BE Big Data

(Text Mining in Python)
Février 2019
CPE

Version Élèves

2019-20

TabMat

I Introduction

NDLR :

☞ **Les exercices de ce documents sont fournis sous forme de NoteBook** (*Fichiers-notebook.tgz*).

☞ Vous trouverez les Datasets (qui ne sont pas toujours fournis par les packages python).

Si le présent document semble volumineux, c'est parce qu'il contient les code des notebooks.

Ces codes sources sont maintenues dans ce document car ils contiennent des commentaires et explications qui ne sont pas tous présents dans les fichiers notebook. Une lecture parallèle de ces deux sources est donc conseillée.

I-1 Etapes du Text Mining

• La préparation (pre-processing) d'un corpus de n documents consiste en la transformation de ce corpus en des données numériques sous la forme d'une matrice terme-document.
Cette matrice aura autant de colonnes (m) que de mots (termes) retenus.
Ainsi, à chaque terme (ou mot retenu) sera associé un vecteur de nombres réels de taille n.
De manière symétrique, un documents sera associé à un vecteur de m réels.

Etapes de préparation en Text Mining :

- tokenisation du corpus (par exemples les tweets)
- comptage des tokens (termes), fréquence de termes
- suppression des mots usuels (stop-words)
- stemming ou Lemmatisation (extraction des souches des mots)
- constitution de n-grams (n=1, 2, 3). $n = 1$ par défaut. $n = 2$: des couples de mots voisins,....

• Suivant les méthodes d'analyse utilisées, on peut avoir également recours à :

- utilisation de *pickle* (sérialisation : efficacité de traitement)
- synonym mapping
- Si nécessaire, passer p.ex. de l'anglais à l'American (et vice versa).
- Etc

I-2 Les Datasets pour ce BE

- Dataset Recettes (recipes)
- Datasets nltk (./nltk, ./nltd-data)
- Dataset twitter

• Enfin, env. 30 points d'exercices sont proposés dans ce document. Choisir vos exercices !

II Corpus Recettes : Introduction aux Text-Mining appliqué

• Dans cette section, nous passons en revue différents types de préparations que l'on applique généralement à un corpus. Le but étant de constituer une matrice Terme-Document.
Pour la partie analyse d'un corpus, voir les sections suivantes.

• **Fichier 2_load_et_explorartion.ipynb**

On utilisera un corpus de recettes de cuisine. Ce corpus vous est normalement fourni pour ce BE.
Les documents de ce corpus ont été découpés : une recette par fichier. Il y a 220 recettes.
On suppose que les 220 documents de ce corpus est placé dans "./data/recipes/1, 2, ..., 220.txt".

Contenu :
- BD : recipes/1, 2, ..., 220.txt (à placer dans le répertoire ./data).
- Chargement du corpus
- Tokenisation
- Quelques fréquences des mots du corpus
- Les mots usuels et leur suppression
- Stemming
- n-grammes

II-1 Chargement du corpus

```
import os

data_folder = os.path.join('./', 'data', 'recipes')
all_recipe_files = (os.path.join(data_folder, fname)
                    for fname in os.listdir(data_folder))

documents = {}
for recipe_fname in all_recipe_files:
    bname = os.path.basename(recipe_fname)
    recipe_number = os.path.splitext(bname)(0)
    with open(recipe_fname, 'r') as f:
        documents(recipe_number) = f.read()

corpus_all_in_one = ' '.join((doc for doc in documents.values()))

print("Nbr de docs: {}".format(len(documents)))
print("Taille Corpus (char): {}".format(len(corpus_all_in_one)))
"""
TRACE :
Nbr de docs: 220
Taille Corpus (char): 161146
"""
```

Étape suivante : on devra construire une matrice (numérique) à partir du texte du corpus. Ce qui est la représentation du corpus dans un espace vectoriel. Voir la suite....

II-2 Tokenisation

Pour simplifier, les tokens sont les lexèmes (les mots qui sont séparés deux à deux par un séparateur) mais aussi les séparateurs (sauf espace) tels que les ponctuations, parenthèses,.....
Les préparations ci-dessous sont souvent nécessaires pour aller plus loin en Text Mining.
Les opérations ci-dessous ont été expliquées en cours (voir support cours). Voir également un exemple de Tokenisation appliqué à quelques phrases en section IX-1 en page 23.

II-2-a Nombre de tokens dans le corpus

Avec word_tokenize de *nltk* :

```
#   Tokenisation :
#====================
# Il s'agit de scinder une chaîne de caractères en une liste de tokens :
# termes, mots et nombres, hashtagnbr, date, unité monnaie, ... (l'unité qui a un sens :  plus formellement
   'lexème').

from nltk.tokenize import word_tokenize

try:   # py3
      all_tokens = (t for t in word_tokenize(corpus_all_in_one))
except UnicodeDecodeError:   # py27
      all_tokens = (t for t in word_tokenize(corpus_all_in_one.decode('utf-8')))

print("Nbr. Total des tokens: {}".format(len(all_tokens)))

# Trace : Nombre Total des tokens: 33719
```

II-3 Quelques tokens fréquents du corpus

Fréquences des 20 mots les plus fréquents (total TF : total Term Frequency) :
☞ Combien de fois un mot apparait dans tout le corpus (nombre total d'occurrences)

```
# Fréquence des mots (avec    'collections.Counter')

# On voudrait trouver (pour les 20  mots les plus communs = les plus fréquents) :
# - Combien de fois un mot apparait dans tout le corpus (nombre total d'occurrences) : dans ce script.
# - Dans combien de documents le mot apparait (le script suivant)

from collections import Counter

# ATTENTION : on a besoin de la variable all_token du code précédent.

total_term_frequency = Counter(all_tokens)

for word, freq in total_term_frequency.most_common(20):
      print("{}\t{}".format(word, freq))

"""
TRACE : fréquence de chaque mot dans tout le corpus (pas dans un seul document)
the             1933
,               1726
.               1568
and             1435
a               1076
of              988
in              811
with            726
it              537
to              452
or              389
is              337
(               295
)               295
be              266
them            248
butter          231
on              220
water           205
little          198
"""
```

II-4 Local Term-frequency (per doc) des mots : Tf

Si on veut savoir le nombre d'occurrences des (p. ex) deux mots les plus fréquents dans chaque document (Tf) :

Pour chaque document, repérez les 2 mots les plus fréquents puis dire pour chacun des deux son nombre d'occurrences dans ce document.

```
# Fréquences par document : les deux tokens les plus fréquens par doc
from collections import Counter

print("No Document\t (Token, Nb_occ du token dans Document)")
Max_nb_lignes=10      # On affichera que 10 réponses
for   ind in documents.keys() :
      doc_tokens = (t for t in word_tokenize(documents(ind)))
      occs = Counter(doc_tokens).most_common(2)
      print(ind,"\t\t", occs)
      Max_nb_lignes-=1
      if Max_nb_lignes < 0 : break

"""
No Document   (Token, Nb_occ du token dans Document)
148             (('it', 7), ('.', 7))
14          (('and', 10), ('.', 8))
145             ((',', 8), ('and', 5))
152             (('the', 5), (',', 4))
50          ((',', 8), ('the', 6))
192             ((',', 17), ('and', 9))
210             (('the', 5), ('is', 3))
40          (('the', 52), (',', 37))
13          (('.', 10), ('the', 9))
64          ((',', 10), ('the', 10))
203             ((',', 15), ('the', 14))
"""
```

II-5 Document-frequency des mots : Df

Fréquences des 20 mots les plus fréquents (DF : Document Frequency) :
☞ Dans combien de documents chaque mot (parmi les 20 les +fréquents) apparait.

A noter : du fait de transformer la liste des termes en ensemble (set : occurrence unique + ordre), tout terme n'est compté qu'une seule fois (par son unicité) dans chaque document. De ce fait, les fréquences sont différentes et moindre dans ce cas.

➛ De ce fait , on obtient pour chaque mot, dans combien de documents ce mot apparait.

```
# Présence des mots dans les documents (DF):
# On commence simple avec     `collections.Counter`

# On voudrait trouver cette fois :
# - Dans combien de documents le mot apparait.

# ATTENTION : on a besoin du corpus (variable "documents") chargé dans l'exemple ci-dessus.

document_frequency=Counter()
for recipe_number, content in documents.items():
      tokens = word_tokenize(content)
      unique_tokens = set(tokens) # Needed ! Les tokens sont ordonnées. La cause de la baisse des fré
 quences
      document_frequency.update(unique_tokens)

for word, freq in document_frequency.most_common(20):
      print("{}\t{}".format(word, freq))
```

```
"""
TRACE :

On note que le nombre d'occurrence de chaque mot tombe à "1" (en passant par "set") si celui-ci est
 présent
dans le document
(d'où Document Frequency). Si le mot n'apparait pas dans un document, sa fréquence reste naturellement
  = 0.

.                220
and               220
,                219
(                218
)                218
the              217
in               215
a                 210
of               210
with             203
it               167
to                165
or                165
is               145
salt             142
butter           137
on                136
be                133
put               126
water             125
"""
```

N.B. : Éventuellement (par exemple, pour calculer *TfIDf*), il faudrait supprimer la transformation de la liste en "set" ci-dessus.

II-6 Les mots usuels et leur suppression

Commençons par vérifier (visualiser) les mots usuels anglais (contenus dans nltk).

II-6-a Les mots usuels Anglais à supprimer

```
from nltk.corpus import stopwords
import string

print(stopwords.words('english'))
print(len(stopwords.words('english')))
print(string.punctuation)

"""
TRACE :
('i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it',
"it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', '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', "didn't", 'doesn', "doesn't", 'hadn', "hadn't",
'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn',
"needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't",
'wouldn', "wouldn't")

179

!"#$%&'()*+,-./:;<=>?@(\)^_`{|}~
"""
```

• A titre indicatif, concernant le Français :

```
from nltk.corpus import stopwords
import string

print("Les mot susuels (Fr) : \n", stopwords.words('french'))
print("Nb mots usuels (Fr)", len(stopwords.words('french')))
print("Ponctuations (Fr) : ", string.punctuation)

"""
Les mot susuels (Fr) :
 ('au', 'aux', 'avec', 'ce', 'ces', 'dans', 'de', 'des', 'du', 'elle', 'en', 'et', 'eux', 'il', 'ils', 'je',
 'la', 'le', 'les', 'leur', 'lui', 'ma', 'mais', 'me', 'même', 'mes', 'moi', 'mon', 'ne', 'nos', 'notre',
 'nous', 'on', 'ou', 'par', 'pas', 'pour', 'qu', 'que', 'qui', 'sa', 'se', 'ses', 'son', 'sur', 'ta', 'te',
 'tes', 'toi', 'ton', 'tu', 'un', 'une', 'vos', 'votre', 'vous', 'c', 'd', 'j', 'l', 'à', 'm', 'n', 's', 't',
 'y', 'été', 'étée', 'étées', 'étés', 'étant', 'étante', 'étants', 'étantes', 'suis', 'es', 'est', 'sommes',
 'êtes', 'sont', 'serai', 'seras', 'sera', 'serons', 'serez', 'seront', 'serais', 'serait', 'serions', 'seriez',
 'seraient', 'étais', 'était', 'étions', 'étiez', 'étaient', 'fus', 'fut', 'fûmes', 'fûtes', 'furent', 'sois',
 'soit', 'soyons', 'soyez', 'soient', 'fusse', 'fusses', 'fût', 'fussions', 'fussiez', 'fussent', 'ayant',
 'ayante', 'ayantes', 'ayants', 'eu', 'eue', 'eues', 'eus', 'ai', 'as', 'avons', 'avez', 'ont', 'aurai',
 'auras', 'aura', 'aurons', 'aurez', 'auront', 'aurais', 'aurait', 'aurions', 'auriez', 'auraient', 'avais',
 'avait', 'avions', 'aviez', 'avaient', 'eut', 'eûmes', 'eûtes', 'eurent', 'aie', 'aies', 'ait', 'ayons',
 'ayez', 'aient', 'eusse', 'eusses', 'eût', 'eussions', 'eussiez', 'eussent')
Nb mots usuels (Fr) 157
Ponctuations (Fr) :  !"#$%&'()*+,-./:;<=>?@(\)^_`{|}~
"""
```

Application à notre corpus de recettes..../...

II-6-b Suppression des mots usuels dans notre corpus

On remarque que certains des mots (parmi les plus communs) ci-dessus ne sont guère "utiles" (ne portent pas d'information précise).
Ces mots appelés **mots usuels** pris isolément ne fournissent aucune information significative. C'est par exemple le cas des déterminants, articles, conjonctions, pronoms, etc.

Remarques :

- Chaque langue a sa propre liste des mots usuels
- La suppression de ces mots peut avoir des effets utiles ou pas selon l'application en vue.

Par exemple, si vous supprimez les mots usuels, des séquences comme "être ou ne pas être" pourraient disparaître alors qu'on pourrait en avoir besoin !

• Ci-dessous, après la suppression des mots usuels du corpus, on recalcule les nouvelles total_word_frequencies des 20 mots les plus fréquents.

```
# Suppression des Mots usuels (Stop words)
# On recalcule le total_term_frequency pour chaque mot conservé.

# Attention : on aura besoin de la variable "all_tokens" (voir ci-dessus)

from nltk.corpus import stopwords
import string

stop_list = stopwords.words('english') + list(string.punctuation)

tokens_no_stop = (token for token in all_tokens if token not in stop_list)

total_term_frequency_no_stop = Counter(tokens_no_stop)

for word, freq in total_term_frequency_no_stop.most_common(20):
    print("{}\t{}".format(word, freq))
```

```
"""
TRACE :
butter          231
water           205
little          198
put             197
one             186
salt            185
fire            169
half            169
two             157
When            132
pepper          128
sauce           128
add             125
cut             125
piece           116
flour           116
The             111
saucepan        100
sugar           100
oil             99
"""

# Noter **When** et **The** ci-dessus (majuscules W et T)
# Les mots en Maj et Min sont distingués (pour l'instant) !
```

☞ Les mots en Maj et Min sont distingués (pour l'instant) !
☞ Noter la présence des mots **When** et **The** dans la trace (majuscules W et T) :

Interrogeons quelques fréquences maintenant.

```
print("Fréquence totale du mot 'olive'", total_term_frequency_no_stop('olive'))
print("Fréquence totale du mot 'olives'",total_term_frequency_no_stop('olives'))
print("Fréquence totale du mot 'Olive'",total_term_frequency_no_stop('Olive'))
print("Fréquence totale du mot 'Olives'",total_term_frequency_no_stop('Olives'))
print("Fréquence totale du mot 'OLIVE'",total_term_frequency_no_stop('OLIVE'))
print("Fréquence totale du mot 'OLIVES'",total_term_frequency_no_stop('OLIVES'))

"""
TRACE :
Fréquence totale du mot 'olive' 27
Fréquence totale du mot 'olives' 3
Fréquence totale du mot 'Olive' 1
Fréquence totale du mot 'Olives' 0
Fréquence totale du mot 'OLIVE' 0
Fréquence totale du mot 'OLIVES' 1
"""
```

☞ Observez ces résultats pour constater que la case (Maj/Min) perturbe les fréquences. La normalisation du texte (Stemming, Lemmatization) se chargent de traiter ce point.

• Après ces quelques manipulations de base, commençons la préparation effective du corpus. ../..

II-7 Normalisation du corpus

On appelle "Normalisation" les étapes de Stemming / Lemmatisation, ...qui interviennent après le nettoyage du texte (cf. opérations précédentes).

Stemming : extraction des souches / radicaux des mots (voir cours).
Voir également un exemple de Tokenisation appliqué à quelques phrases en section IX-1 en page 23.

Pendant le Stemming, on remplace les termes (tokens) par une forme dite "canonique".
Par exemple, on peut regrouper les différentes conjugaisons d'un même verbe.

Ci-dessous, on procède par :
- mettre les documents sous forme minuscule et supprimer les mots usuels (et ponctuations) ;
- stemming (Algorithme Porter)

II-7-a Stemming appliqué à notre corpus

Pour cette phase de normalisation, on transforme le texte en minuscule avant d'appliquer l'algorithme de *Porter* (voir cours) pour obtenir les stems.
Pour montrer la différence, on recalculera les fréquences des mots pour constater les différences.

```
# Normalisation de texte
# On remplace les termes (tokens) par une forme dite "canonique".
# On peut regrouper les différentes conjugaisons d'un même verbe.
# Stemming = retrouver la base/souche d'un mot (stem)

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
all_tokens_lower = (t.lower() for t in all_tokens)

tokens_normalised = (stemmer.stem(t) for t in all_tokens_lower if t not in stop_list)

total_term_frequency_normalised = Counter(tokens_normalised)

for word, freq in total_term_frequency_normalised.most_common(20):
      print("{}\t\t{}".format(word, freq))

"""
TRACE :
put         286
butter    245
salt       215
piec       211
one         210
water      209
cook       208
littl      198
cut          175
half        170
brown      169
fire        169
egg          163
two          162
add          160
boil        154
sauc        152
pepper    130
serv        128
remov      127
"""
```

N.B. :
- Un stem n'est pas toujours un "mot" (qui apparaît dans un dico)
- Se rappeler que les opérations comme la mise en minuscules sont irrévocables
- Ici, on fait abstraction du "bon style" de programmation : mieux vaut faire plutôt des fonctions, regrouper, structurer le code, faire propre !

☞ Pour la lemmatisation, voir en section IX-2 en page 24.

II-7-b Lemmatisation

Alternativement, on peut lemmatiser les tokens (le terme retenu doit figurer dans un dico).
Remarquer p.ex. "piece", "little" ci-dessous. Certains termes ne sont plus parmi les 20 plus fréquents.

```
# Comparer les résultats de la Lemmatization à ceux du stemming
import nltk
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
all_tokens_lower = (t.lower() for t in all_tokens)
tokens_lemmatized = (lemmatizer.lemmatize(t) for t in all_tokens_lower if t not in stop_list)

# Nouvelles fréquences
total_term_frequency_lemmatized = Counter(tokens_lemmatized)

for word, freq in total_term_frequency_lemmatized.most_common(20):
    print(f"{word}\t{freq}")

"""
put 276
butter 243
piece 211
one 210
water 209
little 198
salt 197
half 173
fire 169
cut 166
egg 163
two 162
add 160
sauce 152
pepper 130
flour 123
sugar 116
brown 105
saucepan 101
onion 101
"""
```

II-8 n-grammes

Lorsque nous sommes intéressés par le contexte des mots (dans une phrase par exemple), on peut utiliser les *n-grams* qui représente l'entourage d'un mot (ses voisins). Un n-gram est une séquence de n mots (grams = grammes) adjacents. On utilise généralement des bi-grams ou tri-grams...

Si les n-grams ainsi construits deviennent des attributs (features) de notre corpus, on aura alors besoin de calculer leur fréquence.

☞ Ne pas confondre avec le skip-grams de Word2vect. On n'a pas ici de NN !

Ci-dessous, on calcule la fréquence des bi-grammes après avoir construit ces bi-grams sur les mots en minuscules (**sans la suppression des mots usuels ni l'application du stemming**).

II-8-a Exemple de 2-grammes

```
from nltk import ngrams

# Comptage des bi-grammes (mots usuels tjs présents)
phrases = Counter(ngrams(all_tokens_lower, 2))
for phrase, freq in phrases.most_common(20):
"""
TRACE :
('in', 'the')     175
('in', 'a')  172
('of', 'the')     153
('with', 'a')     142
('.', 'when')     131
('the', 'fire')  129
('on', 'the')     128
('with', 'the')  117
(',', 'and')      117
('salt', 'and')  113
('it', 'is')      109
('a', 'little')  107
('piece', 'of')  102
('and', 'a')      102
('of', 'butter')      94
('and', 'pepper')     87
('.', 'the')      85
('and', 'the')    84
('when', 'the')  82
('with', 'salt')      80
"""
```

Rappel : on avait calculé les mots en minuscules à partir de 'all_tokens' (sans aucun autre traitement) par : `all_tokens_lower = [t.lower() for t in all_tokens]`

Ci-dessous, on construit les tri-grams et on calcule leur fréquence.

II-8-b Fréquences des 3-grammes de notre corpus (avant la suppression des usuels)

```
from nltk import ngrams

# Rappel : on avait calculé les mots en minuscules àpd 'all_tokens' (sans aucun autre traitement) par :
# all_tokens_lower = (t.lower() for t in all_tokens)

phrases = Counter(ngrams(all_tokens_lower, 3))
for phrase, freq in phrases.most_common(20):
      print("{}\t{}".format(phrase, freq))

"""
TRACE :
('on', 'the', 'fire')     90
('salt', 'and', 'pepper')     84
('piece', 'of', 'butter')     73
('a', 'piece', 'of')      63
('with', 'salt', 'and')  62
('.', 'when', 'the')      59
('in', 'a', 'saucepan')  45
('a', 'pinch', 'of')      45
('season', 'with', 'salt')    42
('the', 'fire', 'with')  41
('when', 'it', 'is')      39
('and', 'pepper', '.')    37
('through', 'a', 'sieve')     36
('complete', 'the', 'cooking')    34
('and', 'a', 'half')      33
('of', 'butter', ',')     27
('a', 'taste', 'of')      26
('and', 'when', 'it')     26
('it', 'on', 'the')  26
(',', 'salt', 'and')      25
"""
```

☞ **Remarques :**

Faire attention aux liens entre les n-grammes et les mots usuels (stop-words).

- La suppression des mots usuels affecte les n-grammes.

II-8-c Application des 2-grammes à notre corpus (après la suppression des usuels)

Nous allons donc appliquer une étape de suppression des mots usuels avant la construction des bi-grams et leur fréquence puis de même avec les 3-grammes. A comparer avec les fréquences ci-dessus.

```
# ici bi-grams après la suppression des mots usuels.
phrases = Counter(ngrams(tokens_no_stop, 2))

for phrase, freq in phrases.most_common(20):
    print("{}\t{}".format(phrase, freq))

"""
TRACE :
('salt', 'pepper')  106
('piece', 'butter') 73
('grated', 'cheese')    55
('bread', 'crumbs') 34
('put', 'fire') 32
('tomato', 'sauce') 32
('complete', 'cooking') 31
('brown', 'stock')  29
('thin', 'slices')  29
('season', 'salt')  29
('olive', 'oil')    26
('low', 'fire') 25
('chopped', 'fine') 25
('boiling', 'water')    22
('little', 'pieces')    22
('half', 'ounces')  21
('lemon', 'peel')   18
('one', 'two')  18
('two', 'ounces')   18
('half', 'cooked')  18
"""
```

II-8-d Application des 3-grammes à notre corpus (après la suppression des usuels)

De même pour les 3-grammes :

```
phrases = Counter(ngrams(tokens_no_stop, 3))

for phrase, freq in phrases.most_common(20):
    print("{}\t{}".format(phrase, freq))

"""
TRACE :

('season', 'salt', 'pepper')    28
('Season', 'salt', 'pepper')    16
('bread', 'crumbs', 'ground')   11
('pinch', 'grated', 'cheese')   11
('cut', 'thin', 'slices')   11
('good', 'olive', 'oil')    10
('half', 'inch', 'thick')   9
('greased', 'butter', 'sprinkled')  9
('cut', 'small', 'pieces')  9
('another', 'piece', 'butter')  9
('small', 'piece', 'butter')    9
('salt', 'pepper', 'When')  9
('saucepan', 'piece', 'butter') 9
('medium', 'sized', 'onion')    8
('tomato', 'sauce', 'No')   8
('sauce', 'No', '12')   8
('ounces', 'Sweet', 'almonds')  8
('three', 'half', 'ounces') 8
('crumbs', 'ground', 'fine')    7
('butter', 'salt', 'pepper')    7
"""
```

Nous avions remarqué l'effet de la suppression des mots usuels sur les n-grammes.
Par exmeple, une expression comme "a pinch of salt" devient "pinch salt" après la suppression des mots usuels. Les 3-grams ne seront plus les mêmes.

Comparons quelques tri-grammes (les plus fréquents) avec la présence des mots usuels ... :

```
Sans les usuels :                          Avec les usuels :
_____________________________              _____________________________
('season', 'salt', 'pepper')      28       ('salt', 'and', 'pepper')      84
('Season', 'salt', 'pepper')      16       ('season', 'with', 'salt')     42
('another', 'piece', 'butter')             ('piece', 'of', 'butter')      73
('small', 'piece', 'butter')      9
('salt', 'pepper', 'When')  9              ('salt', 'and', 'pepper')      84
('butter', 'salt', 'pepper')      7
```

Enfin, les 3-grams avec les tokens après lemmatisation :

```
lemm_phrases = Counter(ngrams(tokens_lemmatized_no_stop, 3))

for phrase, freq in lemm_phrases.most_common(20):
    print("{}\t{}".format(phrase, freq))

"""
('season', 'salt', 'pepper')      44
('bread', 'crumb', 'ground')      13
('cut', 'thin', 'slice')      13
('taste', 'lemon', 'peel')   12
('pinch', 'grated', 'cheese')     11
('small', 'piece', 'butter')      10
('good', 'olive', 'oil')      10
('crumb', 'ground', 'fine') 9
('half', 'inch', 'thick')     9
('greased', 'butter', 'sprinkled')   9
('cut', 'small', 'piece')     9
('cut', 'little', 'piece')   9
('another', 'piece', 'butter')   9
('medium', 'sized', 'onion')      9
('ounce', 'sweet', 'almond')      9
('saucepan', 'piece', 'butter') 9
('little', 'piece', 'butter')     8
('tomato', 'sauce', '12')     8
('three', 'half', 'ounce')   8
('butter', 'salt', 'pepper')      7
"""
```

➛ On remarque quelques différences.

II-9 Vers les étapes suivantes

Passage à un espace vectoriel puis application d'algorithmes de Data-Mining.

III Sauvegarde et persistance des modèles

• **Fichier EX-save-complet.ipynb**

Souvent on se pose la question : comment sauvegarder un modèle déjà construit (pour ne pas refaire le travail à chaque utilisation ?).
Un autre problème est la mémoire : conserver TOUT pendant toute la durée d'un traitement coûtera cher tandis que souvent, on peut se délester des éléments gourmands en mémoire, quitte à les recharger plus tard.
Python propose le module *pickle* que l'on traduit par *conserve* (de cornichon !).
☞ Notons : une alternative à *Pickle* est **joblib**.

III-1 Cas de pickle

Voir aussi http ://scikit-learn.org/stable/modules/model_persistence.html

```
from sklearn import svm
from sklearn import datasets

clf = svm.SVC(gamma='auto')
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf.fit(X, y)

"""
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
"""
# Sauvegarder le modèle :
import pickle
s = pickle.dumps(clf)

# Charger le modèle en mémoire :
clf2 = pickle.loads(s)

# tester
print(clf2.predict(X[0:1])) # array([0])
print(y[0]) # 00
```

☞ IMPORTANT dans certains cas d'utilisation de *scikit* (avec beaucoup de données), il est bien plus intéressant d'utiliser **joblib** qui remplace *pickle*. Comme pour pickle, on utlisera *joblib.dump()* et *joblib.load()*.

III-2 Cas de joblib

joblib est bien plus efficace lorsque les objets construits (modèles, données, etc...) contiennent de grandes matrices de numpy comme c'est le cas en apprentissage avec *scikit* (après l'opération "fit"). Ces objets ne pourront parfois même pas être chargés en mémoire dans une chaîne de caractères (étant donné leur taille !).

Exemple :

```
from sklearn.externals import joblib

# La sauvegarde du modèle
joblib.dump(clf, 'filename.pkl')

# Récupération
clf = joblib.load('filename.pkl')
```

joblib.dump() et *joblib.load()* acceptent des objets descripteurs de fichiers.
☞ Voir https ://pythonhosted.org/joblib/persistence.html pour plus d'informations.
joblib peut compresser selon différents formats de compression.

Un exemple plus complet : est développé dans le même fichier.

IV Analyse d'opinion (Polarité de tweet) avec Bayes

- **Fichier ex9-ZZ-un-ex-nltk-sentiment.ipynb**
- Petit exemple de démonstration d'analyse de polarité de tweets.

Contenu :

- BD : phrases dans le code
- Traitement de tweets
- Pickle
- Bayes Naïve

```
import nltk
import sys
from sys import exit
import pickle

pos_tweets = [('I love this car', 'positive'),
              ('This view is amazing', 'positive'),
              ('I feel great this morning', 'positive'),
              ('I am so excited about the concert', 'positive'),
              ('He is my best friend', 'positive'),
              ('Going well', 'positive'),
              ('Thank you', 'positive'),
              ('Hope you are doing well', 'positive'),
              ('I am very happy', 'positive'),
              ('Good for you', 'positive'),
              ('It is all good. I know about it and I accept it.', 'positive'),
              ('This is really good!', 'positive'),
              ('Tomorrow is going to be fun.', 'positive'),
              ('Smiling all around.', 'positive'),
              ('These are great apples today.', 'positive'),
              ('How about them apples? Thomas is a happy boy.', 'positive'),
              ('Thomas is very zen. He is well-mannered.', 'positive')]

neg_tweets = [('I do not like this car', 'negative'),
              ('This view is horrible', 'negative'),
              ('I feel tired this morning', 'negative'),
              ('I am not looking forward to the concert', 'negative'),
              ('He is my enemy', 'negative'),
              ('I am a bad boy', 'negative'),
              ('This is not good', 'negative'),
              ('I am bothered by this', 'negative'),
              ('I am not connected with this', 'negative'),
              ('Sadistic creep you ass. Die.', 'negative'),
              ('All sorts of crazy and scary as hell.', 'negative'),
              ('Not his emails, no.', 'negative'),
              ('His father is dead. Returned obviously.', 'negative'),
              ('He has a bomb.', 'negative'),
              ('Too fast to be on foot. We cannot catch them.', 'negative')]

tweets = []
for (words, sentiment) in pos_tweets + neg_tweets:
    words_filtered = [e.lower() for e in words.split() if len(e) >= 3]
    tweets.append((words_filtered, sentiment))

def mon_get_words_in_tweets(tweets):
# from __future__ import print_function
    all_words = []
    for (words, sentiment) in tweets:
       all_words.extend(words)
    return all_words

def mon_get_word_features(wordlist):
    wordlist = nltk.FreqDist(wordlist)
    word_features = wordlist.keys()
    return word_features

def mon_extract_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features

word_features = mon_get_word_features(mon_get_words_in_tweets(tweets))
```

```
training_set = nltk.classify.apply_features(mon_extract_features, tweets)
classifier = nltk.NaiveBayesClassifier.train(training_set)

# On peut sauvegarder  le classifieur (ici on sauvegarde mais on ne le recharge pas : no need !)
save_classifier = open("tweetposneg.pickle","wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()


# On doit recharger pour tester d'autres données de test : on peut le charger  par les 3 lignes) :
# classifier_f = open("naivebayes.pickle", "rb")
# classifier = pickle.load(classifier_f)
# classifier_f.close()

mon_test_tweets = []  # recupérer une liste de tweet qu'on aurait donné en ligne de commande (TESTEZ)

# NE pas utiliser cet 'if' en console de votre spyder (car sys.argv a une valeur qui nous échappe !)
if len(sys.argv) > 1:  # si le paramètre donné lors de l'exécution = nom un fic de tweets ?
    tweetfile = sys.argv[1]
    with open(tweetfile, "r") as ins:
        for line in ins:
            mon_test_tweets.append(line)

# On y ajoute qq tweets aux cas où on n'aurait rien donné '!
mon_test_tweets.append('I am a bad boy')  # test tweet au cas où
mon_test_tweets.append('Are you a good girl !')  # test tweet au cas où
mon_test_tweets.append('One  night a hotel caught fire')  #  n'importe quoi pour voir !
mon_test_tweets.append('People who were staying there ran out in their night clothes')  #  n'importe quoi
mon_test_tweets.append('Two men were stangig outside talking about the fire')  #  pour voir !


print("="*30, "RESULTATS", "="*30)
poscount = 0
negcount = 0
for tweett in mon_test_tweets:
   valued = classifier.classify(mon_extract_features(tweett.split()))
   print (valued)
   if valued == 'negative':
      negcount = negcount + 1
   else:
      poscount = poscount + 1

print ('\nPositive count: %s \nNegative count: %s' % (poscount,negcount))
#  exit() pour s'arrêter avant la fin.


"""
TRACE :
============================== RESULTATS ==============================
negative
positive
positive
positive
positive

Positive count: 4
Negative count: 1
[Finished in 0.7s]
"""
```

V Exemple complet de clustering de News : K-Means sur SVD

• **Fichier : EX2bis-est-Ex2-nettoyed-from-bigdata-SVD-Kmeans.ipynb**

☞ On peut (si on le souhaite) lancer cet exemple en fournissant des paramètres. Voir le code et le traitement des paramètres de lancement.
☞ Modifier le code pour "jouer" avec ces paramètres. La trace ci-dessous est sans paramètre aucun.

Contenu :

- BD : voir le code (`import fetch_20newsgroups`).
- Traitement de newsgroupes
- K-means et MiniBatchKmeans (une variante)
- SVD
- Quelques mesures d'évaluation

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn import metrics

from sklearn.cluster import KMeans, MiniBatchKMeans

import logging
from optparse import OptionParser
import sys
from time import time

import numpy as np


# Barre de progrès des calculs (un peu longs)
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s')

# Arguments éventuels de la ligne de commande
op = OptionParser()
op.add_option("--lsa",
                    dest="n_components", type="int",
                    help="Preprocess documents with latent semantic analysis.")
op.add_option("--no-minibatch",
                    action="store_false", dest="minibatch", default=True,
                    help="Use ordinary k-means algorithm (in batch mode).")
op.add_option("--no-idf",
                    action="store_false", dest="use_idf", default=True,
                    help="Disable Inverse Document Frequency feature weighting.")
op.add_option("--use-hashing",
                    action="store_true", default=False,
                    help="Use a hashing feature vectorizer")
op.add_option("--n-features", type=int, default=10000,
                    help="Maximum number of features (dimensions)"
                         " to extract from text.")
op.add_option("--verbose",
                    action="store_true", dest="verbose", default=False,
                    help="Print progress reports inside k-means algorithm.")

print(__doc__)
op.print_help()


def is_interactive():
    return not hasattr(sys.modules['__main__'], '__file__')

# Pour Jupyter notebook et IPython
argv = () if is_interactive() else sys.argv[1:]

(opts, args) = op.parse_args(argv)
if len(args) > 0:
    op.error("this script takes no arguments.")
```

```
        sys.exit(1)

"""
TRACE
None
Usage: EX2bis-est-Ex2-nettoyed-from-bigdata-SVD-Kmeans.py (options)

Options:
  -h, --help                  help + exit
  --lsa=N_COMPONENTS          Traiter avec LSA ("latent semantic analysis").
  --no-minibatch              Utiliser simple K-means (en mode "batch").
  --no-idf                    Ne pas utiliser l'indice Idf
  --use-hashing               Utiliser une vectorisation avec Hashage.
  --n-features=N_FEATURES
                              Maximum de features (dimensions) à retenir (matrice)
  --verbose                   Afficher "progress reports" dans l'algo k-means.
"""
```

V-1 Début travaux !

```
# ###############################################################################
# Chageons QQ catégories depuis l'ensemble d'apprentissage
categories = (
        'alt.atheism',
        'talk.religion.misc',
        'comp.graphics',
        'sci.space',
)
# Si décommenté, on fera Toutes les catégoroes !
# categories = None

print("Chargement de 20 newsgroups pour les categories ::")
print(categories)

# Voir les imports
dataset = fetch_20newsgroups(subset='all', categories=categories,
                                        shuffle=True, random_state=42)

print("%d documents" % len(dataset.data))
print("%d categories" % len(dataset.target_names))
print()

labels = dataset.target
true_k = np.unique(labels).shape(0)

"""
TRACE :

Chargement de 20 newsgroups pour les categories ::
('alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space')
3387 documents
4 categories
"""
```

V-2 Extraction des features

```
print("Extraction des features du training set avec vectorisation")
t0 = time()
if opts.use_hashing:
    if opts.use_idf:
        # Perform an IDF normalization on the output of HashingVectorizer
        hasher = HashingVectorizer(n_features=opts.n_features,
                                   stop_words='english', alternate_sign=False,
                                   norm=None, binary=False)
        vectorizer = make_pipeline(hasher, TfidfTransformer())
    else:
        vectorizer = HashingVectorizer(n_features=opts.n_features,
                                       stop_words='english',
                                       alternate_sign=False, norm='l2',
                                       binary=False)
else:
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=opts.n_features,
                                 min_df=2, stop_words='english',
                                 use_idf=opts.use_idf)
X = vectorizer.fit_transform(dataset.data)

print("Fait en  %fs" % (time() - t0))
print("Info : nb samples: %d, nb features: %d" % X.shape)
print()

"""
TRACE :
Extraction des features du training set avec vectorisation
Fait en  0.961714s
Info : nb samples: 3387, nb features: 10000
"""
```

V-3 SVD

```
if opts.n_components:
    print("SVD LSA", end=' ')
    print("de ", X.shape[1], " features à", opts.n_components, " features")
    t0 = time()

    # Les résultats sont déjà normalisés. Ce qui fait que KMeans se comportera comme
    # spherical k-means (mieux).
    # ATTENTION : les résultats de  LSA/SVD ne sont pas Re-normalisés
    # --->> ON DOIT LE REFAIRE
    svd = TruncatedSVD(opts.n_components)
    normalizer = Normalizer(copy=False)
    lsa = make_pipeline(svd, normalizer)

    X = lsa.fit_transform(X)

    print("Fait en %fs" % (time() - t0))

    explained_variance = svd.explained_variance_ratio_.sum()
    print("Variance expliquée par la SVD : {}%".format(
        int(explained_variance * 100)))

    print()

"""
TRACE :
option non demandée cette fois !
Testez-la !
"""
```

V-4 Clustering

```
# clustering

if opts.minibatch:
    km = MiniBatchKMeans(n_clusters=true_k, init='k-means++', n_init=1,
                         init_size=1000, batch_size=1000, verbose=opts.verbose)
else:
    km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1,
                verbose=opts.verbose)

print("Données creuse de Clustering avec %s" % km)
t0 = time()
km.fit(X)
print("Fait en %0.3fs" % (time() - t0));  print()

# Quelques mesures propres au clustering et Kmeans
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels, km.labels_))
print("Completeness: %0.3f" % metrics.completeness_score(labels, km.labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(labels, km.labels_))
print(" Rand-Index ajusté : %.3f" % metrics.adjusted_rand_score(labels, km.labels_))
print("Coefficient Silhouette : %0.3f" % metrics.silhouette_score(X, km.labels_, sample_size=1000))
print()

if not opts.use_hashing:
    print("Meilleurs termes par cluster:")

    if opts.n_components:
        original_space_centroids = svd.inverse_transform(km.cluster_centers_)
        order_centroids = original_space_centroids.argsort()(:, ::-1)
    else:
        order_centroids = km.cluster_centers_.argsort()(:, ::-1)

    terms = vectorizer.get_feature_names()
    for i in range(true_k):
        print("Cluster %d:" % i, end='')
        for ind in order_centroids(i, :10):
            print(' %s' % terms(ind), end='')
        print()

""" TRACE :
Donées creuse de Clustering avec MiniBatchKMeans(batch_size=1000, compute_labels=True, init='k-means++',
        init_size=1000, max_iter=100, max_no_improvement=10, n_clusters=4,
        n_init=1, random_state=None, reassignment_ratio=0.01, tol=0.0,
        verbose=False)
Fait en 0.168s

Homogeneity: 0.391
Completeness: 0.429
V-measure: 0.409
 Rand-Index ajusté : 0.319
Coefficient Silhouette : 0.005

Meilleurs termes par cluster:
Cluster 0: image gif file files format graphics bit color images thanks
Cluster 1: com sandvik keith henry sgi caltech livesey toronto kent apple
Cluster 2: god com people jesus say bible christian don article believe
Cluster 3: space com nasa graphics university access posting host nntp gov
(Finished in 3.1s)
"""
```

VI Exercice

Sur la base de l'exemple ci-dessus :

- Appliquer ce code en utilisant un autre corpus (par exemple movie_review) et consigner les résultats. Cela nécessitera de recréer les fichier pickle du nouveau modèle (dans un autre répertoire).
- **Tranfer-Learning** : appliquer les modèles construits ci-dessus à un autre corpus (le même que le point précédent, par exemple movie_review). Comparer les résultats. Quel est le pourcentage de justesse quand on apprend sur le nouveau corpus par rapport au *Tranfer-Learning*?
- Ajouter deux autres méthodes d'apprentissage pour améliorer le vote.Pour

VII Exemple Word2vect

Petit exemple / démo de Vectorisation à la *Word2vect*.

Contenu :

- BD : dans le code
- word2vect sur un vocabulaire

```
from gensim.models import Word2Vec

# Training data
sentences = (('this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'),
             ('this', 'is', 'the', 'second', 'sentence'),
             ('yet', 'another', 'sentence'),
             ('one', 'more', 'sentence'),
             ('and', 'the', 'final', 'sentence'))

# entrainer le modèle
model = Word2Vec(sentences, min_count=1)

#  infos sur le modèle
print(model)

# infos sur le vocabulaire
words = list(model.wv.vocab)
print("Words = ", words)

# Les vecteurs des mots (le but de word2vect)
print("model('sentence') ", model('sentence'))

# save + charger le modèle
model.save('model.bin')
new_model = Word2Vec.load('model.bin')

# Vérifions que c'est le bon modèle rechargé !'
print("\nnew_model ", new_model)
```

```
# TRACE


"""
Running /home/alex/CPE-DM-17-18/word2vect-Ex6.py
Word2Vec(vocab=14, size=100, alpha=0.025)
Words = ['is', 'second', 'another', 'the', 'first', 'word2vec', 'and', 'final', 'sentence', 'one', 'this', 'more', 'for
  ', 'yet']
model['sentence']  ( -4.16341610e-03   1.02334959e-03   1.57905463e-03   4.85063158e-03
  -9.15832294e-04   1.77958712e-03  -7.82333664e-04  -4.57805116e-03
  -4.57918225e-03  -2.71693338e-04  -4.96282382e-03  -4.34031989e-03
   2.94611393e-03   2.69084936e-03  -2.89399712e-03   2.19605921e-04
  -5.45860967e-04   5.87674091e-04   4.73225815e-03  -3.20032262e-03
  -5.55604602e-05   2.58243107e-03   2.58820271e-03  -1.05416332e-03
   2.98698340e-03   1.49033580e-03  -3.75461444e-04   2.69162792e-05
  -3.33420606e-03   9.54134855e-04  -3.72390077e-03  -1.65057043e-03
  -2.25537879e-04  -1.71130756e-03   1.64663955e-03  -3.05743166e-03
   3.61012877e-03   4.87857178e-04  -8.57718638e-04   2.34226068e-03
   4.68768692e-03  -3.15963174e-03   4.75594122e-03   2.32926593e-03
   2.16187211e-04   2.22291495e-03  -1.20184093e-03  -4.06050077e-03
  -4.07638494e-03   4.76623932e-03  -1.88740680e-03   4.74161562e-03
   3.05079133e-03   3.22582922e-03   2.64309044e-03   4.92485054e-03
   4.01183404e-03   6.06674817e-04  -1.95208297e-03   2.75960122e-03
   4.69434075e-03  -1.33649015e-03   1.44785712e-03   1.19258312e-03
   7.96982727e-04  -2.63210922e-03  -3.40659125e-03  -1.12855365e-03
  -2.81821168e-03  -1.21105043e-03   1.78319623e-03  -4.91397968e-03
   3.93841788e-03   6.12073753e-04  -1.98802096e-03   1.35092332e-03
  -2.57994747e-04   3.39605659e-03  -2.82615633e-03   2.89340969e-04
  -4.54878062e-03  -8.72837380e-04  -1.65895140e-03   7.61264062e-04
   2.43290374e-03  -1.48350431e-03  -5.60902154e-05   3.78510589e-03
  -4.40110825e-03   1.70890440e-03  -2.35772113e-05   4.38308198e-04
   2.37865795e-04  -2.12008948e-03  -1.56214449e-03  -3.14583187e-03
   4.36120015e-03  -4.18594386e-03  -3.53513029e-03  -9.28688154e-04)

new_model  Word2Vec(vocab=14, size=100, alpha=0.025)
```

VIII Exercice

Choisir un corpus parmi ceux fournis (ou des phrases à vous que vous pouvez ajouter à celles de l'exemple précédent), extraire les lemmes extraire puis les vecteurs *word2vect* puis trouver une matrice de similarités entre ces termes.
Établir une graphique montrant (en 2D, tant pis) ces similarités.

IX Annexes : quelques autres exemples

IX-1 Tokenisation

Avant de traiter notre corpus, voyons un exemple de tokenisation.

Contenu (les importations nécessaires sont dans ce code) :

- BD : phrases dans le code
- extraction des mots (split)
- tokenisation de texte (et de tweet, voir `import TweetTokenizer`)

```
# Word Tokenisation
# =======================
text = "The quick brown fox jumped over the lazy dog"
tokens = text.split()
print("1– Tokens par simple split ", tokens)


text = "The quick brown fox, and an Oxford comma"
tokens = text.split()
print("2– Tokens par simple split ", tokens)


from nltk.tokenize import word_tokenize
text = "The quick brown fox, and an Oxford comma"
tokens = word_tokenize(text)
print("3– Tokens par word_tokenize ",tokens)


from nltk.tokenize import word_tokenize
text = "Tweet about #NLProc @PyConUK :)"
tokens = word_tokenize(text)
print("4– Tokens d'un tweet par word_tokenize ",tokens)


from nltk.tokenize import TweetTokenizer
text = "Tweet about #NLProc @PyConUK :)"
tokenizer = TweetTokenizer()
tokens = tokenizer.tokenize(text)
print("5– Tokens d'un tweet par TweetTokenizer (plus spécialisé) ",tokens)


from nltk.tokenize import TweetTokenizer
text = "Tweet about #NLProc @PyConUK :)"
tokenizer = TweetTokenizer(strip_handles=True)
tokens = tokenizer.tokenize(text)
print("6– Tokens d'un tweet par TweetTokenizer (avec option strip_handles) ",tokens)


from nltk.tokenize import word_tokenize
text = "How about currencies (like £100,000.00) and dates (like 19th September)"
tokens = word_tokenize(text)
print("7– Tokens par word_tokenize ",tokens)

"""
Trace :
1– Tokens par simple split  ('The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog')
2– Tokens par simple split  ('The', 'quick', 'brown', 'fox,', 'and', 'an', 'Oxford', 'comma')
3– Tokens par word_tokenize  ('The', 'quick', 'brown', 'fox', ',', 'and', 'an', 'Oxford', 'comma')
4– Tokens d'un tweet par word_tokenize  ('Tweet', 'about', '#', 'NLProc', '@', 'PyConUK', ':', ')')
5– Tokens d'un tweet par TweetTokenizer (plus spécialisé)  ('Tweet', 'about', '#NLProc', '@PyConUK', ':)')
6– Tokens d'un tweet par TweetTokenizer (avec option strip_handles)  ('Tweet', 'about', '#NLProc', ':)')
7– Tokens par word_tokenize  ('How', 'about', 'currencies', '(', 'like', '£', '100,000.00', ')', 'and', 'dates', '(', '
 like', '19th',
'September', ')')
"""
```

IX-2 Lemmatisation : Exemples

Contenu :

- BD : phrases dans le code
- extraction de lems de texte selon les informations "part of speech" *(POS)* (avec *WordNetLemmatizer*)

```
# Lemmatisation
# =================

# Lemmatisation est similaire au stemming mais les resultats doivent faire partie d'un dico (interne).
# Un mot devient un "lem" selon sa POS (part of speech) tel que verbe/adjectif/ nom/ ...

# Il nous faut le package "wordnet" de NLTK, à installer éventuellement MAIS UNE SEULE FOIS.

# Une autre manière d'installer serait :
# python -m nltk.downloader wordnet

from nltk.stem import WordNetLemmatizer

s = WordNetLemmatizer()
print(s.lemmatize('having', pos='v'))
print(s.lemmatize('have', pos='v'))
print(s.lemmatize('had', pos='v'))

print(s.lemmatize('fishing', pos='v'))
print(s.lemmatize('fish', pos='v'))
print(s.lemmatize('fisher', pos='n'))
print(s.lemmatize('fishes', pos='v'))
print(s.lemmatize('fished', pos='v'))

print(s.lemmatize('am', pos='v'))
print(s.lemmatize('is', pos='v'))
print(s.lemmatize('was', pos='v'))
```

Trace :

```
"""
TRACE :
have
have
have
fish
fish
fisher
fish
fish
be
be
be
having
"""
```

☞ "Having" en tant que "verbe" perd son suffixe "ing" mais si on demande à considérer "having" comme un nom (noun), le suffixe est conservé :

```
print(s.lemmatize('having', pos='n'))
# On aura "having" (un nom (noun))
```

IX-3 TfIdf + distance Cosine

Exemple de distance **Cosinus**. A tester !

```
from sklearn.feature_extraction.text import TfidfVectorizer

mydoclist = (
'Julie loves me more than Linda loves me',
'Jane likes me more than Julie loves me',
'He likes basketball more than baseball',
'Linda loves me and I love baseball more than Julie',
'Telling silly sentences like these one is not hard',
'Who loves Julie loves basketball but not baseball')

tfidf_vectorizer = TfidfVectorizer(min_df = 1)
tfidf_matrix = tfidf_vectorizer.fit_transform(mydoclist)

document_distances = (tfidf_matrix * tfidf_matrix.T)
print('Created a ' + str(document_distances.get_shape()(0)) + ' by ' + str(document_distances.get_shape()(1))
   + '
document-document cosine distance matrix.')
print(document_distances.toarray())

"""
TRACE :
((1.          0.73169241  0.16671067  0.67196845  0.          0.33128468)
 (0.73169241  1.          0.32778059  0.47157701  0.          0.19987881)
 (0.16671067  0.32778059  1.          0.28062636  0.          0.25948325)
 (0.67196845  0.47157701  0.28062636  1.          0.          0.28941437)
 (0.          0.          0.          0.          1.          0.09662796)
 (0.33128468  0.19987881  0.25948325  0.28941437  0.09662796  1.        ))
"""
```