assignment4

September 23, 2022

Wordnet is a large lexical database of semantic relationships between words. Wordnet can be used to assign these relationships between different words. The synonyms can then be grouped into synets with short definitions and usage examples. Wordnet works very similarly to a dictionary or thesaurus

[1]:
```
# library imports
import nltk
import math
from nltk.book import text4
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn
from nltk.wsd import lesk
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, …, text9 and sent1, …, sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

[2]:
```
#choose a noun
noun = "bike"
#get its synsets
synsets = wn.synsets(noun)
#print the synsets
synsets
```

[2]:
```
[Synset('motorcycle.n.01'), Synset('bicycle.n.01'), Synset('bicycle.v.01')]
```

[3]:
```
#choose motorcycle from the list of possible synsets
motorcycle = synsets[0]
```

```
#print its definition, its examples, and possible lemmas
print(motorcycle.definition())
print(motorcycle.examples())
print(motorcycle.lemmas())
```

```
a motor vehicle with two wheels and a strong frame
[]
[Lemma('motorcycle.n.01.motorcycle'), Lemma('motorcycle.n.01.bike')]
```

[4]:
```
%%capture --no-display

#traverse and print the WordNet hierarchy tree
hyper = lambda s: s.hypernyms()
list(motorcycle.closure(hyper))
```

[4]:
```
[Synset('motor_vehicle.n.01'),
 Synset('self-propelled_vehicle.n.01'),
 Synset('wheeled_vehicle.n.01'),
 Synset('container.n.01'),
 Synset('vehicle.n.01'),
 Synset('instrumentality.n.03'),
 Synset('conveyance.n.03'),
 Synset('artifact.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

Nouns seem to be classified by their usage, class as an object, then their relation to the rest of the physical world. For example, a motorbike is seen as motor vehicle at its highest level and then slowly describes as self-propelled or wheeled vehicle. Final it ends with synsets that simple refer to it as a physical object, then an entity.

[5]:
```
#print motorcycle hypernyms
print(motorcycle.hypernyms())
#print motorcycle hyponyms
print(motorcycle.hyponyms())
#print motorcycle meronyms
print(motorcycle.part_meronyms())
#print motorcycle holonyms
print(motorcycle.part_holonyms())
#print all motorcycle lemma antonyms
print([lemma.antonyms() for lemma in motorcycle.lemmas()])
```

```
[Synset('motor_vehicle.n.01')]
[Synset('minibike.n.01'), Synset('trail_bike.n.01')]
[Synset('kick_starter.n.01'), Synset('kickstand.n.01'), Synset('mudguard.n.01')]
```

```
[]
[[], []]
```

[6]:
```
#choose a verb
verb = "run"
#get all synsets of verb
synsets = wn.synsets(verb)
#print the verb synsets
synsets
```

[6]:
```
[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')]
```

[7]:
```
#choose the verb melt from possible list of synsets
melt = synsets[-3]

#print melt definitions, examples, and lemmas
print(melt.definition())
print(melt.examples())
print(melt.lemmas())
```

```
reduce or cause to be reduced from a solid to a liquid state, usually by heating
['melt butter', 'melt down gold', 'The wax melted in the sun']
[Lemma('melt.v.01.melt'), Lemma('melt.v.01.run'), Lemma('melt.v.01.melt_down')]
```

[8]:
```
#traverse and print the melt hiearchy tree
list(melt.closure(hyper))
```

[8]:
```
[Synset('dissolve.v.02'),
 Synset('change_integrity.v.01'),
 Synset('change.v.02')]
```

Verbs seem to be organized starting from what specific change they could have on the world to a more general term on how they can effect the world. In this example, melt is specified as a verb that can be dissolving something such as snow. Then it becomes more broad, in that it changes the integrity of an existing object, and then finally that it introduces some type of change.

[9]:
```
#choose a morphy object for the verb "run"
m = wn.morphy("run")
```

```
#get similar words that are all words
similar_words = wn.synsets(m, wn.VERB)
#display the similar words
similar_words
```

[9]:
```
[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')]
```

[10]:
```
#wu path similarity for "hunt" and "move"
hunt = wn.synset("hunt.v.01")
move = wn.synset("move.v.13")

print(wn.path_similarity(hunt, move))
```

```
0.16666666666666666
```

[11]:
```
#lesk algorithm for "hunt"
raw_text = "I went to the forest to hunt for deer ."
sent = raw_text.split()

print(lesk(sent, "hunt"))
```

```
Synset('search.n.01')
```

The Wu-Path similarity algorithm can be used to measure the similiarty between two words. The algorithm seems to work well since it ranked hunt and move with a similarity of 0.16%. This makes sense since each word has a very different connotation. Hunting refers to chasing toward something, while move simply refers to having some type of motion, with no goal in mind. The Lesk algorithm also works quite well in detecting the intension of the target verb. In my sentence I used hunt to refer to the sporting activity of hunting game. The leask algorithm provided the synset of "search". This is accurate since hunting in this context is the action of searching for a deer.

SentiNet is an extension of WordNet that allows you to preform sentiment analysis easily. It works by assigning one of three possible sentiments to each synset: positivity, negativity, or objectivity. NLTK provides an easy access to the SentiNet algorithm that can be used in several ways. For example, a naive sentiment anaylsis would consist of iterating over all synsets, getting the sentiment score, and count how many are positive versus negative.

[12]:
```
#get list of SentiNet synonyms for "awful"
senti_list = list(swn.senti_synsets('awful'))
#print the polarity info for each synonym
for item in senti_list:
    print(item)
```

```
<atrocious.s.02: PosScore=0.0 NegScore=0.875>
<awful.s.02: PosScore=0.0 NegScore=0.625>
<nasty.a.01: PosScore=0.0 NegScore=0.875>
<awed.s.01: PosScore=0.5 NegScore=0.5>
<frightful.s.02: PosScore=0.125 NegScore=0.25>
<amazing.s.02: PosScore=0.875 NegScore=0.125>
<terribly.r.01: PosScore=0.25 NegScore=0.0>
```

[13]:
```
#example emotional sentence
sent = "This chair is very uncomfortable"
#split sentence into list of indivivual words
tokens = sent.split()
```

```
#for each token, print the polarity info
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        print(syn_list[0])
```

```
<chair.n.01: PosScore=0.0 NegScore=0.0>
<be.v.01: PosScore=0.25 NegScore=0.125>
<very.s.01: PosScore=0.5 NegScore=0.0>
<uncomfortable.a.01: PosScore=0.0 NegScore=0.75>
```

The SentiNet is used to derive the sentiment score of each word. By looking at my example, I notice that it assigns to polarity to objects that hold no other context such as chair. It assigns a failr even polarity to verbs that only act to describe a state of being. However, words that are generally used with a heavy conotation have a strong polarity skew. In my example, "uncomfortable" is usually only used to describe something in a negative way, so it has a strong negative polarity score and no positive polarity score. This data can be very useful in NLP contexts, since it allows us generalize a base understanding of any input speech. We can programming learning negative or positive text. This might be useful in applications like learning from customer feedback. We can quickly see if a piece of provided feedback and is negative or positive.

A collocation is a relationship between two words that often occurs in normal human speech of text. A collocation may be formed if we notice that two or more words tend to occur together with a frequency greater than chance would suggest. This suggests that the two words are often used together when refered to. Another indicator that two words may be a collocation is that you cannot substitute synonyms. For example, "wild west" does not mean the same as "feral west"

[14]:
```
#display all collocations for text4
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
```

[15]:
```
#calculate and print pmi probabilities for the possible collocations "fellow␣
 ↪citizens"
text = ' '.join(text4.tokens)

vocab = len(set(text4))
hg = text.count('fellow')/vocab
print("p(fellow citizens) = ",hg )
h = text.count('fellow')/vocab
print("p(fellow) = ", h)
g = text.count('citizens')/vocab
print('p(citizens) = ', g)
pmi = math.log2(hg / (h * g))
```

```
print('pmi = ', pmi)
```

```
p(fellow citizens) =  0.013665835411471322
p(fellow) =  0.013665835411471322
p(citizens) =  0.026932668329177057
pmi =  5.214499019178814
```

PMI works by finding the probability of words x and y occurring next to each other, and dividing this by the probability of x multiplied by the probability of y. A pmi values of 0 would indicate that the words are independent. If the pmi is negative, then there is proof that x and y are not a collocation. Finally a positive pmi could be evidence that there is a strong collocation relationship. For the example that I used above, we got a pmi value of 5. This is a large positive value, so we can conclude that is probably strong evidence that "fellow citizens" is a collocation.