Master Informatique, parcours MALIA-MIASHS

Carnets de note Python pour le cours de Network Analysis for Information Retrieval

Julien Velcin, laboratoire ERIC, Université Lyon 2

Prétraitements (partie 2)

Quelques prétraitements à connaître

- expressions réguières et nettoyages simples
- segmentation en mots (*tokenization*)
- mots-outils
- stemming et lemmatisation
- **n-grammes et collocations**

Au-delà des mots : n-grammes et collocations

Au lieu de traiter de termes composés d'un seul mot, on peut manipuler des séquences composées de *plusieurs* mots. En anglais, on parle parfois de **phrases**. On espère parvenir ainsi à trouver des termes correspondant à des *expressions* qui ont du sens.

Notez qu'on se contentera ici de suites de mots contigus.

Les séquences de mots peuvent être trouvées (et stockées) à l'aide d'algorithmes de programmation dynamique, par ex. celui des tableaux de suffixes (*suffix arrays*).

Le principal avantage à utiliser des expressions est d'identifier le sens d'un terme de manière plus précise, càd de lever une partie de l'ambiguïté des mots qui prennent souvent différents sens en fonction du contexte.

Par exemple, un **bigramme** est une séquence de deux mots consécutifs, comme dans :

> "american president"
> "world war"
> "health care"
> "bird is"
> "the sleepy"

On obtient les bigrammes en glissant une fenêtre de 2 mots sur le texte. On peut bien sûr généraliser à des séquences de 3 mots (trigrammes) voire n mots (n-grammes).

Mots ou expressions ?

Lewis (SIGIR, 1992) montre que les mots (1-gramme) ont des meilleures propriétés statistiques, en particulier pour la classification :

- les mots apparaissent plusieurs fois dans les documents
- les expressions n'apparaissent souvent qu'une seule fois

Les expressions fournissent plus d'information sémantique tandis que les mots ont souvent plusieurs sens (polysémie).

Essayons de calculer des n-grammes à l'aide de la librairie *scikit-learn*.

In [1]:
```
# Nous reviendrons plus tard sur cette classe de la librairie scikit-learn :
from sklearn.feature_extraction.text import CountVectorizer

with open("datasets/Frank Herbert - Dune.txt", "r", encoding='utf8') as f:
    texte_docs = [line.strip() for line in f.readlines()]

# range permet de spécifier la longueur des expressions considérées
# min_df permet de régler la rareté des expressions
vecto_bigrams = CountVectorizer(ngram_range=(2,2), min_df = 0)
matrice_doc_big = vecto_bigrams.fit_transform(texte_docs)
```

In [2]:
```
matrice_doc_big.shape
```

Out[2]:
```
(8608, 93241)
```

On peut observer les n-grammes extraits grâce à la fonction *get_feature_names()* :

In [4]:
```
print([s for s in vecto_bigrams.get_feature_names_out()[0:10]])
```

```
['000 kilos', '000 men', '000 meters', '000 solaris', '000 year', '023 per', '082
0', '092 10', '10 082', '10 092']
```

Le problème est qu'on peut vite se retrouver submergé par les n-grammes. Il devient crucial de filtrer les expressions, par exemple en fixant un seuil sur le nombre d'occurrences minimal.

Mais il existe d'autres moyens afin d'aller au-delà des n-grammes en cherchant les **collocations**.

Collocations

Une **collocation** est un groupe de mots qui ont développé une affinité particulière, de telle sorte à ce que les locuteurs les utilisent naturellement ensemble.

Une collocation comme "*New York City*" prend tout son sens à partir du moment où les trois mots sont utilisés ensemble.

D'autres exemples :

> "rig the election"
> "to cost an awful lot of money"
> "travailler plus pour gagner plus"

On dit que le sens du tout est plus grand que le sens de la somme de ses parties (*the meaning of the whole is greater than the meaning of the sum of its parts*).

Trouver des collocations à partir des n-grammes revient à calculer un **score** d'intérêt relatif à la présence de ces n mots ensemble. Ce score peut être calculé de différentes manières, par ex. :

- Pointwise Mutual Information (PMI)
- C-Value
- etc.

A noter que la PMI peut être utilisée pour calculer la collocation de mots qui ne se suivent pas forcément mais apparaissent dans le même contexte (par ex. la phrase ou le document).

Pointwise Mutual Information

$$PMI(w_1, w_2) = \log \frac{p(w_1 w_2)}{p(w_1)p(w_2)}$$

avec :

$p(w_1, w_2)$ la probabilité d'observer w_1 avec w_2 dans le corpus (*evidence*)
$p(w_1)p(w_2)$ la probabilité d'observer w_1 avec w_2 par chance si les événements étaient indépendants

En général, on utilise la valeur empirique pour estimer ces probabilités :

$p(t) \approx \frac{\#t}{N}$

si $\#t$ est le nombre de fois où le terme est observé et N le nombre de documents dans le corpus.

La PMI pose certains problèmes, en particulier le fait d'être sensible aux motifs très rares. A contrario, un motif composé de mots fréquents peut finir par être supprimé car la valeur sera

trop faible.

Une solution à ce problème consiste à lisser la distribution de probabilité (par ex. avec un lissage de Laplace).

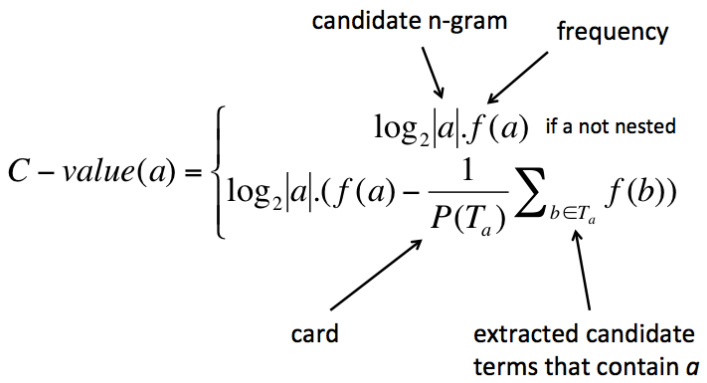
In [5]:
```
# quelques exemples
import nltk
from nltk.collocations import *
bigram_measures = nltk.collocations.BigramAssocMeasures()

print('%0.2f' % bigram_measures.pmi(10, (10, 100), 100000))
print('%0.2f' % bigram_measures.pmi(1, (10, 10), 100000))
print('%0.2f' % bigram_measures.pmi(1, (1, 10), 100000))
```

```
9.97
9.97
13.29
```

C-value

La C-value est une alternative à la PMI qui prend en compte l'écart relatif entre la fréquence d'un n-gramme et celui des (n-1)-gramme qui le composent.



La librairie *nltk* propose des algorithmes efficaces pour extraire des collocations en calculant un certain nombre de mesures (PMI, *likelihood ratio*, chi2...).

In [6]:
```
from nltk.tokenize import word_tokenize

bigram_measures = nltk.collocations.BigramAssocMeasures()
list_tokens = word_tokenize(" ".join(texte_docs))

# word_tokenize() se base sur le Penn Treebank en anglais
# une alternative est wordpunct_tokenize() qui utilise une expression régulière : \
```

In [7]:
```
finder = BigramCollocationFinder.from_words(list_tokens)
```

Nous pouvons afficher le résultat avec le score associé, ici uniquement les bigrammes avec la fréquence associée.

In [8]:
```
for i, t in enumerate(finder.score_ngrams(bigram_measures.raw_freq)):
    (c, v) = t
    print("{} : {:.4f} ".format(c, v))
    if i>20:
        break
```

```
('.', '``') : 0.0150
('.', "''") : 0.0098
(',', "''") : 0.0082
("''", '``') : 0.0073
('.', '.') : 0.0066
('said', '.') : 0.0062
('of', 'the') : 0.0053
('?', "''") : 0.0045
('.', 'The') : 0.0042
('.', 'He') : 0.0035
('in', 'the') : 0.0033
(',', 'the') : 0.0028
('``', 'I') : 0.0026
(':', '``') : 0.0025
("''", 'the') : 0.0023
('to', 'the') : 0.0023
("''", 'Paul') : 0.0023
(',', 'and') : 0.0020
('``', 'You') : 0.0019
('!', "''") : 0.0017
('.', 'She') : 0.0017
(',', 'but') : 0.0017
```

Dans ce qui suit, $nbest(m, k)$ affiche les k termes qui optimisent la mesure m et *apply_freq_filter(s)* permet de supprimer tous les termes qui apparaissent moins de s fois dans le corpus.

In [9]:
```
finder.nbest(bigram_measures.raw_freq, 10)
```

Out[9]:
```
[('.', '``'),
 ('.', "''"),
 (',', "''"),
 ("''", '``'),
 ('.', '.'),
 ('said', '.'),
 ('of', 'the'),
 ('?', "''"),
 ('.', 'The'),
 ('.', 'He')]
```

In [10]:
```
finder.apply_freq_filter(10)
finder.nbest(bigram_measures.pmi, 10)
```

Out[10]:
```
[('Helen', 'Mohiam'),
 ('Shaddam', 'IV'),
 ('Wan', 'na'),
 ('Missionaria', 'Protectiva'),
 ('gom', 'jabbar'),
 ('Salusa', 'Secundus'),
 ('ducal', 'signet'),
 ('per', 'cent'),
 ('Lisan', 'al-Gaib'),
 ('CHOAM', 'Company')]
```

On peut faire de même pour les trigrammes, par exemple :

In [11]:
```
trigram_measures = nltk.collocations.TrigramAssocMeasures()
finder_tri = TrigramCollocationFinder.from_words(list_tokens)
finder_tri.apply_freq_filter(10)
finder_tri.nbest(trigram_measures.pmi, 10)
```

Out[11]:
```
[('Bene', 'Gesserit', 'witch'),
 ('cleared', 'his', 'throat'),
 ('Baron', 'Vladimir', 'Harkonnen'),
 ('=', '=', '='),
 ('Bene', 'Gesserit', 'training'),
 ('Water', 'of', 'Life'),
 ('old', 'Reverend', 'Mother'),
 ('O.C', '.', 'Bible'),
 ('the', 'Missionaria', 'Protectiva'),
 ('the', 'Shield', 'Wall')]
```

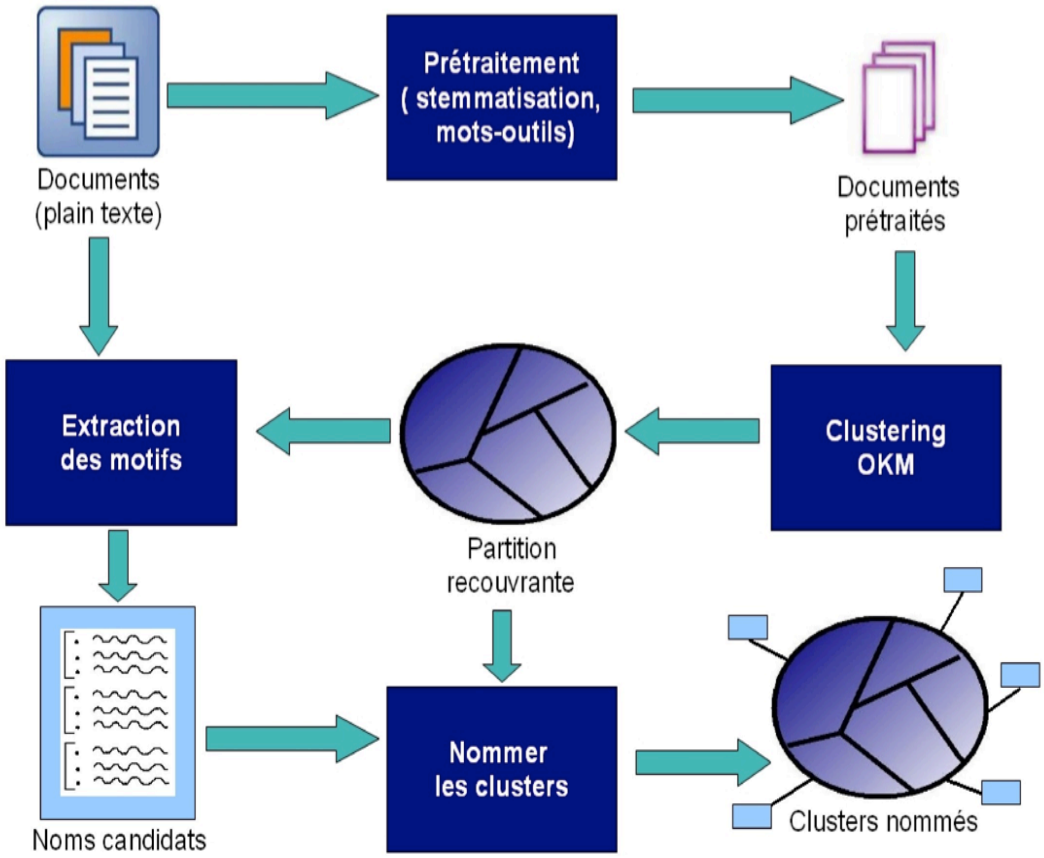
La fonction *apply_ngram_filter()* permet de créer ses propres filtres, par ex. basés sur un certain mot-clef :

In [12]:
```
trigram_measures = nltk.collocations.TrigramAssocMeasures()
finder_tri = TrigramCollocationFinder.from_words(list_tokens)

my_filter = lambda *w: 'Harkonnen' not in w
finder_tri.apply_ngram_filter(my_filter)
finder_tri.nbest(trigram_measures.pmi, 10)
```

Out[12]:
```
[('Harkonnen', 'bondsmen', 'endured'),
 ('Six', 'Harkonnen', 'bravos'),
 ('Harkonnen', 'defensive', 'sector'),
 ('evade', 'Harkonnen', 'restrictions'),
 ('adopt', 'Harkonnen', 'methods'),
 ('Bashar', 'Abulurd', 'Harkonnen'),
 ('Harkonnen', 'mercenaries', 'disguised'),
 ('Harkonnen', 'genetic', 'marker'),
 ('Harkonnen', 'troopers', 'maneuvered'),
 ('Harkonnen', 'orange', 'pennant')]
```

Exemple d'application au topic modeling

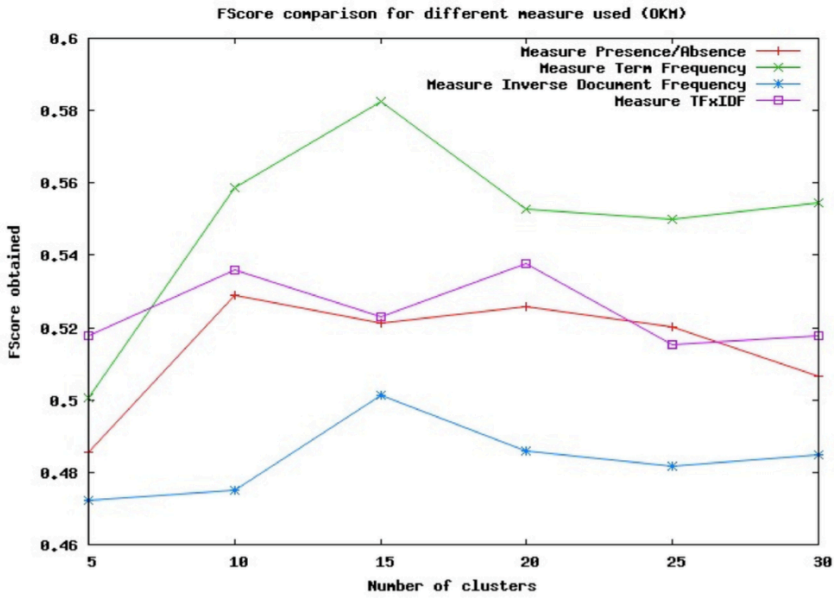


Regrouper les données textuelles et nommer les groupes à l'aide de classes recouvrantes (Rizoiu et al., 2010).

Quelques résultats :

Topic name	Highest rated words (after stemming!)	#	Text excerpt
Cocoa buffer stock	Stock, cocoa, buffer, deleg, icco, censure, produc, rule, meet, council	66	The international Cocoa Organization (ICCO) Council reached aggreement on rules to govern its buffer stock...
Oil and gas company	Oil, min, ga, year, barrel, billion, lt, compani, reserv, natur	169	Hamilton Oil Corp said reserves at the end of 1986 were 59.8 mln barrels of oil and 905.5 billions cubic feet of natural gas, ...
Tonnes of copper	Tonn, copper, cent, price, mine, effect, beef, lb, meat, export	100	Newmont Mining Corp said Magma Copper Co anticipates being able to produce copper at a profit by 1991, assuming copper...
United food and commercial workers	Unit, compani, plant, union, beef, lt, offer, contract, iowa, term	93	Brazil will export 6,000 tonnes of poultry and 10,000 tonnes of frozen meat to Iraq in exchange for oil, Petrobras Commercial Director Carlos Sant'Anna said. Brazil has a barter deal with Iraq...

Au passage, nous avons fait l'observation suivante :



Pour aller plus loin :

Sur l'extraction de collocations avec la librairie NLTK :
http://www.nltk.org/howto/collocations.html

Sur le lien entre les features issus de nltk (notamment les collocations) et scikit-learn :
http://www.nltk.org/_modules/nltk/classify/scikitlearn.html

Références

- Lewis, David D. "An evaluation of phrasal and clustered representations on a text categorization task." Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. 1992.
- M.A. Rizoiu, J. Velcin, J.H. Chauchat. Regrouper les données textuelles et nommer les groupes à l'aide de classes recouvrantes. Actes des 10ème journées francophones en Extraction et Gestion des Connaissances (EGC), Hammamet, Tunisie, 2010.

In [ ]: