Université Mohammed V
Faculté des Sciences
Rabat

2019 - 2020

Analyse des Sentiments
- Cas Twitter -

Avec Python

* Travail réalisé par :
ABBAOUI WAFAE && SAADOUNI CHAIMAE

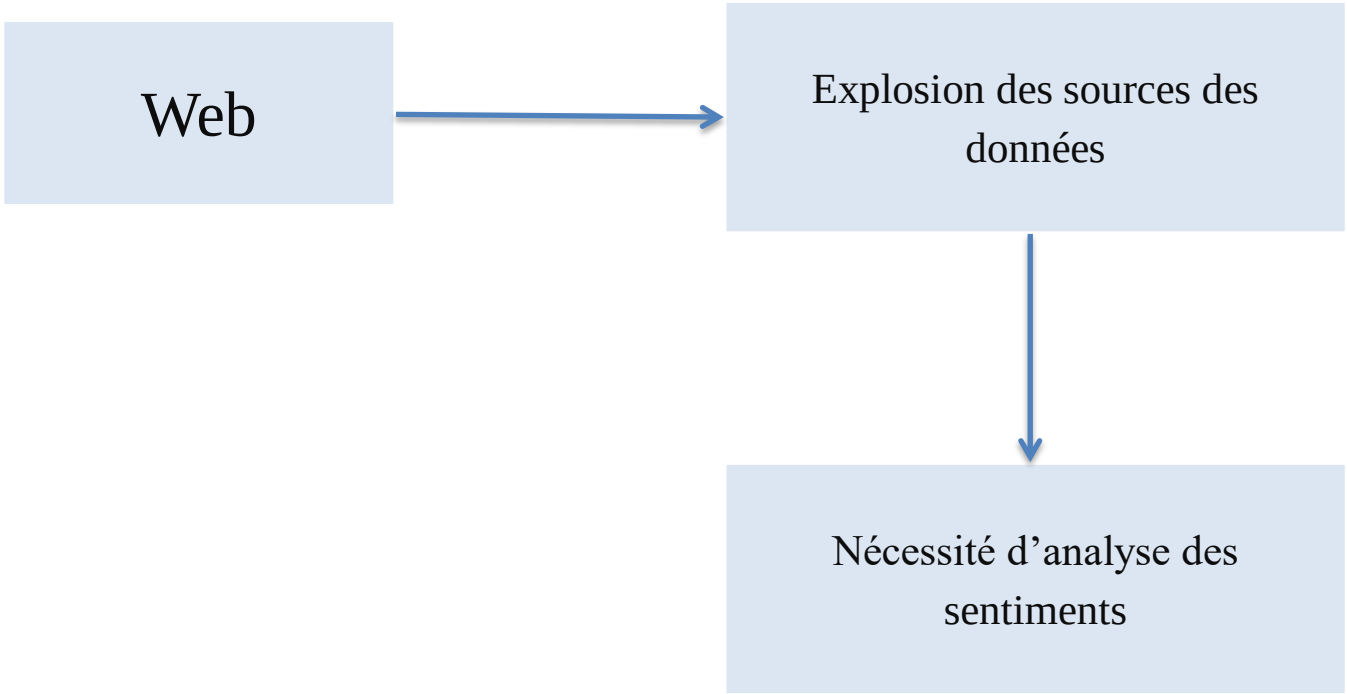
* Professeur Encadrant :
M. MAHMOUDI ABDELHAK

Plan

- Introduction.
- Analyse des sentiments ?
- Pourquoi les données Twitter ?
- Prétraitement des données.
- Words Embedding && Sentiment Classification.
- Topic Modelling à l’aide de LDA.
- Conclusion.

Introduction

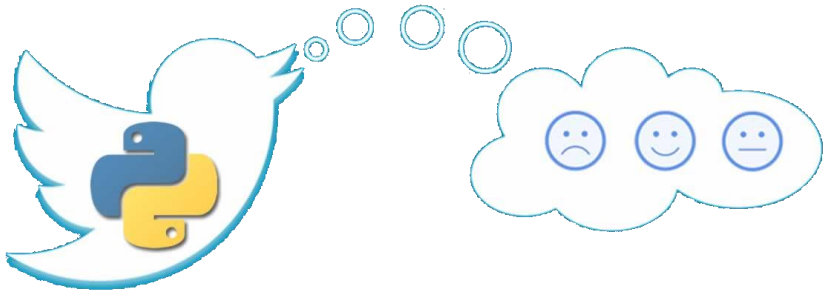
Avec l'avènement du web et l'explosion des sources des données tels que les sites d'avis, les blogs et les micro-blogs est apparu la nécessite d'analyser des millions des postes, des tweets ou d'avis afin de savoir ce que pensent les internautes.



Analyse des sentiments ?

• L'analyse des sentiments est le processus qui consiste à déterminer si un écrit est positif, négatif ou neutre.

• Aussi appelé extraction d'opinion, notre objectif est de déterminer si les données (tweets) sont positives, négatives ou neutres.

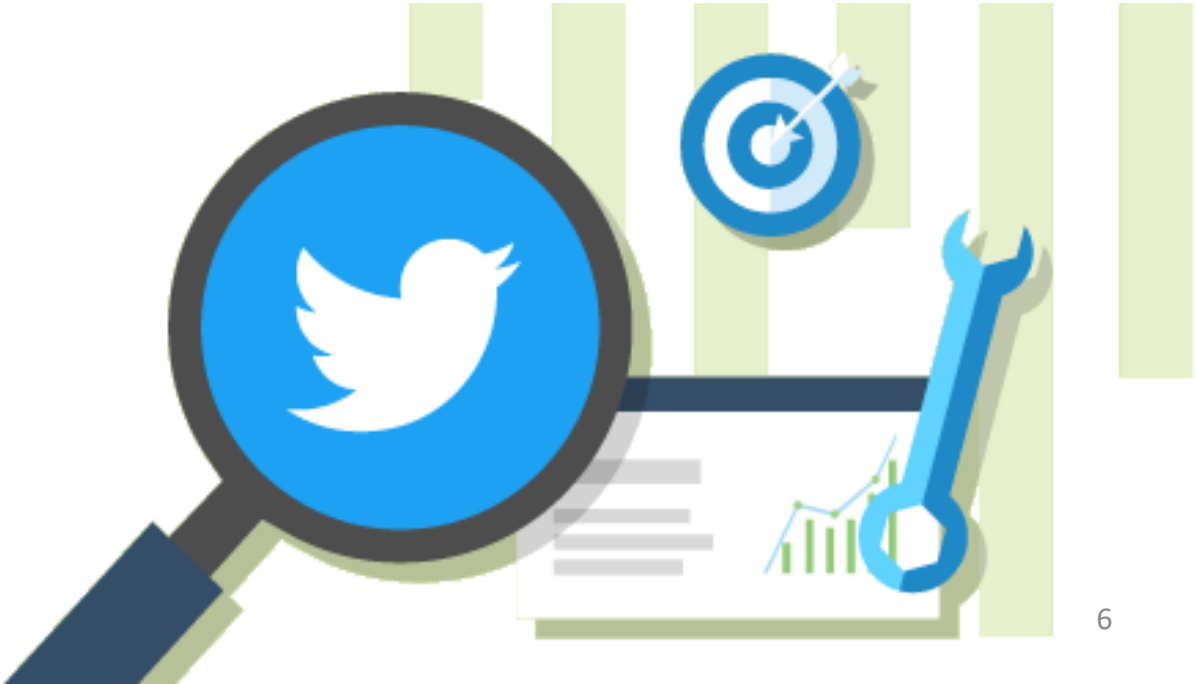


Pourquoi les données Twitter ?

- Site de microblogging populaire.
- Messages texte courts de 140 caractères.
- Plus de 240 millions d'utilisateurs actifs.
- 500 millions de tweets sont générés chaque jour.
- L'audience de Twitter varie d'un homme ordinaire à une célébrité.
- Les utilisateurs discutent souvent des sujets courant (comme covid 19, le sujet de notre dataset).
- Les tweets sont de petite longueur et donc sans ambiguïté.

Prétraitement des données

Une technique utilisée pour convertir les données brutes en un ensemble de données propre.
En d'autres termes, chaque fois que les données sont collectées à partir de différentes sources (Twitter dans notre cas), elles sont collectées au format brut, ce qui n'est pas réalisable pour l'analyse.



* La fonction lower ()

En Python, **lower ()** est une méthode intégrée utilisée pour la gestion des chaînes. La méthode **lower ()** renvoie la chaîne en minuscules de la chaîne donnée. Il convertit tous les caractères majuscules en minuscules. S'il n'existe aucun caractère majuscule, il renvoie la chaîne d'origine.

Exemple :

```
Entrée [18]: df_eng['text'] = df_eng['text'].str.lower() # Lower Casing
```

* La fonction clean ()

Une fonction de la bibliothèque **Preprocessor** qui se charge de prétraitement pour les données de tweet écrites en Python.

La fonction **clean ()** prend actuellement en charge le nettoyage :

- ✓ URLs
- ✓ Hashtags
- ✓ Emojis
- ✓ ……

Exemple :

```
Entrée [17]: import preprocessor as p

             # clean() removes urls, emoticons and hashtags
             df_eng['clean_text'] = df_eng['text'].apply(lambda text: p.clean(text))
```

* Stop words

Dans le NLP, les mots inutiles (données) sont appelés **stop words**.

Stop words: Un **stop word** est un mot couramment utilisé (comme «le», «a», «un», «dans») qu'un moteur de recherche a été programmé pour ignorer, à la fois lors de l'indexation des entrées pour la recherche et lors de leur récupération. à la suite d'une requête de recherche.

Nous ne voudrions pas que ces mots prennent de la place dans notre base de données ou prennent un temps de traitement précieux. Pour cela, nous pouvons les supprimer facilement, en stockant une liste de mots que nous considérons comme des **stop words**.

NLTK (Natural Language Toolkit) en python a une liste de **stop words** stockés dans 16 langues différentes.

Pour vérifier la liste des mots vides, nous pouvons taper les commandes suivantes :

Entrée [19]:
```
from nltk.corpus import stopwords
", ".join(stopwords.words('english'))
```

Out[19]:
```
"i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, h
is, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, wh
om, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing,
a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during,
before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when,
where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very,
s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, d
idn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, nee
dn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't"
```

Remarque: Nous pouvons même modifier la liste en ajoutant les mots de notre choix dans le fichier english.txt dans le répertoire des stop words.

Suppression des stop words avec NLTK

Le programme suivant supprime les stop words :

Entrée [20]:
```
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df_eng['clean_text'] = df_eng['clean_text'].apply(lambda text: remove_stopwords(text))
```

* Les ponctuations

Suppression des ponctuations

Le programme suivant supprime les ponctuations :

Entrée [21]:
```
# Removal of punctions

PUNCT_TO_REMOVE = string.punctuation
def remove_punctuation(text):
    """custom function to remove the punctuation"""
    return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

df_eng['clean_text'] = df_eng['clean_text'].apply(lambda text: remove_punctuation(text))
```

* Stemming

L'objectif principal de **Stemming** est de réduire les mots à leur forme racine.
Cette technique de NLP fonctionne sur le principe que certains mots ayant une orthographe légèrement différente mais presque la même signification doivent être dans le même jeton.
Ainsi, nous coupons les suffixes des mots pour un traitement efficace.

Il existe plusieurs types d'algorithmes de **stemming** disponibles et l'un des plus célèbres est le **PorterStemmer**, qui est largement utilisé.

Exemple :

Entrée [22]:
```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()
def stem_words(text):
    return " ".join([stemmer.stem(word) for word in text.split()])

df_eng['text_stemming'] = df_eng['clean_text'].apply(lambda text: stem_words(text))
```

* Lemmatization

La **lemmatization**, contrairement au Stemming, réduit correctement les mots fléchis en s'assurant que le mot racine appartient à la langue. Dans la **lemmatization**, le mot racine est appelé **Lemma** . Un lemma (lemmas pluriel ou lemmata) est la forme canonique, la forme du dictionnaire ou la forme de citation d'un ensemble de mots.

Python NLTK fournit un lemmatiseur **WordNet** qui utilise la base de données WordNet pour rechercher des **lemmas** de mots.

Exemple :

Entrée [24]:
```
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
wordnet_map = {"N":wordnet.NOUN, "V":wordnet.VERB, "J":wordnet.ADJ, "R":wordnet.ADV}
def lemmatize_words(text):
    pos_tagged_text = nltk.pos_tag(text.split())
    return " ".join([lemmatizer.lemmatize(word, wordnet_map.get(pos[0], wordnet.NOUN)) for word, pos in pos_tagged_text])

df_eng['clean_text'] = df_eng['clean_text'].apply(lambda text: lemmatize_words(text))
```

* Tokenization

La **tokenization** est l'une des tâches les plus courantes lorsqu'il s'agit de travailler avec des données texte, elle consiste essentiellement à diviser une phrase, une expression, un paragraphe ou un document texte entier en unités plus petites, telles que des mots ou des termes individuels. Chacune de ces petites unités est appelée **tokens**.

La figure ci-dessous visualise cette définition :

Natural Language Processing

['Natural', 'Language', 'Processing']

Exemple :

```
Entrée [25]: from nltk.tokenize import word_tokenize

Entrée [26]: df_eng['clean_text'] = df_eng['clean_text'].apply(lambda text: word_tokenize(text))
```

Word Embeddings && Sentiment Classification

Avant de pouvoir commencer avec la classification, nous devons trouver un moyen de représenter les mots.
Maintenant, il existe deux façons populaires de le faire: **Word Vectors** et **Word Embeddings.**

* Les ***Word Vectors*** sont des vecteurs épars de haute dimension (principalement des 0) où chaque vecteur représente un mot qui est simplement codé à chaud.
* Les ***Word Embeddings***, contrairement aux word Vectors, représentent des mots dans des vecteurs denses. Les mots sont mappés dans un espace significatif où la distance entre les mots est liée à leur similitude sémantique.

* GloVe Embeddings ?

L'algorithme Global Vectors for Word Representation, ou GloVe, GloVe word embedding est un modèle de régression log-bilinéaire global, basé sur la cooccurrence et la factorisation de la matrice afin d'obtenir des vecteurs. GloVe est une approche pour marier à la fois les statistiques globales des techniques de factorisation matricielle comme LSA avec l'apprentissage local basé sur le contexte dans word2vec.

* Ce type de représentation de mots est un modèle d'apprentissage non supervisé qui prend en compte toute information portée par le corpus et non pas la seule information portée par une fenêtre de mots, d'où le nom **GloVe**, pour ***VEcteur GLObal***.

* Et pour préparer embedding layer dans ce mini projet, nous avons utilisé la classe Tokenizer du module keras.preprocessing.text pour créer un dictionnaire de mots à index. Dans le dictionnaire mot-à-index, chaque mot du corpus est utilisé comme clé, tandis qu'un index unique correspondant est utilisé comme valeur pour la clé.

* Sentiment Classification problem

La classification des sentiments consiste à regarder un texte et à dire si quelqu'un aime ou n'aime pas ce dont il parle.

* L'entrée **X** est un morceau de texte, le tweet dans notre cas, et la sortie **Y** est le sentiment que nous voulons prédire, comme le classement par étoiles d'une critique de film.

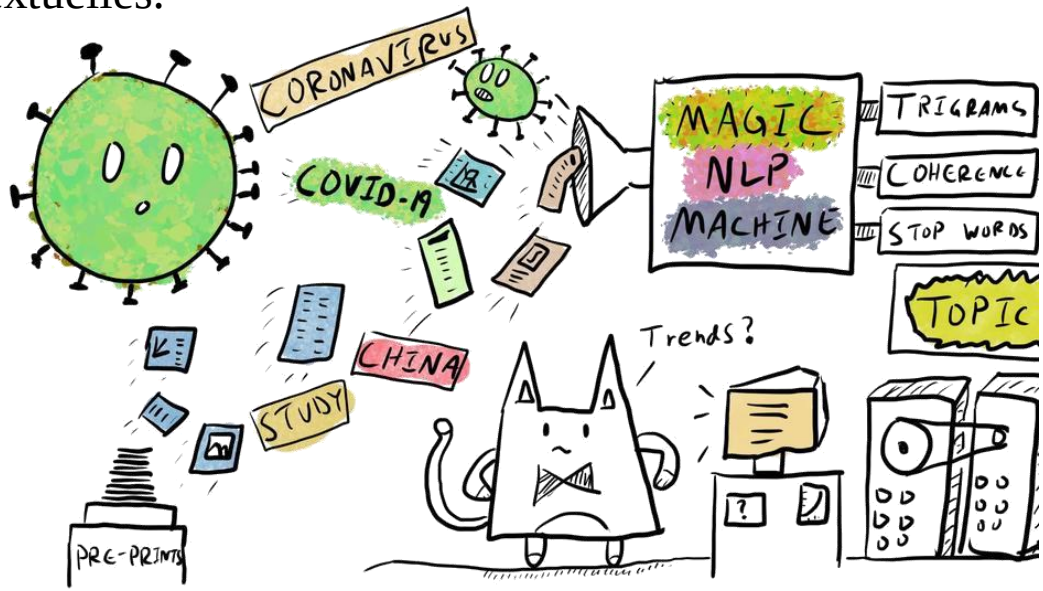
* Dans ce mini projet, nous avons utilisé un jeu de données formé sur un réseau **LSTM** à 6 couches avec des imbrications **GloVe** prétendues. Les réseaux **LSTM** (mémoire à long et court terme) sont des réseaux neuronaux profonds spécialement conçus pour l'entrée de séquences, telles que des phrases qui sont des séquences de mots. Le modèle atteint une précision de 90,25% sur les données d'entraînement (22 563 correctes et 2 437 fausses) et 82,06% de précision sur les données de test.

Topic Modelling à l'aide de LDA

*** Topic Modelling** est une approche non supervisée utilisée pour trouver un ensemble de mots appelés «***topics***» dans un document texte.

* Ces ***topics*** se composent de mots qui se produisent fréquemment ensemble et partagent généralement un thème commun. Et par conséquent, ces ***topics*** avec l'ensemble de mots prédéfini peuvent être utilisés comme facteurs pour décrire au mieux l'ensemble du document.

*** Topic Modelling** nous fournit des méthodes pour organiser, comprendre et résumer de grandes collections de données textuelles.



Il existe de nombreuses approches pour obtenir des sujets à partir d'un document texte. Dans notre projet, nous avons utilisé l'un des modèles de sujets largement utilisés appelé Latent Dirichlet Allocation (LDA).

* Qu'est-ce que LDA ?

Latent Dirichlet Allocation (LDA) est un exemple de topic modelling où chaque document est considéré comme une collection de topics et chaque mot du document correspond à l'un des topics.
Ainsi, étant donné un document, LDA regroupe essentiellement le document en topics où chaque topic contient un ensemble de mots qui décrivent le mieux le topic.

Concernant l'implémentation de LDA en python, la bibliothèque ***Genism*** est le bon choix. Il s'agit d'un outil performant pour topic modelling, l'indexation de documents et la recherche de similitudes avec de grands corpus.

Exemple :

```
Entrée [61]: # Importing Gensim
             import gensim
             from gensim import corpora
```

La figure ci-dessous montre la création du dictionnaire de termes de notre corpus, où chaque terme unique se voit attribuer un index.

Et, elle montre aussi la conversion de la liste de documents (corpus) en matrice de termes de document à l'aide du dictionnaire.

```
# Creating the term dictionary of our corpus, where every unique term is assigned an index.
dictionary = corpora.Dictionary(docs)

# Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.
doc_term_matrix = [dictionary.doc2bow(doc) for doc in docs]
```

Passant à la construction du topic model.

Exemple:

```
Entrée [62]: # Creating the object for LDA model using gensim library
             Lda = gensim.models.ldamodel.LdaModel
```

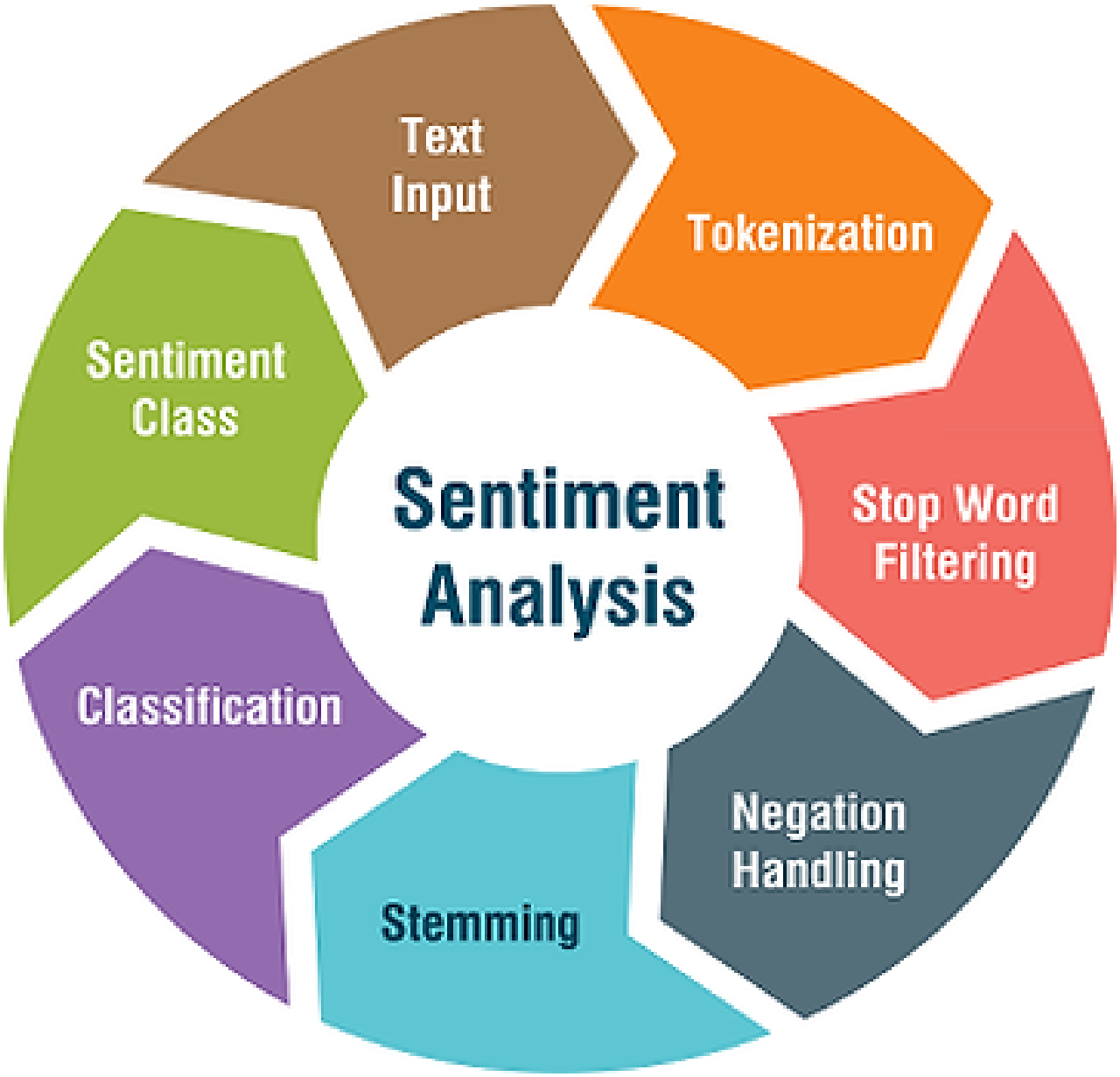
Affichage des rubriques du modèle LDA.

```
Entrée [63]: # Running and Trainign LDA model on the document term matrix.
             ldamodel2 = Lda(doc_term_matrix, num_topics=5, id2word = dictionary, passes=10, iterations=10)
```

Entrée [64]:
```
for idx, topic in ldamodel2.show_topics(formatted=False, num_words= 30):
    print('Topic: {} \nWords: {}'.format(idx+1, '|'.join([w[0] for w in topic])))
```

```
Topic: 1
Words: amp|time|social|worker|health|u|due|work|community|state|service|via|care|s|help|need|share|pm|test|patient|staff|video|
also|county|provide|distancing|hospital|please|first|grant
Topic: 2
Words: get|people|u|take|go|need|life|work|amp|make|stay|world|one|know|please|right|like|you|help|home|thank|time|virus|safe|d
ay|let|many|mask|im|thing
Topic: 3
Words: go|people|get|home|see|stay|love|time|like|one|say|even|im|today|make|thing|think|need|store|still|dont|help|well|pandem
ic|friend|know|use|way|fuck|take
Topic: 4
Words: case|death|new|day|report|u|number|covid19|total|rate|confirm|update|get|first|coronavirus|today|health|test|week|two|re
cover|china|think|double|say|toll|people|year|saturday|march
Topic: 5
Words: amp|test|trump|people|state|get|need|die|new|like|pandemic|support|spread|positive|know|fight|would|big|crisis|quarantin
e|time|act|medical|say|he|u|virus|help|stop|coronavirus
```

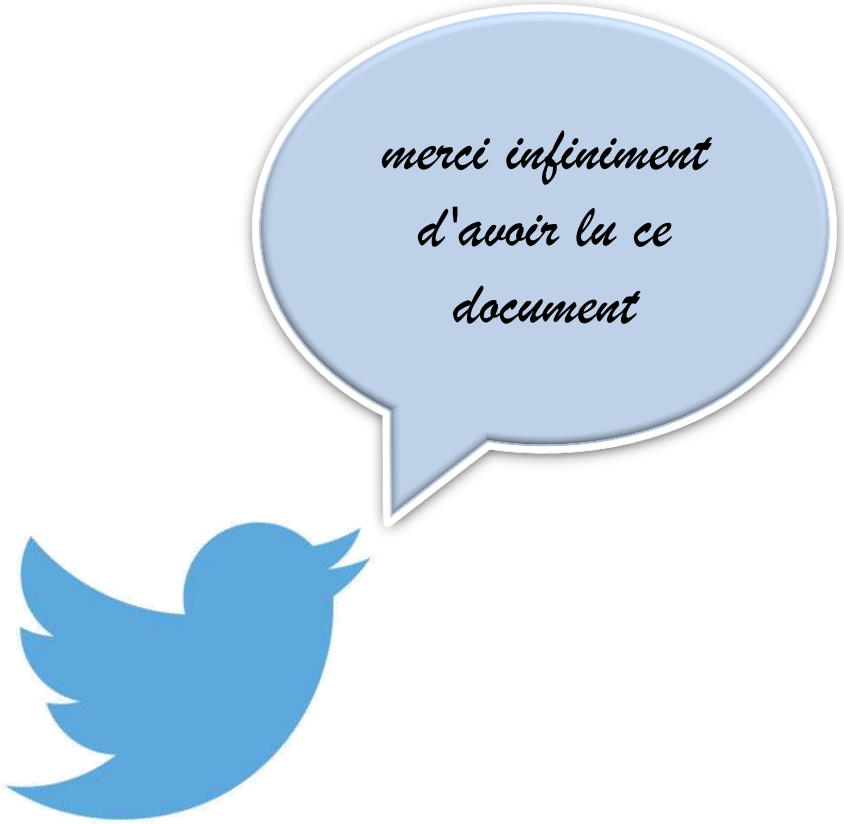
La figure ci-dessus montre que Le modèle LDA a réussi à classer les mots en 5 thèmes abstraits. En lisant les mots individuels d'un sujet, nous pouvons voir que les mots ne sont pas attribués au hasard et que ces mots ensemble ont un sens.


Text Input
Tokenization
Stop Word Filtering
Negation Handling
Stemming
Classification
Sentiment Class
Sentiment Analysis

Conclusion

L'analyse des sentiments apparaît comme un domaine difficile avec de nombreux obstacles car elle implique le traitement du langage naturel. Il dispose d'une grande variété d'applications qui pourraient bénéficier de ses résultats, telles que l'analyse de l'actualité, le marketing, la réponse aux questions, les bases de connaissances, etc. le défi de ce domaine est de développer la capacité de la machine à comprendre les textes comme le font les lecteurs humains.

* Obtenir des informations importantes à partir des opinions exprimées sur Internet, en particulier des blogs de médias sociaux, est vital pour de nombreuses entreprises et institutions, que ce soit en termes de commentaires sur les produits, d'humeur du public ou d'opinions d'investisseurs.
* Dans ce mini projet, nous avons étudié le sentiment des tweets sur la pandémie de Corona Virus. Nous avons utilisé une combinaison de différentes méthodes de prétraitement pour réduire le bruit dans le texte (Text Preprocessing), nous avons rapporté de nombreux résultats expérimentaux, montrant que des méthodes de prétraitement de texte appropriées, y compris la transformation et le filtrage des données, peuvent considérablement améliorer les performances du classificateur.


merci infiniment
d'avoir lu ce
document