wordNet-rcd180001

September 25, 2022

```
[ ]: #Import Libraries
     import nltk
     from nltk.corpus import wordnet as wn
```

1 WordNet

WordNet is a database of nouns, verbs, adjectives, etc. with definitions and examples. It's organized by synonyms sets called synsets that express a concept.

1.1 Noun Synsets

```
[ ]: #Noun Synets
     wn.synsets('country')
```

```
[ ]: [Synset('state.n.04'),
      Synset('country.n.02'),
      Synset('nation.n.02'),
      Synset('country.n.04'),
      Synset('area.n.01')]
```

1.1.1 Synsets Methods

definition() - Retrieves the word's gloss or definition
examples() - Gives examples of using the word
lemmas() - Returns synonyms of the given word

```
[ ]: #Extract Definitions, Usage, and Lemmas
     print(wn.synset('state.n.04').definition(), "\n")
     print(wn.synset('state.n.04').examples(), "\n")
     print(wn.synset('state.n.04').lemmas(), "\n")
```

```
a politically organized body of people under a single government

['the state has elected a new president', 'African nations', "students who had
come to the nation's capitol", "the country's largest manufacturer", 'an
industrialized land']
```

```
[Lemma('state.n.04.state'), Lemma('state.n.04.nation'),
Lemma('state.n.04.country'), Lemma('state.n.04.land'),
Lemma('state.n.04.commonwealth'), Lemma('state.n.04.res_publica'),
Lemma('state.n.04.body_politic')]
```

1.1.2 Noun Traversal

As we traverse through more hypernyms, the more broad the nouns become. This means that multiple nouns are hyponyms of a noun such as location. To prevent a synset to store too many relations with other synsets, they're split up into different defintions and parts of speech to make the different relationships. They're also placed in different levels to prevent one synset to have too many relations.

```
[ ]: #Traversal
country = wn.synset('country.n.02')
hypernyms = lambda s: s.hypernyms()
print(list(country.closure(hypernyms)), "\n")

location = wn.synset('location.n.01')
level = country.hyponyms();
print(level)
```

```
[Synset('administrative_district.n.01'), Synset('district.n.01'),
Synset('region.n.03'), Synset('location.n.01'), Synset('object.n.01'),
Synset('physical_entity.n.01'), Synset('entity.n.01')]

[Synset('african_country.n.01'), Synset('asian_country.n.01'),
Synset('banana_republic.n.01'), Synset('buffer_state.n.01'),
Synset('european_country.n.01'), Synset('fatherland.n.01'),
Synset('kingdom.n.02'), Synset('north_american_country.n.01'),
Synset('south_american_country.n.01'), Synset('sultanate.n.01'),
Synset('tax_haven.n.01')]
```

1.1.3 Synset Relationships

Each synset has hypernyms, hyponyms, meronyms, holonyms, and antonyms. They're used to describe what type of relationship they have with another synset and they're level in relation to another synset.
Hypernym - A higher level than the another synset and is considered more broad
Hyponym - A lower level than another synset and is more specific in its definition
Meronym - A lower level than another synset and the synset is a part of a certain object
Holonym - A higher level than another synset and the synset is a whole object with different parts

```
[ ]: #Synset Relationships
country = wn.synset('country.n.02')

print(country.hypernyms())
print(country.hyponyms(), "\n")

print(country.part_meronyms())
print(country.member_meronyms(), '\n')

print(country.part_holonyms())
print(country.member_holonyms(), '\n')

print(country.lemmas()[0].antonyms())
```

```
[Synset('administrative_district.n.01')]
[Synset('african_country.n.01'), Synset('asian_country.n.01'),
Synset('banana_republic.n.01'), Synset('buffer_state.n.01'),
Synset('european_country.n.01'), Synset('fatherland.n.01'),
Synset('kingdom.n.02'), Synset('north_american_country.n.01'),
Synset('south_american_country.n.01'), Synset('sultanate.n.01'),
Synset('tax_haven.n.01')]

[Synset('domain.n.02'), Synset('midland.n.02')]
[Synset('department.n.02'), Synset('state.n.01')]

[]
[]

[]
```

1.2 Verb Synset

```
[ ]: #Verb Synsets
     wn.synsets("direct")
```

```
[ ]: [Synset('direct.v.01'),
      Synset('target.v.01'),
      Synset('direct.v.03'),
      Synset('direct.v.04'),
      Synset('lead.v.01'),
      Synset('send.v.01'),
      Synset('aim.v.01'),
      Synset('conduct.v.02'),
      Synset('direct.v.09'),
      Synset('calculate.v.05'),
      Synset('steer.v.01'),
      Synset('address.v.03'),
      Synset('mastermind.v.01'),
      Synset('direct.a.01'),
      Synset('direct.s.02'),
      Synset('direct.a.03'),
      Synset('lineal.a.01'),
      Synset('direct.a.05'),
      Synset('direct.a.06'),
      Synset('direct.a.07'),
      Synset('direct.s.08'),
      Synset('direct.s.09'),
      Synset('direct.s.10'),
      Synset('directly.r.01')]
```

1.2.1 Verb Hypernyms

Verb synsets have relationships between other synsets like nouns. Unlike nouns, there's not top level synset for verbs like 'entity' for nouns. This is due to as the more broader the verb, the more the meaning has changed.

```
[ ]: mastermind = wn.synset('mastermind.v.01')

     #Definition, Examples, Lemmas
     print(wn.synset('mastermind.v.01').definition(), "\n")
     print(wn.synset('mastermind.v.01').examples(), "\n")
     print(wn.synset('mastermind.v.01').lemmas(), "\n")

     #Traversal
     hypernyms = lambda s: s.hypernyms()
     print(list(mastermind.closure(hypernyms)), "\n")
```

```
plan and direct (a complex undertaking)

['he masterminded the robbery']

[Lemma('mastermind.v.01.mastermind'), Lemma('mastermind.v.01.engineer'),
Lemma('mastermind.v.01.direct'), Lemma('mastermind.v.01.organize'),
Lemma('mastermind.v.01.organise'), Lemma('mastermind.v.01.orchestrate')]

[Synset('plan.v.02'), Synset('think.v.03')]
```

1.3 Morphy

A function that's rules based to obtain the root form of a a given word.

```
[ ]: #Obtain the root form of a word
     print(wn.morphy('mastermind', wn.VERB))
     print(wn.morphy('stated', wn.VERB))
     print(wn.morphy('emergencies', wn.NOUN))
```

```
mastermind
state
```

```
emergency
```

1.4 Word Similarity

The Wu-Palmer metric measures the similarity of 2 words by comparing their depths and the most specific ancestor. The formula to calculate this metric is

$$2 \times \frac{\text{depth(most specific ancestor(Word 1, Word 2))}}{\text{depth(Word 2)} + \text{depth(Word 1)}}$$

The Lesk algrithim is used to determine what kind of word is being used in a sentence or word sense disambiguation. The algorithm uses the context in the sentence and find the most overlapping synset.

[ ]:
```python
#Import
from nltk.wsd import lesk

book = wn.synset('book.n.01')
pamphlet = wn.synset('pamphlet.n.01')
direct = wn.synset('direct.v.01')

#Wu-Palmer Metric
print("Wu-Palmer: ", wn.wup_similarity(book, pamphlet) * 100)
print("Wu-Palmer: ", wn.wup_similarity(direct, direct) * 100)

#Lesk Algorithm
sentence = ['I', 'tore', 'up', 'my', 'textbook', '.']
print(lesk(sentence, 'textbook', 'n'))

sentence = ['I', 'tore', 'up', 'my', 'book', '.']
print(lesk(sentence, 'book', 'n'))
```

```
Wu-Palmer:  95.23809523809523
Wu-Palmer:  100.0
Synset('textbook.n.01')
Synset('script.n.01')
```

1.5 SentiWordNet

SentiWordNet assigns a positivity, negativity, and objectivity scores to a synset. It can be used to be used to analyze the sentiment of sentences.

[ ]:
```python
from nltk.corpus import sentiwordnet as swn

#Print out hate's sentiment scores
hate = swn.senti_synset('hate.v.01')
print(hate)
print(hate.obj_score(), "\n")
```

```
#Sentence
positive = 0
negative = 0
sentence = "He is amazed with this new sponge"
tokens = sentence.split()

for t in tokens:
    syn_list = list(swn.senti_synsets(t))
    if syn_list:
        syn = syn_list[0]
        positive += syn.pos_score()
        negative += syn.neg_score()
        print(syn)

print("\nPositive: ", positive)
print("Negative: ", negative)
```

```
<hate.v.01: PosScore=0.0 NegScore=0.75>
0.25

<helium.n.01: PosScore=0.0 NegScore=0.0>
<be.v.01: PosScore=0.25 NegScore=0.125>
<amaze.v.01: PosScore=0.0 NegScore=0.25>
<new.a.01: PosScore=0.375 NegScore=0.0>
<sponge.n.01: PosScore=0.0 NegScore=0.0>

Positive:  0.625
Negative:  0.375
```

The output of the sentiment of the sentence, seems to be accurate. But, I feel like there are some words that seems neutral to have a positive or negative emotion and could throw off the output. These sentiment scores could be useful to find the most important words in a text.

1.6 Collocation

A collocation is a pair of words that frequently appear in a body of a text and where you cannot replace a word with a synonym. They're found by finding bigrams and find which bigrams have high frequencies

[ ]:
```
from nltk.book import text4
import math

#Collocations
print(text4.collocations())

#Calculate mutual information
text = ' '.join(text4.tokens)
vocab = len(set(text))
```

```
chiefMagistrate = text.count('Chief Magistrate')/vocab

chief = text.count('Chief')/vocab
magistrate = text.count('Magistrate')/vocab

mutualInfo = math.log2(chiefMagistrate/ (chief * magistrate))

print("\np(Chief Magistrate): ", chiefMagistrate)
print("p(Chief): ", chief)
print("p(Magistrate): ", magistrate)
print("Point-Wise Mutual Information: ", mutualInfo)

oneAnother = text.count('one another')/vocab

one = text.count('one')/vocab
another = text.count('another')/vocab

mutualInfo = math.log2(oneAnother/ (one * another))

print("\np(one another): ", oneAnother)
print("p(one): ", one)
print("p(another): ", another)
print("Point-Wise Mutual Information: ", mutualInfo)
```

```
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
None

p(Chief Magistrate):  0.11904761904761904
p(Chief):  0.3333333333333333
p(Magistrate):  0.13095238095238096
Point-Wise Mutual Information:  1.4474589769712212

p(one another):  0.2619047619047619
p(one):  7.166666666666667
p(another):  0.7976190476190477
Point-Wise Mutual Information:  -4.4479598258014175
```

The first bigram I chose ‘Chief Magistrate’ seems to be a collocation since you can’t replace any of the words and keep the same meaning, the point-wise mutal information (PMI) confirms this since it’s positive. The ‘of another’ bigram is not a bigram since the PMI is negative.