WordNet

September 22, 2022

Wordnet is a lexical database provided by the natural language tool kit. Wordnet is used to find the meanings of words, their usuage, their lemmas and generally to find conceptual relationships between words.

[1]:
```
import nltk
from nltk.corpus import wordnet as wn
```

All Synsets of The noun "Color"

[2]:
```
wn.synsets("color")
```

[2]:
```
[Synset('color.n.01'),
 Synset('color.n.02'),
 Synset('color.n.03'),
 Synset('color.n.04'),
 Synset('semblance.n.01'),
 Synset('coloring_material.n.01'),
 Synset('color.n.07'),
 Synset('color.n.08'),
 Synset('color.v.01'),
 Synset('tinge.v.01'),
 Synset('color.v.03'),
 Synset('color.v.04'),
 Synset('color.v.05'),
 Synset('discolor.v.03'),
 Synset('color.a.01')]
```

Definition, usage, and lemmas for the chosen synset, repectively

[3]:
```
wn.synset("semblance.n.01").definition()
```

[3]:
```
'an outward or token appearance or form that is deliberately misleading'
```

[4]:
```
wn.synset("semblance.n.01").examples()
```

[4]:
```
['he hoped his claims would have a semblance of authenticity',
 'he tried to give his falsehood the gloss of moral sanction',
 'the situation soon took on a different color']
```

[5]:
```
wn.synset("semblance.n.01").lemmas()
```

[5]:
```
[Lemma('semblance.n.01.semblance'),
 Lemma('semblance.n.01.gloss'),
 Lemma('semblance.n.01.color'),
 Lemma('semblance.n.01.colour')]
```

A Traversal up the WordNet Heirarchy, noun

It would seem that entity is the top of the noun heirarchy. This is interesting because abstraction is arguably the opposite of an entity. Entity is the base of all nouns.

[6]:
```
color = wn.synset("color.n.01")
hyper = lambda s: s.hypernyms()
list(color.closure(hyper))
```

[6]:
```
[Synset('visual_property.n.01'),
 Synset('property.n.02'),
 Synset('attribute.n.02'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

Hypernym, hyponym, meronym, holonym, antonym of the chosen synset, respectively

[7]:
```
(wn.synset("semblance.n.01")).hypernyms()
```

[7]:
```
[Synset('appearance.n.01')]
```

[8]:
```
(wn.synset("semblance.n.01")).hyponyms()
```

[8]:
```
[Synset('color_of_law.n.01'),
 Synset('disguise.n.01'),
 Synset('face_value.n.02'),
 Synset('guise.n.01'),
 Synset('simulacrum.n.01'),
 Synset('verisimilitude.n.01')]
```

[9]:
```
(wn.synset("semblance.n.01")).part_meronyms()
```

[9]:
```
[]
```

[10]:
```
(wn.synset("semblance.n.01")).lemmas()[0].antonyms()
```

[10]:
```
[]
```

All Synsets of The verb "meander"

[11]:
```
wn.synsets("meander")
```

[11]:
```
[Synset('meander.n.01'), Synset('ramble.n.01'), Synset('weave.v.04')]
```

Definition, usage, and lemmas for the chosen synset, repectively

[12]:
```
wn.synset("weave.v.04").definition()
```

[12]:
```
'to move or cause to move in a sinuous, spiral, or circular course'
```

[13]:
```
wn.synset("weave.v.04").examples()
```

[13]:
```
['the river winds through the hills',
 'the path meanders through the vineyards',
 'sometimes, the gout wanders through the entire body']
```

[14]:
```
wn.synset("meander.v.01").lemmas()
```

[14]:
```
[Lemma('weave.v.04.weave'),
 Lemma('weave.v.04.wind'),
 Lemma('weave.v.04.thread'),
 Lemma('weave.v.04.meander'),
 Lemma('weave.v.04.wander')]
```

A Traversal up the WordNet Heirarchy, verb

It would seem that it only travels once up the heirarchy, to a more general sense of the word.

[15]:
```
color = wn.synset("meander.v.01")
hyper = lambda s: s.hypernyms()
list(color.closure(hyper))
```

[15]:
```
[Synset('travel.v.01')]
```

using Morphy to find different forms of a word

[16]:
```
wn.morphy("sprinting",wn.VERB)
```

[16]:
```
'sprint'
```

[17]:
```
wn.morphy("sprinted",wn.VERB)
```

[17]:
```
'sprint'
```

[18]:
```
wn.morphy("sprints",wn.VERB)
```

[18]:
```
'sprint'
```

Comparing two words using two distinct methods

it seems that the Wu-Palmer similarity is significantly better at identifying similiar words. They built in similarity is passable at best because it doesn't even identify words that are basically synonyms, or hypernyms. They both return a rating from 0 to 1, dissimilar to similar

[19]:
```
bird = wn.synset('bird.n.01')
pigeon = wn.synset('pigeon.n.01')
bird.path_similarity(pigeon)
```

[19]: 0.25

[20]:
```
run = wn.synset('run.v.01')
sprint = wn.synset('sprint.v.01')
wn.path_similarity(run, sprint)
```

[20]: 0.5

[21]:
```
wn.wup_similarity(bird, pigeon)
```

[21]: 0.8695652173913043

[22]:
```
wn.wup_similarity(run, sprint)
```

[22]: 0.8571428571428571

SentiWordnet

This is used to extract a potential opinion from a sentance or word and it gives a score for postive, neutral, and negative from 0 to 1 such that their probabilities sum to 1. The higher the number, the closer it is to the attribute. This could be potentially used to scan for hateful comments on social media and flag them as toxic. It can be used to quickly get statistics on public speakers and how their diction impacts how their meaning is conveyed.

[23]:
```
from nltk.corpus import sentiwordnet as swn
```

[24]:
```
senti_list = list(swn.senti_synsets('rancid'))
for item in senti_list:
    print(item)
```

```
<rancid.s.01: PosScore=0.0 NegScore=0.625>
<sour.s.01: PosScore=0.0 NegScore=0.375>
```

The negative score for rancid seems a little low at .625, I don't know anyone who would enjoy a rancid odor, taste, or anything for that matter. It's interesting that it rates opulent (deluxe) and expensive on the positve scale. This rating has a positve bias towards words that imply wealth. This functionality can be used in NLP for a chat bot in finding the appropriate response to a prompt.

[25]:
```
sent = 'Martha absolutely recommends the opulent, expensive caviar'
negTot = 0
posTot = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
```

```
    if syn_list:

        syn = syn_list[0]
        print(syn)
        negTot += syn.neg_score()
        posTot += syn.pos_score()

print("neg\tpos counts")
print(negTot, '\t', posTot)
```

```
<absolutely.r.01: PosScore=0.5 NegScore=0.0>
<recommend.v.01: PosScore=0.0 NegScore=0.125>
<expensive.a.01: PosScore=0.5 NegScore=0.0>
<caviar.n.01: PosScore=0.0 NegScore=0.0>
neg     pos counts
0.125    1.0
```

0.0.1 collocations

collocations are defined as words that are often found together in such a way that changing one of the words with a synonym would change the meaning. An example would be to make your bed, to pay interest to someone. These are especially hard for machines understand because their patterns are only observed through experiencing the language and differentiating between literal and figurative language.

[30]:
```
from nltk.book import *
text4.collocations()
text = ' '.join(text4.tokens)
```

```
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
```

[31]:
```
import math
vocab = len(set(text4))
hg = text.count('foreign nations')/vocab
print("p(foreign nations) = ",hg )
h = text.count('foreign')/vocab
print("p(foreign) = ", h)
g = text.count('nations')/vocab
print('p(nations) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(foreign nations) =  0.0014962593516209476
p(foreign) =  0.01027431421446384
p(nations) =  0.020448877805486283
pmi =  2.8322245851494996
```

[32]:
```
hg = text.count('public debt')/vocab
print("p(public debt) = ",hg )
h = text.count('public')/vocab
print("p(public) = ", h)
g = text.count('debt')/vocab
print('p(debt) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(public debt) =  0.001396508728179551
p(public) =  0.03341645885286783
p(debt) =  0.004189526184538653
pmi =  3.318334830163354
```

[33]:
```
hg = text.count('Chief Magistrate')/vocab
print("p(Chief Magistrate) = ",hg )
h = text.count('chief')/vocab
print("p(chief) = ", h)
g = text.count('Magistrate')/vocab
print('p(Magistrate) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(Chief Magistrate) =  0.000997506234413965
p(chief) =  0.0017955112219451373
p(Magistrate) =  0.0010972568578553616
pmi =  8.983886091037398
```

The above collocations are calculated with a probablistic mutual information process. It seems that public debt ranks higher than foreign nations, which sound about right because foreign nations are literal whereas you are not literally in debt to the public. This, however, marks chief magistrate as very high because they are seen together often. This is wrong, but it makes sense how the result was achieved. The set of words would need to be expanded to see that they are only seen together because they are essentially one word