自然言語処理

[Python][Python ライブラリ]

言語処理

■ NLTK(Natural Language Toolkit) のインストール

・Python NLTK(Natural Language Toolkit)
・NLTK
・NLTK How to

■ Python MeCab(日本語形態素解析) のインストール

・Python MeCab(日本語形態素解析)

■ Beautiful Soup (HTML 解析) のインストール

・Beautiful Soup (HTML 解析)

グラフ

■ NumPy (数学関数) のインストール

・NumPy (数学関数)

■ matplotlib (グラフ機能) のインストール

・matplotlib (グラフ機能)

テキストコーパスと語彙資源へのアクセス

コーパス

・テキストコーパスとは巨大なテキストのこと
・1 つ以上のジャンルから集められた素材をバランスよく含むようにデザインされる

・Python NLTK(Natural Language Toolkit) をインストールして、以下を試す。

テキストを検索する

・text1: Moby Dick by Herman Melville 1851 (白鯨) から、"monstrous" という単語を調べる

```
>>> text1.concordance('monstrous')
Building index...
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
```

```
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

簡単な統計処理

分散プロットを用いて表示

・ある単語がテキストの最初から最後までの間にどの位置に何回出現するのかを調べる
・以下のライブラリが必要
  ・Python NumPy
  ・Python matplotlib

```
>>> text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
```

グラフウィンドウの設定

CentOS にて上記手順でグラフウィンドウを表示できなかったので、ブラウザにグラフを表示する設定、matplotlib.use("WebAgg") を行っている。

・Python matplotlib
・モジュールロード時に一度だけ行えばよい
・Windows 環境では、デフォルトの TkAgg でグラフ表示された

```
import matplotlib
matplotlib.use("WebAgg")
```

頻度分布

・頻度分布は言語処理では頻繁に必要になる。
・NLTK ではそれを標準でサポートしている。

■ FreqDist を使って「白鯨 (Moby Dick)」から 頻出 50 単語を取り出す例

```
>>> fdist = FreqDist(text1)
>>> fdist
<FreqDist with 19317 samples and 260819 outcomes>
>>> vocabulary = fdist.keys()
>>> vocabulary[:50]
[',', 'the', '.', 'of', 'and', 'a', 'to', ';', 'in', 'that', "'", '-', 'his', 'it', 'I', 's', 'is',
'he', 'with', 'was', 'as', '"', 'all', 'for', 'this', '!', 'at', 'by', 'but', 'not', '--', 'him',
'from', 'be', 'on', 'so', 'whale', 'one', 'you', 'had', 'have', 'there', 'But', 'or', 'were', 'now',
'which', '?', 'me', 'like']
```

■試しに NLTK のホームページ語彙の出現頻度を数えてみる

```
import nltk
from nltk.probability import FreqDist
import urllib2
from BeautifulSoup import BeautifulSoup
import re

soup = BeautifulSoup(urllib2.urlopen("http://nltk.org/"))
l = []
for t in soup.findAll(text=True):
    l.extend(re.split('[ \n\t]',str(t)))
freq = FreqDist(l)
print freq
print freq.keys()[:50]
```

結果

```
 <FreqDist: '': 547, 'NLTK': 23, '&gt;&gt;&gt;': 11, 'and': 10, 'with': 8, '.': 7, 'a': 7, 'for': 7,
'=': 6, 'Python': 5, ...>
 ['', 'NLTK', '&gt;&gt;&gt;', 'and', 'with', '.', 'a', 'for', '=', 'Python', 'is', 'the', 'to',
''IN'),', '&#8211;', '(', 'Installing', 'Language', 'Natural', 'list', 'mailing', 'nltk', 'of', '|',
'\xc2\xb6',  ''JJ'),',  ')',  'tagged',  'tokens',  ''CD'),',  ''NNP'),',  ''RB'),',  '('Thursday',',
'('eight',',  '('morning',',  '('on',',  '(&quot;o'clock&quot;,',  '0',  '3',  '3.0',  'API',  'Contents',
'Data', 'Development', 'HOWTO', 'Index', 'NLTK,', 'News', 'Processing', 'Search']
```

上記では BeautifulSoup を利用したが、nltk.clean_html() で、HTML 文書からタグを取り除くことができる

```
html = urllib2.urlopen("http://nltk.org/").read()
untagged = nltk.clean_html(html)
tokens = nltk.word_tokenize(untagged)
```

■ NLTK のホームページ語彙の頻出 50 語累積頻度をプロットしてみる

```
import nltk
from nltk.probability import FreqDist
import urllib2
from BeautifulSoup import BeautifulSoup
import re
import matplotlib
matplotlib.use("WebAgg")

soup = BeautifulSoup(urllib2.urlopen("http://nltk.org/"))
l = []
for t in soup.findAll(text=True):
    l.extend(re.split('[ \n\t]',unicode(t)))
freq = FreqDist(l)
print freq
print freq.keys()[:50]
freq.plot(50, cumulative=True)
```

■ NLTK に定義されている頻度分布に関する関数

例	説明
fdist = FreqDist(samples)	samples で指定されたデータの頻度分布を生成
fdist.inc(sample)	sample で指定されたデータ数を 1 増やす
fdist['monstrous']	指定されたデータの出現数
fdist.freq('monstrous')	指定されたデータの頻度
fdist.N()	サンプルの総数
fdist.keys()	頻度の多い順ソートされたサンプル
for sample in fdist	サンプルを頻度の多い順に処理
fdist.max()	数のもっとも多いサンプル
fdist.tabulate()	頻度分布を表形式で表示
fdist.plot()	頻度分布のプロットを生成する
fdist.plot(cumulative=True)	累積頻度プロットを生成
fdist1 < fdist2	fdist1 のサンプルの頻度が fdist2 よりも少ないかをテスト

よりきめ細かい単語選択

・特性 P をもつ単語を探し出す場合、「V(語彙) に属し、特性 P を持つすべての W の集合」と言うことができる。[参考]

```
{ w | w ∈ V & P(w) }
```

Python の内包表記で表現すると

```
[ w for w in V if p(w) ]
```

となる

■ 'a' から始まる単語を抽出してみる

```
import nltk
import urllib2
from BeautifulSoup import BeautifulSoup
import re

soup = BeautifulSoup(urllib2.urlopen("http://nltk.org/"))
l = []
for t in soup.findAll(text=True):
    l.extend(re.split('[ \n\t]',unicode(t)))
v = set(l)
start_with_a = [w for w in v if w.startswith('a')]
print sorted(start_with_a)
```

結果

```
 [u'a', u'about', u'alike.', u'all,', u'along', u'alongside', u'amazing', u'analyzing', u'and',
u'announcements', u'as', u'available']
```

■ 5 文字以上かつ 3 回以上出現する単語を抽出してみる

```
import nltk
from nltk.probability import FreqDist
import urllib2
from BeautifulSoup import BeautifulSoup
import re

soup = BeautifulSoup(urllib2.urlopen("http://nltk.org/"))
l = []
for t in soup.findAll(text=True):
    l.extend(re.split('[ \n\t]',unicode(t)))
freq = FreqDist(l)
print sorted([w for w in set(l) if len(w) >= 5 and freq[w] >= 3])
```

結果

```
 [u''IN'),',  u''JJ'),',  u'&#8211;',  u'&gt;&gt;&gt;',  u'Installing',  u'Language',  u'Natural',
u'Python', u'mailing', u'tagged', u'tokens']
```

コロケーションとバイグラム

・コロケーションとは非常に頻繁に共起する一連の単語列
　・「red wine」(赤ワイン) はコロケーションだが、「the wine」(そのワイン) は異なる
・コロケーションの特徴は類似した意味を持つ単語で置き換えしづらい
　・例えば、赤ワインの赤を茶色に変えることは非常に違和感がある

・単語のペアは、バイアグラムとよばれ、bigrams() 関数で簡単に抜き出すことができる。

```
>>> from nltk import *
>>> bigrams(['this','is','a','pen'])
[('this', 'is'), ('is', 'a'), ('a', 'pen')]
```

・ここでタプルで表されるペアがバイグラムであり、本質的には頻出するバイグラムがコロケーション。

■コロケーションの作成

・http://nltk.org/howto/collocations.html
・About Python のページのコロケーションを作成してみる

```
import nltk
from nltk.collocations import *
import urllib2
from BeautifulSoup import BeautifulSoup
import re
from xml.sax.saxutils import unescape

soup = BeautifulSoup(urllib2.urlopen("http://www.python.org/about/"))
l = []
for t in soup.body.findAll(text=True):
    l.extend(re.split('[ \t\n]', unicode(t)))
l = [unescape(x,{"'":"'"}) for x in l]
txt = " ".join(l)

tokens = nltk.wordpunct_tokenize(txt)
bigram_measures = nltk.collocations.BigramAssocMeasures()
trigram_measures = nltk.collocations.TrigramAssocMeasures()
finder = BigramCollocationFinder.from_words(tokens)
print finder.nbest(bigram_measures.pmi, 10) # NORMALIZE WHITESPACE
```

結果

```
 [(u'Applications', u'Success'), (u'CORBA', u'objects'), (u'CPython', u'uses'), (u'Communications',
u'Engine'), (u'Core', u'Development'), (u'Documentation', u'Download'), (u'Download', u'\u4e0b\u8f7
d'), (u'English', u'Resources'), (u'Getting', u'Started'), (u'Insider', u'Blog')]
```

WordNet

・意味により整列された英語辞書

同義語

■ 'motorcar' の同義語を調べる例

・'car.n.01' というエントリは、synset(同義語集合) で「car」の最初の名詞の語義を表す
・synset.definition で定義文を参照
・synset.examples で例文を参照
・synset.lemmas で同義語集合を取得

```
from nltk.corpus import wordnet as wn

def print_syn_attr(w):
    for synset in wn.synsets(w):
        print '\n*** %s ***' % (synset)
        print '1.definition:\n\t%s' % (synset.definition)
        print '2.examples:\n\t%s' % (synset.examples)
        print '3.lemma_names'
```

```
        for lemma in synset.lemma_names:
            print '\t%s' % (lemma)

if __name__ == '__main__':
    print_syn_attr('motorcar')
```

結果

```
*** Synset('car.n.01') ***
1.definition:
  a motor vehicle with four wheels; usually propelled by an internal combustion engine
2.examples:
  ['he needs a car to get to work']
3.lemma_names
  car
  auto
  automobile
  machine
  motorcar
```

階層構造

・WordNet の同義語集合は抽象概念に対応しており、階層構造となっている。
・下位語を調べるには、synset.hyponyms() を利用する。
・上位語を調べるには、synset.hypernyms() を利用する
・下位語に至る経路が複数ある場合、synset.hypernym_paths()
・もっとも一般的な譲位後は、synset.root_hypernyms() で得られる。

■ 'motorcar' の階層構造を調べる

```
# -*- coding:"utf-8" -*-
from nltk.corpus import wordnet as wn

def print_syn_hierarchy(w):
    for synset in wn.synsets(w):
        print '\n*** %s ***' % (synset)
        print '1.definition:\n\t%s' % (synset.definition)
        print '2.examples:\n\t%s' % (synset.examples)
        print '3.hypernym:'
        for hypernym in synset.hypernyms():
            print '\t%s' % (hypernym)
        path_idx = 1
        print '4.hypernym paths:'
        for hypernym_path in synset.hypernym_paths():
            print '\t4.%d:%s' % (path_idx, hypernym_path)
            path_idx += 1
        print '5.hypernym root:'
        path_idx = 1
        for hypernym_root in synset.root_hypernyms():
            print '\t5.%d:%s' % (path_idx, hypernym_root)
            path_idx += 1

if __name__ == '__main__':
    print_syn_hierarchy('motorcar')
```

結果

```
*** Synset('car.n.01') ***
1.definition:
  a motor vehicle with four wheels; usually propelled by an internal combustion engine
2.examples:
  ['he needs a car to get to work']
3.hypernym:
  Synset('motor_vehicle.n.01')
4.hypernym paths:
       4.1 :[Synset('entity.n.01 '),    Synset('physical_entity.n.01 '),    Synset('object.n.01 '),
Synset('whole.n.02'), Synset('artifact.n.01'), Synset('instrumentality.n.03'), Synset('container.n.01
```

```
'), Synset('wheeled_vehicle.n.01'), Synset('self-propelled_vehicle.n.01'), Synset('motor_vehicle.n.01
'), Synset('car.n.01')]
        4.2 :[Synset('entity.n.01 '),    Synset('physical_entity.n.01 '),    Synset('object.n.01 '),
Synset('whole.n.02   '),        Synset('artifact.n.01   '),        Synset('instrumentality.n.03   '),
Synset('conveyance.n.03   '),       Synset('vehicle.n.01   '),       Synset('wheeled_vehicle.n.01   '),
Synset('self-propelled_vehicle.n.01'), Synset('motor_vehicle.n.01'), Synset('car.n.01')]
 5.hypernym root:
   5.1:Synset('entity.n.01')
```

語彙の関係

・構成要素への関係 (メロニム)
・構成要素とそれを含む要素の関係 (ホロニム)

■ 'tree' の例

```
from nltk.corpus import wordnet as wn

def print_syn_relation(w):
    for synset in wn.synsets(w):
        print '\n*** %s ***' % (synset)
        print '1.definition:\n\t%s' % (synset.definition)
        print '2.examples:\n\t%s' % (synset.examples)
        print '3.part meronyms:'
        for part_meronym in synset.part_meronyms():
            print '\t%s' % (part_meronym)
        print '4.substance meronyms:'
        for substance_meronym in synset.substance_meronyms():
            print '\t%s' % (substance_meronym)
        print '5.member holonyms:'
       for member_holonym in synset.member_holonyms():
            print '\t%s' % (member_holonym)

if __name__ == '__main__':
    print_syn_relation('tree')
```

結果

・part meronyms: 幹 (trunk) や枝 (limb) は、木 (tree) の一部
・substance meronyms: 材質的には、心材 (heartwood) と辺材 (sapwood) で構成
・member holonyms: 木 (tree) は集まると森 (forest) を形成する

```
*** Synset('tree.n.01') ***
1.definition:
  a tall perennial woody plant having a main trunk and branches forming a distinct elevated crown;
includes both gymnosperms and angiosperms
2.examples:
  []
3.part meronyms:
  Synset('burl.n.02')
  Synset('crown.n.07')
  Synset('stump.n.01')
  Synset('trunk.n.01')
  Synset('limb.n.02')
4.substance meronyms:
  Synset('heartwood.n.01')
  Synset('sapwood.n.01')
5.member holonyms:
  Synset('forest.n.01')
    :( 略 )
```

テキストの正規化

ステマー

・単語からすべての接辞を取り除く処理をステミングと呼ぶ。

・NLTK には、すぐ利用可能なステマーがいくつか用意されている
・ステミング処理は形式的に定義されているわけではなく、一般的には利用しようと思っているアプリケーションにもっとも適切なステマーを選ぶ。
・単語が異なる語形であっても検索できるようテキストのインデックスを作成したいなら、Porter ステマーはよい選択。

見出し語化

・語形を辞書に記述されている形に変換する作業を " 見出し語化 " と呼ぶ
・WordNet のレマタイザ (見出し語化ツール) は、結果が辞書に存在する場合にのみ、接辞を削除するようになっている。
・テキストの語彙を収集する、または有効な見出し語のリストが必要な場合は、WordNet のレマタイザを使うのはよい選択だろう。

■ Porter ステマー、Lancaster ステマーと WordNet レマタイザの使用例

```
import nltk
import urllib2

html = urllib2.urlopen("http://nltk.org").read()
untagged = nltk.clean_html(html)
tokens = nltk.word_tokenize(untagged)
tokens.sort()
tokens = set(tokens)

porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
lemmatizer = nltk.WordNetLemmatizer()

print "****************** porter ********************"
print [t + "->" + porter.stem(t) for t in tokens if t != porter.stem(t)]
print "****************** lancaster ********************"
print [t + "->" + lancaster.stem(t) for t in tokens if t != lancaster.stem(t)]
print "****************** lemmatizer ********************"
print [t + "->" + lemmatizer.lemmatize(t) for t in tokens if t != lemmatizer.lemmatize(t)]
```

結果例

```
****************** porter ********************
  ['semantic->semant',    'programming->program',    'sentence->sentenc',    'resources->resourc',
'tagged->tag',  'using->use',  'linguistics->linguist',  'terms->term',  'classification->classif',
'simple->simpl', 'writing->write', 'only->onli', 'has->ha'...(略)]
****************** lancaster ********************
  ['all->al',    'semantic->sem',    'programming->program',    'sentence->sent',    'over->ov',
'resources->resourc',  'tagged->tag',  'JJ->jj',  'using->us',  'Bird->bird',  'linguistics->lingu',
'terms->term', 'classification->class', 'Klein->klein'...(略)]
****************** lemmatizer ********************
  ['resources->resource',    'terms->term',    'has->ha',    'modules->module',    'updates->update',
'linguists->linguist',  'educators->educator',  'users->user',  'entities->entity',  'guides->guide',
'libraries->library', 'students->student', 'interfaces->interface'...(略)]
```