WordNet_nxw180009

September 25, 2022

1 WordNet

[315]:
```
### Imports

from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import *
import math
```

1.0.1 Summary of Wordnet

WordNet is a lexical database of semantic relations between words in more than 200 languages. WordNet links words into semantic relations including synonyms, hyponyms, and meronyms. The synonyms are grouped into synsets with short definitions and usage examples. The original goal of Wordnet was to support theories of human semantic memory that suggested that people organize concepts mentally in some kind of hierarchy.

1.1 This program selects a noun, then outputs all synsets.

[316]:
```
print("The chosen noun is phone.")

print('Sysnset list:', wn.synsets('phone'))
```

```
The chosen noun is phone.
Sysnset list: [Synset('telephone.n.01'), Synset('phone.n.02'),
Synset('earphone.n.01'), Synset('call.v.03')]
```

This section selects one synset from the list of synsets. Then it extracts its definition, usage examples, and lemmas.

[317]:
```
# Definition
selection = wn.synset('telephone.n.01')
print('Phone -', selection.definition().title(), '\n')

# Example
print('Example:',selection.examples(), '\n')
```

```
# Lemmas
print(selection.lemmas())
```

```
Phone - Electronic Equipment That Converts Sound Into Electrical Signals That
Can Be Transmitted Over Distances And Then Converts Received Signals Back Into
Sounds

Example: ['I talked to him on the telephone']

[Lemma('telephone.n.01.telephone'), Lemma('telephone.n.01.phone'),
Lemma('telephone.n.01.telephone_set')]
```

Using the same noun, traverse up the WordNet hierarchy as far as it can, outputting the synsets as it goes.

[318]:
```
hyp = selection.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('electronic_equipment.n.01')
Synset('equipment.n.01')
Synset('instrumentality.n.03')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

1.2 How WordNet is organized with nouns

Nouns in WordNet have a max hypernym of 'entity.n.01', where the nouns are organized into a hierarchy structure with the max being on the very top. It seems that all the nouns are hyponyms of this one synset.

The following code outputs the following (or an empty list if none exist): hypernyms, hyponyms, meronyms, holonyms, antonym.

[319]:
```
# hypernyms
print('hypernyms: ', selection.hypernyms(), '\n')

# hyponyms
print('hyponyms: ', selection.hyponyms(), '\n')
```

```
# meronyms
print('meronyms: ', selection.part_meronyms(), '\n')

# holonyms
print('holonyms: ', selection.part_holonyms(), '\n')

# antonym
print('antonym: ', selection.lemmas()[0].antonyms())
```

```
hypernyms:  [Synset('electronic_equipment.n.01')]

hyponyms:  [Synset('desk_phone.n.01'), Synset('dial_telephone.n.01'),
Synset('extension.n.10'), Synset('handset.n.01'), Synset('pay-phone.n.01'),
Synset('radiotelephone.n.02'), Synset('speakerphone.n.01')]

meronyms:  [Synset('mouthpiece.n.02'), Synset('telephone_receiver.n.01')]

holonyms:  [Synset('telephone_system.n.01')]

antonym:  []
```

1.2.1 This program selects a verb, then outputs all synsets.

[320]:
```
print("The chosen verb is run.")

print('Sysnset list:', wn.synsets('run'))
```

```
The chosen verb is run.
Sysnset list: [Synset('run.n.01'), Synset('test.n.05'), Synset('footrace.n.01'),
Synset('streak.n.01'), Synset('run.n.05'), Synset('run.n.06'),
Synset('run.n.07'), Synset('run.n.08'), Synset('run.n.09'), Synset('run.n.10'),
Synset('rivulet.n.01'), Synset('political_campaign.n.01'), Synset('run.n.13'),
Synset('discharge.n.06'), Synset('run.n.15'), Synset('run.n.16'),
Synset('run.v.01'), Synset('scat.v.01'), Synset('run.v.03'),
Synset('operate.v.01'), Synset('run.v.05'), Synset('run.v.06'),
Synset('function.v.01'), Synset('range.v.01'), Synset('campaign.v.01'),
Synset('play.v.18'), Synset('run.v.11'), Synset('tend.v.01'),
Synset('run.v.13'), Synset('run.v.14'), Synset('run.v.15'), Synset('run.v.16'),
Synset('prevail.v.03'), Synset('run.v.18'), Synset('run.v.19'),
Synset('carry.v.15'), Synset('run.v.21'), Synset('guide.v.05'),
Synset('run.v.23'), Synset('run.v.24'), Synset('run.v.25'), Synset('run.v.26'),
Synset('run.v.27'), Synset('run.v.28'), Synset('run.v.29'), Synset('run.v.30'),
Synset('run.v.31'), Synset('run.v.32'), Synset('run.v.33'), Synset('run.v.34'),
Synset('ply.v.03'), Synset('hunt.v.01'), Synset('race.v.02'),
Synset('move.v.13'), Synset('melt.v.01'), Synset('ladder.v.01'),
Synset('run.v.41')]
```

This section selects one synset from the list of synsets. Then it extracts its definition, usage examples, and lemmas.

[321]:
```
# Definition
selection2 = wn.synset('run.v.01')
print('Run -', selection2.definition().title(), '\n')

# Example
print('Example:',selection2.examples(), '\n')

# Lemmas
print(selection2.lemmas())
```

```
Run - Move Fast By Using One'S Feet, With One Foot Off The Ground At Any Given
Time

Example: ["Don't run--you'll be out of breath", 'The children ran to the store']

[Lemma('run.v.01.run')]
```

Using the same verb, traverse up the WordNet hierarchy as far as it can, outputting the synsets as it goes.

[322]:
```
x = selection2
while True:
    print(x)
    if x.hypernyms():
        x = x.hypernyms()[0]
    else:
        break
```

```
Synset('run.v.01')
Synset('travel_rapidly.v.01')
Synset('travel.v.01')
```

1.2.2 How WordNet is organized with nouns

Unlike nouns, for verbs there is no uniform top level synset. They can still be grouped into different hierarchies, but there isn't a max level that groups them together as we saw for the nouns example with 'entity.n.01'.

1.2.3 This code uses morphy to find as many different forms of the word

[323]:
```
# selected word is run
run = 'run'
print(wn.morphy(run))
print(wn.morphy(run, wn.ADJ))
print(wn.morphy(run, wn.VERB))
print(wn.morphy(run, wn.NOUN))
```

```
print(wn.morphy(run, wn.ADJ_SAT))
```

```
run
None
run
run
None
```

1.2.4 This portion selects two words that I think might be similar and find the specific synsets

[324]:
```
print('Girl: ', wn.synsets('Girl'), '\n')

print('Female: ', wn.synsets('Female'), '\n')
```

```
Girl:  [Synset('girl.n.01'), Synset('female_child.n.01'),
Synset('daughter.n.01'), Synset('girlfriend.n.02'), Synset('girl.n.05')]

Female:  [Synset('female.n.01'), Synset('female.n.02'), Synset('female.a.01'),
Synset('female.s.02'), Synset('female.s.03')]
```

1.2.5 This program runs the Wu-Palmer similarity metric and the Lesk algorithm

[325]:
```
girl = wn.synset('girl.n.01')
female = wn.synset('female.n.01')

print('Wu-Palmer similarity metric:', wn.wup_similarity(girl,female))

# Lesk Algorithm to disambiguate the meaning of football
sent = ['Are', 'you', 'a', 'girl']
print('Lesk Algorithm:', lesk(sent, 'girl'))
```

```
Wu-Palmer similarity metric: 0.6666666666666666
Lesk Algorithm: Synset('girlfriend.n.02')
```

A similarity score between two word senses can be extracted from WordNet, where the similarity ranges from 0 (little similarity) to 1 (identity). We see that the words girl and female are similar. The Lesk Algorithm was a bit funny and disambiguated the word girl to girlfriend.n.02.

1.3 SentiWordNet

What is SentiWordNet? - SentiWordNet is a lexical resource built on top of WordNet that assigns 3 sentiment scores - For each synset: positivity, negativity, and objectivity.

What is a use case? - An example of using this would be to see if a Twitter tweet is negative or violating.

The Program below select an emotionally charged word, find its senti-synsets and output the

polarity scores for each word. Then it makes up a sentence and outputs the polarity for each word in the sentence

[326]:
```
#print(wn.synsets('anger'))

# emotionally charged word
anger = swn.senti_synset('hurt.n.01')

# scores
print("Positive score: ", anger.pos_score(), '\n')
print("Negative score: ", anger.neg_score(), '\n')
print("Object score: ", anger.obj_score(), '\n')

sent = 'I hate having to wake up early'
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()
        print('Word --->', token)
        print(str(syn))
        print('neg score:', syn.neg_score(), '\tpos score:',syn.pos_score(),␣
 ↪'\tobj score:', syn.obj_score(), '\n')
```

```
Positive score:  0.0

Negative score:  0.625

Object score:  0.375

Word ---> I
<iodine.n.01: PosScore=0.0 NegScore=0.0>
neg score: 0.0  pos score: 0.0  obj score: 1.0

Word ---> hate
<hate.n.01: PosScore=0.125 NegScore=0.375>
neg score: 0.375        pos score: 0.125        obj score: 0.5

Word ---> having
<have.v.01: PosScore=0.25 NegScore=0.0>
neg score: 0.0  pos score: 0.25         obj score: 0.75

Word ---> wake
<aftermath.n.01: PosScore=0.0 NegScore=0.0>
```

```
neg score: 0.0  pos score: 0.0  obj score: 1.0

Word ---> up
<up.v.01: PosScore=0.0 NegScore=0.0>
neg score: 0.0  pos score: 0.0  obj score: 1.0

Word ---> early
<early.a.01: PosScore=0.0 NegScore=0.0>
neg score: 0.0  pos score: 0.0  obj score: 1.0
```

1.3.1 Observatioons of the scores

It was pretty cool to use this functionality to see how words get scored. For the word 'anger', it seemed accurate to give it a negative score. The word 'hate' has a 0.125 score of being positive, which was eye catching. The utility of knowing these scores in an NLP application is huge! Although some words were scored incorrectly, I believe we could someday have models that are more ideal and very accurate.

2 Collocations

When two or more words usually occur together with a frequency greater than chance would suggest, the words may form a collocation. A key indication that two words form a collocation is that you cannot substitute synonyms. An examples would be 'fast food'. These two words are used together and does not mean that the food is fast. These pairs of words are known as collocation.

[327]:
```
# Output collocations for text4, the Inaugural corpus
print(text4.collocations())

x = ' '.join(text4.tokens)
y = len(set(text4))

# Calculation of mutual information
phrase = x.count('American people')/ y
phrase2 = x.count('American')/ y
phrase3 = x.count('people')/ y
pmi = math.log2(phrase/(phrase2*phrase3))

# print results
print('\nAmerican people:', phrase)
print('\nAmerican:', phrase2)
print('\npeople:', phrase3)
print('\npmi: ', pmi)

# Calculation of mutual information
phrase4 = x.count('Indian tribes')/ y
phrase5 = x.count('Indian')/ y
phrase6 = x.count('tribes')/ y
```

```
pmi = math.log2(phrase/(phrase5*phrase6))

# print results
print('\nIndian tribes:', phrase4)
print('\nIndian:', phrase5)
print('\ntribes:', phrase6)
print('\npmi: ', pmi)
```

```
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

American people: 0.00399002493765586

American: 0.025735660847880298

people: 0.06264339152119701

pmi:  1.3073947068021263

Indian tribes: 0.000598503740648379

Indian: 0.0010972568578553616

tribes: 0.000598503740648379

pmi:  12.568848591758554
```

Looking at the results, the pmi for 'American people' was 1.3073947068021263 and 12.568848591758554 for Indian tribes; therefore, according to collocations, Indian tribes is more likely to be a collocations.