Algorithme Bayésien naïf

Exercice 1 Filtrage de spams

Le jeu de données contient des exemples de SMS et l'indication spam ou ham (non spam).

1. Préparation des données

   (a) Télécharger le jeu de données `sms_spam.csv`, observer les données dans le fichier, les charger dans Python dans un dataframe `sms_raw`. Combien d'exemples de SMS contient le jeu de données ?

   Parmi les exemples, combien y a-t-il de spams et de non-spams ?

   (b) Créer une variable X contenant les SMS et une variable Y contenant le label spam/-ham

   (c) Standardisation des données textes

   Les SMS sont des chaînes de caractères composées de mots, d'espaces, de chiffre et de ponctuation. On va donc supprimer les chiffes et la ponctuation. Il faut aussi voir comment gérer les mots inintéressants comme *and, but, or* et comment séparer les phrases en mots.

   1ère étape : création d'un corpus (collection de documents textes), dans notre cas ce sera une collection de messages SMS.

   On doit maintenant diviser le message en mots.

```
import nltk
nltk.download("punkt")

from nltk.tokenize import sent_tokenize, word_tokenize
token_words=word_tokenize(X[0])
```

   Une autre normalisation nécessaire consiste à réduire les mots à leur racine (processus appelé *stemming*). Le processus prend des mots comme *learned, learning, learns* et les transforme en *learn*. Cela permet aux algorithmes de machine learning de traiter les termes relatifs à un unique concept.

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer

#create an object of class PorterStemmer
porter = PorterStemmer()
lancaster=LancasterStemmer()

#A list of words to be stemmed
word_list = ["Friend", "friendship", "friends", "
   friendships", "stabil", "destabilize", "
   misunderstanding", "railroad", "moonlight", "
   football"]
```

```
10          print("{0:20}{1:20}{2:20}".format("Word","Porter␣
                Stemmer","lancaster␣Stemmer"))
11          for word in word_list:
12              print("{0:20}{1:20}{2:20}".format(word,porter.stem
                    (word),lancaster.stem(word)))
```

Vous pouvez afficher quelques exemples.

On peut ensuite supprimer les *stop words* (comme *to, and*) et la ponctuation.

```
1           tokenizer = nltk.RegexpTokenizer("\w+")
2
3           new_words = tokenizer.tokenize(X[0])
4           new_words
```

```
1           nltk.download('stopwords')
2
3           from nltk.corpus import stopwords
4           from nltk.tokenize import word_tokenize
5
6           porter=PorterStemmer()
7
8           def stemSentence(sentence):
9               tokenizer = nltk.RegexpTokenizer("[a-zA-Z]+")
10
11              token_words = tokenizer.tokenize(sentence)
12              stop_words = set(stopwords.words('english'))
13              stem_sentence=[]
14
15              for word in token_words:
16                  if word not in stop_words:
17                      stem_sentence.append(porter.stem(word))
18                      stem_sentence.append("␣")
19              return "".join(stem_sentence)
```

Vous pouvez afficher quelques exemples.

```
1           x=stemSentence(X[0])
2           x
```

Il faut ensuite appliquer cela à l'ensemble des sms.

(d) Division du texte en mots

La dernière étape consiste à séparer les messages en termes individuels en utilisant un processus appelé *tokenization*. Dans notre cas, un token est un mot. On utilisera la méthode `CountVectorizer` qui convertit une collection de documents texte en une matrice document-terme (DTM, Document-Term Matrix) dans laquelle les lignes indiquent les documents (messages SMS) et les colonnes indiquent les termes (mots). Le paramètre `min_df` définit la fréquence minimale autorisée pour un terme dans la matrice document-terme.

(e) Création des jeux d'entrainement et jeux de test

On va diviser les 2 données en 2 : 75% pour l'entrainement et 25% pour le test.

On vérifie ensuite que la proportion des spams et des message normaux dans le sous-ensemble est proche de celle du jeu complet.

(f) Visualisation des données textes : Nuages de mots
Un nuage de mots est un moyen de visualiser la fréquence des mots dans un texte. Les mots apparaissant le plus souvent sont représentés avec une taille plus importante. Le package `wordcloud` fournit une fonction python permettant de créer ce type de diagramme.
Si le package wordcloud n'est pas disponible, entrez la commande `%pip install wordcloud` dans la cellule précédente.
Vous pouvez afficher les nuages de mots des sous-ensemble de spams et des messages normaux. Quels sont les mots les plus fréquents dans chacun des messages ?

2. Entrainement du modèle : utilisation de Naive Bayes de Scikit-Learn

3. Evaluation des performances du modèle
Evaluer les performances de votre modèle en calculant le nombre de vrais positifs TP, vrais négatifs TN, faux positifs FP, faux négatifs FN et l'accuracy.