language extractement

September 28, 2017

1 TP1

1.1 importer module nltk, charger les données du fichier txt

```
In [2]: from nltk import *
        #nltk.download()
        import nltk

In [3]: path=nltk.data.find(r"C:\Users\Lucas Yeoh\Desktop\TC3\tbbt\s3\txt\tbbts03e03.txt")
        raw=open(path,'rU').read()
```

1.2 normalizationchanger le format du type Text de module nltk

```
In [4]: tokens = word_tokenize(raw)
        type(tokens)  #<class 'list'>
        text = nltk.Text(tokens)
        type(text)  #<class 'nltk.text.Text'>

Out[4]: nltk.text.Text
```

1.3 tester sur les fonctions

1.3.1 1.segmentation en mots

```
In [5]: tokens=sorted(set(text))
```

1.3.2 2. segmentation en phrases

```
In [6]: sent_tokenize_list = sent_tokenize(raw)
        sent_tokenize_list[2]

Out[6]: 'Morning, Sheldon.'
```

1.3.3 3. annotation en parties du discours ou étiquetage morpho-syntaxique (PoS tagging)

```
In [7]: words=word_tokenize(raw)
        word_tag=pos_tag(words)
```

vérifier si les caractères dans le text sont bien encodé

```
In [8]: list_non_utf8=[]
        for w in words:
            for c in w:
                if int(ord(c)) > 128:
                    x=words.index(w)
                    list_non_utf8.append(x)
                    print c
        print list_non_utf8

        for l in list_non_utf8:
            print l
            print words[l]
```

```
[2950, 2950]
2950
ménage
2950
ménage
```

on observe que il y a une caractère français qui n'est pas correctement encodé donc il faut les reformaliser

```
In [9]: import unicodedata

        for l in list_non_utf8:
            words[l]=unicode(words[l],'utf-8')
            words[l]=unicodedata.normalize('NFD',words[l]).encode('ascii', 'ignore')
            print words[l]
```

```
menage
menage
```

1.3.4 4. stemming et lemmatisation

stemming

```
In [10]: porter=PorterStemmer()
         lancaster=LancasterStemmer()
         porter_list=[porter.stem(t) for t in words]
         lancaster_list=[lancaster.stem(t) for t in words]
```

lemmatisation

```
In [11]: wnl=WordNetLemmatizer()
         lemmatized_list=[wnl.lemmatize(t) for t in words]
```

1.3.5 5. reconnaissance d'entités nommées

```
In [12]: word_tag=pos_tag(words)
         chunked_words=ne_chunk(word_tag)
```

1.3.6 6. analyse syntaxique en constituants (constituency parsing)

```
In [13]: groucho_grammar = CFG.fromstring("""
         ... S -> NP VP
         ... PP -> P NP
         ... NP -> Det N | Det N PP | 'I'
         ... VP -> V NP | VP PP
         ... Det -> 'an' | 'my'
         ... N -> 'elephant' | 'pajamas'
         ... V -> 'shot'
         ... P -> 'in'
         ... """)
         sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
         print 'Chart parser'
         parser = nltk.ChartParser(groucho_grammar)
         for tree in parser.parse(sent):
             print(tree)
         print '--------------------------------------------------------------------------------
         '''
         print 'recursive descent parser'
         parser = nltk.RecursiveDescentParser(groucho_grammar)
         for tree in parser.parse(sent):
             count=count+1
             print(tree)
         '''

         print '--------------------------------------------------------------------------------
         print 'shift reduce parser'
         parser = nltk.ShiftReduceParser(groucho_grammar)
         for tree in parser.parse(sent):
             print(tree)
```

```
Chart parser
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas)))))
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
shift reduce parser
```

1.3.7 7. analyse syntaxique en dépendances (dependency parsing)

```
In [14]: groucho_dep_grammar = DependencyGrammar.fromstring("""
         ... 'shot' -> 'I' | 'elephant' | 'in'
         ... 'elephant' -> 'an' | 'in'
         ... 'in' -> 'pajamas'
         ... 'pajamas' -> 'my'
         ... """)
         print groucho_dep_grammar

         print'----------------------------'
         pdp = ProjectiveDependencyParser(groucho_dep_grammar)
         sent = 'I shot an elephant in my pajamas'.split()
         trees = pdp.parse(sent)
         for tree in trees:
             print tree

Dependency grammar with 7 productions
  'shot' -> 'I'
  'shot' -> 'elephant'
  'shot' -> 'in'
  'elephant' -> 'an'
  'elephant' -> 'in'
  'in' -> 'pajamas'
  'pajamas' -> 'my'
----------------------------
(shot I (elephant an (in (pajamas my))))
(shot I (elephant an) (in (pajamas my)))
```

2 analyse de corpus

2.1 1. identifier quels sont les personnages principals dans le corpus

D'abord, on va diviser le corpus en phrases et encore en mots. Ensuite on marque des étiquettes en chaque mot dans les phases. Et la prochain étape est de reconnaître les entité nommé, c'est à dire faire 'chunking' affecté dans chacun des éléments

```
In [15]: sentences = sent_tokenize(raw)
         tokenized_sentences = [word_tokenize(sentence) for sentence in sentences]
         tagged_sentences = [pos_tag(sentence) for sentence in tokenized_sentences]
         sentences
         tokenized_sentences
         tagged_sentences
```

```
chunked_sentences = ne_chunk_sents(tagged_sentences, binary=True)
chunked_sentences
```

```
Out[15]: <generator object <genexpr> at 0x000000001840B6C0>
```

On va trouver toutes les éléments qui est nommé comme 'NE'(personne)

```
In [16]: def extract_entity_names(t):
             entity_names = []

             if hasattr(t, 'label') and t.label:
                 if t.label() == 'NE':
                     entity_names.append(' '.join([child[0] for child in t]))
                 else:
                     for child in t:
                         entity_names.extend(extract_entity_names(child))

             return entity_names

         entity_names = []
         for tree in chunked_sentences:


             entity_names.extend(extract_entity_names(tree))


         print entity_names
```

```
['Goin', 'Sheldon', 'Come', 'Penny', 'French', 'French', 'Dear Lord', 'French', 'Look', 'Leonar
```

On va trouver les 15 mots plus fréquente qui sont pensé comme les nom de personne

```
In [17]: fdist=FreqDist(text)
         fname=[]
         for name in entity_names:
             if [fdist[name],name] not in fname:
                 fname.append([fdist[name],name])
         fname=sorted(fname, reverse=True)
         fname[0:15]
```

```
Out[17]: [[20, 'No'],
          [14, 'Yeah'],
          [8, 'Penny'],
          [8, 'Okay'],
          [6, 'Sheldon'],
          [5, 'French'],
          [5, 'Come'],
          [4, 'Raj'],
```

```
[4, 'Look'],
[3, 'Uh'],
[3, 'Kim'],
[3, 'Howard'],
[3, 'Ha'],
[3, 'Chocolate'],
[2, 'wikiHow']]
```

Evidemment il y en a beaucoup qui n'est pas nom de personne. âpre identifier à la main, on trouve que 'Penny','Shelton','Raj','Kim','Howard' dans les résultats sont prénoms de personnes.Comme penny apparaît 8 fois et Sheldon apparaît 6 fois, ces deux personnes apparaissent beaucoup plus que les autres, donc on assume que 'Penny' et 'Sheldon' sont les rôles principals dans cette corpus.

3 conclusions

Penny et Shelton sont les rôles principals de cette corpus

```
In [ ]:
```