جـامعة الحسن الثـاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSAM
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

MASTER BIG DATA ET INTERNET DES OBJETS

Rapport de projet pour la création d'un modèle de détection des courriers électroniques (Spam/Ham)

Rédigé par:

HAZMIRI Younes

EL ABDOUNI Abderrazzaq

Proposé par :

Pr. BAHASSINE Said

Table of Contents

I. Introduction :

L'objectif de ce projet est d'analyser les courriers électroniques et de les classifier en tant que spam ou ham (non-spam). Nous avons utilisé des techniques de text mining pour nettoyer et prétraiter les données, puis entraîné plusieurs modèles de classification pour prédire la nature des courriers électroniques.

II. À propos de l'ensemble de données

1. Contexte :

Le SMS Spam Collection est un ensemble de messages SMS étiquetés qui ont été collectés dans le cadre de recherches sur le spam SMS. Il contient un ensemble de messages SMS en anglais comprenant 5 574 messages, étiquetés comme étant soit ham (légitimes) soit spam.

2. Contenu :

Les fichiers contiennent un message par ligne. Chaque ligne est composée de deux colonnes : v1 contient l'étiquette (ham ou spam) et v2 contient le texte brut.

Ce corpus a été collecté à partir de sources gratuites ou gratuites à des fins de recherche sur Internet :

- Une collection de 425 messages SMS spam a été extraite manuellement du site Web Grumbletext. Il s'agit d'un forum britannique où les utilisateurs de téléphones portables font des déclarations publiques sur les messages spam SMS, la plupart du temps sans signaler le message spam reçu.
- Un sous-ensemble de 3 375 messages ham (légitimes) choisis au hasard dans le corpus SMS de la NUS (NSC), qui est un ensemble de données d'environ 10 000 messages légitimes collectés à des fins de recherche au département d'informatique de la National University of Singapore.
- Une liste de 450 messages ham (légitimes) SMS collectés à partir de la thèse de doctorat de Caroline Tag,
- Enfin, le corpus SMS Spam v.0.1 Big. Il contient 1 002 messages ham (légitimes) et 322 messages spam

Inspiration :

Pouvons-nous utiliser cet ensemble de données pour construire un modèle de prédiction qui classera de manière précise quels textes sont des spams ?

III. Les étapes suivies :

1. Chargement des données :

Nous avons utilisé un ensemble de données contenant des courriers électroniques étiquetés comme spam ou ham. Les données ont été chargées à l'aide de la bibliothèque pandas ainsi que d'autre.

```
In [1]: import pandas as pd
        import numpy as np
        import nltk
        import matplotlib.pyplot as plt
        import string
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        import re
        from sklearn.model_selection import train_test_split
        from collections import Counter
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.svm import SVC
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn import metrics
```

Ce code importe les bibliothèques nécessaires et les classes/modules spécifiques pour effectuer des tâches liées au texte et à l'apprentissage automatique. Ces importations sont nécessaires pour les étapes ultérieures du code, qui comprennent le prétraitement du texte, la division des données en ensembles d'entraînement et de test, la création de modèles de classification et l'évaluation de leurs performances.

```
In [2]: #DATA LOADING

In [3]: mail = pd.read_csv('spam.csv', encoding='latin-1')

In [4]: mail
```

Out[4]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will Ì_ b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

5572 rows × 5 columns

Ce code charge un fichier CSV appelé "spam.csv" dans un objet DataFrame appelé "mail" à l'aide de la fonction **pd.read_csv()**. Le paramètre **encoding='latin-1'** est utilisé pour spécifier l'encodage des caractères du fichier CSV.

```
In [5]: #sms.drop(['Unnamed: 2','Unnamed: 3', 'Unnamed: 4'], axis = 1, inplace = True)
mail.dropna(how="any", inplace=True, axis=1)
mail.columns = ['label', 'message']
mail
```

Out[5]:

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

Ce code supprime les colonnes qui contiennent des valeurs manquantes dans le DataFrame "mail" à l'aide de la méthode **dropna()** avec les paramètres **how="any" et axis=1**. Cela signifie que si une colonne contient au moins une valeur manquante, elle sera supprimée.

Ensuite, le code renomme les colonnes du DataFrame en utilisant l'attribut columns et la liste ['label', 'message']. Cela donne à chaque colonne du DataFrame un nouveau nom, où 'label' représente l'étiquette (ham ou spam) et 'message' représente le contenu du message.

2. Analyse exploratoire des données (EDA) :

Nous avons effectué une analyse exploratoire des données pour comprendre la répartition des étiquettes (spam/ham) et la longueur des messages. Nous avons également examiné les statistiques des messages ham et spam séparément.

```
In [6]: #Exploratory Data Analysis (EDA)
```

```
In [7]: mail.describe()
```

Out[7]:

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

```
In [8]: mail.groupby('label').describe()
```

Out[8]:

	message			
	count	**unique**	**top**	**freq**
label				
ham	4825	4516	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

La méthode describe() renvoie un résumé statistique du DataFrame "mail". Voici ce que chaque mesure représente :

- count : le nombre total d'observations non manquantes pour chaque colonne
- unique : le nombre de valeurs uniques pour chaque colonne
- top : la valeur la plus fréquente dans chaque colonne
- freq : le nombre de fois que la valeur la plus fréquente apparaît dans chaque colonne

Pour notre cas, on remarque qu'on a deux label (ham – spam) dont ham se répète 4825 fois tandis que spam a une valeur de 747, ainsi que le message « Sorry, I'll call later » est celui le plus fréquent pour les mails non-spam.

In [9]:
```
mail['label_num'] = mail.label.map({'ham':0, 'spam':1})
mail
```

Out[9]:

	label	message	label_num
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0
...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	1
5568	ham	Will Ì_ b going to esplanade fr home?	0
5569	ham	Pity, * was in mood for that. So...any other s...	0
5570	ham	The guy did some bitching but I acted like i'd...	0
5571	ham	Rofl. Its true to its name	0

5572 rows × 3 columns

Cette instruction crée une nouvelle colonne dans le DataFrame "mail" appelée "label_num". Cette colonne contient la conversion des valeurs de la colonne "label" en valeurs numériques. Plus précisément, elle attribue la valeur 0 à toutes les occurrences du label "ham" et la valeur 1 à toutes les occurrences du label "spam".

Cette transformation permet de représenter les labels de manière numérique, ce qui est souvent nécessaire pour l'entraînement de modèles d'apprentissage automatique qui requièrent des données numériques.

Au fur et à mesure que nous poursuivons notre analyse, nous voulons commencer à réfléchir aux fonctionnalités que nous allons utiliser. Plus notre connaissance du domaine sur les données est bonne, meilleure est notre capacité à concevoir plus de fonctionnalités à partir de celles-ci.

In [10]:
```
mail['message_len'] = mail.message.apply(len)
mail
```

Out[10]:

	label	message	label_num	message_len
0	ham	Go until jurong point, crazy.. Available only ...	0	111
1	ham	Ok lar... Joking wif u oni...	0	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1	155
3	ham	U dun say so early hor... U c already then say...	0	49
4	ham	Nah I don't think he goes to usf, he lives aro...	0	61
...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	1	161
5568	ham	Will Ì_ b going to esplanade fr home?	0	37
5569	ham	Pity, * was in mood for that. So...any other s...	0	57
5570	ham	The guy did some bitching but I acted like i'd...	0	125
5571	ham	Rofl. Its true to its name	0	26

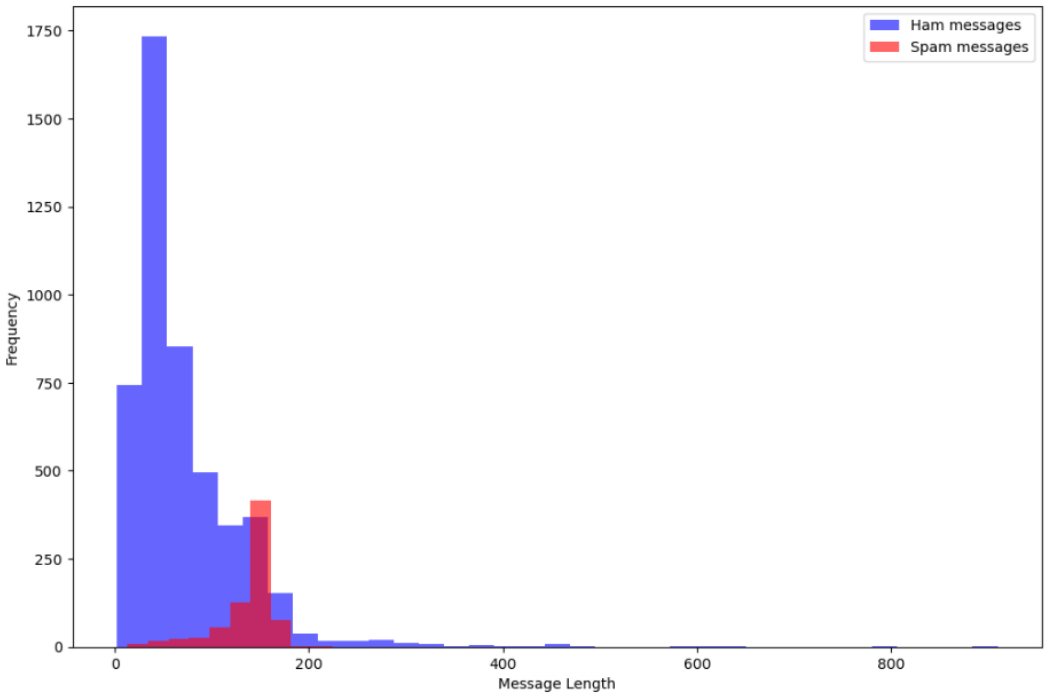
5572 rows × 4 columns

Cette instruction crée une nouvelle colonne dans le DataFrame "mail" appelée "message_len". Cette colonne contient la longueur de chaque message de la colonne "message".

In [11]:
```
plt.figure(figsize=(12, 8))

mail[mail.label=='ham'].message_len.plot(bins=35, kind='hist', color='blue',
                                         label='Ham messages', alpha=0.6)
mail[mail.label=='spam'].message_len.plot(kind='hist', color='red',
                                         label='Spam messages', alpha=0.6)
plt.legend()
plt.xlabel("Message Length")
```

Out[11]: Text(0.5, 0, 'Message Length')



Ce code génère un histogramme de la longueur des messages dans le DataFrame "mail", en différenciant les messages "ham" (non-spam) et les messages "spam".

Bien, nous remarquons que grâce à l'EDA, nous avons pu découvrir une tendance selon laquelle les messages de spam tendent à avoir plus de caractères.

```
In [12]: mail[mail.label=='ham'].describe()
```

Out[12]:

	label_num	message_len
count	4825.0	4825.000000
mean	0.0	71.023627
std	0.0	58.016023
min	0.0	2.000000
25%	0.0	33.000000
50%	0.0	52.000000
75%	0.0	92.000000
max	0.0	910.000000

```
In [13]: mail[mail.label=='spam'].describe()
```

Out[13]:

	label_num	message_len
count	747.0	747.000000
mean	1.0	138.866131
std	0.0	29.183082
min	1.0	13.000000
25%	1.0	132.500000
50%	1.0	149.000000
75%	1.0	157.000000
max	1.0	224.000000

Ces lignes de code permettent de calculer les statistiques descriptives pour les messages « ham » et « spam » pour nos données. Cela inclut le compte, la moyenne, l'écart-type, les valeurs minimum et maximum, ainsi que les quartiles (25%, 50% et 75%) de la longueur de ces messages.

On peut visualiser le message ayant le maximum de caractère dans la catégories de « ham », soit 910 caractères, pour faire cela :

```
In [14]: mail[mail.message_len == 910].message.iloc[0]
```

```
Out[14]: "For me the love should start with attraction.i should feel that I need her every time around me.she should be the first thing
         which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then w
         hen my every breath has her name.my life should happen around her.my life will be named to her.I would cry for her.will give al
         l my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the
         craziest things for her.love will be when I don't have to proove anyone that my girl is the most beautiful lady on the whole pl
         anet.I will always be singing praises for her.love will be when I start up making chicken curry and end up makiing sambar.life
         will be the most beautiful then.will get every morning and thank god for the day because she is with me.I would like to say a l
         ot..will tell later.."
```

3. Prétraitement des données :

Nous avons appliqué plusieurs étapes de prétraitement pour nettoyer les messages. Cela comprenait la suppression des balises HTML, des caractères non alphabétiques, des mentions et des hashtags. Nous avons également effectué la suppression des mots vides (stop words) et la racinisation des mots à l'aide d'un stemmer.

In [15]:
```
#TEXT PreProcessing
```

In [16]:
```
# Initialize the stemmer and stopwords
stemmer = PorterStemmer()
stopwords = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])

def clean_text(text):
    # Remove HTML tags
    text = re.sub('<.*?>', '', text)
    # Remove non-alphabetic characters and convert to lowercase
    text = re.sub('[^a-zA-Z]', ' ', text).lower()
    # Remove Mentions
    text = re.sub(r'@\S+', '', text)
    # Remove Hashtags
    text = re.sub(r'#\S+', '', text)
    # Remove stopwords and stem the words
    words = [stemmer.stem(w) for w in text.split() if w not in stopwords]
    # Join the words back into a string
    cleaned_text = ' '.join(words)
    return text
```

Le code ci-dessus définit une fonction clean_text qui prend un texte en entrée et effectue plusieurs étapes de prétraitement pour nettoyer le texte.

Voici les étapes effectuées par la fonction clean_text :

- Suppression des balises HTML : La fonction utilise l'expression régulière **re.sub('<.*?>', '', text)** pour supprimer toutes les balises HTML du texte.
- Suppression des caractères non alphabétiques : La fonction utilise l'expression régulière **re.sub('[^a-zA-Z]', ' ', text)** pour remplacer tous les caractères non alphabétiques par des espaces.
- Conversion en minuscules : La fonction utilise la méthode **lower()** pour convertir le texte en minuscules.
- Suppression des mentions : La fonction utilise l'expression régulière **re.sub(r'@\S+', '', text)** pour supprimer toutes les mentions (texte commençant par '@') du texte.
- Suppression des hashtags : La fonction utilise l'expression régulière **re.sub(r'#\S+', '', text)** pour supprimer tous les hashtags (texte commençant par '#') du texte.
- Suppression des stopwords et racinisation des mots : La fonction divise le texte en mots à l'aide de la méthode **split()** et vérifie si chaque mot n'est pas présent dans l'ensemble des stopwords défini précédemment. Les mots qui ne sont pas des stopwords sont racinisés à l'aide de l'objet stemmer (instance de PorterStemmer).

On a ajouté ces mots : ‘u’, ‘ü’, ‘ur’, ‘4’, ‘2’, ‘im’, ‘dont’, ‘doin’, ‘ure’. Vu qu’en anglais, on peut trouver des abréviations telles que 4 ➔ ‘for’ ou 2➔ ‘to’.

- Reconstitution du texte : Les mots nettoyés sont ensuite joints pour reconstituer le texte nettoyé à l'aide de l'instruction **cleaned_text = ' '.join(words).**

La fonction renvoie le texte nettoyé.

In [17]:
```
mail['clean_msg'] = mail.message.apply(clean_text)

mail
```

Out[17]:

	label	message	label_num	message_len	clean_msg
0	ham	Go until jurong point, crazy.. Available only ...	0	111	go until jurong point crazy available only ...
1	ham	Ok lar... Joking wif u oni...	0	29	ok lar joking wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1	155	free entry in a wkly comp to win fa cup fina...
3	ham	U dun say so early hor... U c already then say...	0	49	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...	0	61	nah i don t think he goes to usf he lives aro...
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	1	161	this is the nd time we have tried contact u...
5568	ham	Will Ì_ b going to esplanade fr home?	0	37	will b going to esplanade fr home
5569	ham	Pity, * was in mood for that. So...any other s...	0	57	pity was in mood for that so any other s...
5570	ham	The guy did some bitching but I acted like i'd...	0	125	the guy did some bitching but i acted like i d...
5571	ham	Rofl. Its true to its name	0	26	rofl its true to its name

5572 rows × 5 columns

Ce code ajoute une nouvelle colonne appelée "clean_msg" au DataFrame "mail". Cette colonne contient les messages du DataFrame "mail" après avoir été nettoyés en utilisant la fonction "clean_text". La fonction **"apply"** est utilisée pour appliquer la fonction "clean_text" à chaque valeur de la colonne "message" du DataFrame "mail" et stocker les résultats dans la nouvelle colonne "clean_msg".

In [18]:
```
words = mail[mail.label=='ham'].clean_msg.apply(lambda x: [word.lower() for word in x.split()])
ham_words = Counter()

for msg in words:
    ham_words.update(msg)

print(ham_words.most_common(50))
```

```
[('i', 2948), ('you', 1944), ('to', 1554), ('the', 1126), ('a', 1060), ('u', 1026), ('and', 857), ('in', 820), ('me', 772), ('m
y', 750), ('is', 732), ('it', 713), ('that', 558), ('of', 525), ('for', 501), ('s', 490), ('have', 440), ('can', 440), ('so', 4
35), ('but', 434), ('your', 417), ('not', 415), ('are', 414), ('m', 412), ('t', 393), ('on', 393), ('do', 384), ('at', 378),
('we', 355), ('if', 354), ('will', 341), ('be', 335), ('gt', 318), ('lt', 316), ('get', 311), ('how', 304), ('now', 300), ('n
o', 298), ('just', 293), ('ok', 287), ('when', 287), ('up', 287), ('what', 273), ('with', 272), ('ll', 265), ('go', 253), ('thi
s', 252), ('all', 245), ('ur', 241), ('know', 236)]
```

In [19]:
```
words = mail[mail.label=='spam'].clean_msg.apply(lambda x: [word.lower() for word in x.split()])
spam_words = Counter()

for msg in words:
    spam_words.update(msg)

print(spam_words.most_common(50))
```

```
[('to', 688), ('a', 391), ('call', 370), ('you', 299), ('your', 264), ('free', 228), ('the', 206), ('for', 204), ('now', 203),
('or', 192), ('u', 186), ('p', 180), ('txt', 170), ('is', 158), ('on', 144), ('ur', 144), ('have', 135), ('mobile', 129), ('fro
m', 128), ('text', 126), ('stop', 126), ('and', 122), ('claim', 113), ('with', 109), ('reply', 104), ('t', 99), ('s', 99), ('ww
w', 98), ('of', 97), ('prize', 93), ('this', 89), ('get', 86), ('our', 85), ('only', 84), ('in', 82), ('are', 80), ('just', 7
8), ('no', 76), ('cash', 76), ('won', 76), ('uk', 74), ('win', 72), ('nokia', 71), ('i', 70), ('send', 70), ('new', 69), ('c',
63), ('urgent', 63), ('week', 60), ('out', 60)]
```

Ce code effectue l'analyse des mots les plus fréquents dans les messages "ham" et "spam" du DataFrame "mail" après avoir été nettoyés. Voici une explication du code :

La ligne de code **words = mail[mail.label=='ham'].clean_msg.apply(lambda x: [word.lower() for word in x.split()])** récupère les mots de chaque message "ham" en les divisant en mots individuels et en les convertissant en minuscules.

Ensuite, la variable **ham_words** est initialisée comme un objet **Counter**, qui est utilisé pour compter le nombre d'occurrences de chaque mot.

Une boucle est utilisée pour parcourir chaque message de mots et mettre à jour le compteur **ham_words** avec les mots du message.

Enfin, la ligne de code **print(ham_words.most_common(50))** affiche les 50 mots les plus fréquents dans les messages "ham".

Même chose pour les messages « spam ».

4. Création de la matrice document-terme :

Nous avons utilisé la classe **CountVectorizer** de la bibliothèque **scikit-learn** pour créer une représentation numérique des messages. Cette matrice document-terme a été utilisée comme entrée pour les modèles de classification.

In [20]:
```
#SplitDATA and Vectorize
```

In [21]:
```
# split X and y into training and testing sets

X = mail.clean_msg
y = mail.label_num
print(X.shape)
print(y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5572,)
(5572,)
(4179,)
(1393,)
(4179,)
(1393,)
```

Dans ce code, nous avez utilisé la fonction **train_test_split** pour diviser les données en ensembles d'entraînement et de test. Les variables X et y contiennent respectivement les données des messages nettoyés et les étiquettes de classe.

La commande **train_test_split(X, y, random_state=1)** divise les données en utilisant les proportions par défaut, c'est-à-dire 75% pour l'ensemble d'entraînement et 25% pour l'ensemble de test. La graine aléatoire **random_state=1** est utilisée pour garantir que la division des données est la même à chaque exécution du code, ce qui permet d'obtenir des résultats reproductibles.

Après l'exécution de cette commande, les variables suivantes sont créées :

- X_train : les données des messages nettoyés pour l'ensemble d'entraînement.
- X_test : les données des messages nettoyés pour l'ensemble de test.
- y_train : les étiquettes de classe correspondantes à l'ensemble d'entraînement.
- y_test : les étiquettes de classe correspondantes à l'ensemble de test.

Or, Si nous n'indiquons pas explicitement la valeur de l'argument **test_size** dans la fonction **train_test_split,** sa valeur par défaut est de 0.25. Cela signifie que les données seront divisées de manière à ce que 25% des données soient attribuées à l'ensemble de test et 75% des données à l'ensemble d'entraînement.

```
In [22]: # instantiate the vectorizer
         vect = CountVectorizer()

         # learn training data vocabulary, then use it to create a document-term matrix
         X_train_dtm = vect.fit_transform(X_train)

         # examine the document-term matrix
         print(type(X_train_dtm), X_train_dtm.shape)

         # transform testing data (using fitted vocabulary) into a document-term matrix
         X_test_dtm = vect.transform(X_test)
         print(type(X_test_dtm), X_test_dtm.shape)
```

```
<class 'scipy.sparse._csr.csr_matrix'> (4179, 6664)
<class 'scipy.sparse._csr.csr_matrix'> (1393, 6664)
```

Le morceau de code que vous avez fourni effectue les tâches suivantes :

- Instancie un objet **CountVectorizer,** qui sera utilisé pour convertir les données textuelles en une représentation matricielle terme-document.
- Apprend le vocabulaire à partir des données d'entraînement (X_train) en utilisant la méthode fit_transform du vectoriseur. Cette étape permet à la fois de calibrer le vectoriseur sur les données d'entraînement et de transformer les données d'entraînement en une matrice terme-document (X_train_dtm).
- Affiche le type et la forme de la matrice terme-document X_train_dtm, fournissant des informations sur la structure des données et les dimensions.
- Utilise le vocabulaire appris pour transformer les données de test (X_test) en une matrice terme-document (X_test_dtm) à l'aide de la méthode transform du vectoriseur.
- Affiche le type et la forme de la matrice terme-document X_test_dtm, fournissant des informations sur la structure des données et les dimensions.

5. Entraînement des modèles de classification :

Nous avons entraîné trois modèles de classification : KNN Classifier, Decision Tree et SVM. Chaque modèle a été entraîné sur les données d'entraînement et évalué sur les données de test.

```
In [23]: #Building and evaluating the model

In [24]: SVM = SVC()

In [25]: SVM.fit(X_train_dtm, y_train)
```

```
Out[25]: ▾ SVC
         SVC()
```

```
In [27]: KN = KNeighborsClassifier()

In [28]: KN.fit(X_train_dtm, y_train)

Out[28]: KNeighborsClassifier
         KNeighborsClassifier()
```

```
In [30]: DT = DecisionTreeClassifier()

In [31]: DT.fit(X_train_dtm, y_train)

Out[31]: DecisionTreeClassifier
         DecisionTreeClassifier()
```

Ces deux lignes de code créent un modèle SVM, KNN et DT et le font apprendre à partir des données d'entraînement fournies. Une fois que les modèle sont ajustés, ils peuvent être utilisés pour faire des prédictions sur de nouvelles données.

6. Évaluation des modèles :

Nous avons évalué les performances des modèles en utilisant la précision de prédiction (accuracy) comme mesure de performance. De plus, nous avons affiché les matrices de confusion pour chaque modèle, ce qui permet d'évaluer les performances en termes de vrais positifs, vrais négatifs, faux positifs et faux négatifs.

```
In [26]: # make class predictions for X_test_dtm
         y_pred_class = SVM.predict(X_test_dtm)

         # calculate accuracy of class predictions
         print("=======Accuracy Score==========")
         print(metrics.accuracy_score(y_test, y_pred_class))

         # print the confusion matrix
         print("=======Confision Matrix==========")
         metrics.confusion_matrix(y_test, y_pred_class)

         =======Accuracy Score==========
         0.9849246231155779
         =======Confision Matrix==========

Out[26]: array([[1212,    1],
                [  20,  160]], dtype=int64)
```

In [29]:
```
y_pred_class = KN.predict(X_test_dtm)

# calculate accuracy of class predictions
print("=======Accuracy Score===========")
print(metrics.accuracy_score(y_test, y_pred_class))

# print the confusion matrix
print("=======Confision Matrix===========")
print(metrics.confusion_matrix(y_test, y_pred_class))
```

```
=======Accuracy Score===========
0.9296482412060302
=======Confision Matrix===========
[[1212    1]
 [  97   83]]
```

In [32]:
```
y_pred_class = DT.predict(X_test_dtm)

# calculate accuracy of class predictions
print("=======Accuracy Score===========")
print(metrics.accuracy_score(y_test, y_pred_class))

# print the confusion matrix
print("=======Confision Matrix===========")
print(metrics.confusion_matrix(y_test, y_pred_class))
```

```
=======Accuracy Score===========
0.9655419956927495
=======Confision Matrix===========
[[1190   23]
 [  25  155]]
```

Ces codes utilisent les modèles SVM (Support Vector Machine), DT (Decision Tree) et KNN (K-Nearest Neighbors) préalablement entraînés pour effectuer des prédictions sur des données de test. Ils calculent ensuite le score d'exactitude (accuracy) pour évaluer les performances du modèle et affichent la matrice de confusion qui représente les résultats des prédictions.

7. Comparaison des modèles :

En arrivant à la fin, nous pouvons conclure en effectuant une brève comparaison et en déterminant le meilleur modèle à utiliser.

In [34]:
```
# Classifiers
names = [
    "KNN Classifier",
    "Decision Tree",
    "SVM",
]

models = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    SVC(),

]
```

In [35]:
```
def score(X_train, y_train, X_test, y_test, names, models):
    score_df = pd.DataFrame()
    score_train = []
    confusion_matrices = []

    for name, model in zip(names, models):
        model.fit(X_train, y_train)
        y_pred_class = model.predict(X_test)
        score_train.append(metrics.accuracy_score(y_test, y_pred_class))
        confusion_matrices.append(metrics.confusion_matrix(y_test, y_pred_class))

    score_df["Classifier"] = names
    score_df["Training accuracy"] = score_train
    score_df["Confusion matrix"] = confusion_matrices
    score_df.sort_values(by='Training accuracy', ascending=False, inplace=True)

    return score_df
```

In [36]:
```
score(X_train_dtm, y_train, X_test_dtm, y_test, names=names, models=models)
```

Out[36]:

	Classifier	Training accuracy	Confusion matrix
2	SVM	0.984925	[[1212, 1], [20, 160]]
1	Decision Tree	0.966978	[[1188, 25], [21, 159]]
0	KNN Classifier	0.929648	[[1212, 1], [97, 83]]

Le code ci-dessus définit une fonction appelée **"score"** qui prend en paramètres les données d'entraînement et de test, les noms des classifieurs et les modèles correspondants. Cette fonction itère sur chaque modèle, ajuste le modèle aux données d'entraînement, effectue des prédictions sur les données de test, calcule la précision de prédiction et la matrice de confusion correspondante. Les résultats sont ensuite stockés dans un DataFrame qui est trié par ordre décroissant de précision de prédiction. Finalement, le DataFrame contenant les scores des différents classifieurs est retourné.

Lorsque nous comparons les trois modèles de classification, nous observons ce qui suit :

Le modèle SVM présente la meilleure précision, avec un score d'exactitude de 0.984925. Sa matrice de confusion indique qu'il a correctement prédit 1212 échantillons comme étant "ham" (non-spam) et 160 échantillons comme étant "spam". Il a commis une erreur en classant 20 échantillons "ham" comme "spam" et 1 échantillon "spam" comme "ham".

Le modèle Decision Tree obtient également de bons résultats, avec un score d'exactitude de 0.966978. Sa matrice de confusion montre qu'il a correctement prédit 1188 échantillons comme étant "ham" et 159 échantillons comme étant "spam". Il a commis 25 erreurs en classant des échantillons "ham" comme "spam" et 21 échantillons "spam" comme "ham".

Le modèle KNN Classifier obtient un score d'exactitude légèrement inférieur, avec une valeur de 0.929648. Sa matrice de confusion révèle qu'il a correctement classé 1212 échantillons comme étant "ham" et 83 échantillons comme étant "spam". Cependant, il a commis 97 erreurs en classant des échantillons "ham" comme "spam" et 1 échantillon "spam" comme "ham".

En résumé, le modèle SVM présente la meilleure performance globale, suivi par le modèle Decision Tree, tandis que le modèle KNN Classifier obtient des résultats légèrement inférieurs.

8. Tester les modèles :

Pour le modèle DT :

In [37]:
```
#Test the models
```

In [38]:
```
def classify_email(email):
    cleaned_email = clean_text(email)
    email_dtm = vect.transform([cleaned_email])
    prediction = DT.predict(email_dtm)
    if prediction[0] == 0:
        return 'ham'
    else:
        return 'spam'
```

In [39]:
```
email = "Hello, this is a legitimate email."
classification = classify_email(email)
print(classification)  # Output: ham

email = "Congratulations! You have won a prize. Click here to claim."
classification = classify_email(email)
print(classification)
```

```
ham
spam
```

Pour le modèle SVM :

In [46]:
```
def classify_email(email):
    cleaned_email = clean_text(email)
    email_dtm = vect.transform([cleaned_email])
    prediction = SVM.predict(email_dtm)
    if prediction[0] == 0:
        return 'ham'
    else:
        return 'spam'
```

In [47]:
```
email = "Hello, this is a legitimate email."
classification = classify_email(email)
print(classification)  # Output: ham

email = "Congratulations! You have won a prize. Click here to claim."
classification = classify_email(email)
print(classification)
```

```
ham
spam
```

Et enfin KNN :

In [50]:
```
def classify_email(email):
    cleaned_email = clean_text(email)
    email_dtm = vect.transform([cleaned_email])
    prediction = KN.predict(email_dtm)
    if prediction[0] == 0:
        return 'ham'
    else:
        return 'spam'
```

In [51]:
```
email = "Hello, this is a legitimate email."
classification = classify_email(email)
print(classification)  # Output: ham

email = "Congratulations! You have won a prize. Click here to claim."
classification = classify_email(email)
print(classification)
```

```
ham
ham
```

Le code fournit une fonction **classify_email** qui prend un email en entrée, le nettoie en utilisant la fonction **clean_text**, le transforme en une représentation vectorielle à l'aide du vectoriseur **vect**, puis prédit la classification de l'email à l'aide des modèles KNN, SVC et DT entraîné KN, SVM et DT. Si la prédiction est 0, l'email est classé comme "ham", sinon il est classé comme "spam". L'exemple montre comment utiliser cette fonction pour classer deux emails différents et affiche la classification correspondante pour chaque email.

Ce qu'on peut conclure est que les modèle SVM et DT sont mieux à prédire les mails que le modèle KNN.

IV. Interprétation :

En se basant sur les résultats d'exactitude, le modèle SVM a obtenu la meilleure performance avec une exactitude de 98,49%, suivi de près par l'arbre de décision avec une exactitude de 96,41%. Le modèle KNN Classifier a obtenu une performance légèrement inférieure avec une exactitude de 92,96%.

La performance supérieure du modèle SVM peut être attribuée à sa capacité à trouver une frontière de décision complexe entre les classes, ce qui lui permet de bien généraliser et de classer avec précision les nouveaux emails. Les SVM exploitent également les vecteurs de support pour maximiser la marge de séparation entre les classes, ce qui peut aider à réduire les erreurs de classification.

D'autre part, les arbres de décision ont également donné de bons résultats en utilisant une approche basée sur des règles de décision. Les arbres de décision peuvent être efficaces pour capturer les schémas non linéaires et les interactions entre les caractéristiques des emails. Cependant, ils peuvent être sensibles aux variations des données d'entraînement et peuvent avoir tendance à surapprendre (overfit) si l'arbre devient trop complexe.

Le modèle KNN Classifier a donné des performances légèrement inférieures, peut-être en raison de sa sensibilité aux valeurs aberrantes et à la distance entre les points voisins. Il est possible que les caractéristiques des emails ne soient pas bien capturées par une mesure de similarité basée sur les voisins les plus proches, conduisant à une moins bonne performance par rapport aux autres modèles.

V. Conclusion :

Dans ce projet, nous avons réalisé une analyse des courriers électroniques en utilisant des techniques de text mining. Les modèles de classification ont montré de bonnes performances dans la prédiction de la nature des courriers électroniques (spam/ham). Cependant, des améliorations peuvent être apportées en explorant d'autres méthodes de prétraitement des données et en utilisant des ensembles de données plus vastes.