Cours TAL – Labo 4

Nathan Gonzalez Montes et Vincent Guidoux

Exercice 1 - exécuter la NER dans NLTK

En utilisant nltk.ne_chunk (voir livre NLTK p. 283 - 4),extraire les entités nommées les plus fréquentes d'un texte de votre choix

In [1]:

```
# import sys
# !{sys.executable} -m pip install numpy
```

Importation des librairies nécessaires

In [2]:

```
import numpy
import nltk
import os, codecs
from nltk.tokenize import sent_tokenize
from nltk.tag import pos_tag
from nltk import ne_chunk
from nltk import FreqDist
from urllib import request
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('words')
nltk.download('maxent_ne_chunker')
```

```
[nltk_data] Downloading package punkt to C:\Users\Vincent
[nltk_data]     Guidoux\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Vincent Guidoux\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package words to C:\Users\Vincent
[nltk_data]     Guidoux\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to C:\Users\Vincent
[nltk_data]     Guidoux\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
```

Out[2]:

```
True
```

Importation du livre et nous prenons que la partie intéressante de celui-ci

In [3]:

```
url1 = "http://www.gutenberg.org/files/18488/18488.txt"

response = request.urlopen(url1)
raw = response.read().decode('utf8')[4340:489067]
```

Tokenisation en phrases, et des phrases en mots

In [4]:

```
sentences = nltk.sent_tokenize(raw)

sentences_of_words = [nltk.word_tokenize(sent) for sent in sentences]

len(sentences_of_words)
```

Out[4]:

```
6701
```

Nous taguons les mots dans chaque phrase grace à NLTK

In [5]:

```
tagged_sentences = [nltk.pos_tag(sentence) for sentence in sentences_of_words]

len(tagged_sentences)
```

Out[5]:

```
6701
```

Grace à `ne_chunk` les phrases sont transformées en arbre et les entités nommées sont mise en évidence grace au label qui nous permet de les retrouver et de les stocker

In [6]:

```
trees = []

for sentence in tagged_sentences:
    for chunk in ne_chunk(sentence):
        if hasattr(chunk, 'label'):
            trees.append((chunk.label(), ' '.join(c[0] for c in chunk)))
```

Nous en sortons les 15 entités nommées les plus fréquentes

In [7]:

```
nltk_most_commons = FreqDist(trees).most_common()
```

En vous inspirant du POS de Stanford, utiliser leur NER (https://nlp.stanford.edu/software/CRF-NER.html (https://nlp.stanford.edu/software/CRF-NER.html)) et la classe StanfordNERTagger de NLTK, pour extraire les entités nommées du même texte

importation du `StanfordNERTagger` et tokenization des mots.

In [8]:

```
from nltk.tag import StanfordNERTagger
st = StanfordNERTagger('stanford/classifiers/english.all.3class.distsim.crf.ser.g
z',
                       'stanford/stanford-ner.jar',
                       encoding='utf-8')
sentences = nltk.word_tokenize(raw)

len(sentences)
```

Out[8]:

```
106728
```

Taguage des mots pour trouver les entités nommées

In [9]:

```
# tagged_sentences = [st.tag(sentence) for sentence in sentences]
tagged_sentences = st.tag(sentences)
tagged_sentences[:10]
```

Out[9]:

```
[('CHAPTER', 'O'),
 ('I', 'O'),
 ('Priscilla', 'PERSON'),
 ('Glenn', 'PERSON'),
 ('stood', 'O'),
 ('on', 'O'),
 ('the', 'O'),
 ('little', 'O'),
 ('slope', 'O'),
 ('leading', 'O')]
```

Il faut désormais merger le mots tagués comme `('Priscilla', 'PERSON')` et `('Glenn', 'PERSON')` car `StanfordNERTagger` ne le fait pas

In [10]:

```
empty = 'O'

entities = []

last_entity = empty
last_word = empty
for tagged_word in tagged_sentences: # on parcourd chaque mot tagué
    current_entity = tagged_word[1]
    current_word = tagged_word[0]

    if current_entity != empty: # Si l'entity courante est une PERSON, ORGANIZATIO
N, GPE, ou LOCATION
        if last_entity == current_entity: # et quelle est la même que le mot précé
dent, nous les jumelons
            last_word = last_word + ' ' + current_word
        else: # on garde en mémoire son entité et le mot
            last_entity = current_entity
            last_word = current_word
    else: # si l'entity courante n'est pas nommée
        if last_entity != empty: # et qu'en mémoire il existe un mot tagué
            entities.append((last_entity,last_word))
            last_entity = empty
            last_word = empty

entities[:10]
```

Out[10]:

```
[('PERSON', 'Priscilla Glenn'),
 ('ORGANIZATION', 'Kenmore'),
 ('ORGANIZATION', 'Across Priscilla'),
 ('PERSON', 'Nathaniel Glenn'),
 ('PERSON', 'Nathaniel Glenn'),
 ('LOCATION', 'Kenmore'),
 ('PERSON', 'Glenn'),
 ('PERSON', 'Nathaniel'),
 ('LOCATION', 'Kenmore'),
 ('PERSON', 'Glenn')]
```

Nous en sortons les 15 entités nommées les plus fréquentes

In [11]:

```
stanfords_most_commons = FreqDist(entities).most_common()
```

Comparez la liste des NE les plus fréquentes et leurs types reconnus

In [12]:

```
print('(nltk, stanford)')

for nltk_freq, stanford in zip(nltk_most_commons[:15], stanfords_most_commons[:15
]):
    print(nltk_freq, stanford)
```

```
(nltk, stanford)
(('PERSON', 'Priscilla'), 227) (('PERSON', 'Priscilla'), 531)
(('PERSON', 'Farwell'), 194) (('PERSON', 'Farwell'), 311)
(('GPE', 'Priscilla'), 193) (('LOCATION', 'Ledyard'), 135)
(('PERSON', 'Ledyard'), 106) (('PERSON', 'Boswell'), 134)
(('PERSON', 'Master Farwell'), 64) (('LOCATION', 'Kenmore'), 71)
(('ORGANIZATION', 'Boswell'), 61) (('PERSON', 'Nathaniel'), 59)
(('ORGANIZATION', 'Travers'), 59) (('PERSON', 'Margaret'), 57)
(('PERSON', 'Nathaniel'), 55) (('PERSON', 'Priscilla Glenn'), 54)
(('GPE', 'Kenmore'), 55) (('ORGANIZATION', 'Travers'), 53)
(('PERSON', 'Margaret'), 52) (('PERSON', 'Dick'), 44)
(('GPE', 'Boswell'), 47) (('PERSON', 'McAlpin'), 40)
(('PERSON', 'Dick'), 42) (('PERSON', 'Margaret Moffatt'), 39)
(('ORGANIZATION', 'Priscilla'), 41) (('PERSON', 'Mary McAdam'), 36)
(('ORGANIZATION', 'McAlpin'), 40) (('PERSON', 'Theodora'), 34)
(('ORGANIZATION', 'Farwell'), 39) (('LOCATION', 'States'), 34)
```

Priscilla Glenn est le personnage principal de ce roman, il est donc normal de considérer que **Stanford** l'a mieux nommée alors que **nltk** pense à plusieurs reprise que Priscilla Glenn est autre chose.

In [13]:

```
for entity in nltk_most_commons:
    if "Priscilla" in entity[0][1] or "Glenn" in entity[0][1] :
        print(entity)


print(" - ")
for entity in stanfords_most_commons:
    if "Priscilla" in entity[0][1] or "Glenn" in entity[0][1]:
        print(entity)
```

```
(('PERSON', 'Priscilla'), 227)
(('GPE', 'Priscilla'), 193)
(('ORGANIZATION', 'Priscilla'), 41)
(('PERSON', 'Glenn'), 38)
(('PERSON', 'Priscilla Glenn'), 31)
(('ORGANIZATION', 'Priscilla Glenn'), 13)
(('PERSON', 'Nathaniel Glenn'), 4)
(('PERSON', 'Priscilla Glynn'), 4)
(('ORGANIZATION', 'Glenns'), 2)
(('PERSON', 'Mr. Glenn'), 2)
(('PERSON', 'Miss Priscilla Glenn'), 2)
(('ORGANIZATION', 'Priscilla Glynn'), 2)
(('PERSON', 'Glenns'), 1)
(('ORGANIZATION', 'Glenn'), 1)
(('PERSON', 'Miss Glenn'), 1)
(('ORGANIZATION', 'XIV Priscilla Glenn'), 1)
(('ORGANIZATION', 'XXIV Priscilla'), 1)
(('ORGANIZATION', 'Priscilla Travers'), 1)
(('PERSON', 'Priscilla Travers'), 1)
 - 
(('PERSON', 'Priscilla'), 531)
(('PERSON', 'Priscilla Glenn'), 54)
(('PERSON', 'Glenn'), 30)
(('PERSON', 'Priscilla Glynn'), 12)
(('PERSON', 'Nathaniel Glenn'), 5)
(('ORGANIZATION', 'Glenns'), 3)
(('PERSON', 'Theodora Glenn'), 3)
(('LOCATION', 'Priscilla'), 2)
(('ORGANIZATION', 'Across Priscilla'), 1)
(('PERSON', 'Miss Glenn'), 1)
(('PERSON', 'June Priscilla Glenn'), 1)
(('PERSON', '-- Priscilla'), 1)
(('ORGANIZATION', 'Priscilla Travers'), 1)
(('PERSON', 'Priscilla Travers'), 1)
```

Exercice 2 - comparer les deux NER sur les données CoNLL2003 (eng.test-a et test-b)

In [14]:

```
from nltk.metrics.scores import accuracy

#source : https://pythonprogramming.net/testing-stanford-ner-taggers-for-accuracy/

# Group NE data into tuples
def group(lst, n):
    for i in range(0, len(lst), n):
        val = lst[i:i+n]
        if len(val) == n:
            yield tuple(val)

def nltk_stanfortd_accuracy(filepath):

    raw_annotations = open(filepath).read()
    split_annotations = raw_annotations.split()

    # Amend class annotations to reflect Stanford's NERTagger
    for n,i in enumerate(split_annotations):
        if i == "I-PER":
            split_annotations[n] = "PERSON"
        if i == "I-ORG":
            split_annotations[n] = "ORGANIZATION"
        if i == "I-LOC":
            split_annotations[n] = "LOCATION"

    reference_annotations = list(group(split_annotations, 4)) # le fichier de test
contient 4 colonnes au lieu de 2 comme dans l'exemple

    #Nous devons prendre la première et dernière colonne
    reference_annotations = [(reference_annotation[0],reference_annotation[3]) for
reference_annotation in reference_annotations]

    # Ok, that looks good! But we'll also need the “clean” form of that data to st
ick into our NER classifiers. Let's make that happen too.
    pure_tokens = split_annotations[::4]

    # Let's go ahead and test the NLTK classifier.
    tagged_words = nltk.pos_tag(pure_tokens)
    nltk_unformatted_prediction = nltk.ne_chunk(tagged_words)

    #Convert prediction to multiline string and then to list (includes pos tags)
    multiline_string = nltk.chunk.tree2conllstr(nltk_unformatted_prediction)
    listed_pos_and_ne = multiline_string.split()

    # Delete pos tags and rename
    del listed_pos_and_ne[1::3]
    listed_ne = listed_pos_and_ne

    # Amend class annotations for consistency with reference_annotations
    for n,i in enumerate(listed_ne):
        if i == "B-PERSON":
            listed_ne[n] = "PERSON"
        if i == "I-PERSON":
            listed_ne[n] = "PERSON"
        if i == "B-ORGANIZATION":
```

```
                listed_ne[n] = "ORGANIZATION"
            if i == "I-ORGANIZATION":
                listed_ne[n] = "ORGANIZATION"
            if i == "B-LOCATION":
                listed_ne[n] = "LOCATION"
            if i == "I-LOCATION":
                listed_ne[n] = "LOCATION"
            if i == "B-GPE":
                listed_ne[n] = "LOCATION"
            if i == "I-GPE":
                listed_ne[n] = "LOCATION"

    # Group prediction into tuples
    nltk_formatted_prediction = list(group(listed_ne, 2))

    # Now we can test the accuracy of NLTK:
    nltk_accuracy = accuracy(reference_annotations, nltk_formatted_prediction)
    print("-------------------")
    print("NLTK accuracy")
    print(filepath, nltk_accuracy)

    print("-------------------")
    print("Stanford accuracy")
    stanford_prediction = st.tag(pure_tokens)
    stanford_accuracy = accuracy(reference_annotations, stanford_prediction)
    print(filepath, stanford_accuracy)



nltk_stanfortd_accuracy('eng.testa')
nltk_stanfortd_accuracy('eng.testb')
```

```
-------------------
NLTK accuracy
eng.testa 0.9118034821047734
-------------------
Stanford accuracy
eng.testa 0.9614370468029004
-------------------
NLTK accuracy
eng.testb 0.9002700038571979
-------------------
Stanford accuracy
eng.testb 0.9540779153987914
```

On remarque une meilleure performance du côté de Stanford

Sources

- FreqDist (https://stackoverflow.com/questions/4634787/freqdist-with-nltk)
- str.contains() (https://stackoverflow.com/questions/3437059/does-python-have-a-string-contains-substring-method)
- stanford_nltk_accuracy (https://pythonprogramming.net/testing-stanford-ner-taggers-for-accuracy/)
- The Place beyond the winds (http://www.gutenberg.org/ebooks/18488?msg=welcome_stranger)