WordNet

September 25, 2022

1 1

Wordnet is a hierarchical database of words, organized and connected by semantic meaning. It contains definitions, synonyms, and examples for each word. Wordnet is incredibly useful for all facets of natural language processing, as it provides semantic meaning to every word.

[66]:
```
from nltk.corpus import wordnet as wn
```

2 2

The following cell gets the synsets of the word "good".

[67]:
```
synsets = wn.synsets("good")
synset = synsets[1]

print(f"Synset: {synsets}")
```

```
Synset: [Synset('good.n.01'), Synset('good.n.02'), Synset('good.n.03'),
Synset('commodity.n.01'), Synset('good.a.01'), Synset('full.s.06'),
Synset('good.a.03'), Synset('estimable.s.02'), Synset('beneficial.s.01'),
Synset('good.s.06'), Synset('good.s.07'), Synset('adept.s.01'),
Synset('good.s.09'), Synset('dear.s.02'), Synset('dependable.s.04'),
Synset('good.s.12'), Synset('good.s.13'), Synset('effective.s.04'),
Synset('good.s.15'), Synset('good.s.16'), Synset('good.s.17'),
Synset('good.s.18'), Synset('good.s.19'), Synset('good.s.20'),
Synset('good.s.21'), Synset('well.r.01'), Synset('thoroughly.r.02')]
```

3 3

It seems like the wordnet hierarchy is ordered from top to bottom. At the bottom, there are specific words. As you go further up, the hypernyms encapsulate everything until you reach the highest level, entity. There are many more nouns at the bottom of the hierarchy than the top. At the highest level of all nouns is the "entity" synset.

[68]:
```
print(f"Definition: {synset.definition()}")
print(f"Examples: {synset.examples()}")
print(f"Lemmas: {synset.lemmas()}")
```

```
current = [synset]
while len(current) > 0:
        print(current)
        current = current[0].hypernyms()
```

```
Definition: moral excellence or admirableness
Examples: ['there is much good to be found in people']
Lemmas: [Lemma('good.n.02.good'), Lemma('good.n.02.goodness')]
[Synset('good.n.02')]
[Synset('morality.n.01')]
[Synset('quality.n.01')]
[Synset('attribute.n.02')]
[Synset('abstraction.n.06')]
[Synset('entity.n.01')]
```

4 4

The following cell prints the synset's hypernyms, hyponyms, meronyms, holonyms, and antonyms.

[69]:
```
print(f"Hypernyms: {synset.hypernyms()}")
print(f"Hyponyms: {synset.hyponyms()}")
print(f"Meronyms: {synset.part_meronyms()}")
print(f"Holonyms: {synset.part_holonyms()}")
antonyms = []
for l in synset.lemmas():
    for a in l.antonyms():
        antonyms.append(a)
print(f"Antonyms: {antonyms}")
```

```
Hypernyms: [Synset('morality.n.01')]
Hyponyms: [Synset('beneficence.n.02'), Synset('benignity.n.01'),
Synset('kindness.n.01'), Synset('saintliness.n.01'),
Synset('summum_bonum.n.01'), Synset('virtue.n.01'), Synset('virtue.n.04')]
Meronyms: []
Holonyms: []
Antonyms: [Lemma('evil.n.03.evil'), Lemma('evil.n.03.evilness')]
```

5 5

The following cell outputs all synsets of the given verb.

[70]:
```
verb = wn.synsets("jog")
```

6 6

The following cell will find the hierarchy of the given verb. The hierarchy of verbs in wordnet is similar to the hierarchy of nouns. As you go up the hierarchy from the bottom, the verbs get more general. Unlike the noun hierarchy, the verb hierarchy doesn't originate from the same synset.

```
[71]: current = [verb[5]]
      while len(current) > 0:
              print(current)
              current = current[0].hypernyms()
```

```
[Synset('jog.v.03')]
[Synset('run.v.01')]
[Synset('travel_rapidly.v.01')]
[Synset('travel.v.01')]
```

7 7

The following cell uses morphy to find different forms of the word.

```
[72]: print(wn.morphy('jog', wn.NOUN))
      print(wn.morphy('jogs', wn.NOUN))
      print(wn.morphy('jogged', wn.VERB))
```

```
jog
jog
jog
```

8 8

The following cell computes the similarity between two words. The Wu-Palmer algorithm shows that lion and tiger are somewhat similar, however they aren't the exact same word. This makes sense, given they are in the same family of words (felines), however they refer to two distinct entities. For the Lesk algorithm, it seems longer sentences result in more context which in turn leads to better results.

```
[73]: from nltk.wsd import lesk

      lion = wn.synsets("lion")[0]
      tiger = wn.synsets("tiger")[0]
      print(wn.wup_similarity(lion, tiger))

      print(lesk("He jogged to work".split(), "jog"))
      print(lesk("The experience jogged his memory".split(), "jog"))
      print(lesk("He forgot about this, but the experience jogged his memory".
       ↪split(), "jog"))
```

```
0.5217391304347826
Synset('jog.v.06')
Synset('trot.v.01')
Synset('square_up.v.02')
```

9 9

SentiWordNet is a sentiment analysis dataset for identifying intention and opinion in natural language. It uses three different sentiment scores in order to try to quantify sentiment. SentiWordNet can be used for identifying bias in news articles, or for monitoring polarization on social media.

The biggest thing I observed is that each score is completely independent of the context around it. Having these scores can be incredibly useful for NLP applications, however they can be misleading in edge cases such as double negatives.

[74]:
```python
from nltk.corpus import sentiwordnet as swn
word = "outrage.n.02"
breakdown = swn.senti_synset(word)
print("Positive score = ", breakdown.pos_score())
print("Negative score = ", breakdown.neg_score())
print("Objective score = ", breakdown.obj_score())

sentence = "I was walking my beautiful dog in the ugly park down the street"
neg = 0
pos = 0
for word in sentence.split():
    syn_list = list(swn.senti_synsets(word))
    if syn_list:
        syn = syn_list[0]
        neg += syn.pos_score()
        pos += syn.neg_score()
        print(f"Pos: {syn.pos_score()}, Neg: {syn.neg_score()}")
print(f"Total Pos: {pos}, Total Neg: {neg}")
```

```
Positive score =  0.125
Negative score =  0.0
Objective score =  0.875
Pos: 0.0, Neg: 0.0
Pos: 0.0, Neg: 0.0
Pos: 0.0, Neg: 0.0
Pos: 0.75, Neg: 0.0
Pos: 0.0, Neg: 0.0
Pos: 0.0, Neg: 0.0
Pos: 0.0, Neg: 0.375
Pos: 0.0, Neg: 0.0
Pos: 0.125, Neg: 0.0
Pos: 0.0, Neg: 0.0
Total Pos: 0.375, Total Neg: 0.875
```

10 10

Collocation is when two or more words are combined more than statistically is normal, and when you can't substitute any of the words to get the same meaning. This can be useful to determine idioms and phrases that act as an atomic unit. In the following cell, I compute the mutual information between the words "criminal" and "justice". The mutual information formula gave a relatively high score, but not the highest I've seen, which makes sense as criminal justice is a legitimate collocation. I would interpret this score as criminal justice is a collocation, however they are often used without eachother which results in a lower mutual information score.

[75]:
```python
import itertools
import math
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.book import text4

def bigram_word_feats(words, score_fn=BigramAssocMeasures.chi_sq, n=200):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(score_fn, n)
    return [ngram for ngram in itertools.chain(words, bigrams)
                if type(ngram) == tuple]
bigram = bigram_word_feats(text4)[11]
print(bigram)
text = ' '.join(text4.tokens)

vocab = len(set(text))
p01 = text.count(bigram[0] + " " + bigram[1])/vocab
p0 = text.count(bigram[0])/vocab
p1 = text.count(bigram[1])/vocab
pmi = math.log2(p01/(p0*p1))
print(pmi)
```

```
('CRIMINAL', 'JUSTICE')
6.392317422778761
```