BE Text Mining

(Text Mining in Python)
Mars 2024
ECL-LIRIS

Version Élèves

2023-24

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

Il y a plusieurs techniques pour procéder à la fouille de données textuelles (Text Mining, voir cours) :
Approche "Historique" : construction d'arbre syntaxique (non traitée ici)

Approche Sac de mot : les mots sont extraits des documents puis transformés en tokens puis et on travaille sur ces termes.

Approche LSA/LSI : suite à l'étape d'extraction de tokens, on créé un espace vectoriel où on associe à chaque terme (après pré-traitement) un vecteur de réels. Partant, on associera un vecteur de réels à chaque document du corpus.
Dans cette technique, on peut s'intéresser à la fréquence d'un terme (**tf** : term fequency), la valeur **tfidf** (term fequency, inverse document frequencey) du corpus. On obtient ainsi une (grande et creuse) matrice Termes-Documents.
Dans cette technique, la taille du vecteur associé à un terme dépend de la taille du corpus (le nombre de documents). Pour palier un trop grand nombre de termes, on procède à une réduction de dimension (PCA, SVD, NMF, t-SNE, ...)

Approche Word2Vec : Google a mise en place cette technique en associant un vecteur de taille fixe (300) à un terme (voir cours). L'approche Word2Vec se décline en deux techniques : **Skip-Gram** et **CBOW** (continuous Bag of Word). Word2Vec a également donné lieu au calcul de **Doc2Vec**. Notons que **Glove** et **FastText** sont deux autres techniques similaires à Word2Vec.
Le terme **Word Embedding** apparaît dès lors que le term est représenté par un vecteur de nombres.
Une fois l'embedding préparé, on peut appliquer différentes techniques du Data Mining aux résultats.

Approche Transformer est semblable à *Word2Vce* en ce qui concerne le word Embedding. Simplement, le vecteur représentant le terme est plus complexe, tient compte de la représentation (de taille 512) du terme (token) mais aussi de la position du token dans la phrase.
De plus et via une série d'opérations matricielles, la notion d'**Attention** a été introduite. Le résultat permet de mieux tenir compte du contexte de chaque terme.

Une différence majeur entre l'embedding du Word2Vec et celui du Tranformer est que le second sera traité par des réseaux de neurones tandis que Word2Vec permet l'emploi de n'importe quelle méthode de Data Mining.

I-2 Contenu de ce BE

Dans ce BE, nous utiliseront l'approche LSA et Word2Vec.

II LSA/LSI

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.

Étapes 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

II-1 Les Datasets pour ce BE

- Dataset Recettes (recipes)
- Datasets nltk (./nltk, ./nltd-data)
- Dataset twitter
- Df_papers.csv
- news20

II-2 Vérifications des packages Python3

Utiliser le fichier **1-install-load-packages.ipynb** pour vérifier que vous disposez des packages et les datasets python nécessaires.

II-3 Fichiers d'exercice

Utiliser les notebooks (et les datasets) fournis pour apprendre le Text Mining.

Le reste de ce document donne des explications pour d'autres exemples.

II-4 Travail à Réaliser

Démarche : suivant les notebooks étudiés, prendre un autre corpus et appliquer les mêmes opérations.
On utilisera la BD. movie_reviews.csv fourni pour y effectuer une analayse d'optinion / sentiments.
Pour cela, faites vous guider par le contenu du fichier **2-Polarite-sentiments-Sujet-principal.ipynb**.

III 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 8-1-load_et_explorartion-recipes.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

III-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....

III-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 **??** en page **??**.

III-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
```

III-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
"""
```

III-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))
"""
```

III-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.

III-6 Les mots usuels et leur suppression

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

III-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..../...

III-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. ../..

III-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 **??** en page **??**.

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)

III-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 **??** en page **??**.

III-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
"""
```

III-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**).

III-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.

III-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.

III-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
"""
```

III-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.

III-9 Vers les étapes suivantes

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

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

- **Fichier 4-ex-analyse-Bayes-sentiments-tweets.ipynb**
- Petit exemple de démonstration d'analyse de polarité de tweets.

Contenu :

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

- Quelques tweets d'exemple :

```
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')]
```

- Quelques fonctions utilitaires :

```
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
```

- Le traitement :

```
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.
```

• La trace d'exécution :

```
"""
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 : 6-4-bis-Data-NewsGroupes-SVD-Kmeans.ipynb**

  Voir le code !

VI Exemple Word2vect

Vous pouvez utiliser le note book suivant :

5-3bis-Bien-Un-ex-simple-doc2vect-avec-Plein-de-Similarities.ipynb

Mais avant cela, vous trouverez ci-dessous un petit exemple / démo de Vectorisation à la *Word2vect*.

Contenu :

- BD : dans le code
- word2vect sur un vocabulaire

- Créer un NoteBook avec ce code, exécuter cet exemple, essayer de mettre en place une analyse....

```
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)
```