TP : Indexation des documents textuels avec NLTK

Objectifs

Ce TP vise à découvrir la Natural Language ToolKit (NLTK) et tester ses algorithmes les plus courants, tels que la tokenisation, la racinisation et la lemmatisation, en manipulant quelques exemples textuels, afin de rendre l'étudiant capable d'implémenter un programme d'indexation et de recherche des documents textuels.

I. Prétraitement de Texte avec NLTK

NLTK est une suite de bibliothèques et de programmes pour le traitement symbolique et statistique du langage naturel pour l'anglais écrit dans le langage de programmation Python.

Travail à faire

1. **Etape fondamentale**

   Montez votre Google Drive sur Collaboratory.

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Chargement de NLTK**

   Importer toutes les ressources pour le traitement du langage naturel avec Python.

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

3. **Introduction des éléments textuels**

   a. Tapez le texte directement

```
EXAMPLE_TEXT = "Breast cancer is the most common form of cancer worldwi
de among women, with a high mortality rate. In fact, in most cases, bre
ast cancer leads to death. Nevertheless, thanks to the early detection,
the number of breast cancer related deaths was been reduced in the last
decade. The best tool to carry out the early breast cancer detection
is mammography, where through certain typical signatures like masses an
d microcalcifications can help in the early diagnosis of this dangerous
cancer."
```

b. Lire le texte à partir d'un fichier

```
f = open("/content/drive/MyDrive/INDEXATION/fileTest.txt", "r")
print(f.read())
```

PS : pour lire une seule ligne, il faut utiliser la fonction readline()

*Pour lire un nombre de caractère bien déterminé **n**, il faut utiliser la fonction read(**n**)*

c. Utiliser l'un des ensembles de données de corpus fournit par NLTK

```
from nltk.corpus import state_union
text = state_union.raw("2005-GWBush.txt")
print(text)
```

4. Tokenisation

```
from nltk.tokenize import sent_tokenize, word_tokenize
```

a. Tokenisation des phrases
Extraire les phrases du texte.

```
print(sent_tokenize(EXAMPLE_TEXT))
```

b. Tokenisation des mots
Extraire les mots du texte.

```
print(word_tokenize(EXAMPLE_TEXT))
```

5. Identification des mots vides

```
from nltk.corpus import stopwords
```

a. Afficher les mots vides

```
set(stopwords.words('english'))
```

Q : Afficher la liste des mots vides de la langue française et arabe.

b. Eliminer les mots vides

```
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
```

```
print(filtered_sentence)
```

PS : il faut déclarer "word_tokens" et "stop_words"

6. **Racinisation**

   Utilisation de l'algorithme de *Porter*

   Abréviation d'un texte à des mots racines afin d'éliminer les différences dues à des formes particulières des mots.

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
for w in filtered_sentence:
    print(ps.stem(w))
```

Q : Donner le résultat de la racinisation des mots trouvés dans la liste ci-dessous et commenter le résultat trouvé.

```
["provide","provided","provides","providers","provider","providing"]
```

7. **Etiquetage morpho-syntaxique**

   Attribuer une étiquette à chaque mot d'une phrase en mentionnant sa fonction grammaticale (nom propre, adjective, déterminant, verbe,…).

```
def process_content():
    try:
        for i in filtered_sentence[:]:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            print(tagged)

    except Exception as e:
        print(str(e))
process_content()
```

Q : Donner le résultat de l'étiquetage morpho-syntaxique des mots extraites initialement du texte (sans l'élimination des mots vides).

Catégorie grammaticale

```
POS tag list:

CC      coordinating conjunction
CD      cardinal digit
DT      determiner
EX      existential there (like: "there is", "there exists")
FW      foreign word
IN      preposition/subordinating conjunction
JJ      adjective    'big'
```

```
JJR     adjective, comparative      'bigger'
JJS     adjective, superlative      'biggest'
LS      list marker   1)
MD      modal  could, will
NN      noun, singular 'desk'
NNS     noun plural   'desks'
NNP     proper noun, singular       'Harrison'
NNPS    proper noun, plural 'Americans'
PDT     predeterminer'all the kids'
POS     possessive ending   parent\'s
PRP     personal pronoun    I, he, she
PRP$    possessive pronoun  my, his, hers
RB      adverb very, silently,
RBR     adverb, comparative better
RBS     adverb, superlative best
RP      particle      give up
TO      to     go 'to' the store.
UH      interjection errrrrrrrm
VB      verb, base form     take
VBD     verb, past tense    took
VBG     verb, gerund/present participle  taking
VBN     verb, past participle      taken
VBP     verb, sing. present, non-3d      take
VBZ     verb, 3rd person sing. present   takes
WDT     wh-determiner which
WP      wh-pronoun    who, what
WP$     possessive wh-pronoun      whose
WRB     wh-abverb     where, when
```

8. Lemmatisation

Appliquer une analyse morphologique aux mots, visant à supprimer uniquement les fins flexionnelles et à renvoyer la forme de base ou de dictionnaire d'un mot, qui est connue sous le nom de lemme.

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

a. Simple

i. Liste

```
for words in filtered_sentence:
    print(words + " ---> " + lemmatizer.lemmatize(words))
```

Q : Donner le résultat de la lemmatisation de la liste des mots suivante :

```
list=['kites', 'babies', 'dogs', 'flying', 'smiling', 'driving', 'died',
 'tried', 'feet']
```

ii. Mot

```
print(lemmatizer.lemmatize("cats"))
print(lemmatizer.lemmatize("cacti"))
print(lemmatizer.lemmatize("geese"))
print(lemmatizer.lemmatize("rocks"))
print(lemmatizer.lemmatize("python"))
```

b. Avec étiquette

Spécifier le type de lemme du mot.

a	adjective
v	verbe
n	nom
r	adverbe

i. Liste

```
for words in filtered_sentence:
    print(words + " ---> " + lemmatizer.lemmatize(words, "v"))
    print(words + " ---> " + lemmatizer.lemmatize(words, "n"))
    print(words + " ---> " + lemmatizer.lemmatize(words, pos="r"))
    print(words + " ---> " + lemmatizer.lemmatize(words, pos="a"))
```

ii. Mot

```
print(lemmatizer.lemmatize("better", "a"))
print(lemmatizer.lemmatize("meeting", "v"))
```

9. WordNet

WordNet est une base de données lexicale pour la langue anglaise, peut être utilisée pour trouver les significations, les exemples, les synonymes et les antonymes des mots.

```
from nltk.corpus import wordnet
```

```
syns = wordnet.synsets("text")
print(syns[0].name())
print(syns[0].lemmas()[0].name())
```

a. Significations

```
print(syns[0].definition())
```

b. Exemples

```
print(syns[0].examples())
```

c. Synonymes et Antonymes

```
synonyms = []
antonyms = []

for syn in syns:
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

Q : Donner la signification, les exemples, les antonymes et les synonymes du mot « good ».

d. Similarité de deux mots

```
w1 = wordnet.synset('good.n.01')
w2 = wordnet.synset('hello.n.01')
print(w1.wup_similarity(w2))
```

Q : Calculer la similarité entre les mots « good et text » et « text et hello ».

10. Détermination du nombre de phrases, de mots et de caractères

La fonction *len()* permet de renvoyer le nombre d'éléments d'un objet.
Cette fonction peut être utilisée pour calculer le nombre de mots d'une liste, de caractères d'un mot et de phrases d'un texte.

```
print("The number of characters is", len(EXAMPLE_TEXT))
print("The number of sentences is", len(sent_tokenize(EXAMPLE_TEXT)))
print("The number of tokens is",len(word_tokens))
print("The number of the removed stopwords are",len(word_tokens)-
len(filtered_sentence))
print("The number of total tokens after removing stopwords are",len(fil
tered_sentence))
print("The number of characters in the word", filtered_sentence[30], "i
s", len(filtered_sentence[30]))
```

11. Fréquence des mots

Calculer le nombre d'occurrence des mots dans un document.

```
all_words = []
for w in filtered_sentence:
    all_words.append(w.lower())
all_words = nltk.FreqDist(all_words)
```

a. Un seul mot

```
print(all_words["early"])
```

b. Les mots les plus pertinents

```
print(all_words.most_common(5))
```

c. Tous les mots d'un document

```

```

Q : Ecrire le code qui permet d'afficher le nombre d'occurrence de tous les mots de la liste 'filtered_sentence'.

12. Conditions sur les mots

Il existe des fonctions permettant d'exprimer des conditions sur les mots.
Par exemple, l'affichage des mots se terminant par 'er'.

```
import numpy
words= sorted([w for w in set(filtered_sentence) if w.endswith('er')])
print(words)
```

Q : Afficher les mots de la liste 'filtered_sentence' qui se terminent par la lettre 'e'.

13. Sélection des mots pertinents par tf-idf

L'analyse tf-idf permet de caractériser la pertinence des mots pour distinguer des textes pris dans un corpus.

```
mytexts = nltk.TextCollection([filtered_sentence, word_tokens])
```

a. tf

```
tf=mytexts.tf('and',word_tokens)
print("tf of 'and' in 'word_tokens' is", tf)
```

b. idf

```
idf=mytexts.idf('and')
print("idf of 'and' is:",idf)
```

c. tf-idf

```
tf_idf=mytexts.tf_idf('and', word_tokens)
print("tf_idf of 'and' is:",tf_idf)
```

Q : Choisir aléatoirement quelques mots des listes 'filtered_sentence' et 'word_tokens' et calculer leurs tf, idf et tf-idf.

II. Mini-projet : Indexation et Recherche des Documents Textuels

Ce projet vise à implémenter un programme d'indexation et de recherche des documents textuels avec python en utilisant NLTK.

Pour ce faire, il faut commencer, tout d'abord, par la préparation d'un ensemble des documents textuels (**corpus**), qui va être utilisé dans le programme à implémenter.

Le programme proposé doit être composé de deux phases fondamentales.

La première phase est la phase d'**indexation**, qui consiste à effectuer un prétraitement des documents préparés (tokenisation, racinisation…).

La deuxième phase est celle de **recherche**, qui consiste à recevoir en paramètre un mot afin d'afficher

- La liste des documents contenant ce mot
- Le nombre d'occurrence de ce mot dans chaque document retourné
- Le poids de ce mot dans chaque document retourné
- Le tf-idf de ce mot dans chaque document retourné
- Le document le plus pertinent à ce mot

PS : le poids d'un mot : $\mathbf{W_{t,d} = (1+\log (tf_{t,d})) * \log (N/df_{t,d})}$

Avec :

$\mathbf{tf_{t,d}}$ est le nombre d'occurrence de mot **t** dans le document **d**

N est le nombre total des documents

$\mathbf{df_{t,d}}$ est le nombre de documents contenant **t**

Exemple: t= 'good', N=5, $df_{t,d}$= 3, $tf_{t,d1}$=2, $tf_{t,d2}$=6, $tf_{t,d3}$=4

$W_{t,d1}$= (1+log (2))* log (5/3)

$W_{t,d2}$= (1+log (6)) * log (5/3)

$W_{t,d3}$= (1+log (4)) * log (5/3)