Ingénierie des données textuelles

De nombreuses applications utilisent des données textuelles pour faire de la prédiction : détection d'opinions, classification automatique de documents en fonction du contenu : spam - no spam, article sport vs article économie, etc...

La classification se fait de manière tout à fait classique par contre il est indispensable de traiter les documents pour pouvoir les faire interpréter par un classifieur. Le traitement des données textuelles est particulièrement difficile car il dépend des données disponibles et tout traitement n'est pas forcément justifié. Par exemple le fait de convertir tout le texte en minuscule peut faire perdre de l'information (*e.g., Mr Play indique une personne et play un verbe*), la suppression des ponctuations peut avoir des conséquences (*! est très souvent utilisé pour la détection d'opinions*), etc. En outre chaque langue possède aussi ses particularités et les librairies disponibles considèrent souvent l'anglais même s'il existe de plus en plus de ressources en différentes langues comme le français.

Le but de ce notebook est de présenter différentes approches d'ingénierie de données textuelles afin de pré-traiter les données.

Comme nous le verrons tout au cours de ce notebook, il existe de nombreuses librairies qui offrent des fonctionnalités pour pouvoir facilement traiter les données.

Installation

Avant de commencer, il est nécessaire de déjà posséder dans son environnement toutes les librairies utiles. Dans la seconde cellule nous importons toutes les librairies qui seront utiles à ce notebook. Il se peut que, lorsque vous lanciez l'éxecution de cette cellule, une soit absente. Dans ce cas il est nécessaire de l'installer. Pour cela dans la cellule suivante utiliser la commande :

! pip install nom_librairie

Attention : il est fortement conseillé lorsque l'une des librairies doit être installer de relancer le kernel de votre notebook.

Remarque : même si toutes les librairies sont importées dès le début, les librairies utiles pour des fonctions présentées au cours de ce notebook sont ré-importées de manière à indiquer d'où elles viennent et ainsi faciliter la réutilisation de la fonction dans un autre projet.

```
# utiliser cette cellule pour installer les librairies manquantes
# pour cela il suffit de taper dans cette cellule : !pip install nom_librairie_m
# d'exécuter la cellule et de relancer la cellule suivante pour voir si tout se
# recommencer tant que toutes les librairies ne sont pas installées ...

# sous Colab il faut déjà intégrer ces deux librairies

! pip install langdetect
!pip install contractions

! pip install wordcloud
# eventuellement ne pas oublier de relancer le kernel du notebook
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: langdetect in /usr/local/lib/python3.8/dist-
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-package
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: contractions in /usr/local/lib/python3.8/dis
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.8/di
Requirement already satisfied: anyascii in /usr/local/lib/python3.8/dist-pa
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: wordcloud in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.8/dis
Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dis
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/pytho
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-pa
```

```
# Importation des différentes librairies utiles pour le notebook

#Sickit learn met régulièrement à jour des versions et
#indique des futurs warnings.
#ces deux lignes permettent de ne pas les afficher.
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# librairies générales
import pickle
import pandas as pd
from scipy.stats import randint
import numpy as np
import string
import time
import base64
import re
import sys

import contractions

# librairie BeautifulSoup
from bs4 import BeautifulSoup

# librairie affichage
import matplotlib.pyplot as plt
import seaborn as sns
import wordcloud

## detection de language
import langdetect

import nltk
from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

from nltk import RegexpParser
# il est possible de charger l'ensemble des librairies en une seule fois
# décocher le commentaire de la ligne ci-dessous
#nltk.download('all')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag
nltk.download('tagsets')
nltk.download("stopwords")
nltk.download('wordnet')
```

```
from nltk.corpus import stopwords

import spacy
from spacy.tokens import Span
# il faut sélectionner pour quelle langue les traitements vont être faits.

# !python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")
from spacy.lang.fr import French
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Package tagsets is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Pour pouvoir sauvegarder sur votre répertoire Google Drive, il est nécessaire de fournir une autorisation. Pour cela il suffit d'éxecuter la ligne suivante et de saisir le code donné par Google.

```
# pour monter son drive Google Drive local
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, c
```

Corriger éventuellement la ligne ci-dessous pour mettre le chemin vers un répertoire spécifique dans votre répertoire Google Drive :

```
my_local_drive='/content/gdrive/My Drive/Colab Notebooks/ML_FDS'
# Ajout du path pour les librairies, fonctions et données
sys.path.append(my_local_drive)
# Se positionner sur le répertoire associé
%cd $my_local_drive

%pwd
```

```
/content/gdrive/My Drive/Colab Notebooks/ML_FDS
'/content/gdrive/My Drive/Colab Notebooks/ML_FDS'
```

```
# fonctions utilities (affichage, confusion, etc.)
from MyNLPUtilities import *
```

Une première analyse des documents

Très souvent pour commencer à appréhender un texte, l'une des approches consiste à déjà regarder les mots principaux d'un texte. Les word clouds offrent cette fonctionnalité. Il est également utile de connaître la langue du document. Par exemple, cela va permettre de pouvoir utiliser des librairies spécifiques, supprimer des mots inutiles pour cette langue, etc.

Il existe heureusement des librairies spécifiques comme *wordcloud* ou *langdetect*.

Nous présentons par la suite quelques premiers petits traitements pratiques qui peuvent être effectués pour nettoyer un peu les données.

```
import wordcloud

## detection de language
import langdetect
document = "Alice opened the door and found that it led into a small passage, no
How she longed to get out of that dark hall, and wander about among those beds o

# affichage des word clouds
wc = wordcloud.WordCloud(background_color='black', max_words=100,
                         max_font_size=35)
wc = wc.generate(str(document))
fig = plt.figure(num=1)
plt.axis('off')
plt.imshow(wc, cmap=None)
plt.show()


print(" Le document '", document, "' est en  ", langdetect.detect(document))
print ("la phrase il fait beau est en ", langdetect.detect("il fait beau"))
```


```
 Le document ' Alice opened the door and found that it led into a small pas
la phrase il fait beau est en  fr
```

Encodage des données

Les données textuelles sont souvent sujettes à des problèmes d'encodage (“Latin”, “UTF8” etc). Le plus simple est de les convertir dans un format classique (UTF8).

```
import unicodedata
chaine = u"Klüft skräms inför på fédéral électoral große"
chaine=unicodedata.normalize('NFKD', chaine).encode('ascii','ignore')
print (chaine)
```

```
b'Kluft skrams infor pa federal electoral groe'
```

Suppression des tags XML/HTML

Les données textuelles peuvent être issues de pages web, contenir des entêtes, etc.. L'une des premières étapes consistent à les nettoyer pour ne retenir que le texte. La librairie BeautifulSoup permet de récupérer directement le texte en supprimant les tags : https://www.crummy.com/software/BeautifulSoup/bs4/doc/

```
page = """
<!DOCTYPE html>
<html> <head> <title>Machine Learning – Apprentissage</title> </head>
  <body>

<h1>Le cours de Machine Learning est à a FDS </h1> (<a href=https://sciences.edu

 Situé à Montpellier [où il fait toujours beau]
</body> </html>"""
print (page)
```

```
<!DOCTYPE html>
<html> <head> <title>Machine Learning – Apprentissage</title> </head>
  <body>

<h1>Le cours de Machine Learning est à a FDS </h1> (<a href=https://science

 Situé à Montpellier [où il fait toujours beau]
</body> </html>
```

```
from bs4 import BeautifulSoup

def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

page=strip_html (page)
print (page)
```

```
  Machine Learning – Apprentissage

Le cours de Machine Learning est à a FDS  ().

 Situé à Montpellier [où il fait toujours beau]
```

Utilisation d'expressions régulières

De nombreuses modifications peuvent être réalisées en utilisant des expressions régulières (utilisation de la librairie *re*). Par exemple la fonction suivante permet de supprimer les textes entre crochets [].

Nous verrons d'autres exemples d'expressions régulières par la suite.

```
import re
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

page=remove_between_square_brackets(page)
print (page)
```

```
       Machine Learning – Apprentissage

     Le cours de Machine Learning est à a FDS  ().

      Situé à Montpellier
```

Plus loin dans les pré-traitements des documents

La phase de pré-traitement est la phrase de préparation des données pour que ces dernières soient utilisables par un modèle d'apprentissage. Outre l'étape de nettoyage des données, il y a de nombreux pré-traitements qui peuvent ou doivent être effectués en fonction de leur type et de la tâche visée. Par exemple, l'extraction des tokens composant les phrases, la racinisation/"stemmatisation" qui vise à garder la racine des mots, la lemmatisation qui consiste à appliquer une analyse lexicale d'un texte, la suppression de mots vides ou creux (i.e., des mots qui ne sont pas discriminants pour la classification), la détection d'entité nommée, etc.

L'étape de nettoyage peut contenir elle même différentes sous-étapes selon les données (Cf. exemple précédent avec des données HTML) et la tâche visée. Elle comprend souvent la conversion des documents en minuscule et la suppression des signes de ponctuations.

Comme nous l'avons vu précédemment, il existe de nombreuses librairies pour effectuer ces différentes tâches. Dans ce notebook nous nous intéresserons plus particulièrement à :

- la librairie NLTK (Natural Language Toolkit) : http://www.nltk.org
- la librairie SpaCY : https://spacy.io/

Nous présentons comment ces dernières peuvent être utilisées pour réaliser les différents pré-traitements. L'importation de ces librairie se fait de la manière suivante :

```
import nltk
# il est possible de charger l'ensemble des librairies en une seule fois
# décocher le commentaire de la ligne ci-dessous
#nltk.download('all')

import spacy
# il faut sélectionner pour quelle langue les traitements vont être faits.
```

```
import nltk
# il est possible de charger l'ensemble des librairies en une seule fois
# décocher le commentaire de la ligne ci-dessous
#nltk.download('all')

import spacy
# il faut sélectionner pour quelle langue les traitements vont être faits.
nlp = spacy.load("en_core_web_sm")
```

Utilisation de NLTK

NLTK (Natural Language Toolkit - http://www.nltk.org) est une bibliothèque Python développée par Steven Bird et Edward Loper du département d'informatique de l'université de Pennsylvanie. Elle offre de très nombreuses fonctionnalités pour manipuler les textes dans différentes langues dont le français.

L'importation de la librairie se fait par :

```
import nltk
# il est possible de charger l'ensemble des librairies associées en une seule fo
# pour cela décocher le commentaire de la ligne ci-dessous
#nltk.download('all')
```

```
document = "Alice opened the door and found that it led into a small passage, no
How she longed to get out of that dark hall, and wander about among those beds o
```

Sous NLTK, le découpage en phrase peut se faire à l'aide de la fonction *sent_tokenize* :

```
import nltk
nltk.download('punkt')
from nltk import sent_tokenize

phrases = sent_tokenize(document)
for phrase_nltk in phrases:
  print ("phrases : ",phrase_nltk)
```

```
phrases :  Alice opened the door and found that it led into a small passage
phrases :  How she longed to get out of that dark hall, and wander about am
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Découpage en tokens (tokenisation)

Un texte sous python est généralement considéré comme *string*. Il est donc tout à fait possible d'utiliser les fonctions associées comme *lower* (conversion en minuscule) ou la fonction *split* associée pour découper en tokens.

```
print ("conversion document en minuscule")
print (document.lower())

document_splitted = document.split()
print(document_splitted)
```

```
conversion document en minuscule
alice opened the door and found that it led into a small passage, not much
['Alice', 'opened', 'the', 'door', 'and', 'found', 'that', 'it', 'led', 'in
```

Via NLTK, le découpage en tokens se fait via la fonction *word_tokenize*. Contrairement à la fonction *split* précédente, les caractères de ponctuations sont considérés comme tokens.

Remarque : comme nous pouvons le constater les ponctuations sont soient intégrées au dernier mot (*split*), soient correspondent à des tokens. Nous verrons jusque après que NTLK peut les reconnaître spécifiquement via une analyse grammaticale.

```
from nltk.tokenize import word_tokenize
# la liste des tokens de la première phrase
tokens = word_tokenize(phrases[0])
print(tokens)
```

```
['Alice', 'opened', 'the', 'door', 'and', 'found', 'that', 'it', 'led', 'in
```

Etiquetage grammatical (*Part of Speech Tagging*)

L'étiquetage morpho-syntaxique (ou étiquetage grammatical) permet d'associer à chaque mot d'un texte les informations grammaticales correpondantes (e.g. verbe, préposition, ...).
Elle se fait via la fonction *pos_tag*. Elle s'applique à une phrase composée d'un ensemble de tokens et retourne les différents composants de la phrase.

```
from nltk import pos_tag
nltk.download('averaged_perceptron_tagger')

for phrase_nltk in phrases:
  print ("phrases : ",phrase_nltk)
  tokens = word_tokenize(phrase_nltk)
  tokens_tag = nltk.pos_tag(tokens)
  print (tokens_tag)
```

```
phrases :  Alice opened the door and found that it led into a small passage
[('Alice', 'NNP'), ('opened', 'VBD'), ('the', 'DT'), ('door', 'NN'), ('and'
phrases :  How she longed to get out of that dark hall, and wander about am
[('How', 'WRB'), ('she', 'PRP'), ('longed', 'VBD'), ('to', 'TO'), ('get', '
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Il est possible de connaître la liste de tous les tags disponibles :

```
nltk.download('tagsets')
nltk.help.upenn_tagset()
```

```
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
RBR: adverb, comparative
    further gloomier grander graver greater grimmer harder harsher
    healthier heavier higher however larger later leaner lengthier less-
    perfectly lesser lonelier longer louder lower more ...
RBS: adverb, superlative
    best biggest bluntest earliest farthest first furthest hardest
    heartiest highest largest least less most nearest second tightest worst
RP: particle
    aboard about across along apart around aside at away back before behind
    by crop down ever fast for forth from go high i.e. in into just later
    low more off on open out over per pie raising start teeth that through
    under unto up up-pp upon whole with you
SYM: symbol
    % & ' '' ''. ) ). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker
    to
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly
    man baby diddle hush sonuvabitch ...
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
```

```
    VBD: verb, past tense
        dipped pleaded swiped regummed soaked tidied convened halted registered
        cushioned exacted snubbed strode aimed adopted belied figgered
        speculated wore appreciated contemplated ...
    VBG: verb, present participle or gerund
        telegraphing stirring focusing angering judging stalling lactating
        hankerin' alleging veering capping approaching traveling besieging
        encrypting interrupting erasing wincing ...
    VBN: verb, past participle
        multihulled dilapidated aerosolized chaired languished panelized used
        experimented flourished imitated reunifed factored condensed sheared
        unsettled primed dubbed desired ...
    VBP: verb, present tense, not 3rd person singular
        predominate wrap resort sue twist spill cure lengthen brush terminate
        appear tend stray glisten obtain comprise detest tease attract
        emphasize mold postpone sever return wag ...
    VBZ: verb, present tense, 3rd person singular
        bases reconstructs marks mixes displeases seals carps weaves snatches
        slumps stretches authorizes smolders pictures emerges stockpiles
        seduces fizzes uses bolsters slaps speaks pleads ...
    WDT: WH-determiner
        that what whatever which whichever
    WP: WH-pronoun
        that what whatever whatsoever which who whom whosoever
    WP$: WH-pronoun, possessive
        whose
    WRB: Wh-adverb
        how however whence whenever where whereby whereever wherein whereof why
    ``: opening quotation mark
        ` ``
    [nltk_data] Downloading package tagsets to /root/nltk_data...
    [nltk_data]   Package tagsets is already up-to-date!
```

A partir de cet étiquetage, il est donc possible de ne sélectionner que des tokens correspondant à une catégorie.

```
word_tokens = word_tokenize(document)
pos = nltk.pos_tag(word_tokens)
selective_pos = ['NN','VBD']
selective_pos_words = []
for word,tag in pos:
    if tag in selective_pos:
        selective_pos_words.append((word,tag))
print(selective_pos_words)
```

```
    [('opened', 'VBD'), ('door', 'NN'), ('found', 'VBD'), ('led', 'VBD'), ('pas
```

Mots vides *Stop words*

Les mots vides correspondent à des mots qui sont tellement commun qu'il n'est pas nécessaire de les considérer dans l'apprentissage. NLTK possède une liste prédéfinie de mots vides faisant référence aux mots les plus courants. La première fois, il est nécessaire de télécharger les mots vides en utilisant : *nltk.download («stopwords»)*.

```
nltk.download("stopwords")
from nltk.corpus import stopwords
the_stopwords=set(stopwords.words("english"))
print (the_stopwords)
```

```
{'out', 'who', 'had', 'couldn', 'ourselves', 'd', 'our', 'his', 'mustn', 'l
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Il est tout à fait possible de supprimer des stopwords de cette liste :

```
not_stopwords = {'this', 'd', 'o'}
#new_stopwords_list=stopwords
final_stop_words = set([word for word in the_stopwords if word not in not_stopwo

print (final_stop_words)
```

```
{'out', 'who', 'had', 'couldn', 'ourselves', 'our', 'his', 'mustn', 'll', "
```

ou d'étendre la liste des stop words.

```
new_stopwords=['stopword1', 'stopword2']
final_stop_words=final_stop_words.union(new_stopwords)

print (final_stop_words)
```

```
{'out', 'who', 'had', 'couldn', 'ourselves', 'our', 'his', 'mustn', 'll', "
```

Pour supprimer les stopwords d'un document, il suffit de rechercher les tokens qui sont inclus dans les stopwords et de les supprimer.

```
print ("Avant suppression des stopwords")
print (word_tokens)
tokens_Alice=[word for word in word_tokens if word not in the_stopwords]
print ("Après suppression des stopwords")
print (tokens_Alice)
```

```
Avant suppression des stopwords
['Alice', 'opened', 'the', 'door', 'and', 'found', 'that', 'it', 'led', 'in
Après suppression des stopwords
['Alice', 'opened', 'door', 'found', 'led', 'small', 'passage', ',', 'much'
```

Racinisation (*stemming*) **et lemmatisation**

NLTK utilise l'algorithme de racinisation de Porter et propose une fonction de lemmatisation. Il s'agit de deux approches différentes de transformation des flexions en leur radical ou racine. Voir https://fr.wikipedia.org/wiki/Racinisation.

```
import nltk
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
ps=nltk.stem.porter.PorterStemmer()

print ("tokens")
print (tokens_Alice)
print("Stemming")
print([ps.stem(word) for word in tokens_Alice])

print("Lemmatisation")
lem = nltk.stem.wordnet.WordNetLemmatizer()
print([lem.lemmatize(word) for word in tokens_Alice])
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
tokens
['Alice', 'opened', 'door', 'found', 'led', 'small', 'passage', ',', 'much'
Stemming
['alic', 'open', 'door', 'found', 'led', 'small', 'passag', ',', 'much', 'l
Lemmatisation
['Alice', 'opened', 'door', 'found', 'led', 'small', 'passage', ',', 'much'
[nltk_data]   Package omw-1.4 is already up-to-date!
```

NLTK propose aussi un stemmatiseur pour le Français :

```
# un autre stemmatiseur qui accepte le français
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("french")
phrase = "malade malades maladie maladies maladive"
tokens = word_tokenize(phrase)
print ("Avant transformation \n")
print (tokens)
stemmed = [stemmer.stem(word) for word in tokens]
print ("\n Après transformation\n")
print (stemmed)
```

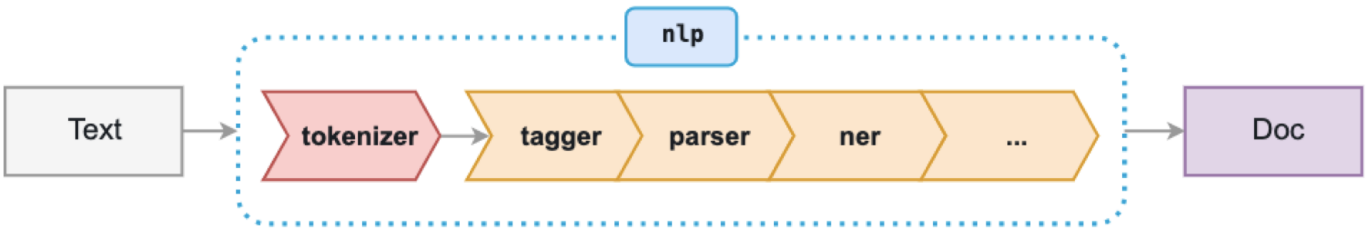
```
Avant transformation

['malade', 'malades', 'maladie', 'maladies', 'maladive']

 Après transformation

['malad', 'malad', 'malad', 'malad', 'malad']
```

Utilisation de Spacy

L'importation de la librairie se fait par :

```
import spacy
```

Spacy utilise un objet particulier, généralement appelé *nlp*, qui va créer un pipeline sur tous les éléments d'un document afin de générer un objet de type doc. Le pipeline de base est le suivant :



Il existe bien entendu différents modèles de pipeline en fonction de la langue.

Les ressources sont par défaut en anglais. Pour le français, il faut au préalable télécharger la ressource associée par :

```
#!python -m spacy download fr
spacy.load('fr_core_news_sm')
```

```
<spacy.lang.fr.French at 0x7fc2f30ba880>
```

```
from spacy.lang.fr import French

# Création d'un objet nlp
nlp = French()

# Créé en traitant une chaine de caractères avec l'objet nlp
doc = nlp("Bonjour monde !")

# Itère sur les tokens dans un Doc
for token in doc:
    print(token.text)

# remise des ressources en anglais pour la suite
nlp = spacy.load("en_core_web_sm")
```

```
Bonjour
monde
!
```

```
document = "Alice opened the door and found that it led into a small passage, no
How she longed to get out of that dark hall, and wander about among those beds o
```

Découpage en phrases, tokenisation et analyse grammaticale

Le découpage en phrases se fait via l'attribut (*sents*) lors de la création du pipeline.

```
# le document est en anglais
nlp = spacy.load("en_core_web_sm")

doc=nlp(document)
for phrases_spacy in doc.sents:
  print ("phrases : ", phrases_spacy)

#sauvegarde des phrases dans un tableau pour les manipulations ultérieures
sentences = [sent.text.strip() for sent in doc.sents]
```

```
phrases :  Alice opened the door and found that it led into a small passage
phrases :  How she longed to get out of that dark hall, and wander about am
```

Spacy permet d'avoir de très nombreuses information sur les tokens : le token, l'index (associé au nombre de caractères), le lemme associé (voir plus loin), s'il s'agit d'un caractère de ponctuation, d'un espace, la forme (un X représente une majuscule et un x une minuscule), sa catégorie (Part of Speech tagging), le tag associé, s'il s'agit dune entité nommée, etc.

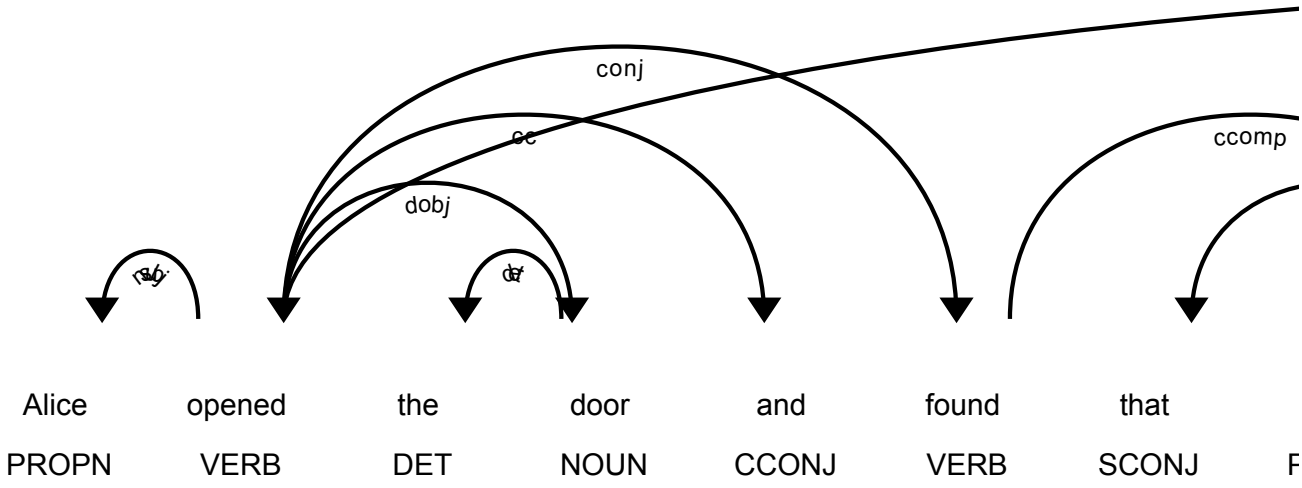
La liste des différents attributs est disponible ici : https://spacy.io/api/token

```
#traitement de la première phrase
first_sentence=nlp(sentences[0])

for token in first_sentence:
    # l'objet token contient différents attributs
    print("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}".format(
        token.text,
        token.idx,
        token.lemma_,
        token.is_punct,
        token.is_space,
        token.shape_,
        token.pos_,
        token.tag_,
        token.ent_type_
    ))
```

```
Alice    0       Alice    False   False   Xxxxx   PROPN   NNP     PERSON
opened   6       open     False   False   xxxx    VERB    VBD
the      13      the      False   False   xxx     DET     DT
door     17      door     False   False   xxxx    NOUN    NN
and      22      and      False   False   xxx     CCONJ   CC
found    26      find     False   False   xxxx    VERB    VBD
that     32      that     False   False   xxxx    SCONJ   IN
it       37      it       False   False   xx      PRON    PRP
led      40      lead     False   False   xxx     VERB    VBD
into     44      into     False   False   xxxx    ADP     IN
a        49      a        False   False   x       DET     DT
small    51      small    False   False   xxxx    ADJ     JJ
passage  57      passage  False   False   xxxx    NOUN    NN
,        64      ,        True    False   ,       PUNCT   ,
not      66      not      False   False   xxx     PART    RB
much     70      much     False   False   xxxx    ADV     RB
larger   75      large    False   False   xxxx    ADJ     JJR
than     82      than     False   False   xxxx    ADP     IN
a        87      a        False   False   x       DET     DT
rat      89      rat      False   False   xxx     NOUN    NN
–        92      –        True    False   –       NOUN    NN
hole     93      hole     False   False   xxxx    NOUN    NN
:        97      :        True    False   :       PUNCT   :
she      99      she      False   False   xxx     PRON    PRP
knelt    103     kneel    False   False   xxxx    VERB    VBD
down     109     down     False   False   xxxx    ADP     RP
and      114     and      False   False   xxx     CCONJ   CC
looked   118     look     False   False   xxxx    VERB    VBD
along    125     along    False   False   xxxx    ADP     IN
the      131     the      False   False   xxx     DET     DT
passage  135     passage  False   False   xxxx    NOUN    NN
into     143     into     False   False   xxxx    ADP     IN
the      148     the      False   False   xxx     DET     DT
loveliest        152      lovely  False   False   xxxx    ADJ     JJS
garden   162     garden   False   False   xxxx    NOUN    NN
you      169     you      False   False   xxx     PRON    PRP
ever     173     ever     False   False   xxxx    ADV     RB
saw      178     see      False   False   xxx     VERB    VBD
.        181     .        True    False   .       PUNCT   .
```

Il est également possible de visualiser les résultats de l'analyse grammaticale :

```
from spacy import displacy
displacy.render(first_sentence, style='dep', jupyter=True, options={'distance':
```



Entité nommée (*name entity*)

Spacy permet également d'extraire les entités nommées d'un texte.

```
example_withnamedentities="Donald Trump was a President of the US and now it is
sentence=nlp(example_withnamedentities)
for entity in sentence.ents:
    print(entity.text + ' - ' + entity.label_ + ' - ' + str(spacy.explain(entity
```

```
Donald Trump - PERSON - People, including fictional
US - GPE - Countries, cities, states
Joe Biden - PERSON - People, including fictional
```

D'autres librairies ou traitements pratiques

Nous présentons ici différentes librairies ou traitements souvents utilisés.

```
#inflect est une librairie qui permet de convertir les nombres en mots
import inflect

phrase="They are 100"
tokens = word_tokenize(phrase)

print ("Nombre à convertir \n")
words = [word for word in tokens if word.isdigit()]
print(words)
p = inflect.engine()
numbertransf = [p.number_to_words(word) for word in tokens if word.isdigit()]

print ("Nombre après conversion \n")
print(numbertransf)
```

```
Nombre à convertir

['100']
Nombre après conversion

['one hundred']
```

```
tokens = [w.lower() for w in tokens]
print (tokens)
```

```
['they', 'are', '100']
```

```
# Suppression de tous les termes qui ne sont pas alphanumériques
words = [word for word in tokens if word.isalpha()]
print(words)
```

```
['they', 'are']
```

```
import contractions

phrase="They're 100"
tokens = word_tokenize(phrase)

def replace_contractions(text):
    return contractions.fix(text)

print ("Avant remplacement\n")
print (phrase)
print ("\nAprès remplacement\n")
laphrase=replace_contractions(phrase)
print (laphrase)
```

```
Avant remplacement

They're 100

Après remplacement

They are 100
```

Les tweets ont une syntaxe très particulière et généralement les traitements se font à l'aide d'expressions régulières.

```
import re
tweet = '#ML is thus a good example :D ;) RT @theUser: see http://ml.example.com
#traitement des émoticones
emoticons_str = r"""
    (?:
        [:=;] # Eyes
        [oO\-]? # Nose (optional)
        [D\)\]\(\]/\\OpP] # Mouth
    )"""

#Prise en compte des éléments qui doivent être regroupés
regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @-mentions
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&amp;+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+'

    r'(?:(?:\d+,?)+(?:\.?\d+)?)', # nombres
    r"(?:[a-z][a-z'\-_]+[a-z])", # mots avec - et '
    r'(?:[\w_]+)', # autres mots
```

```
    r'(?:\S)' # le reste
]

tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASE)

def tokenize(s):
    return tokens_re.findall(s)

def preprocess(s, lowercase=False):
    tokens = tokenize(s)
    if lowercase:
        tokens = [token if emoticon_re.search(token) else token.lower() for toke
    return tokens

# un example de tweet

print ("Un exemple de tweet : \n",tweet)

print ("\nLe tweet avec un processus normal de transformation\n")
print (word_tokenize(tweet))
print ("\nLe tweet avec des expressions régulières\n")
words=preprocess(tweet)
print(words)
```

```
Un exemple de tweet :
 #ML is thus a good example :D ;) RT @theUser: see http://ml.example.com

Le tweet avec un processus normal de transformation

['#', 'ML', 'is', 'thus', 'a', 'good', 'example', ':', 'D', ';', ')', 'RT',

Le tweet avec des expressions régulières

['#ML', 'is', 'thus', 'a', 'good', 'example', ':D', ';)', 'RT', '@theUser',
```

Une petite mise en pratique

Il est temps à présent de mettre en pratique ce que nous avons vu.

Considérez le document suivant :

```
testpratique=[u"""“Curiouser and curiouser!” cried Alice (she was so much surpri
And she went on planning to herself how she would manage it. “They must go by th
     Alice’s Right Foot, Esq.,
       Hearthrug,
         near the Fender,
           (with Alice’s love).
Oh dear, what nonsense I’m talking!”
""",u"""After a time she heard a little pattering of feet in the distance, and s
u"""They were indeed a queer-looking party that assembled on the bank—the birds
The first question of course was, how to get dry again: they had a consultation
At last the Mouse, who seemed to be a person of authority among them, called out
“Ahem!” said the Mouse with an important air, “are you all ready? This is the dr
“Ugh!” said the Lory, with a shiver.
“I beg your pardon!” said the Mouse, frowning, but very politely: “Did you speak
“Not I!” said the Lory hastily.
“I thought you did,” said the Mouse. “—I proceed. ‘Edwin and Morcar, the earls o
“Found what?” said the Duck.
“Found it,” the Mouse replied rather crossly: “of course you know what ‘it’ mean
“I know what ‘it’ means well enough, when I find a thing,” said the Duck: “it’s
```

Il contient différentes phrases plus ou moins longues. Vous pourrez réaliser les opérations en utilisant soit NLTK ou Spacy.

Exercice :

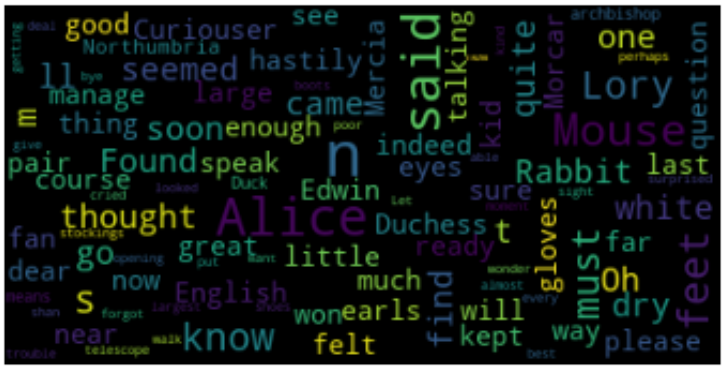
1. Afficher les wordclouds associés aux documents
2. Transformer les documents de telle sorte qu'ils soient en minuscule, qu'il ne possède plus de caractères spéciaux ni uniques.
3. Transformer les en tokens de manière à ce qu'ils ne contiennent plus que des tokens de type NN et VB.
4. Enfin, transformer les tokens pour n'avoir que leur racine.

```

```

Solution :

```
wc = wordcloud.WordCloud(background_color='black', max_words=100,
                         max_font_size=35)
wc = wc.generate(str(testpratique))
fig = plt.figure(num=1)
plt.axis('off')
plt.imshow(wc, cmap=None)
```

```
plt.show()

print ("Document initial ", testpratique)
# suppression des caractères spéciaux
sentence = re.sub(r'[^\w\s]',' ', str(testpratique))
# suppression de tous les caractères uniques
sentence = re.sub(r'\s+[a-zA-Z]\s+', ' ', sentence)

# tokenisation
word_tokens = word_tokenize(sentence)
print ("Premiers tokens après nettoyage des données ", word_tokens)

# ne retenir que les categories NN et VB
# sauvegarde des mots dans un tableau selective_words.
# selective_pos_words contient le mot et son tag
pos = nltk.pos_tag(word_tokens)
selective_pos = ['NN','VB']
selective_pos_words = []
selective_words=[]
for word,tag in pos:
     if tag in selective_pos:
         selective_pos_words.append((word,tag))
         selective_words.append(word)

print ("tokens conservés de catégorie NN ou VB")
print (selective_pos_words)
# racinisation


ps=nltk.stem.porter.PorterStemmer()


print("Racinisation")
print([ps.stem(word) for word in selective_words])
```




```
Document initial  ['“Curiouser and curiouser!” cried Alice (she was so much
Premiers tokens après nettoyage des données  ['Curiouser', 'and', 'curiouse
tokens conservés de catégorie NN ou VB
[('curiouser', 'NN'), ('moment', 'NN'), ('speak', 'VB'), ('telescope', 'NN'
Racinisation
['curious', 'moment', 'speak', 'telescop', 'bye', 'be', 'sight', 'put', 'sh
```

Nous savons maintenant nettoyer nos données et les transformer sous la forme de tokens. Aussi nous allons comment ces derniers peuvent être utilisés comme représentation des documents pour faire .... de la classification. La classification de données textuelles est présentée dans un autre notebook.

Produits payants Colab - Résilier les contrats ici

0 s terminée à 14:36

0 s terminée à 14:36