



الكلية متعددة التخصصات الناحور

ⵜⴰⵎⵓⵔⵜ ⵜⴰⵎⵓⵔⵜ ⵜⴰⵎⵓⵔⵜ ⵜⴰⵎⵓⵔⵜ

Faculté Pluridisciplinaire de Nador

RAPPORT DE MINI-PROJET

Master Sciences des Données et Systèmes Intelligents

Sous le thème :

**DÉTECTION DE SPAM SMS EN UTILISANT DES
TECHNIQUES D'APPRENTISSAGE AUTOMATIQUE.**

Élaboré par
Bilal EL HAYANI
Mohamed IKEN

Encadré par
Pr. Anas EL ANSARI

ANNÉE UNIVERSITAIRE 2022 - 2023

20 juillet 2023

Table des matières

Introduction générale	2
1 Compréhension du problème et du dataset	3
1.1 Introduction	3
1.2 Problématique	3
1.3 Solution propose	3
1.4 Objectif	3
1.5 Les outils et les bibliothèques utilisées	4
1.6 Conclusion	4
2 Explication du code source	5
2.1 Importation des bibliothèques	5
2.2 Importer notre dataset	5
2.3 Description des features	5
2.4 Analyse descriptive et Préparation du dataset	6
2.5 Analyse exploratoire des données EDA	8
2.6 Natural Language Processing	13
2.6.1 NLTK Library	13
2.6.2 Tokenization	13
2.6.3 Remove stops words	13
2.6.4 Stemming	14
2.6.5 TF-IDF	14
2.7 Moduling	20
2.7.1 SVC (Support Vector Classifier)	20
2.7.2 KNN (K plus proches voisins)	21
2.7.3 Naïve Bayes	22
2.7.4 Arbre de décision	22
2.7.5 L'algorithme Random Forest	23
2.8 Déploiement	27
Conclusion générale	31

Introduction générale

Le Data Mining (fouille de données) est un champ situé au croisement de la statistique et des technologies de l'information dont le but est de découvrir des structures dans de vastes ensembles de données et d'en extraire des informations intéressantes.

C'est un processus qui consiste à explorer et analyser de grandes quantités de données afin de découvrir des formes, des règles, des relations, des tendances, des corrélations et/ou dépendances en utilisant des méthodes d'intelligence artificielle, de statistiques et de reconnaissances de formes.

L'objectif de notre projet est de résoudre un problème de classification des SMS en tant que spam ou non spam. Ce sujet relève du domaine du traitement automatique du langage naturel (NLP) et implique l'utilisation d'algorithmes d'apprentissage automatique. Nous appliquons la méthodologie CRISP-DM (Cross Industry Standard Process for Data Mining), qui est une approche standard pour la gestion de projets de fouille de données. Cette méthodologie comprend six étapes clés : la compréhension du domaine, la compréhension des données, la préparation des données, la modélisation, l'évaluation et le déploiement. Notre objectif est de construire un modèle capable de prédire si un SMS est un spam ou non spam. Pour cela, nous suivons les étapes de CRISP-DM afin de garantir la qualité de notre modèle et de permettre une mise en œuvre réussie dans un environnement opérationnel.

Chapitre 1

Compréhension du problème et du dataset

1.1 Introduction

Dans le chapitre suivant, nous examinerons les problématiques rencontrées dans notre étude et nous fournirons une description générale du dataset afin de mieux la comprendre. Nous aborderons les questions spécifiques liées à notre sujet et explorerons les caractéristiques clés des données pour mettre en contexte notre analyse.

1.2 Problématique

L'estimation et la détection des SMS spam sont essentielles pour améliorer la sécurité et la pertinence des services de messagerie. Les SMS spam peuvent contenir du contenu indésirable, de la publicité non sollicitée, des escroqueries ou des liens malveillants. Les utilisateurs ont besoin de mécanismes efficaces pour filtrer les SMS indésirables et protéger leur expérience utilisateur.

1.3 Solution propose

Pour résoudre ce problème, des techniques avancées d'apprentissage automatique et de traitement du langage naturel sont utilisées. Des modèles sont entraînés sur de grands ensembles de données contenant des exemples de SMS spam et de SMS non-spam. Ces modèles analysent les caractéristiques des SMS, telles que les mots-clés, les schémas de ponctuation et la longueur du message, pour prédire s'il s'agit d'un spam ou non.

La détection des SMS spam permet de bloquer ou de marquer automatiquement les messages indésirables, offrant ainsi aux utilisateurs une expérience plus sûre et plus pertinente. Cependant, il est important de mettre à jour régulièrement les modèles de détection pour rester efficace face aux nouvelles techniques utilisées par les spammeurs.

En somme, la détection des SMS spam est essentielle pour protéger les utilisateurs contre le contenu indésirable et les risques associés. Grâce à l'utilisation de techniques d'apprentissage automatique, il est possible de fournir des services de messagerie plus sécurisés et plus agréables pour les utilisateurs.

1.4 Objectif

L'objectif des problématiques liées à la détection des SMS spam est d'améliorer la sécurité, la pertinence, la confidentialité et l'expérience utilisateur dans le domaine des services de messagerie.

1.5 Les outils et les bibliothèques utilisées

1. **Python** est en effet un langage de programmation open source largement utilisé dans le domaine du data mining et de l'analyse des données. Il offre une grande variété de bibliothèques et d'outils qui facilitent le développement de code pour les différentes phases du modèle CRISP-DM.



2. **Scikit-learn** est une bibliothèque dédiée au machine learning et à la data-science dans l'univers Python. Elle Donne accès à des versions efficaces d'un grand nombre d'algorithmes courants et elle s'appuie sur certaines des technologies comme : Numpy, pandas et Matplotlib



3. **Streamlit** est une bibliothèque Python open-source qui facilite la création d'applications web pour des projets de science des données et d'apprentissage automatique. Elle est conçue pour simplifier le processus de transformation de scripts de données en applications web interactives, permettant aux scientifiques des données et aux développeurs de partager facilement leur travail avec d'autres personnes. Nous avons utilisé le micro-framework Streamlit pour développer notre application qui utilise le modèle sélectionné.



4. **Pickle** est un module Python qui facilite la sauvegarde et la récupération de variables dans un fichier. Nous avons utilisé ce module pour enregistrer le modèle sélectionné afin de pouvoir l'utiliser ultérieurement lors de la phase de déploiement de notre application.
5. **Pandas**
Pandas est une bibliothèque Python dédiée à la Data Science. Nous l'avons utilisée pour transformer nos données en DataFrame et effectuer diverses opérations de traitement sur l'ensemble de données.

1.6 Conclusion

Dans ce chapitre, Nous avons abordé la problématique ainsi que le dataset associé. Nous avons également présenté l'ensemble des outils nécessaires pour mener à bien ma tâche.

Chapitre 2

Explication du code source

2.1 Importation des bibliothèques

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import matplotlib as mpl
```

2.2 Importer notre dataset

```
[ ]: data = pd.read_csv("spam.csv",encoding='latin-1')
data.head()
```

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

      Unnamed: 3 Unnamed: 4
0           NaN          NaN
1           NaN          NaN
2           NaN          NaN
3           NaN          NaN
4           NaN          NaN
```

2.3 Description des features

Dans ce dataset de SMS spam où il existe deux features, voici une description de ces deux caractéristiques :

Texte du message : Cette feature représente le contenu textuel du message SMS. Il s'agit de la partie principale du SMS à partir de laquelle les informations sont extraites pour la détection de spam. Les algorithmes de détection de spam analysent le texte du message pour identifier les motifs, les mots-clés ou les schémas associés aux SMS spam.

Étiquette de spam : Cette feature est une variable binaire qui indique si le message est classé comme spam (1) ou non-spam (0). Cette étiquette est généralement fournie avec le dataset et est utilisée comme la variable cible dans les tâches de classification supervisée pour entraîner les modèles de détection de spam.

2.4 Analyse descriptive et Préparation du dataset

Nombre de lignes et de colonnes

```
[ ]: data.shape
```

```
[ ]: (5572, 5)
```

Data set contient 5 colonnes. La colonne v1 est l'étiquette « ham » ou « spam » et la colonne v2 contient le texte du message SMS. Les colonnes « Sans nom : 2 », « Sans nom : 3 », « Sans nom : 4 » ne sont pas nécessaires, elles peuvent donc être supprimées car elles ne seront pas utiles dans la construction du modèle

```
[ ]: data.drop(columns=["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], inplace=True)

#rename the label and text columns
data = data.rename(columns={"v1": "target", "v2": "text"})
data.head()
```

Le tableau devrait maintenant se présenter comme suit :

```
[ ]:   target      text
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...
```

Analyse de duplications

```
[ ]: data.duplicated().sum()
```

```
[12]: 403
```

```
[ ]: print("before removing duplicates;",data.shape)
```

```
before removing duplicates; (5572, 2)
```

```
[13]: data.drop_duplicates(keep='first',inplace=True)
      print("after removing duplicates",data.shape)
```

```
after removing duplicates (5169, 2)
```

Vérification des valeurs manquantes

```
[ ]: data.isnull().sum()
```

```
[ ]: target    0
     text      0
     dtype: int64
```

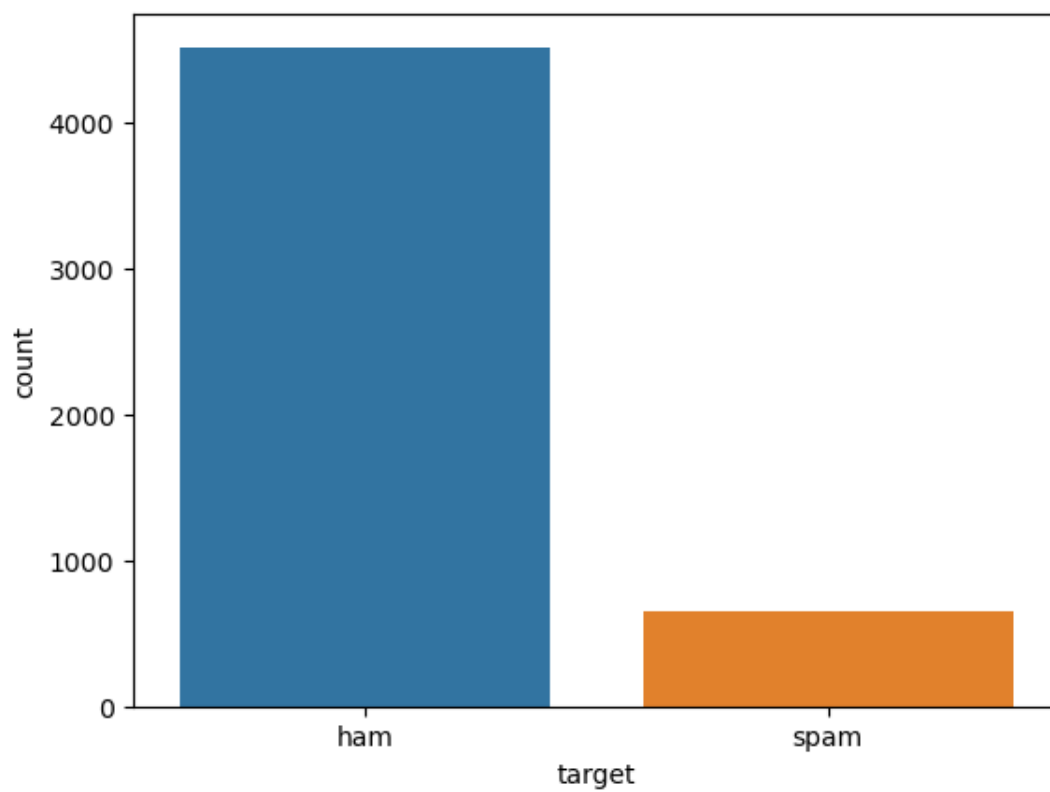
Cela montre qu'il y a 653 messages de spam dans dataset et 4516 messages ham.

```
[ ]: data["target"].value_counts()
```

```
[ ]: ham      4516  
    spam      653  
    Name: target, dtype: int64
```

Puis nous avons montrer le nombre de message spam et non spam a travers un schéma

```
[ ]: sns.countplot(data['target'])  
     plt.show()
```



→ represente en % le taux de message ham (non spam) ou de message spam

2.5 Analyse exploratoire des données EDA

Séparation des caractéristiques et des étiquettes de classe.

```
[ ]: y = data['target']  
     X = data.drop('target', axis=1)
```

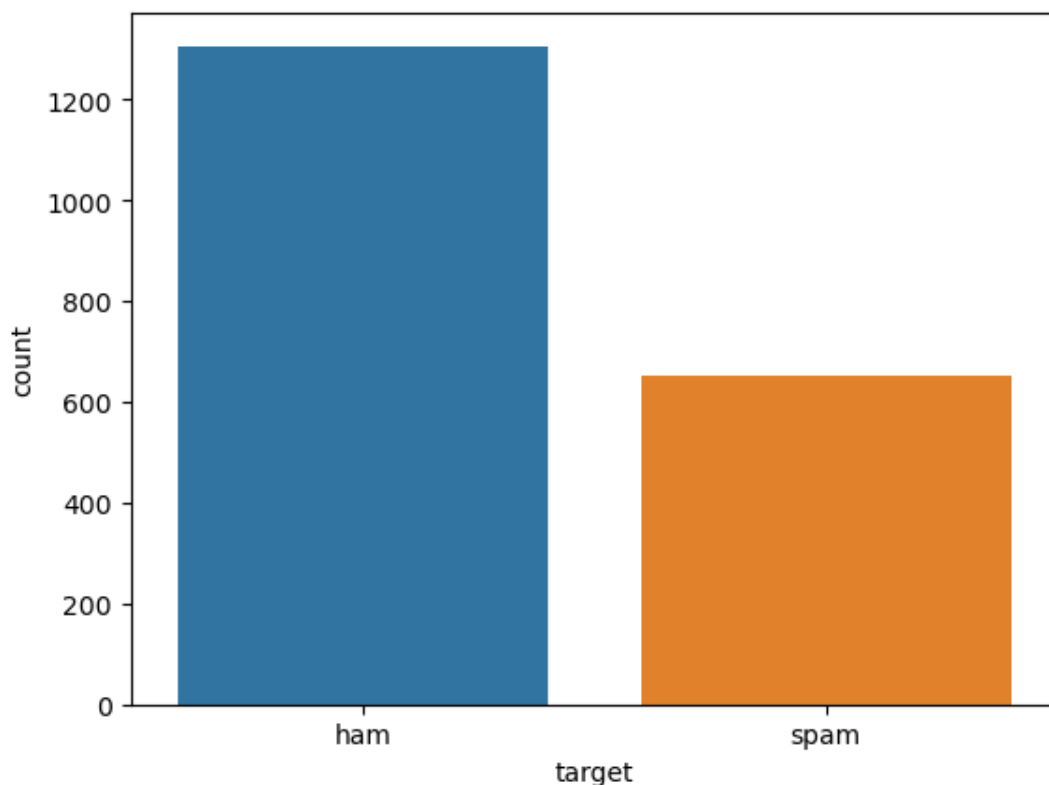
En utilisant le **RandomUnderSampler**, vous pouvez équilibrer les classes du jeu de données pour faire face aux problèmes de déséquilibre, ce qui peut améliorer les performances des modèles d'apprentissage automatique, en particulier lorsque vous traitez avec des ensembles de données où les classes sont fortement déséquilibrées.

```
[ ]: from imblearn.under_sampling import RandomUnderSampler  
     rus = RandomUnderSampler(sampling_strategy= 0.5, random_state=42)  
     X_resampled, y_resampled = rus.fit_resample(X, y)
```

```
[ ]: y_resampled.value_counts()
```

```
[22]: ham      1306  
      spam      653  
      Name: target, dtype: int64
```

```
[ ]: sns.countplot(y_resampled)  
     plt.show()
```



Après cette opération, `combined_data` contiendra l'ensemble de données combiné où les caractéristiques et les étiquettes cibles sous-échantillonnées sont fusionnées en une seule matrice. Chaque ligne de cette matrice représente un exemple d'entraînement, où les colonnes correspondent aux caractéristiques et à l'étiquette cible pour cet exemple. Cela permettra de préparer les données pour l'entraînement des modèles d'apprentissage automatique sur l'ensemble de données équilibré.

```

[]: y_resampled_array = y_resampled.values.reshape(-1, 1)

# Combine X_resampled and y_resampled_array horizontally (along columns)
combined_data = np.concatenate((X_resampled, y_resampled_array), axis=1)

[]: #combined_data

[]: data = pd.DataFrame(combined_data, columns=list(X_resampled.columns)+ ['target'])

[]: data['num_characters'] = data['text'].apply(len)
data.head()

```

Calculer le nombre de caractères dans chaque message.

```

[]:

```

	text	target	num_characters
0	Come to me, slave. Your doing it again ... Goi...	ham	128
1	U meet other fren dun wan meet me ah... Muz b ...	ham	59
2	G says you never answer your texts, confirm/deny	ham	48
3	K so am I, how much for an 8th? Fifty?	ham	38
4	HMM yeah if your not too grooved out! And im l...	ham	83

```

[]: data.shape

```

```

[]: (1959, 3)

```

```

[]: data.hist(column='num_characters', by='target', bins=50,figsize=(11,5))

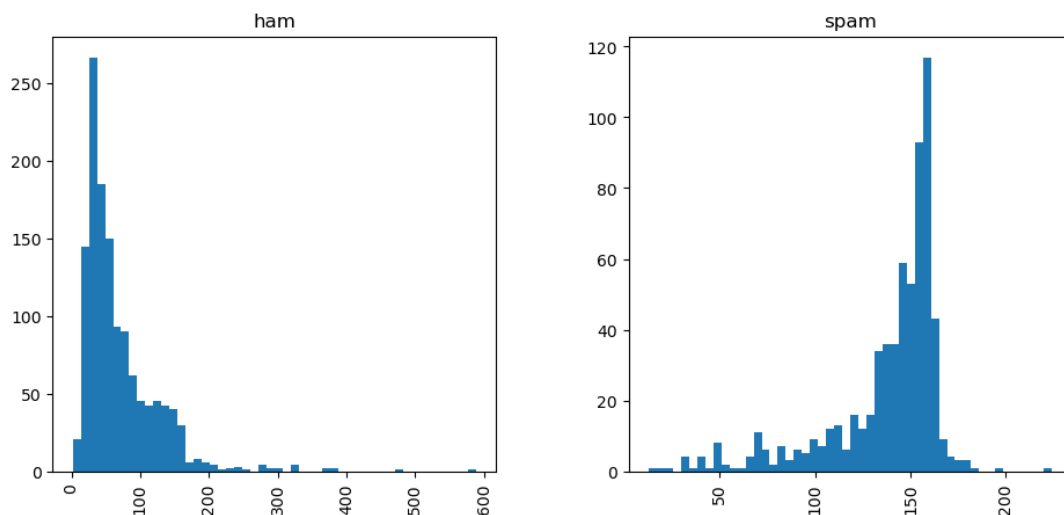
```

Chaque histogramme montrera la distribution des valeurs de 'num_characters' pour la classe correspondante. Cela permettra de visualiser la répartition des nombres de caractères dans les données en fonction des classes cibles, ce qui peut être utile pour comprendre la relation entre ces deux variables.

```

[]: array([<AxesSubplot:title={'center':'ham'}>,
        <AxesSubplot:title={'center':'spam'}>], dtype=object)

```



Après avoir exécuté ces deux lignes de code, le DataFrame data aura deux nouvelles colonnes : 'numwords' et 'numsentences', qui indiqueront le nombre de mots et de phrases respectivement pour chaque texte dans la colonne 'text'. Ces informations peuvent être utiles pour l'analyse exploratoire des données et pour mieux comprendre la structure et la complexité des textes dans dataset.

```
[ ]: # num of words
data['num_words'] = data['text'].apply(lambda x:len(nltk.word_tokenize(x)))
#num of sentences
data['num_sentences'] = data['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

les étiquettes de la colonne 'target' seront encodées en valeurs numériques. Cette transformation est utile lorsqu'on travaille avec des algorithmes d'apprentissage automatique qui nécessitent des étiquettes numériques plutôt que catégoriques.

```
[ ]: from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
data['target']=encoder.fit_transform(data['target'])
```

En utilise la méthode **describe()** pour obtenir des statistiques descriptives sur les colonnes 'num_characters', 'num_words' et 'num_sentences' du DataFrame 'data'.

```
[ ]: data[['num_characters', 'num_words', 'num_sentences']].describe()
```

```
[ ]:
      num_characters  num_words  num_sentences
count      1959.000000    1959.000000      1959.000000
mean         92.009699      20.399183         2.186830
std         58.083535      12.455720         1.467319
min          2.000000       1.000000         1.000000
25%         40.000000      10.000000         1.000000
50%         81.000000      19.000000         2.000000
75%        145.000000      29.000000         3.000000
max        588.000000     154.000000        16.000000
```

```
[ ]: data.groupby('target').describe().T
```

```
[ ]: target
num_characters count  1306.000000  653.000000
      mean      69.068913  137.891271
      std      55.027223   30.137753
      min       2.000000   13.000000
      25%      32.000000  132.000000
      50%      51.000000  149.000000
      75%      89.000000  157.000000
      max     588.000000  224.000000
num_words      count  1306.000000  653.000000
      mean     16.764931   27.667688
      std     12.983952    7.008418
      min      1.000000    2.000000
      25%       8.000000   25.000000
      50%      13.000000   29.000000
      75%      22.000000   32.000000
      max     154.000000   46.000000
num_sentences count  1306.000000  653.000000
      mean      1.796325   2.967841
      std      1.293610   1.483201
      min      1.000000   1.000000
      25%      1.000000   2.000000
      50%      1.000000   3.000000
      75%      2.000000   4.000000
      max     16.000000   8.000000
```

un histogramme des nombres de caractères pour les classes 'Non-spam' (en bleu) et 'Spam' (en rouge), ce qui permettra de visualiser la distribution des caractères dans chaque classe du DataFrame 'data'. Cela peut être utile pour comprendre les différences entre les deux classes en termes de nombre

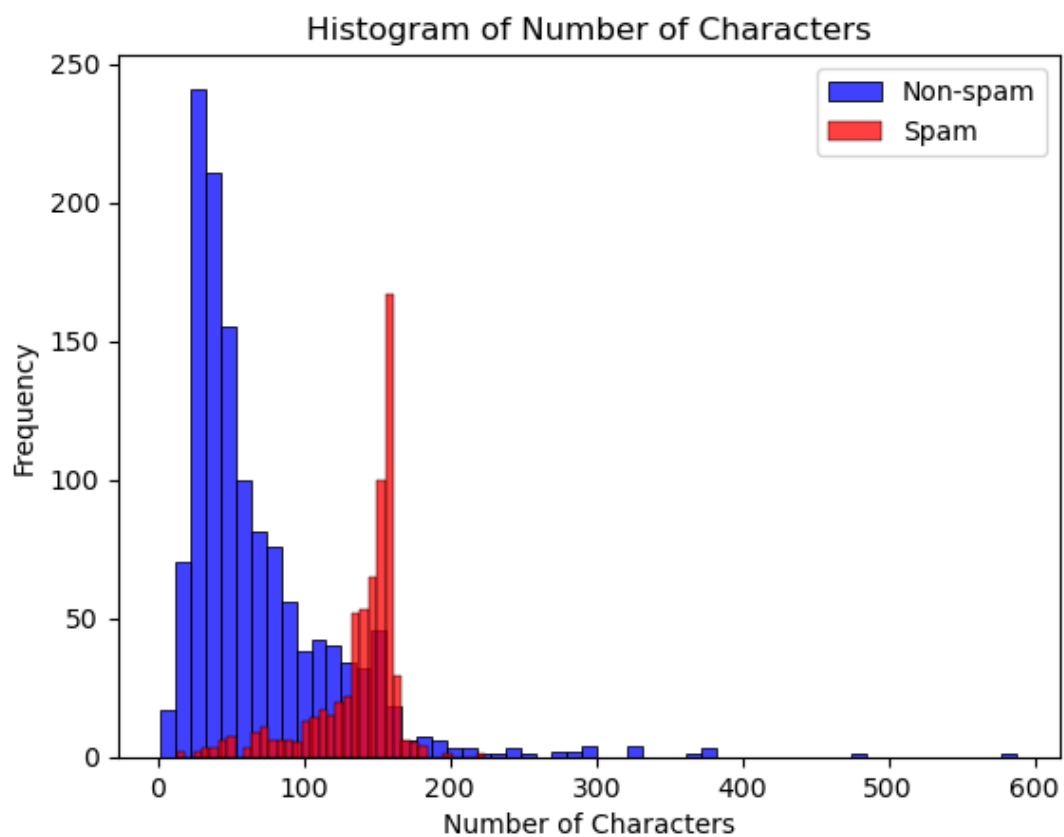
de caractères dans les textes.

```
[ ]: #num_characters
import seaborn as sns
sns.histplot(data[data['target'] == 0]['num_characters'], color='blue',
             label='Non-spam')
sns.histplot(data[data['target'] == 1]['num_characters'], color='red', label='Spam')

plt.xlabel('Number of Characters')
plt.ylabel('Frequency')
plt.title('Histogram of Number of Characters')

plt.legend()

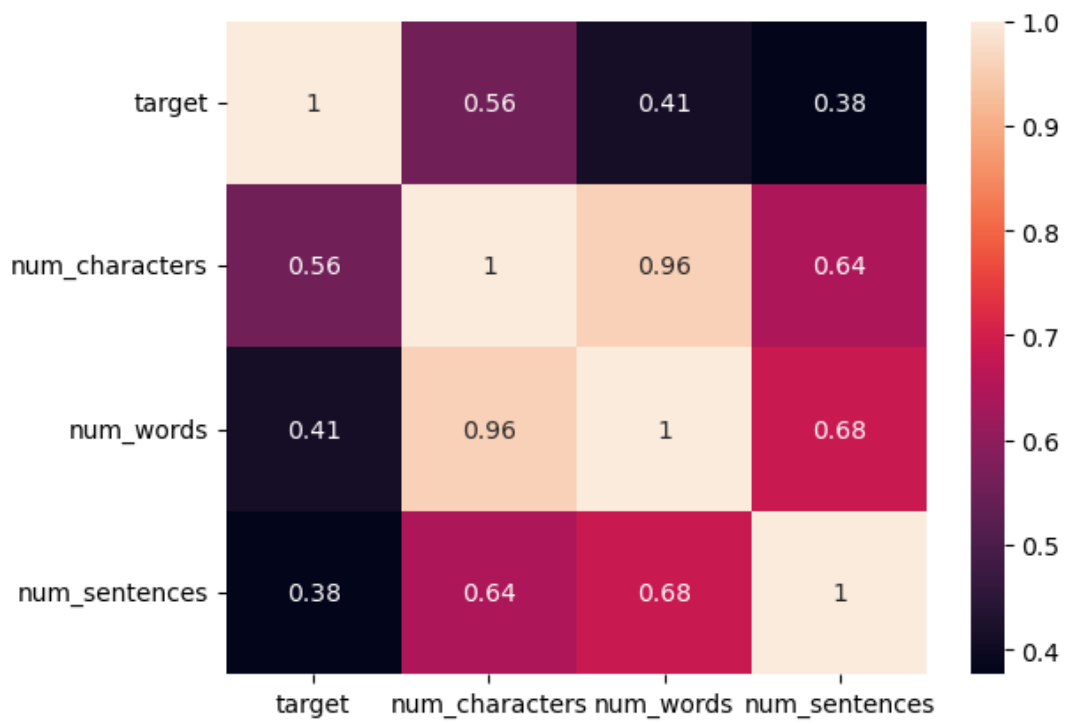
plt.show()
```



Matrice de corrélation

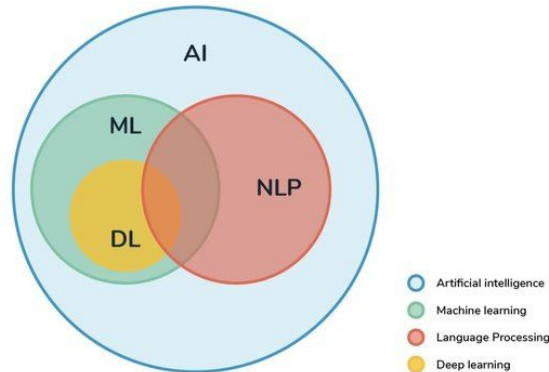
```
[ ]: sns.heatmap(data.corr(),annot=True)
```

```
[ ]: <AxesSubplot:>
```



2.6 Natural Language Processing

Natural Language Processing (NLP) est un sous-domaine de l'IA qui a vocation à donner la faculté à un programme informatique de comprendre et d'interpréter le langage tel qu'il est parlé et écrit par les êtres humains, dans toutes ses complexités et ses nuances. Ainsi, un algorithme qui utilise le NLP est capable d'analyser les phrases, de saisir le sens des mots dans leur contexte.



2.6.1 NLTK Library

Natural Language Toolkit (NLTK) est une bibliothèque Python open-source utilisée pour le traitement du langage naturel (NLP). Elle fournit une grande variété de fonctionnalités pour faciliter l'analyse et la compréhension des données textuelles.

2.6.2 Tokenization

La tokenisation cherche à transformer un texte en une série de tokens individuels. Dans l'idée, chaque token représente un mot, et identifier des mots semble être une tâche relativement simple.

```
[ ]: import nltk
text = "I know you mood off today"
print("Before tokenization : ",text)
print("-"*65)
text = nltk.word_tokenize(text)
print("After tokenization : ",text)
```

Before tokenization : I know you mood off today

After tokenization : ['I', 'know', 'you', 'mood', 'off', 'today']

2.6.3 Remove stops words

Certains mots se retrouvent très fréquemment dans la langue française. En anglais, on les appelle les "stop words". Ces mots, bien souvent, n'apportent pas d'information dans les tâches suivantes. Lorsque l'on effectue par exemple une classification par la méthode TF-IDF, on souhaite limiter la quantité de mots dans les données d'entraînement.

Les "stop words" sont établis comme des listes de mots. Ces listes sont généralement disponibles dans une librairie appelée NLTK (Natural Language Tool Kit), et dans beaucoup de langues différentes. On accède aux listes en anglais de cette manière :

```
[ ]: from nltk.corpus import stopwords
nltk.download('stopwords')
print(stopwords.words('english'))
```

2.6.4 Stemming

Le stemming consiste à réduire un mot dans sa forme " racine ". Le but du stemming est de regrouper de nombreuses variantes d'un mot comme un seul et même mot.

Par exemple, une fois que l'on applique un stemming sur "walked" ou "walk" , le mot résultant est le même. Cela permet notamment de réduire la taille du vocabulaire dans les approches de type sac de mots ou TF-IDF.

```
[ ]: import nltk
from nltk.stem import PorterStemmer
text = "The dogs are barking at the cat"
stemmer = PorterStemmer()
text = nltk.word_tokenize(text)
filterd_words = []
for word in text:
    stemmed_word = stemmer.stem(word)
    filterd_words.append(stemmed_word)
print(filterd_words)
```

```
['the', 'dog', 'are', 'bark', 'at', 'the', 'cat']
```

2.6.5 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) est une technique couramment utilisée en traitement du langage naturel qui mesure l'importance d'un terme dans un document ou un corpus de documents.

Elle prend en compte à la fois la fréquence du terme dans le document (TF) et sa rareté dans le corpus (IDF) pour attribuer un poids à chaque terme.

Le score TF-IDF est le produit des scores TF et IDF, et est utilisé pour identifier les termes les plus importants dans le texte.

Term Frequency (TF) est la fréquence d'apparition d'un mot dans un document, divisée par le nombre de mots.

$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$

Inverse Document Frequency (IDF) pour attribuer un poids à chaque terme du document ou du corpus.

$$IDF(t) = \log_e(\text{TotalNumberOfDocuments} / \text{NumberOfDocumentsWithTerm } t \text{ In It})$$

$$TF\text{-}IDF = TF(t) * IDF(t)$$

2.6.6 Nettoyage du texte

- Remplacer toutes les majuscules par des minuscules
- Tokeniser le texte en mots individuels.
- Retirer la ponctuation
- Enlever les "stop words"
- Stemming

```
[ ]: # Function to transform the text
def transform_text(text):
    # Convert text to lowercase
```

```

text = text.lower()

# Tokenize the text into individual words
text = nltk.word_tokenize(text)

# Create an empty list to store filtered words
filtered_words = []

# Iterate over each word in the text
for word in text:
    # Check if the word contains only alphanumeric characters
    if word.isalnum():
        # Add the word to the filtered list
        filtered_words.append(word)

# Update the text with the filtered words
text = filtered_words[:]

# Clear the filtered words list for reuse
filtered_words.clear()

# Remove stop words and punctuation from the text
for word in text:
    # Check if the word is not a stop word or punctuation mark
    if word not in stopwords.words('english') and word not in string.
→ punctuation:
        # Add the word to the filtered list
        filtered_words.append(word)

# Update the text with the filtered words
text = filtered_words[:]

# Clear the filtered words list for reuse
filtered_words.clear()

# Apply stemming to the words in the text
stemmer = PorterStemmer()
for word in text:
    # Perform stemming on each word
    stemmed_word = stemmer.stem(word)
    # Add the stemmed word to the filtered list
    filtered_words.append(stemmed_word)

# Join the filtered words to form the transformed text
transformed_text = " ".join(filtered_words)

# Return the transformed text
return transformed_text

```

```
[ ]: s = data["text"][100]
```

```
[234]: s
```

```
[ ]: 'I know you mood off today'
```

```
[ ]: transform_text(s)
```



```
[ ]: 'know mood today'
```

```
[ ]: data['transformed_text'] = data['text'].apply(transform_text)
```

```
[ ]: data.head()
```

```
[ ]:
      text      target  num_characters  \
0  Come to me, slave. Your doing it again ... Goi...      0          128
1  U meet other fren dun wan meet me ah... Muz b ...      0           59
2    G says you never answer your texts, confirm/deny      0           48
3      K so am I, how much for an 8th? Fifty?          0           38
4  HMM yeah if your not too grooved out! And im l...      0           83

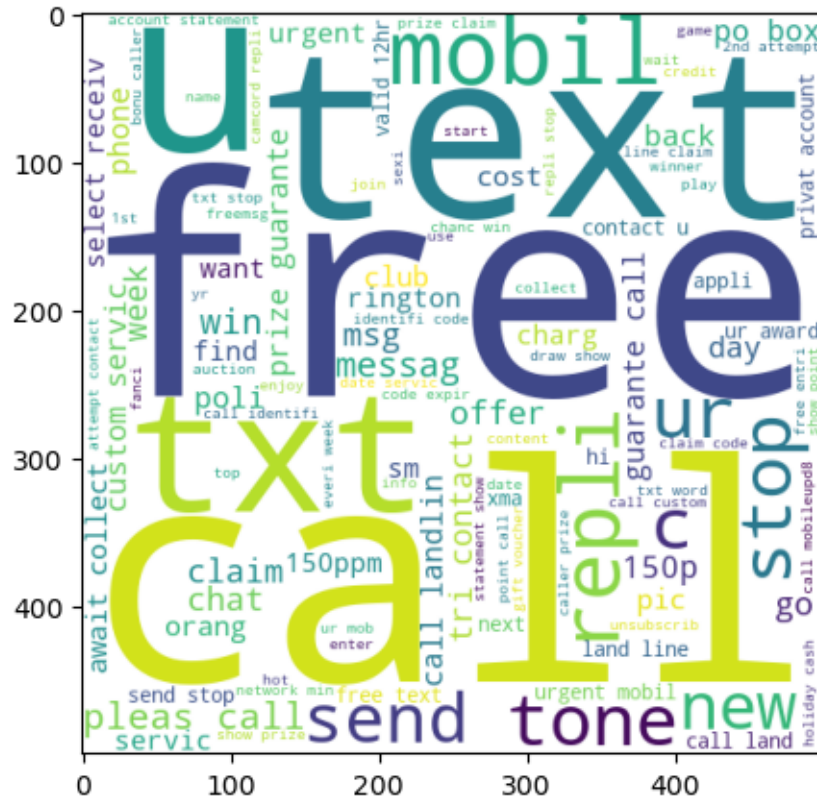
      num_words  num_sentences      transformed_text
0           28           3  come slave go shell unconsci avoid make unhappi
1           16           1      u meet fren dun wan meet ah muz b guy rite
2            9           1      g say never answer text
3           13           2      k much 8th fifti
4           19           2  hmm yeah groov im look forward pound special
```

```
[ ]: #!pip install wordcloud
```

```
[ ]: from wordcloud import WordCloud
wc=WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

→ Nombre le plus courant de mots dans les messages spam.

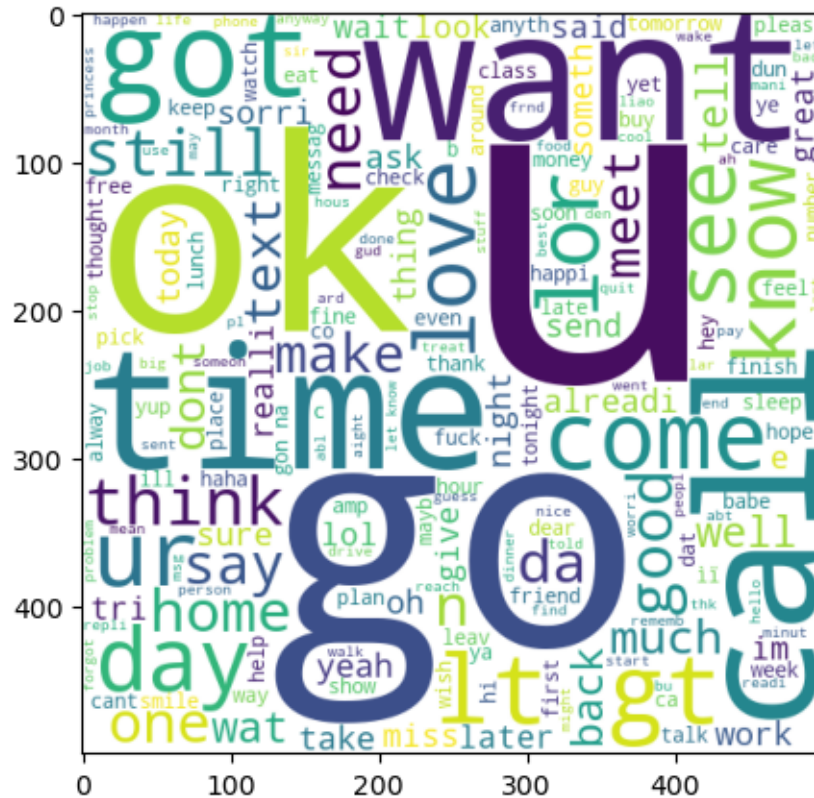
```
[ ]: #generating Word cloud for only Spam words
spam_wc = wc.generate(data[data['target'] == 1]['transformed_text'].str.cat(sep="␣
↵"))
plt.figure(figsize=(10,5))
plt.imshow(spam_wc)
plt.show()
```



→ Nombre le plus fréquent de mots dans le message non spam (ham).

```
[ ]: #generating Word cloud for only ham words

ham_wc = wc.generate(data[data['target'] == 0]['transformed_text'].str.cat(sep=" "))
plt.figure(figsize=(10,5))
plt.imshow(ham_wc)
plt.show()
```



```

[]: spam_corpus = []
    for msg in data[data['target'] == 1]['transformed_text'].tolist():
        for word in msg.split():
            spam_corpus.append(word)

```

```

[]: len(spam_corpus)

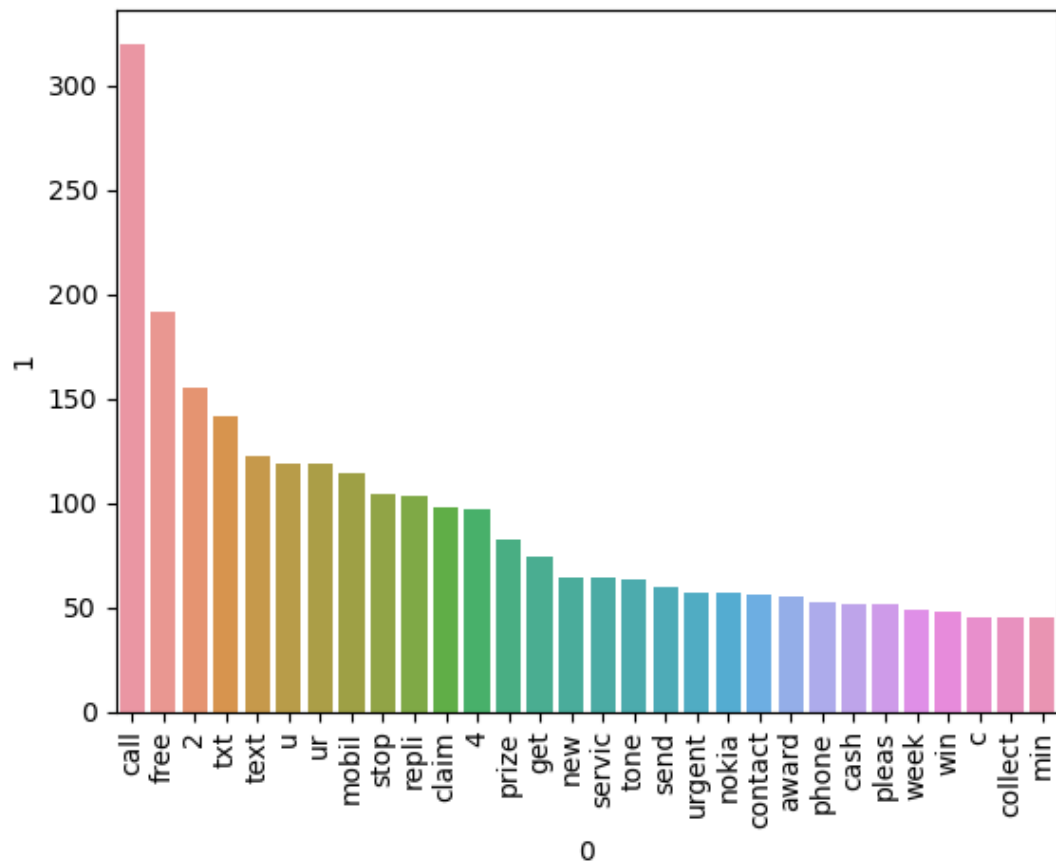
```

[] : 9939

```

[]: from collections import Counter
sns.barplot(pd.DataFrame(Counter(spam_corpus).most_common(30))[0], pd.
    ↳ DataFrame(Counter(spam_corpus).most_common(30))[1])
plt.xticks(rotation='vertical')
plt.show()

```

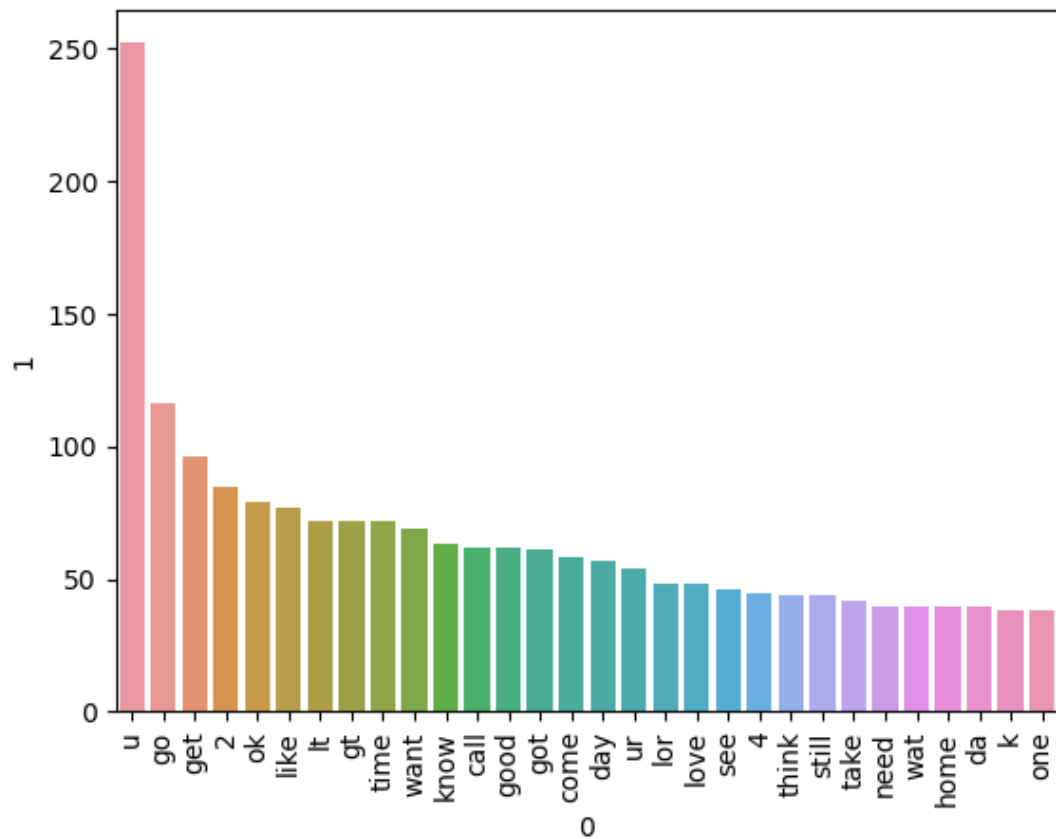


```
[ ]: ham_corpus = []
for msg in data[data['target'] == 0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
```

```
[ ]: len(ham_corpus)
```

```
[ ]: 9963
```

```
[ ]: from collections import Counter
sns.barplot(pd.DataFrame(Counter(ham_corpus).most_common(30))[0],pd.
↳ DataFrame(Counter(ham_corpus).most_common(30))[1])
plt.xticks(rotation='vertical')
plt.show()
```



```
[ ]: from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
      cv = CountVectorizer()
      tfidf = TfidfVectorizer(max_features=3000)
```

```
[ ]: X = tfidf.fit_transform(data['transformed_text']).toarray()
```

```
[ ]: X
```

2.7 Moduling

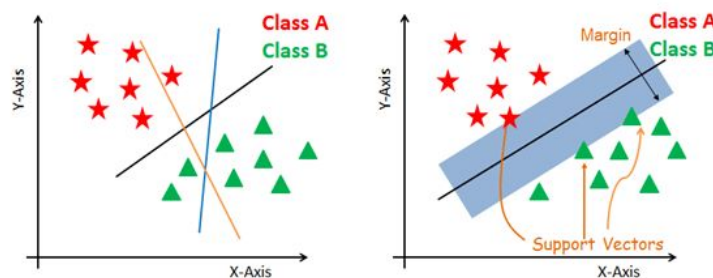
2.7.1 SVC (Support Vector Classifier)

La SVM est un algorithme passionnant et les concepts sont relativement simples. Le classificateur sépare les points de données à l'aide d'un hyperplan avec la plus grande quantité de marge. C'est pourquoi un classificateur SVM est également connu sous le nom de classificateur discriminatif. La SVM trouve un hyperplan optimal qui aide à classer les nouveaux points de données.

Comment fonctionne la SVM ?

L'objectif principal est de séparer l'ensemble de données donné de la meilleure façon possible. La distance entre les points les plus proches est connue sous le nom de marge. L'objectif est de sélectionner un hyperplan avec la marge maximale possible entre les vecteurs de support dans le jeu de données donné. La SVM recherche l'hyperplan marginal maximal dans les étapes suivantes :

1. Générez des hyperplans qui séparent les classes de la meilleure façon. Figure de gauche montrant trois hyperplans noir, bleu et orange. Ici, le bleu et l'orange ont une erreur de classification plus élevée, mais le noir sépare correctement les deux classes.
2. Sélectionnez l'hyperplan droit avec la ségrégation maximale à partir des points de données les plus proches, comme indiqué dans la figure de droite.



Importation, initialisation et entraînement du modèle

```
[ ]: from sklearn.svm import SVC

[ ]: svm = SVC(kernel='sigmoid')

[ ]: svm.fit(X_train,y_train)
     y_pred = svm.predict(X_test)
     print(classification_report(y_test,y_pred))
     print("MCC :",matthews_corrcoef(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.93	0.99	0.96	387
1	0.98	0.86	0.92	201
accuracy			0.95	588
macro avg	0.96	0.93	0.94	588
weighted avg	0.95	0.95	0.95	588

MCC : 0.8834243850099469

- Le modèle s'adapte bien aux données d'entraînement et aux données de test
- Le modèle SVM représente un accuracy de 95%, alors ce modèle est un bon modèle et il peut effectuer des bonnes prédictions.
- On remarque que ces deux valeurs qu'on a trouvés que les deux valeurs sont proches et représente un

très bon score, alors on peut conclure que notre classifieur fonctionne très bien sur les éléments négatifs et positifs.

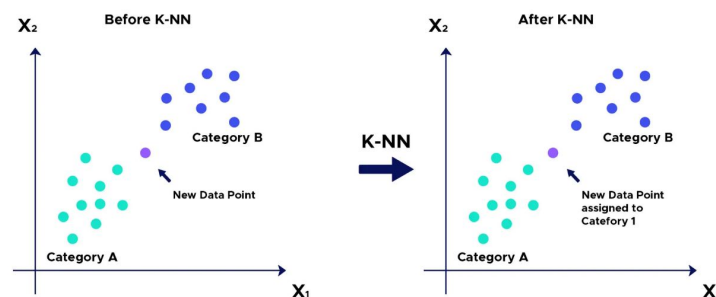
2.7.2 KNN (K plus proches voisins)

KNN est un algorithme d'apprentissage supervisé qui nous permet de faire des prédictions sur des variables quantitatives et aussi des variables qualitatives. Et elle permet de résoudre des problèmes de régression et des problèmes de classification.

Principe de fonctionnement

Le principe de fonctionnement de l'algorithme KNN (K plus proches voisins) peut être résumé en plusieurs étapes :

1. **Étape 1** : Sélectionnez le nombre K de voisins.
2. **Étape 2** : Calculez la distance :
 - **Distance de Manhattan** : $\sum_{i=1}^n |x_i - y_i|$
 - **Distance euclidienne** : $\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
3. **Étape 3** : Prenez les K voisins les plus proches selon la distance calculée.
4. **Étape 4** : Parmi ces K voisins, comptez le nombre de points appartenant à chaque catégorie.
5. **Étape 5** : Attribuez le nouveau point à la catégorie la plus présente parmi ces K voisins.



Importation, initialisation et entraînement du modèle

```
[ ]: from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]: knc = KNeighborsClassifier()
```

```
[ ]: knc.fit(X_train,y_train)
y_pred = knc.predict(X_test)
print(classification_report(y_test,y_pred))
print("MCC :",matthews_corrcoef(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.71	1.00	0.83	387
1	1.00	0.23	0.37	201
accuracy			0.74	588
macro avg	0.86	0.61	0.60	588
weighted avg	0.81	0.74	0.68	588

MCC : 0.40423761822839754

2.7.3 Naïve Bayes

Le Naive Bayes multinomial est l'une des variations de l'algorithme Naive Bayes en apprentissage automatique, qui est très utile à utiliser sur un ensemble de données distribué selon une loi multinomiale. Lorsqu'il y a plusieurs classes à classifier, cet algorithme peut être utilisé pour prédire l'étiquette du texte en calculant la probabilité de chaque étiquette pour le texte d'entrée, puis en générant l'étiquette avec la probabilité la plus élevée comme résultat.

Importation, initialisation et entraînement du modèle

```
[ ]: from sklearn.naive_bayes import MultinomialNB
```

```
[ ]: mnb = MultinomialNB()
```

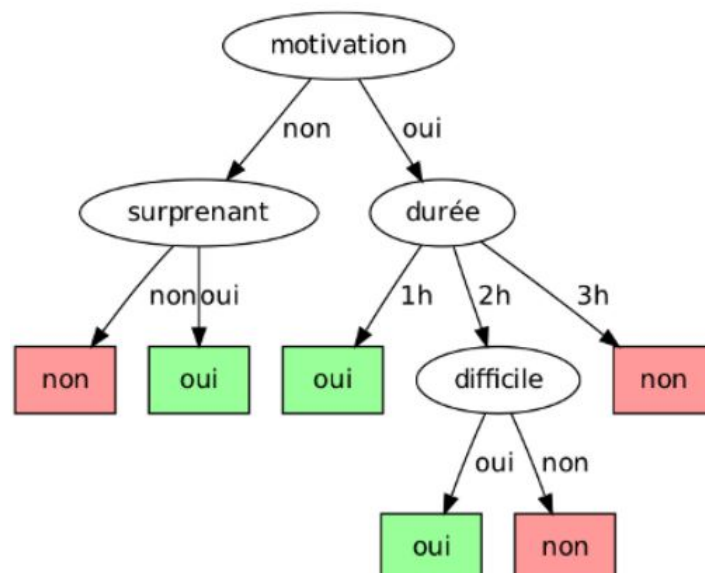
```
[ ]: mnb.fit(X_train,y_train)
y_pred = mnb.predict(X_test)
print(classification_report(y_test,y_pred))
print("MCC :",matthews_corrcoef(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.93	0.99	0.96	387
1	0.99	0.86	0.92	201
accuracy			0.95	588
macro avg	0.96	0.93	0.94	588
weighted avg	0.95	0.95	0.95	588

MCC : 0.8875523383591435

2.7.4 Arbre de décision

L'arbre de décision est un algorithme qui représente un ensemble de choix sous la forme graphique d'un arbre.



Principe de fonctionnement

Ce genre d'algorithme permet de répartir une population d'individus en groupes homogènes selon des attributs discriminants en fonction d'un objectif fixé et connu. Pour ce faire, l'algorithme va chercher

à partitionner les individus en groupes d'individus les plus similaires possibles du point de vue de la variable à prédire. Le résultat de l'algorithme produit un arbre qui révèle des relations hiérarchiques entre les variables. Il est ainsi possible de rapidement comprendre des règles métiers expliquant votre variable cible.

Importation, initialisation et entraînement du modèle

```
[ ]: from sklearn.tree import DecisionTreeClassifier
```

```
[ ]: dtc = DecisionTreeClassifier(max_depth=5)
```

```
[ ]: dtc.fit(X_train,y_train)
y_pred = dtc.predict(X_test)
print(classification_report(y_test,y_pred))
print("MCC :",matthews_corrcoef(y_test,y_pred))
```

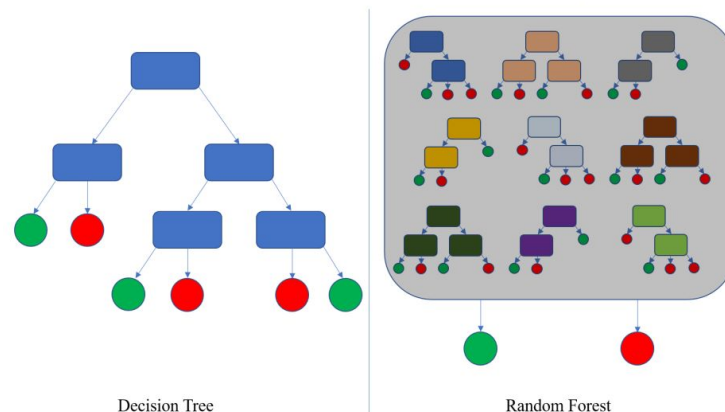
	precision	recall	f1-score	support
0	0.85	0.96	0.90	387
1	0.89	0.67	0.76	201
accuracy			0.86	588
macro avg	0.87	0.81	0.83	588
weighted avg	0.86	0.86	0.85	588

MCC : 0.680417112204023

→ Le modèle arbre de décision représente un accuracy de 86%, alors la performance de ce modèle est assez bien

2.7.5 L'algorithme Random Forest

L'algorithme des forêts d'arbres décisionnels effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents.



Principe de fonctionnement

Un Random Forest est constitué d'un ensemble d'arbres de décision indépendants. Chaque arbre dispose d'une vision parcelaire du problème du fait d'un double tirage aléatoire :

- Un tirage aléatoire avec remplacement sur les observations (les lignes de votre base de données). Ce processus s'appelle le tree bagging, 35

- Un tirage aléatoire sur les variables (les colonnes de votre base de données). Ce processus s'appelle le feature sampling.

A la fin, tous ces arbres de décisions indépendants sont assemblés. La prédiction faite par le random forest pour des données inconnues est alors la moyenne (ou le vote, dans le cas d'un problème de classification) de tous les arbres.

Importation, initialisation et entraînement du modèle

```
[ ]: from sklearn.ensemble import RandomForestClassifier
```

```
[ ]: rfc = RandomForestClassifier(n_estimators=50, random_state=2)
```

```
[ ]: rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
print(classification_report(y_test,y_pred))
print("MCC :",matthews_corrcoef(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.99	0.95	387
1	0.99	0.83	0.90	201
accuracy			0.94	588
macro avg	0.95	0.91	0.93	588
weighted avg	0.94	0.94	0.94	588

MCC : 0.8617079130061838

diviser Dataset

Nous divisons les données en 70 % pour la formation et 30 % pour les tests, avec et étant les messages dans les ensembles de formation et de test, et étant les étiquettes numériques de jambon ou de spam correspondantes. Par défaut, le fractionnement est effectué de manière aléatoire à chaque exécution du script. Le dernier paramètre rend le fractionnement cohérent et reproductible.

```
[]: from sklearn.model_selection import train_test_split
```

```
[]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

Les fonctions importées depuis la bibliothèque sklearn.metrics offrent des outils essentiels pour évaluer les performances des modèles d'apprentissage automatique.

```
[]: from sklearn.metrics import _  
    ↪ accuracy_score, confusion_matrix, precision_score, matthews_corrcoef, classification_report, f1_score
```

```
[]: from sklearn.linear_model import LogisticRegression  
    from sklearn.svm import SVC  
    from sklearn.naive_bayes import MultinomialNB  
    from sklearn.tree import DecisionTreeClassifier  
    from sklearn.neighbors import KNeighborsClassifier  
    from sklearn.ensemble import RandomForestClassifier  
    from sklearn.ensemble import AdaBoostClassifier  
    from sklearn.ensemble import BaggingClassifier  
    from sklearn.ensemble import ExtraTreesClassifier  
    from sklearn.ensemble import GradientBoostingClassifier  
    from xgboost import XGBClassifier
```

```
[]: svc = SVC(kernel='sigmoid')  
    knc = KNeighborsClassifier()  
    mnb = MultinomialNB()  
    dtc = DecisionTreeClassifier(max_depth=5)  
    lrc = LogisticRegression(solver='liblinear', penalty='l1')  
    rfc = RandomForestClassifier(n_estimators=50, random_state=2)  
    abc = AdaBoostClassifier(n_estimators=50, random_state=2)  
    bc = BaggingClassifier(n_estimators=50, random_state=2)  
    etc = ExtraTreesClassifier(n_estimators=50, random_state=2)  
    gbdt = GradientBoostingClassifier(n_estimators=50, random_state=2)  
    xgb = XGBClassifier(n_estimators=50, random_state=2)
```

```
[]: clfs = {  
    'SVC' : svc,  
    'KN' : knc,  
    'NB' : mnb,  
    'DT' : dtc,  
    'LR' : lrc,  
    'RF' : rfc,  
    'AdaBoost' : abc,  
    'BgC' : bc,  
    'ETC' : etc,  
    'GBDT' : gbdt,  
    'xgb' : xgb  
}
```

```
[]: def train_classifier(clf,X_train,y_train,X_test,y_test):  
    clf.fit(X_train,y_train)  
    y_pred = clf.predict(X_test)
```

```

accuracy = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
mcc = matthews_corrcoef(y_test,y_pred)
#f1_sc = f1_score(y_test,y_pred)

return accuracy,precision,mcc

```

```

[]: accuracy_scores = []
precision_scores = []
mcc_scores = []
f1_scores = []
for name,clf in clfs.items():

    current_accuracy,current_precision,mcc = train_classifier(clf,X_train,y_train,X_test,y_test)
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
    mcc_scores.append(mcc)
    #f1_scores.append(f1_sc)

```

→ Comparaison des différents algorithmes sur la base de l'accuracy, de la précision et du coefficient de corrélation de Matthews (MCC).

```

[]: performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores,
    'Precision':precision_scores,'MCC':mcc_scores}).
    sort_values('Precision',ascending=False)

```

```

[]: performance_df

```

```

[]:

```

	Algorithm	Accuracy	Precision	MCC
1	KN	0.729592	1.000000	0.404484
8	ETC	0.954082	0.991803	0.900423
2	NB	0.956633	0.984000	0.905353
5	RF	0.951531	0.983740	0.894368
0	SVC	0.956633	0.976378	0.904957
6	AdaBoost	0.920918	0.957265	0.826556
10	xgb	0.936224	0.944882	0.859304
9	GBDT	0.915816	0.933884	0.814036
7	BgC	0.928571	0.916667	0.842346
4	LR	0.895408	0.914530	0.768193
3	DT	0.859694	0.867257	0.686524

```

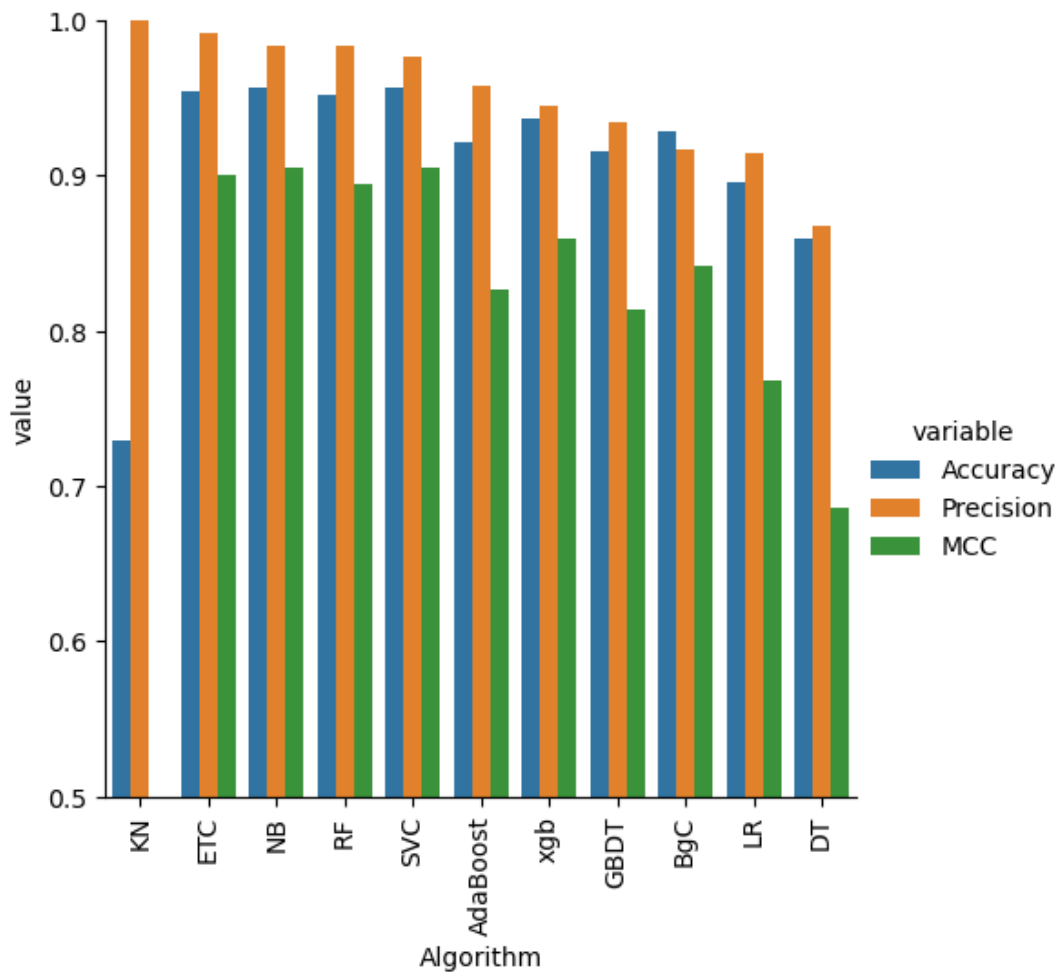
[]: performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")

```

```

[]: sns.catplot(x = 'Algorithm', y='value',
    hue = 'variable',data=performance_df1, kind='bar',height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()

```



2.8 Déploiement

Après la phase de Évaluation et avoir mis le modèle dans la phase de réalité du test, on a constaté que l'algorithme de **Naive Bayes** était le plus précis dans la prédiction.

Donc, nous avons choisi l'algorithme des "**Naive Bayes**" pour l'utiliser lors de la phase de déploiement.

```
[ ]: import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```

Voici le code que nous avons utilisé pour la partie de déploiement et la mise en mode de test. Nous avons utilisé le framework Streamlit.

```
import streamlit as st
import pickle
import string
from nltk.corpus import stopwords
import nltk
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

def transform_text(text):
```

```

text = text.lower()
text = nltk.word_tokenize(text)

y = []
for i in text:
    if i.isalnum():
        y.append(i)

text = y[:]
y.clear()

for i in text:
    if i not in stopwords.words('english') and i not in string.punctuation:
        y.append(i)

text = y[:]
y.clear()

for i in text:
    y.append(ps.stem(i))

return " ".join(y)

tfidf = pickle.load(open('vectorizer.pkl','rb'))
model = pickle.load(open("model.pkl",'rb'))

st.title("SMS Spam Classifier")

input_sms = st.text_input("Enter the message")
if st.button('Predict'):
    transformed_sms = transform_text(input_sms)

    vect_input = tfidf.transform([transformed_sms])

    result = model.predict(vect_input)[0]

    if result== 1:
        st.header("spam")
    else:
        st.header("Not spam")

```

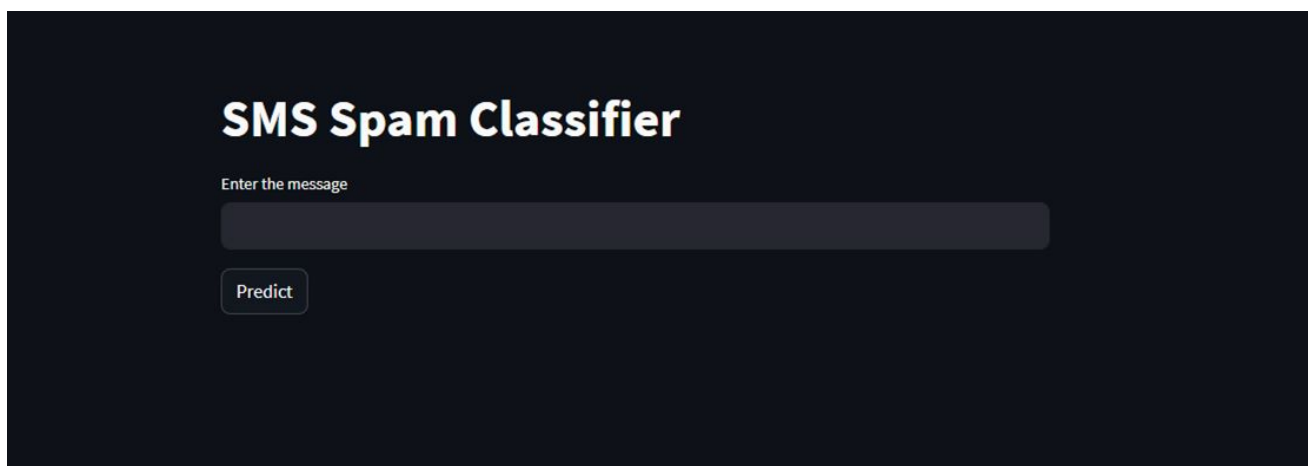
```
Anaconda Prompt (anaconda3) - streamlit run app.py

(base) C:\Users\iknem>cd C:\Users\iknem\OneDrive\Desktop\test
(base) C:\Users\iknem\OneDrive\Desktop\test>streamlit run app.py

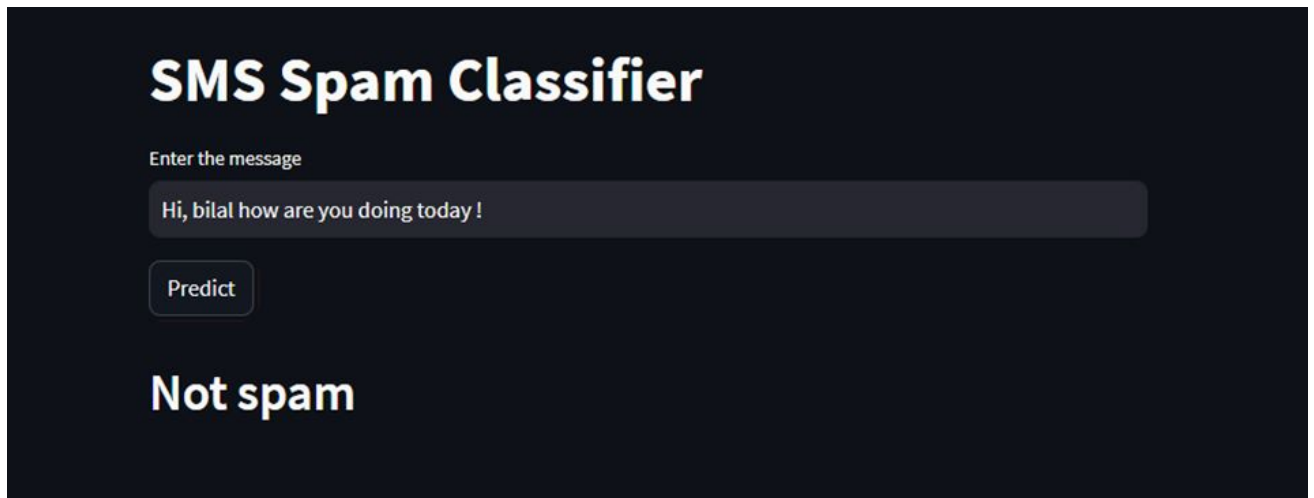
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://10.0.0.15:8501
```

Voici la première interface graphique où l'utilisateur peut simplement entrer un message pour prédire si le message est un spam ou non.

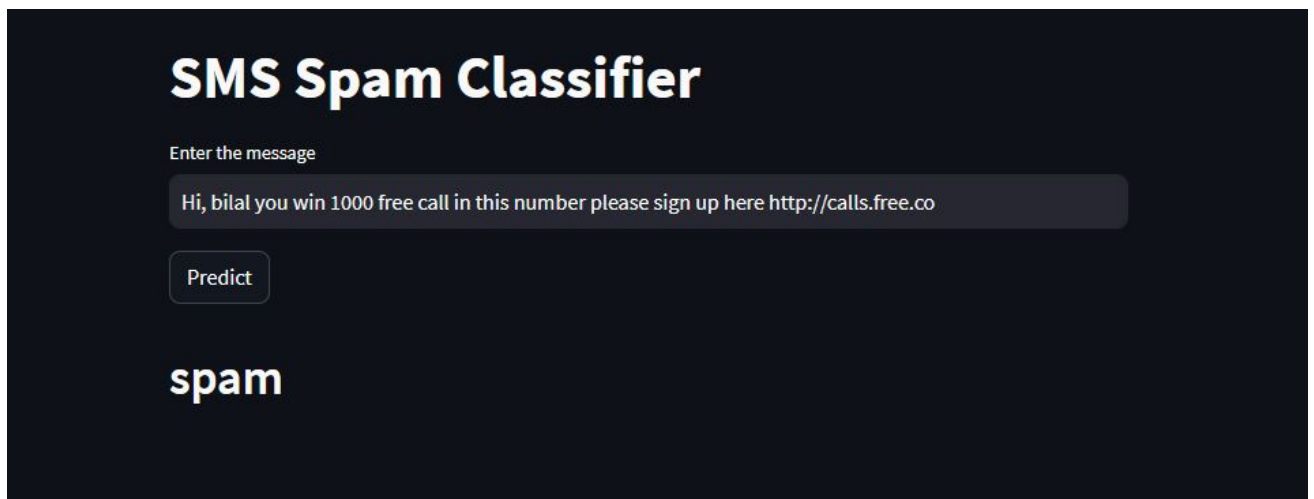


Voici un exemple d'un message que le modèle prédit comme non spam :
'Hi, bilal how are you doing today !'



The screenshot shows a web interface titled "SMS Spam Classifier" on a dark background. Below the title is a label "Enter the message" above a text input field containing the message "Hi, bilal how are you doing today !". Below the input field is a button labeled "Predict". At the bottom of the interface, the prediction result "Not spam" is displayed in a large, bold, white font.

Voici un exemple d'un message que le modèle prédit comme spam :
'Hi, bilal you win 1000 free call in this number please sign up here [http ://calls.free.co](http://calls.free.co)'



The screenshot shows the same "SMS Spam Classifier" web interface. The text input field now contains the message "Hi, bilal you win 1000 free call in this number please sign up here <http://calls.free.co>". The "Predict" button is still present. At the bottom, the prediction result "spam" is displayed in a large, bold, white font.

Conclusion générale

Dans ce projet, nous avons créé un modèle de classification de spam pour les messages SMS (Short Message Service).

Nous avons également créé une interface graphique où l'utilisateur peut simplement entrer un message et le site lui indiquera si le message est un spam ou non.

Le problème de ce projet est que le modèle n'a pas la capacité de prédire tous les SMS qui sont des spams, Car le dataset est déséquilibré et ne contient pas beaucoup d'exemples, et les textes des SMS ne sont pas au bon format de langage. Ils contiennent des mots tels que 'idk', qui signifie 'I don't know', Etc...