WordNet

Summary

WordNet is a database containing nouns, verbs, adjectives, and adverbs. It groups words into sets of synonyms called '*synsets*'. Each synset can list its definiton, examples, and its relations with other synsets.

Synsets of the noun 'paper'

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn

wn.synsets('paper')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[Synset('paper.n.01'),
 Synset('composition.n.08'),
 Synset('newspaper.n.01'),
 Synset('paper.n.04'),
 Synset('paper.n.05'),
 Synset('newspaper.n.02'),
 Synset('newspaper.n.03'),
 Synset('paper.v.01'),
 Synset('wallpaper.v.01')]
```

Definition of synset 'composition.n.08'

```
wn.synset('composition.n.08').definition()
```

```
'an essay (especially one written as an assignment)'
```

Usage Examples of synset 'composition.n.08'

```
wn.synset('composition.n.08').examples()
```

```
['he got an A on his composition']
```

Lemmas of synset 'composition.n.08'

```
wn.synset('composition.n.08').lemmas()
```

```
[Lemma('composition.n.08.composition'),
 Lemma('composition.n.08.paper'),
 Lemma('composition.n.08.report'),
 Lemma('composition.n.08.theme')]
```

Traversing the hierarchy for synset 'composition.n.08'

Observation:

By looking at the output below, I can see that each noun is a part of a category. To reach the top synset, we just need to go through the parent categories of each noun on the way.

```
hyp = wn.synset('composition.n.08').hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
  print(hyp)
  if hyp == top:
    break
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
```

```
Synset('essay.n.01')
Synset('writing.n.02')
Synset('written_communication.n.01')
Synset('communication.n.02')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

Hypernyms of synset 'composition.n.08'

```
wn.synset('composition.n.08').hypernyms()
```

```
[Synset('essay.n.01')]
```

Hyponyms of synset 'composition.n.08'

```
wn.synset('composition.n.08').hyponyms()
```

```
[Synset('term_paper.n.01')]
```

Meronyms of synset 'composition.n.08'

```
wn.synset('composition.n.08').part_meronyms()
```

```
[]
```

Holonyms of synset 'composition.n.08' text

```
wn.synset('composition.n.08').part_holonyms()
```

```
[]
```

Antonyms of synset 'composition.n.08'

```
record = wn.synset('composition.n.08')
record.lemmas()[0].antonyms()
```

```
[]
```

Synsets of the verb 'talk'

```
wn.synsets('talking')
```

```
[Synset('talk.n.01'),
 Synset('talk.v.01'),
 Synset('talk.v.02'),
 Synset('speak.v.03'),
 Synset('spill.v.05'),
 Synset('spill_the_beans.v.01'),
 Synset('lecture.v.01')]
```

Definition of synset 'speak.v.03'

```
wn.synset('speak.v.03').definition()
```

```
'use language'
```

Usage Examples of synset 'speak.v.03'

```
wn.synset('speak.v.03').examples()
```

```
['the baby talks already',
 "the prisoner won't speak",
 'they speak a strange dialect']
```

Lemmas of synset 'speak.v.03'

```
wn.synset('speak.v.03').lemmas()
```

```
[Lemma('speak.v.03.speak'), Lemma('speak.v.03.talk')]
```

Traversing the hierarchy for synset 'speak.v.03'

Observation:

After running several different verbs through the code, I realize that, unlike nouns, all verbs are not defined in a single hierarchy. Different verbs have different root synsets.
In the case below, I am ending the loop at 'act.v.01' because I have found that this is the root synset for the verb 'speak'.

```
hyp = wn.synset('speak.v.03').hypernyms()[0]
top = wn.synset('act.v.01')
while hyp:
  print(hyp)
  if hyp == top:
    break
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
```

```
Synset('communicate.v.02')
Synset('interact.v.01')
Synset('act.v.01')
```

Morphy

```
wn.morphy('talking', wn.NOUN)
```

```
'talking'
```

```
wn.morphy('talking', wn.VERB)
```

```
'talk'
```

Wu-Palmer Similarity Metric

Observation:

By looking at the output below, I realize that any two words that we might use interchangeably might not be so similar, if we consider all of their possible uses.

```
see = wn.synset('see.v.01')
observe = wn.synset('look.v.01')
wn.wup_similarity(see,observe)
```

```
0.4
```

Lesk Algorithm

Observation:

By looking at the output below, I can see that the algorithm is pretty accurate in figuring out which form of the word I used in the given sentence.

```
from nltk.wsd import lesk
sent1 = ['I','can','see','the','whole','blue','sky','from','here','.']
print(lesk(sent1, 'see'))
```

```
Synset('watch.v.03')
```

```
wn.synset('watch.v.03').definition()
```

```
'see or watch'
```

SentiWordNet

Summary

SentiWordNet is a resource that allocates 3 sentiment scores to each synset contained in the WordNet database. This resource can be used to determine the tone of a sentence.
One application for this feature can be a system that sorts the reviews of a product into different categories, based on if they were good, bad, or neutral.

```
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
```

```
sad = swn.senti_synsets('sad')
for item in sad:
  print(item)
  print("Positive score = ", item.pos_score())
  print("Negative score = ", item.neg_score())
  print("Objective score = ", item.obj_score())
  print()
```

```
    <sad.a.01: PosScore=0.125 NegScore=0.75>
    Positive score =  0.125
    Negative score =  0.75
    Objective score =  0.125

    <sad.s.02: PosScore=0.0 NegScore=0.25>
    Positive score =  0.0
    Negative score =  0.25
    Objective score =  0.75

    <deplorable.s.01: PosScore=0.0 NegScore=1.0>
    Positive score =  0.0
    Negative score =  1.0
    Objective score =  0.0

    [nltk_data] Downloading package sentiwordnet to /root/nltk_data...
    [nltk_data]   Package sentiwordnet is already up-to-date!
```

Observation:

By looking at the output below, I can see that the function ignored the words that did not contain any sentiments. That can be pretty useful when we need a certain type of data for some application, and using this feature can get rid of data that is not needed.

```
sent = 'I am sad about how the movie ended'
tokens = sent.split()
for token in tokens:
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    print(token)
    print(syn)
    print("Positive score = ", syn.pos_score())
    print("Negative score = ", syn.neg_score())
    print("Objective score = ", syn.obj_score())
    print()
```

```
    I
    <iodine.n.01: PosScore=0.0 NegScore=0.0>
    Positive score =  0.0
    Negative score =  0.0
    Objective score =  1.0
```

```
am
<americium.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

sad
<sad.a.01: PosScore=0.125 NegScore=0.75>
Positive score =  0.125
Negative score =  0.75
Objective score =  0.125

about
<about.s.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

movie
<movie.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

ended
<end.v.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
```

Collocation

Summary

A Collocation is when two words appearing together mean something that the two words would not mean, if they appeared separately. Also, the words in a collocation would not convey the same meaning if we replace each word with its synonyms.

```
import nltk
from nltk.book import *

text4.collocations()
```

```
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
```

Interpretation:

Looking at the output below, I can see that the calculated PMI is positive. According to the book, this means that the words 'foreign' and 'nations' do appear in the text together more than they each were expected to; this means that these two words do form a collocation.

```
text = ' '.join(text4.tokens)
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
```