Traitement du langage naturel (NLP):

Qu’est-ce que la NLP ?

NLP est une branche de l'intelligence artificielle qui permet de comprendre et de traiter automatiquement les langages humain

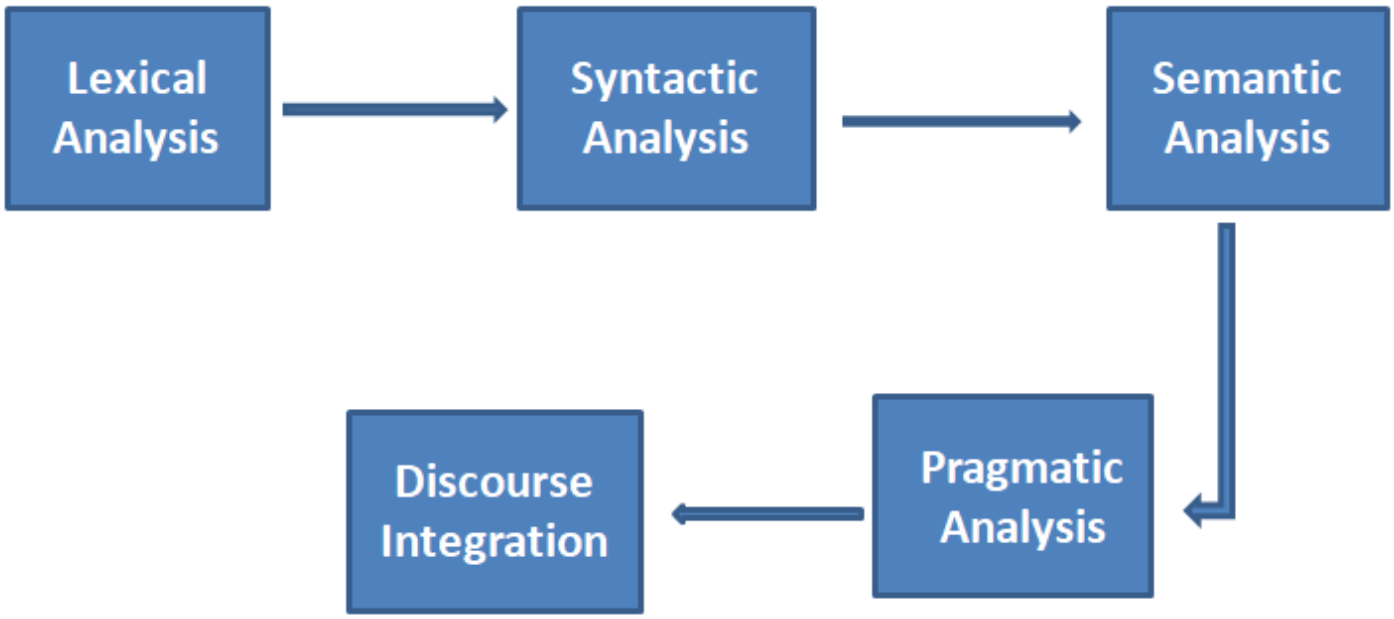
Avantages de NLP:

- Analyser des données provenant de sources structurées et non structurées
- Rapide et efficace
- Fournir des réponses précises et complètes à la question du début à la fin.

Applications de la NLP:

- Traduction automatique : Traduire automatiquement du texte d'une langue à une autre. Exemple : **Google Traduction**.
- Analyse des sentiments: Evaluer si un élément de contenu est positif, négatif ou neutre
- Chatbot

Phases du Traitement du Langage Naturel:



Lexical or **Morphological** Analysis:

- Première étape du traitement du langage naturel.
- Analyser les structures des mots.
- Décomposition d'un fichier texte en paragraphes, phrases et mots.

Syntactic Analysis or **Parsing**:

- Garantir qu'un morceau de texte donné possède une structure correcte.
- Analyser la phrase pour vérifier la grammaire correcte au niveau de la phrase.

Semantic Analysis:

- Chercher le sens dans la phrase donnée.
- Traiter la combinaison des mots en phrases.

Discourse Analysis:

- La signification d'une phrase dépend de celle qui la précède immédiatement, et elle établit également le sens de la phrase suivante**.**
- Les phrases précédentes influent sur cette intégration du discours**.**

Exemple: Marie est une athlète talentueuse .Elle s'entraîne tous les jours au gymnase.

La première phrase présente **Marie** comme une athlète talentueuse. La deuxième phrase utilise "**elle**" pour se référer à **Marie**, établie dans la **phrase précédente**.

Pragmatic Analysis:

- La dernière étape du NLP
- Interpréter le texte donné en utilisant les informations des étapes précédentes.

Bibliothèques PNL:

- NLTK
- Spacy
- Gensim
- fastText
- Stanford toolkit (Glove)
- Apache OpenNLP

Approches classiques

Approches classiques du traitement du langage naturel:

- Prétraitement du texte:

++ La tokenisation : divise le texte en parties plus petites pour faciliter l'analyse automatique

- **Tokenisation par mot.** Cette méthode divise le texte en mots individuels

  Exemple:

  Texte d'entrée : "La belle journée ensoleillée."

  Résultat de la tokenisation : ["La", "belle", "journée", "ensoleillée."]

- **Tokenisation par caractère**: le texte est segmenté en caractères individuels

  Exemple:

  Texte d'entrée : "Bonjour"

  Résultat de la tokenisation : ["B", "o", "n", "j", "o", "u", "r"]

- **Tokenisation par sous-mot:** divise le texte en unités qui peuvent être plus grandes qu'un seul caractère mais plus petites qu'un mot complet

  Exemple:

  Texte d'entrée : "Château"

  Résultat de la tokenisation : ["Ch", "â", "teau"]

++ Vectoristaion: est le processus de conversion de texte en vecteurs numériques afin de permettre aux algorithmes de machine learning de comprendre et de traiter le langage humain

- **Bag of words (BOW):** transforme le texte en une représentation comptabilisant l'occurrence des mots les plus fréquemment rencontrés.
- **Term Frequency-Inverse Document Frequency** (**TF-IDF**): attribue un poids à chaque mot en fonction de sa fréquence dans un document spécifique et de sa fréquence dans l'ensemble du corpus
- **N-gram :** une sous-séquence de n éléments construite à partir d'une séquence

donnée

exemple: Le chat dort dans le jardin

- **Uni-gram**: ["Le", "chat", "dort", "dans", "le", "jardin"]
- **Bi-gram**: ["Le chat", "chat dort", "dort dans", "dans le", "le jardin"]
- **Tri-gram:** ["Le chat dort", "chat dort dans", "dort dans le", "dans le jardin"]

++ Lemmatization: représente les mots sous leur forme canonique, appelée **lemme.**

exemple : rapidement => rapide

++ Stemming(racinisation): réduit les mots à leur racine en supprimant les suffixes, préfixes

et autres afin de ne conserver que leur origine

exemple : marchant => march

Après avoir exploré les concepts clés du traitement du langage naturel, nous plongerons dans l'**analyse des sentiments** sur les **tweets**. Cette approche nous permettra d'évaluer les opinions et les émotions exprimées dans les tweets, offrant ainsi des insights précieux pour comprendre les tendances et les sentiments des utilisateurs sur les réseaux sociaux. En utilisant des méthodes d'analyse automatisée, nous pourrons extraire des informations significatives des tweets pour mieux comprendre les attitudes et les perceptions des utilisateurs.

Voici une simple implémentation étape par étape pour l'**analyse des sentiments sur Twitter** :

Introduction:

dans ce travail vous découvrirez l'analyse des sentiments sur Twitter à l'aide de Sklearn et NLTK

Scikit-learn (**sklearn**) est une bibliothèque open-source en Python qui propose :

- Une large gamme d'algorithmes pour la classification, la régression et le clustering ;
- Des utilitaires pour le prétraitement des données, la validation croisée et l'évaluation des modèles.

NLTK (**Natural Language Toolkit**) est une bibliothèque Python populaire pour le traitement du langage naturel, offrant :

- Une gamme d'outils et de ressources pour le traitement de texte ;
- L'analyse de sentiments et d'autres tâches linguistiques.

Le dataset utilisé est disponible sur Kaggle et contient 1,6 million de tweets.
Les tweets **négatifs** sont étiquetés avec 0 et les tweets **positifs** avec 4.

Importation des bibliothéques:

```
# importing some required libraries
import pandas as pd
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import string
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
nltk.download('stopwords')
```

Importation de l’ensemble de données:

Avec pandas, je charge l'ensemble de données du fichier CSV dans le dataframe. J'ai simplement renommé la colonne "target" en "sentiment" car cela décrit mieux les données**.**

```
DATASET_ENCODING = "ISO-8859-1"
DATASET_COLUMNS = ["sentiment", "id", "date", "query", "user_id", "text"]
df= pd.read_csv('../input/sentiment140/training.1600000.processed.noemoticon.csv', encoding=DATASET_ENCODING, names=DATASET_COLUMNS)
df.head()
```

J'ai supprimé les colonnes "id", "date", "query" et "user_id" car elles ne contiennent pas d'informations pertinentes pour l'analyse de sentiment. En revanche, j'ai conservé les colonnes "text" et "sentiment", car elles représentent respectivement le contenu des tweets et les étiquettes de sentiment, qui sont essentielles pour l'entraînement et l'évaluation du modèle de sentiment analysis.

```
# Removing the unnecessary columns.
df = df.drop(["id", "date", "query", "user_id"], axis=1)
```

les avis positifs sont marqués comme étant 4, je les ai changés en 1, et les avis négatifs sont représentés par 0

```
# Postive review is marked as 4, changing it to 1 and Negative review is 0.

df['sentiment'] = df['sentiment'].replace(4,1)
```

Visualisation des données.

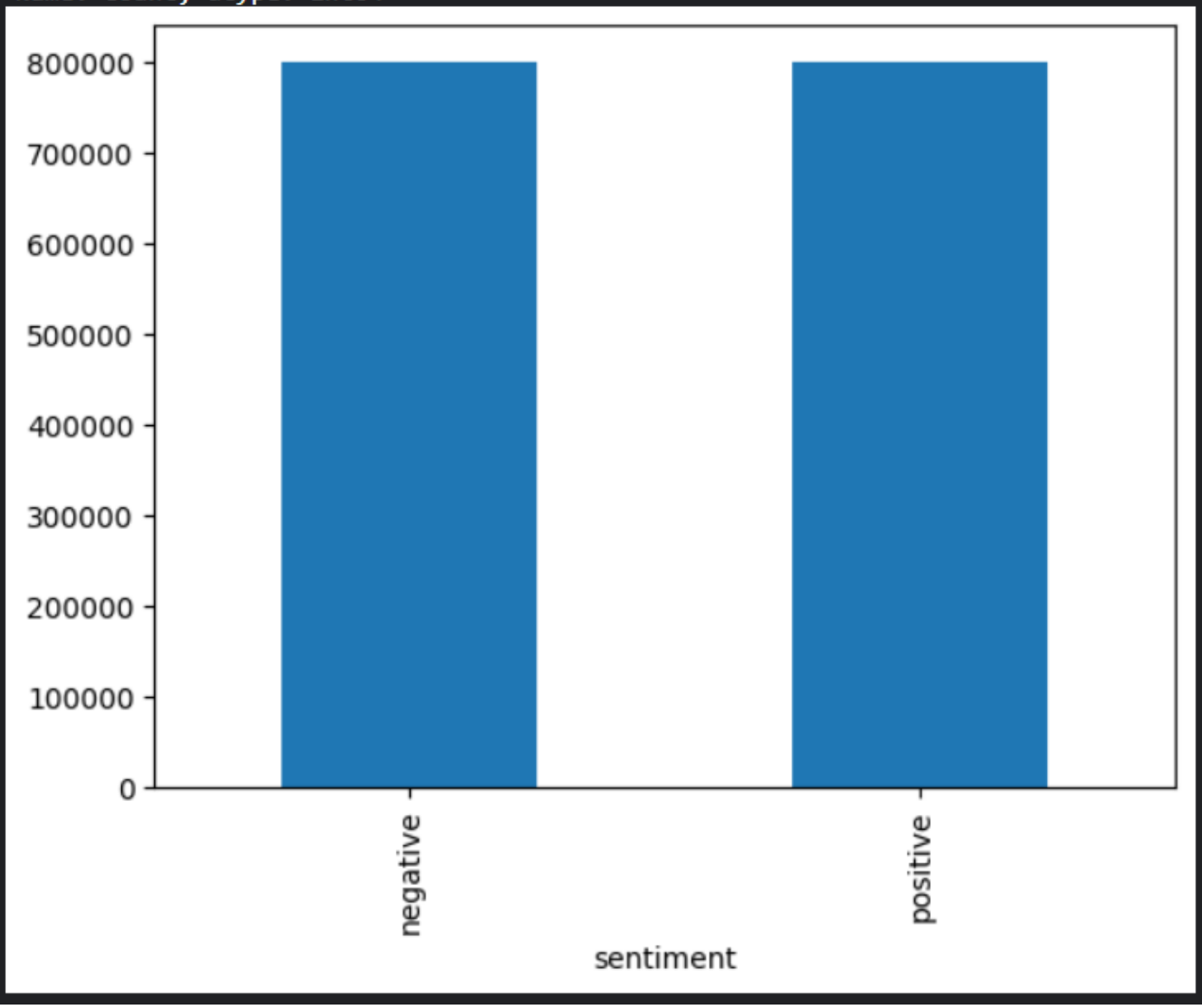
```
# Visualising the distribution of the sentiment variable.

sentiment = {0:"negative", 1:"positive"}
print(df.sentiment.apply(lambda x: sentiment[x]).value_counts())
df.sentiment.apply(lambda x: sentiment[x]).value_counts().plot(kind = 'bar')
plt.show()
```

output:

Comme le montre la figure, le jeu de données présente une distribution équilibrée entre les sentiments positifs et négatifs, avec 800 000 tweets exprimant chaque sentiment. Cette égalité dans les étiquettes de sentiment suggère une répartition uniforme des opinions exprimées dans les tweets entre les deux catégories, ce qui peut être avantageux pour l'entraînement des modèles d'apprentissage automatique. En ayant un nombre égal de données pour chaque classe de sentiment, les modèles peuvent être plus robustes et généralisés, ce qui conduit à des performances plus fiables lors de la prédiction des sentiments pour de nouveaux tweets.

```
sentiment
negative    800000
positive    800000
Name: count, dtype: int64
```



Prétraitement des données:

Ce code définit trois éléments clés pour le prétraitement des données dans le cadre de l'analyse de texte :
Il crée une liste de mots vides (**stopwords**) (exp: "the",”and”,”is”) en anglais à partir de la bibliothèque NLTK. Les mots vides sont des mots très courants qui ne portent généralement pas de sens dans une analyse de texte et peuvent être ignorés.

Le **stemmer** est utilisé pour **réduire** les mots à leur forme racine, ce qui permet de normaliser le texte et de **réduire la dimensionnalité** de l'espace de caractéristiques.

L'**expression régulière** (**regex**) identifie et supprime les **liens URL** (commençant par "http://" ou "https://") ainsi que les caractères spéciaux autres que les lettres et les chiffres.

```
stopWords = stopwords.words('english')
stemmer = SnowballStemmer('english')

text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"
```

cette fonction nettoie des liens URL et des caractères spéciaux, puis le divise en mots individuels (tokens), en supprimant les mots vides, avant de renvoyer le texte prétraité.

```
def preprocess(text):
    text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
    tokens = []
    for token in text.split():
        if token not in stopWords:
            tokens.append(token)
    return " ".join(tokens)
```

Cette ligne de code applique la fonction preprocess que nous avons définie précédemment à la colonne 'text' du DataFrame 'df'. Cela signifie qu'elle va nettoyer et prétraiter le texte de chaque tweet dans la colonne 'text' en utilisant la fonction preprocess, et stocker le résultat dans la même colonne 'text'.

```
df['text'] = df['text'].map(preprocess)
```

Division des données:

J'ai divisé les données de telle sorte que 80% sont utilisées pour l'entraînement (X_train et y_train), tandis que les 20% restants sont réservés pour les tests (X_test et y_test)

```
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['sentiment'],
                                                    test_size = 0.2, random_state = 0)
```

TF-IDF (Term Frequency-Inverse Document Frequency) :

La vectorisation **TF-IDF** permet de convertir un corpus de documents en une matrice numérique où chaque document est représenté par un vecteur contenant des scores TF-IDF pour chaque terme du vocabulaire.

```
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
```

```
#transforming the dataset into matrix of TF-IDF Features by using the TF-IDF Vectoriser
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

Construction du modèle:

```
# Train the Naive Bayes classifier
naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train, y_train)
# Predict with the Naive Bayes classifier on the test set
naive_bayes_predictions = naive_bayes_classifier.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, naive_bayes_predictions)
print("Accuracy:", accuracy)

# Calculate the F1 score
f1 = f1_score(y_test, naive_bayes_predictions)
print("F1 Score:", f1)

# Compute and plot the Confusion matrix
conf_matrix = confusion_matrix(y_test, naive_bayes_predictions)
categories  = ['Negative','Positive']
group_names = ['TN','FP', 'FN','TP']
group_percentages = ['{0:.2%}'.format(value) for value in conf_matrix.flatten() / np.sum(conf_matrix)]

labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
labels = np.asarray(labels).reshape(2,2)

sns.heatmap(conf_matrix, annot = labels, cmap = 'Reds',fmt = '',
               xticklabels = categories, yticklabels = categories)

plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
plt.ylabel("Actual values"   , fontdict = {'size':14}, labelpad = 10)
plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Le modèle utilisé est le classificateur Naive Bayes .Ce modèle a une précision de 77,83% et un score F1 de 77,80%

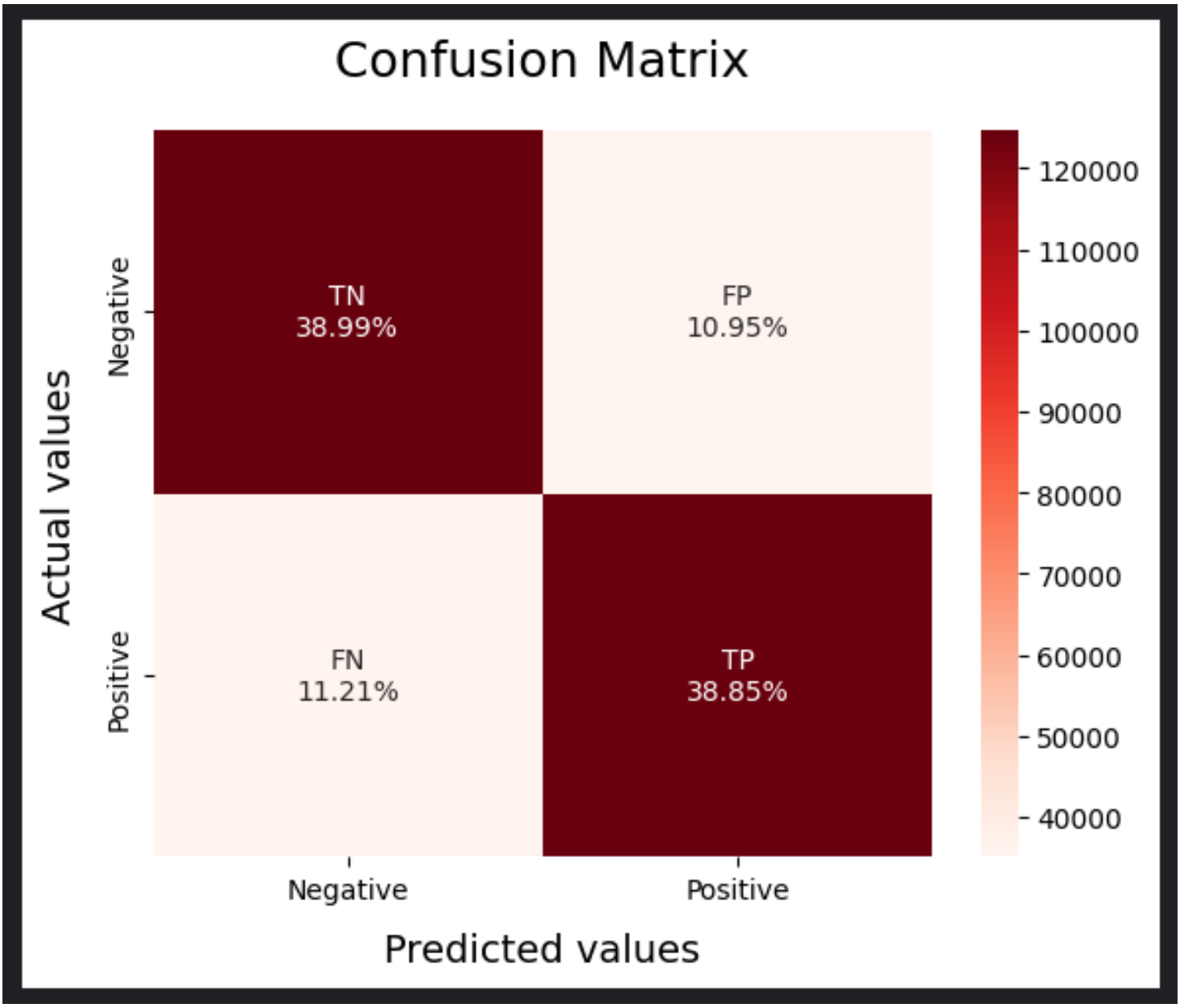
Dans la matrice de confusion :

TN (**True Negative**) représente la proportion des échantillons négatifs correctement classés. Dans ce cas, 38,99 % des échantillons négatifs ont été correctement classés comme négatifs.

TP (**True Positive**) représente la proportion des échantillons positifs correctement classés. Ici, 38,85 % des échantillons positifs ont été correctement classés comme positifs.

FP (**False Positive**) représente la proportion des échantillons négatifs incorrectement classés comme positifs. Nous avons 10,95 % des échantillons négatifs qui ont été incorrectement classés comme positifs.

FN (**False Negative**) représente la proportion des échantillons positifs incorrectement classés comme négatifs. Nous avons 11,21 % des échantillons positifs qui ont été incorrectement classés comme négatifs



En conclusion, le modèle Naive Bayes semble capable de fournir des prédictions acceptables pour la tâche de classification des sentiments des tweets. Cependant, il pourrait bénéficier d'une amélioration afin de réduire les erreurs de classification. L'exploration de l'utilisation d'autres modèles d'apprentissage automatique et d'apprentissage profond, comme les réseaux neuronaux récurrents (RNN), pourrait être une avenue prometteuse pour améliorer davantage les performances du modèle

Vous pouvez trouver le code dans mon référentiel GitHub

Sana Arfaoui