "are", etc.
  - PorterStemmer : est une classe du module **stem** (qui comprend différentes classes et fonctions pour effectuer le stemming) de la bibliothèque **NLTK** utilisée pour réduire les mots à leur forme de base (ou racine) en supprimant les suffixes.
  - TweetTokenizer : est une classe du module tokenize de la bibliothèque NLTK spécialement conçue pour la tokenization des

tweets. Contrairement à la tokenization standard, TweetTokenizer est capable de gérer les particularités des tweets, telles que les hashtags, les mentions d'utilisateurs et les émoticônes.

Code du prétraitement:

```python
# Fonction pour nettoyer un tweet
def clean_tweet(tweet):

    # Convertir en minuscules
    tweet = tweet.lower()

    # Supprimer les balises HTML
    tweet = re.sub(r'<.*?>', '', tweet)

    # Supprimer les URL
    tweet = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', tweet)

    # Supprimer les mentions
    tweet = re.sub(r'@[^\s]+', '', tweet)

    # Supprimer les hashtags et leur mot suivant
    tweet = re.sub(r'#\w+\b\s*', '', tweet)

    # Normalisation todaaaay ==> today
    tweet = re.sub(r'([A-Za-z])\1{2,}', r'\1', tweet)

    # Supprimer les caractères non alphabétiques et la ponctuation
    tweet = re.sub(r'[^\w\s]', '', tweet)

    # Supprimer les mots vides (stop words)
    stop_words = set(stopwords.words('english'))
    stop_words.update(['new', 'way'])  # Ajouter des mots personnalisés à supprimer
    tweet = ' '.join(word for word in tweet.split() if word not in stop_words)

    # Racinisation des mots
    stemmer = PorterStemmer()
    tweet = ' '.join(stemmer.stem(word) for word in tweet.split())

    # Supprimer les caractères speciaux
    tweet = ''.join(char for char in tweet if char in string.printable)

    return tweet
```

✓ 0.0s

Explication:

Tout d'abord on charge le fichier CSV 'training.1600000.processed.noemoticon.csv' dans un DataFrame pandas qu'on a appelé Data, en spécifiant l'encodage à utiliser ('latin1') et en indiquant qu'il n'y a pas de ligne d'en-tête dans le fichier. Ensuite, on affiche les cinq premières lignes du DataFrame à des fins de permettre l'exploration initiale des données.

```
# Charger le fichier CSV en utilisant la fonction définie
Data = pd.read_csv('training.1600000.processed.noemoticon.csv', encoding='latin1', header=None)
Data.head()
```

✓ 5.1s

	0	1	2	3	4	5
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

Vu que on aura besoin dans le cas d'analyse de sentiments que du texte(tweet) et son target on va créer un nouveau dataframe comme suit :

récupère la première colonne du DataFrame Data et la stocke dans une variable appelée target.

récupère la dernière colonne du DataFrame Data et la stocke dans une variable appelée text.

concatèné les deux variables target et text le long de l'axe horizontal (axis=1) pour former un nouveau DataFrame appelé New_Data.

Il enregistre le DataFrame New_Data dans un fichier CSV nommé 'New_Data.csv', sans inclure les index des lignes (index=False) ni la ligne d'en-tête (header=False).

```
# Récupérer la première colonne dans une variable target
target = Data.iloc[:, 0]
# Récupérer la dernière colonne dans une variable text
text = Data.iloc[:, -1]

# Concaténer les deux variables
New_Data = pd.concat([target, text], axis=1)

# Enregistrer le résultat dans un fichier CSV
New_Data.to_csv('New_Data.csv', index=False, header=False)
```

✓ 8.1s

L'étape du prétraitement de nos données est implémenté dans la méthode ‘clean_tweet’ qui prendra toutes les lignes du dataframe et seulement la colonne qui contient le text (la colonne 1)

a- Conversion des majuscules en minuscules:

Pour réaliser cela nous avons utilisé ***lower()*** qui convertit tout le texte du tweet en minuscules. Cela permet d'uniformiser la casse des mots, ce qui simplifie le traitement ultérieur.

b- Suppression des balises HTML:

Pour réaliser cela nous avons utilisé une expression régulière ***re.sub()*** pour supprimer toutes les balises HTML du tweet. Elle remplace toutes les occurrences de texte compris entre < et > par une chaîne vide.

c- Suppression des URL:

Pour réaliser cela nous avons utilisé une autre expression régulière pour supprimer les URL du tweet. Elle recherche les motifs correspondant aux URL et les remplace par une chaîne vide.

d-Supprimer les mentions:

supprime les mentions d'utilisateurs dans le tweet pour réaliser cela nous avons utilisé re.sub(r'@[^\s]+', '', tweet) qui recherche les motifs commençant par @ suivis de un ou plusieurs caractères qui ne sont pas des espaces (\s) et les remplace par une chaîne vide.

e-Suppression des hashtags et leur mot suivant:

supprime les hashtags et les mots qui les suivent dans le tweet. re.sub() recherche les motifs commençant par # suivis de un ou plusieurs caractères de mot (\w+) et les remplace par une chaîne vide.

f- Normalisation :

En utilisant une expression régulière pour effectuer une normalisation. Plus précisément, elle vise à remplacer les occurrences répétées de caractères dans les mots.

L'expression régulière r'([A-Za-z])\1{2,}' identifie toute séquence de trois caractères ou plus ({2,}) qui se répète, c'est-à-dire où un caractère alphabétique ([A-Za-z]) est suivi par deux occurrences ou plus (\1) du même caractère.

La fonction re.sub() remplace ces occurrences répétées par une seule occurrence du caractère correspondant (r'\1'), ce qui normalise les mots en réduisant les répétitions inutiles.

j- Supprimer les caractères non alphabétiques et la ponctuation :

en utilisant une expression régulière pour supprimer tous les caractères qui ne sont pas des lettres ou des espaces dans le tweet, en remplaçant ces caractères par une chaîne vide. Cela permet de se débarrasser de la ponctuation et d'autres symboles non alphabétiques qui pourraient ne pas être pertinents pour l'analyse.

h- Supprimer les mots vides :

en utilisant la liste de stopwords de la langue anglaise à partir de NLTK pour filtrer les mots vides du tweet.

La variable stop_words contient une liste de mots vides, et la compréhension de liste '***word for word in tweet.split() if word not in stop_words***' filtre les mots du tweet en ne conservant que ceux qui ne sont pas des mots vides.

i- Racinisation des mots :

En utilisant le PorterStemmer de NLTK pour réduire chaque mot à sa forme racine.

Pour chaque mot du tweet, le PorterStemmer applique le stemming, ce qui consiste à supprimer les suffixes afin de normaliser les mots et de les ramener à leur forme racine.

g- Supprimer les caractères spéciaux :

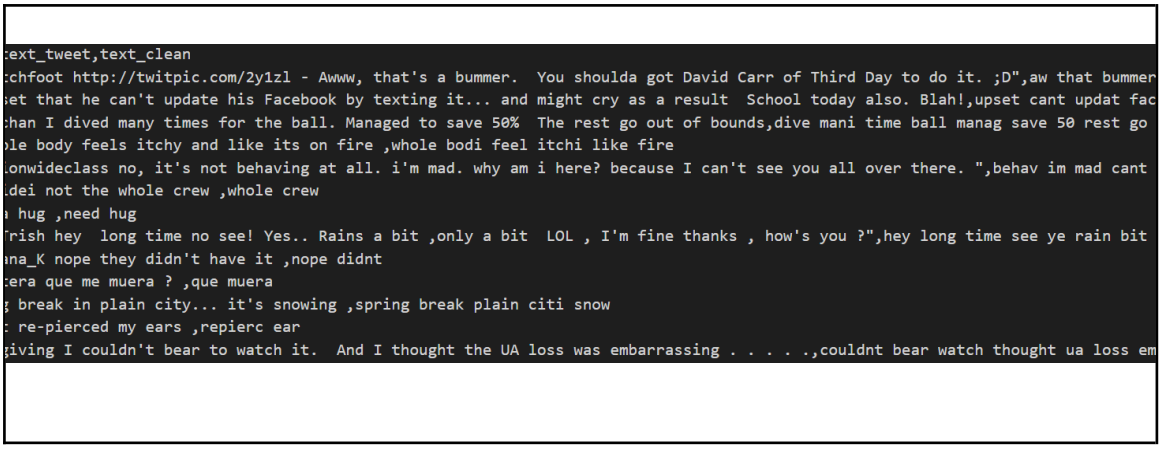
En filtrant tous les caractères spéciaux qui ne font pas partie du jeu de caractères imprimables ASCII standard. Cela élimine tout caractère non standard qui pourrait être présent dans le tweet après les étapes précédentes de prétraitement, assurant ainsi que le texte est propre et prêt pour l'analyse ultérieure.

Exécution:

Appliquer la fonction **clean_tweet** définie précédemment avec la deuxième colonne du dataframe (la colonne du tweet) puis appliquer le tokenizer (tknzr) à chaque tweet nettoyé de la colonne 'text_clean', les résultats seront donc enregistrés dans des fichiers CSV.

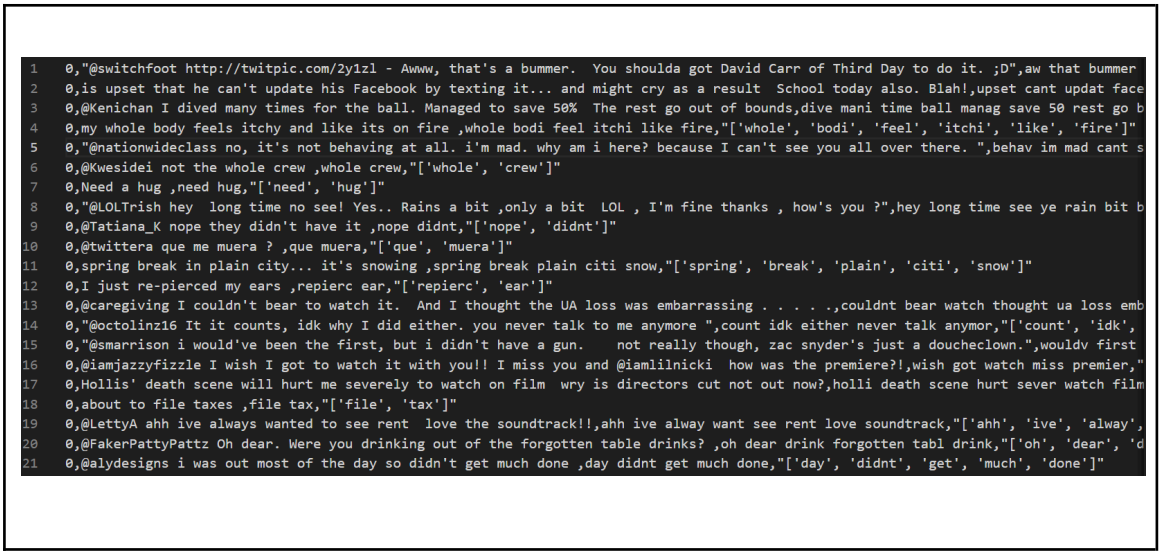
```
from nltk.tokenize import TweetTokenizer
import numpy as np
# Initialiser le tokenizer
tknzr = TweetTokenizer()

text_clean = dataframe[1].apply(clean_tweet)
dataframe = pd.DataFrame(np.column_stack([dataframe, text_clean]), columns=['target', 'text_tweet', 'text_clean'])

# Enregistrer le DataFrame dans un fichier CSV
dataframe.to_csv('cleaned_data.csv', index=False)

# Appliquer la tokenisation sur la deuxième colonne (index 1)
tokens_column = text_clean.apply(tknzr.tokenize)

# Utiliser np.column_stack() pour empiler les colonnes contenant les tokens
dataframe = pd.DataFrame(np.column_stack([dataframe, tokens_column]), columns=['target', 'text_tweet', 'text_clean', 'tokens'])

# Afficher les premières lignes pour vérification
dataframe.head()

# Enregistrer le DataFrame dans un fichier CSV
dataframe.to_csv('tokenize_data.csv', index=False, header=False)
✓ 15m 51.0s
```

l'image suivante montre la structure du fichier cleaned_data.csv qui est le résultat de l'application de la méthode **clean_tweet**. il contient le target, le tweet et le tweet nettoyé:



```
:ext_tweet,text_clean
:chfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D",aw that bummer
:et that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!,upset cant updat fac
:han I dived many times for the ball. Managed to save 50%  The rest go out of bounds,dive mani time ball manag save 50 rest go
)le body feels itchy and like its on fire ,whole bodi feel itchi like fire
:onwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there. ",behav im mad cant
:dei not the whole crew ,whole crew
) hug ,need hug
'rish hey  long time no see! Yes.. Rains a bit ,only a bit  LOL , I'm fine thanks , how's you ?",hey long time see ye rain bit
)na_K nope they didn't have it ,nope didnt
:era que me muera ? ,que muera
; break in plain city... it's snowing ,spring break plain citi snow
: re-pierced my ears ,repierc ear
:iving I couldn't bear to watch it.  And I thought the UA loss was embarrassing . . . . .,couldnt bear watch thought ua loss em
```

l'image suivante montre la structure du fichier tokenize_data.csv qui est le résultat de la tokenization des tweets nettoyés. il contient le target, le tweet, le tweet nettoyé, et la liste des tokens :



```
0,"@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D",aw that bummer
0,is upset that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!,upset cant updat face
0,@Kenichan I dived many times for the ball. Managed to save 50%  The rest go out of bounds,dive mani time ball manag save 50 rest go b
0,my whole body feels itchy and like its on fire ,whole bodi feel itchi like fire,"['whole', 'bodi', 'feel', 'itchi', 'like', 'fire']"
0,"@nationwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there. ",behav im mad cant s
0,@Kwesidei not the whole crew ,whole crew,"['whole', 'crew']"
0,Need a hug ,need hug,"['need', 'hug']"
0,"@LOLTrish hey  long time no see! Yes.. Rains a bit ,only a bit  LOL , I'm fine thanks , how's you ?",hey long time see ye rain bit b
0,@Tatiana_K nope they didn't have it ,nope didnt,"['nope', 'didnt']"
0,@twittera que me muera ? ,que muera,"['que', 'muera']"
0,spring break in plain city... it's snowing ,spring break plain citi snow,"['spring', 'break', 'plain', 'citi', 'snow']"
0,I just re-pierced my ears ,repierc ear,"['repierc', 'ear']"
0,@caregiving I couldn't bear to watch it.  And I thought the UA loss was embarrassing . . . . .,couldnt bear watch thought ua loss emb
0,"@octolinz16 It it counts, idk why I did either. you never talk to me anymore ",count idk either never talk anymor,"['count', 'idk',
0,"@smarrison i would've been the first, but i didn't have a gun.    not really though, zac snyder's just a doucheclown.",wouldv first
0,@iamjazzyfizzle I wish I got to watch it with you!! I miss you and @iamlilnicki  how was the premiere?!,wish got watch miss premier,"
0,Hollis' death scene will hurt me severely to watch on film  wry is directors cut not out now?,holli death scene hurt sever watch film
0,about to file taxes ,file tax,"['file', 'tax']"
0,@LettyA ahh ive always wanted to see rent  love the soundtrack!!,ahh ive alway want see rent love soundtrack,"['ahh', 'ive', 'alway',
0,@FakerPattyPattz Oh dear. Were you drinking out of the forgotten table drinks? ,oh dear drink forgotten tabl drink,"['oh', 'dear', 'd
0,@alydesigns i was out most of the day so didn't get much done ,day didnt get much done,"['day', 'didnt', 'get', 'much', 'done']"
```

Deuxième étape:

1. Extraction du vocabulaire :

Après la phase du prétraitement viens l'extraction du vocabulaire à partir de tout le dataset, cette phase est implémenté dans la méthode suivante

Choix de bibliothèque
la seule bibliothèque manipuler dans cette étape est la classe Counter de la bibliothèque Collections

Collections est une partie de la bibliothèque standard de Python qui fournit des structures de données spécialisées en plus des types de données intégrés de Python.
Parmi les classes les plus utilisées de *collections*, il y a *Counter,* qui est une sous-classe de dictionnaire que nous avons utilisée pour compter les occurrences des éléments dans une séquence

Code de la construction du vocabulaire :

```python
#Construction de vocabulaire
from collections import Counter
def create_vocab():
    # Compter le nombre d'occurrences de chaque mot dans l'ensemble des tweets
    word_counts = Counter()

    for tokens in dataframe['tokens']:
        word_counts.update(tokens)

    # Filtrer les mots qui apparaissent au moins K fois dans le corpus de tweets
    K = 2000  # Choisir empiriquement K
    vocab = [word for word, count in word_counts.items() if count >= K]

    # Enregistrer les mots sélectionnés dans un fichier vocab.txt
    with open('vocab.txt', 'w', encoding='utf-8') as f:
        for word in vocab:
            f.write(word + '\n')

    # Mapper chaque mot prétraité à son index dans la liste de vocabulaire
    word_to_index = {word: idx for idx, word in enumerate(vocab)}

    # Afficher quelques mots du vocabulaire pour vérification
    print("Quelques mots du vocabulaire : ", vocab[:10])

    return word_to_index
word_to_index = create_vocab()
```

Explications :

Tout d'abord, elle importe la classe Counter de la bibliothèque collections, qui sera utilisée pour compter le nombre d'occurrences de chaque mot dans le dataset. Ensuite, elle charge un fichier CSV prétraité contenant les données dans un DataFrame pandas. À partir de ce DataFrame, elle récupère la dernière colonne qui contient les tokens des tweets. La fonction commence par compter le nombre d'occurrences de chaque mot dans l'ensemble des tweets à l'aide de la classe Counter. Ensuite, elle filtre les mots qui apparaissent au moins un certain nombre de fois, défini empiriquement comme K (qu'on a fixé à 2000 pour avoir un fichier vocab peu volumineux), dans l'ensemble des tweets. Les mots sélectionnés sont stockés dans une liste appelée vocab. Cette liste est ensuite enregistrée dans un fichier texte nommé 'vocab.txt', où chaque mot est écrit sur une nouvelle ligne. En

parallèle, la fonction crée un dictionnaire nommé word_to_index qui mappe chaque mot du vocabulaire à son index dans la liste vocab. Enfin, quelques mots du vocabulaire sont affichés à des fins de vérification. Cette fonction renvoie le dictionnaire word_to_index, qui est utilisé pour représenter chaque mot par un entier unique dans le vocabulaire, facilitant ainsi la manipulation et l'analyse ultérieure des données.

l'image suivante montre quelque vocabulaire extrait par cette fonction

```
text
might
cri
school
today
also
mani
time
manag
save
rest
go
whole
bodi
feel
like
im
mad
see
need
hug
hey
long
ye
rain
bit
lol
fine
thank
how
```

Troisième étape :

1. Représentation binaire des caractéristiques :

Code de l'extraction des caractéristiques :

```python
def extract_features_binary_batch(dataframe, word_to_index, batch_size=1000):
    num_rows = len(dataframe)
    num_batches = (num_rows + batch_size - 1) // batch_size

    features = []
    for i in range(num_batches):
        start_idx = i * batch_size
        end_idx = min((i + 1) * batch_size, num_rows)
        batch_df = dataframe.iloc[start_idx:end_idx]
        batch_features = []

        for tokens in batch_df['tokens']:
            word_features = [1 if word in tokens else 0 for word in word_to_index]
            batch_features.append(word_features)

        features.extend(batch_features)

    return features
```

Explication :

La fonction **extract_features_binary_batch** prend en entrée un dataframe, un dictionnaire word_to_index contenant les mots et leurs indices correspondants, et une taille de lot (batch_size) pour traiter les données par lots.

Elle divise les données du dataframe en lots de taille batch_size, puis parcourt chaque lot pour extraire les caractéristiques binaires.

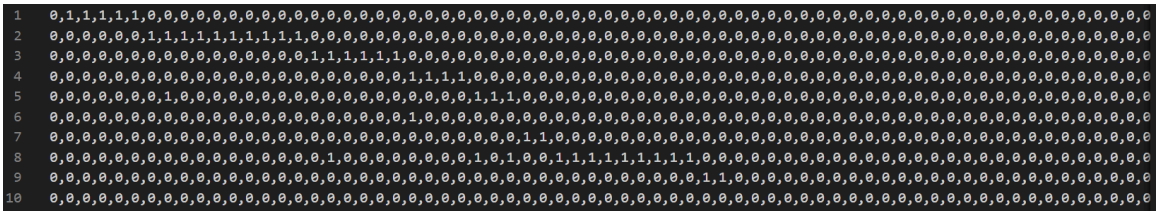
Les caractéristiques binaires sont représentées par la présence ou l'absence de chaque mot du dictionnaire word_to_index dans les tokens de chaque tweet. Si un mot est présent dans les tokens du tweet, sa caractéristique binaire est définie sur 1 ; sinon, elle est définie sur 0.

Les caractéristiques binaires de chaque lot sont ensuite étendues à la liste features, qui est retournée à la fin.

Exécution :

```python
# Utilisation de la fonction pour extraire les caractéristiques binaires
features_binary = extract_features_binary_batch(dataframe, word_to_index)

# Créer un DataFrame avec les cibles et les caractéristiques binaires
data_binary = pd.DataFrame(features_binary, columns=[f'feat_{i}' for i in range(len(word_to_index))])
data_binary.insert(0, 'target', dataframe['target'])

# Enregistrer le DataFrame dans un fichier CSV
data_binary.to_csv('features_binary.csv', index=False, header=False)
```

l'image suivante montre la structure du fichier features_binary.csv qui est le résultat de l'extraction binaire.



2. Représentation par comptage des caractéristiques :

Code de l'extraction des caractéristiques :

La fonction ***extract_features_count_batch*** est conçue pour extraire des caractéristiques de comptage de mots à partir d'un DataFrame contenant des tweets, en les traitant par lots (batches) afin de gérer la mémoire de manière efficace.L'explication détaillée de chaque étape et instruction de la methode:

```python
def extract_features_count_batch(dataframe, word_to_index, batch_size=1000):
    num_samples = len(dataframe)
    num_batches = (num_samples + batch_size - 1) // batch_size

    # Parcourir chaque lot (batch) de tweets
    for batch_idx in range(num_batches):
        start_idx = batch_idx * batch_size
        end_idx = min((batch_idx + 1) * batch_size, num_samples)

        # Initialiser un tableau pour stocker les caractéristiques par comptage du lot actuel
        batch_features = np.zeros((end_idx - start_idx, len(word_to_index)), dtype=int)

        # Parcourir chaque tweet du lot
        for i, tokens in enumerate(dataframe['tokens'].iloc[start_idx:end_idx]):
            # Compter le nombre d'apparitions de chaque mot dans le tweet
            for word in tokens:
                if word in word_to_index:
                    word_index = word_to_index[word]
                    batch_features[i, word_index] += 1

        # Créer un DataFrame pour le lot actuel
        batch_df = pd.DataFrame(batch_features, columns=[f'count_{i}' for i in range(len(word_to_index))])
        batch_df.insert(0, 'target', dataframe['target'].iloc[start_idx:end_idx].values)
```

```python
        # Écrire le lot actuel dans un fichier CSV (mode append)
        if batch_idx == 0:
            batch_df.to_csv('features_count.csv', index=False, header=False)
        else:
            batch_df.to_csv('features_count.csv', index=False, header=False, mode='a')

        # Libérer la mémoire du lot actuel
        del batch_features
        del batch_df

batch_size = 1000

# Utilisation de la fonction pour extraire les caractéristiques par comptage par lots
extract_features_count_batch(dataframe, word_to_index, batch_size=batch_size)
```

Elle prend en entrée DataFrame contenant les tweets (dataframe), un dictionnaire associant chaque mot à un index (word_to_index), et une taille de lot (batch_size), par défaut fixée à 1000. Elle extrait les caractéristiques de comptage de mots des tweets et les écrit dans un fichier CSV par lots.

Nombre d'Échantillons : num_samples est le nombre total de tweets dans le DataFrame.

Nombre de Lots : num_batches est calculé pour déterminer combien de lots sont nécessaires pour traiter tous les échantillons, en arrondissant vers le haut si nécessaire.

Boucle sur les Lots : La boucle for itère sur chaque lot en utilisant batch_idx.

Indices de début et de fin : start_idx et end_idx déterminent les indices de début et de fin pour le lot actuel. end_idx est limité au nombre total d'échantillons pour éviter de dépasser les limites du DataFrame.

Tableau de Comptage : batch_features est un tableau de zéros de taille (nombre de tweets dans le lot, nombre de mots dans le dictionnaire). Il stockera le nombre d'occurrences de chaque mot pour chaque tweet du lot.

Boucle sur les Tweets : La boucle for parcourir chaque tweet du lot.
Boucle sur les Mots : La boucle interne parcourt chaque mot dans les tokens du tweet.
Comptage des Mots : Si le mot est dans word_to_index, son index est récupéré, et le compteur correspondant dans batch_features est incrémenté.

DataFrame de Caractéristiques : batch_features est converti en Data Frame avec des colonnes nommées count_0, count_1, etc., correspondant aux indices des mots.

Ajout de la Colonne Target : La colonne target est insérée au début du DataFrame pour associer les cibles des tweets aux caractéristiques extraites.

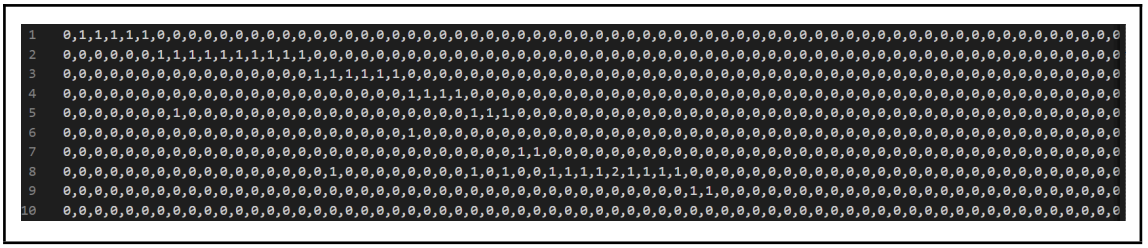
Écriture du Premier Lot : Si c'est le premier lot (batch_idx == 0), le DataFrame est écrit dans un fichier CSV sans index et sans en-têtes.

Écriture des Lots Suivants : Pour les lots suivants, les données sont ajoutées (mode='a') au fichier existant.

Suppression des Objets : Les objets batch_features et batch_df sont supprimés pour libérer la mémoire utilisée par le lot actuel avant de passer au suivant.

En résumé *extract_features_count_batch* traite un DataFrame de tweets en extrayant les caractéristiques de comptage de mots par lots, afin de gérer efficacement la mémoire. Pour chaque lot de tweets, elle compte les occurrences de mots définis dans un dictionnaire, stocke ces comptes dans un tableau, convertit ce tableau en DataFrame, et écrit les résultats dans un fichier CSV. Cette approche par lots permet de traiter de grands ensembles de données sans surcharger la mémoire disponible.

l’image suivante montre la structure du fichier features_count.csv qui est le résultat de l’extraction par comptage.



Quatrième étape :

Classification:

Définition:

Les Bibliothèques utilisé pour la classification:

```
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.neighbors import KNeighborsClassifier
```

train_test_split de sklearn.model_selection:

On l'utilise pour diviser un ensemble de données en deux sous-ensembles : un ensemble d'entraînement et un ensemble de test.

svm de sklearn:

On utilise svm.SVC pour créer et entraîner un modèle SVM pour la classification des sentiments des tweets. Les SVMs sont particulièrement utiles pour les problèmes de classification binaire.

confusion_matrix de sklearn.metrics:

La matrice de confusion nous montre le nombre de prédictions correctes et incorrectes, ventilées par chaque classe. Dans le contexte de classification des sentiments, elle nous permet de voir combien de tweets positifs et négatifs ont été correctement ou incorrectement classés.

LogisticRegression de sklearn.linear_model:

La régression logistique est couramment utilisée pour la classification des sentiments des tweets en raison de sa simplicité et de son efficacité. Elle nous permet de modéliser la probabilité qu'un tweet appartienne à une classe de sentiment spécifique.

accuracy_score de sklearn.metrics:

métrique qui nous donne une mesure globale de la performance du modèle, indiquant combien de tweets ont été correctement classés par rapport au total.

recall_score de sklearn.metrics:

On l' utilise pour le calcul du score de rappel, qui est la proportion de vrais positifs parmi les vrais positifs et les faux négatifs.
Le rappel est crucial lorsqu'il est important de minimiser les faux négatifs, par exemple, pour s'assurer que les tweets négatifs sont correctement identifiés.

precision_score de sklearn.metrics:

On l'utilise pour le calcul du score de précision, c'est-à-dire la proportion de vrais positifs parmi les vrais positifs et les faux positifs.
La précision est importante lorsque les faux positifs ont un coût élevé, par exemple, pour s'assurer que les tweets classés comme positifs le sont réellement.

KNeighborsClassifier de sklearn.neighbors:

Pour implémenter l'algorithme des k-plus proches voisins (k-NN) pour les tâches de classification.

Avant d'appliquer les algorithmes de classification qu'on a appris on doit d'abord:

- Charger les données (Résultat de la représentation binaire):

```
#Chargement de données (Résultat de la représentation binaire)
data_features_binary = pd.read_csv('features_binary.csv', encoding='latin1', header=None)
data_features_binary.head()
```

	0	1	2	3	4	5	6	7	8	9	...	842	843	844	845	846	847	848	849	850	851
0	0	1	1	1	1	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 852 columns

- on a visualiser combien de tweet a trajet 4 et a target 0:

```
data_features_binary[0].value_counts()
```

```
0
0    800000
4    800000
Name: count, dtype: int64
```

- Puis nous avons convertis tous les tweets de target 4 a 1:

```
data_features_binary[0] = data_features_binary[0].replace(4,1)
```

```
data_features_binary[0].value_counts()
```

```
0
0    800000
1    800000
Name: count, dtype: int64
```

- diviser les caractéristique en des données d'entraînement et de tests:

```
# Sélectionner toutes les colonnes sauf la première
X_Binary = data_features_binary.iloc[:, 1:]

# Sélectionner la première colonne
Y_Binary = data_features_binary.iloc[:, 0]
```

```
X_Binary.shape, Y_Binary.shape
```

```
((1600000, 851), (1600000,))
```

```
# Séparation des données en ensemble d'entraînement et ensemble de test
X_train, X_test, y_train, y_test = train_test_split(X_Binary, Y_Binary, test_size=0.3, stratify=Y_Binary, random_state=2)
```

classification avec KNN:

Définition:

L'algorithme des K plus proches voisins ou K-nearest neighbors (kNN) est un algorithme de Machine Learning qui appartient à la classe des algorithmes d'apprentissage supervisé simple et facile à mettre en œuvre qui peut être utilisé pour résoudre les problèmes de classification et de régression.[1]

1. **Nous créons un modèle k-NN avec k=3 pour**

trouver les voisins les plus proches de manière efficace. Ensuite, nous entraînons le modèle sur l'ensemble d'entraînement.

Création du model KNN:

```
# Créer un modèle KNN avec k=5 (vous pouvez ajuster ce paramètre)
knn_model = KNeighborsClassifier(n_neighbors=3)
```

Entrainement du modèle sur l'ensemble des données d'entraînement:

```
# Entraîner le modèle sur l'ensemble d'entraînement
knn_model.fit(X_train, y_train)
```

Effectuer la prédiction sur l'ensemble de test:

```
# Prédiction sur l'ensemble de test
y_pred = knn_model.predict(X_test)
```

2. Calcul des Métriques de Performance

Pour évaluer les performances du modèle k-NN, nous calculons diverses métriques telles que la précision, le rappel, et la précision. Nous utilisons également la matrice de confusion pour visualiser les performances du modèle par classe.[2]

Calculer la matrice de confusion:

```
# Calculer la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

On a calculé l'accuracy:

L'accuracy permet de connaître la proportion de bonnes prédictions par rapport à toutes les prédictions. L'opération est simplement : Nombre de bonnes prédictions / Nombre total de prédictions[2]

Formule:

$$\text{Accuracy} = \frac{tn + tp}{tn + fp + fn + tp}$$

```
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

On a calculé le recall:

```
recall = recall_score(y_test, y_pred)
print(recall)
```

On a calculé le rappel:

Le rappel correspond au nombre de documents **correctement** attribués à la classe i par rapport au nombre total de documents **appartenant** à la classe i (total true positive).[2]

La formule:

$$\text{Recall} = \frac{tp}{tp + fn}$$

```
recall = recall_score(y_test, y_pred)
print(recall)
```

On a calculé la précision:

La précision correspond au nombre de documents **correctement** attribués à la classe i par rapport au nombre total de documents **prédits** comme appartenant à la classe i (total **predicted** positive).[2]

La Formule:

$$\text{Precision} = \frac{tp}{tp + fp}$$

```
precision = precision_score(y_test, y_pred)
print(precision)
```

classification avec Régression Logistique:

Définition:

La régression logistique est un modèle statistique permettant d'étudier les relations entre un ensemble de variables qualitatives Xi et une variable qualitative Y. Il s'agit d'un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien.

Un modèle de régression logistique permet aussi de prédire la probabilité qu'un événement arrive (valeur de 1) ou non (valeur de 0) à partir de l'optimisation des coefficients de régression

Création du model Régression Logistique:

```
# Initialisation du modèle de régression logistique avec régularisation L2 (Ridge)
# et un nombre maximal d'itérations de 10000
model = LogisticRegression(penalty="l2", max_iter=1000)
```

Entrainement du modèle sur l'ensemble des données d'entraînement:

```
#Entraîne le modèle sur l'ensemble d'entraînement
model.fit(X_train, y_train)
```

Effectuer la prédiction sur l'ensemble de test:

```
# Prédiction sur l'ensemble de test
y_pred = model.predict(X_test)
```

Calculer la matrice de confusion:

```
# Calculer la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

On a calculé l'accuracy:

```
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
0.7503979166666667
```

On a calculé le recall:

```
recall = recall_score(y_test, y_pred)
print(recall)
```

```
0.7952916666666666
```

On a calculé la précision:

```
precision = precision_score(y_test, y_pred)
print(precision)
```

```
0.7297676534798451
```

Conclusion:

Dans cette étude, nous avons comparé les performances de K-Nearest Neighbors (KNN) et de la régression logistique pour l'analyse des sentiments des tweets. Voici les points clés :

1. **Efficacité Computationnelle :**
   - KNN : Lent et impraticable pour de grands ensembles de données (1 120 000 lignes, 851 caractéristiques) en raison de la complexité de calcul des distances.
   - Régression Logistique : Rapide et scalable, adaptée aux grands ensembles de données.

2. **Performance Prédictive :** Les deux modèles ont des performances comparables en termes de précision et de rappel.

3. **Simplicité et Interprétabilité :** La régression logistique est plus interprétable et offre une meilleure compréhension de l'impact des caractéristiques sur les prédictions.

Choix du Modèle : La régression logistique est recommandée pour l'analyse des sentiments des tweets en raison de sa rapidité, de son évolutivité et de son interprétabilité, surpassant KNN pour des ensembles de données volumineux et complexes.

Références

[1]

https://datascientest.com/knn

[2]

https://beranger.medium.com/ml-accuracy-pr%C3%A9cision-f1-score-courbe-roc-que-choisir-5d4940b854d7