Wordnet and SentiWordNet

Wordnet is a network of relations between different words mostly of the same part of speech. It connects words theat are similar/synonymous, antonyms, and part - whole relationships. It is not completely hierchical. As most words are connected only to others of the same part of speech, there are 4 distinct subnetworks within wordnet.

Sentiwordnet is an extension of this that assigns synsets a sentiment score for the purpose of sentiment analysis.

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[Synset('dogma.n.01'), Synset('dogma.n.02')]
```

```
# get a definition for one of the synsets. Here I chose the second.

wn.synset('dogma.n.02').definition()
```

```
'a doctrine or code of beliefs accepted as authoritative'
```

```
#An example of the usage of the word in this context
wn.synset('dogma.n.02').examples()
```

```
['he believed all the Marxist dogma']
```

```
#The base forms of this word (it is the base form)
wn.synset('dogma.n.02').lemmas()
```

```
[Lemma('dogma.n.02.dogma')]
```

```
#This climbs the hypernym ladder for the first definition (not the one we used, but still interesting)
hyp = wn.synset('dogma.n.01').hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('religious_doctrine.n.01')
Synset('doctrine.n.01')
Synset('belief.n.01')
Synset('content.n.05')
Synset('cognition.n.01')
Synset('psychological_feature.n.01')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

```
wn.synset('dogma.n.01').hyponyms()
```

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

```
#This climbs the hypernym ladder for the second definition (the one we used)
hyp = wn.synset('dogma.n.02').hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
    Synset('doctrine.n.01')
    Synset('belief.n.01')
    Synset('content.n.05')
    Synset('cognition.n.01')
    Synset('psychological_feature.n.01')
    Synset('abstraction.n.06')
    Synset('entity.n.01')
```

This demonstrates that dogma can be viewed in two contexts, which are not too dissimilar- either religious, or purely ideological. Religion and ideaology have much overlap however- religion, according to wordnet, is a subset of ideology. I find it strange however that abstractions are considered entitites- I would personally consider them to be a concept, rather than any sort of thing that exists in actuality, which is what I think an entity entails.

```
print(wn.synset('dogma.n.02').hyponyms())
wn.synset('dogma.n.02').hypernyms()
wn.synset('dogma.n.02').part_meronyms()
wn.synset('dogma.n.02').substance_meronyms()
wn.synset('dogma.n.02').part_holonyms()
#wn.synset('dogma.n.02').antonyms()
```

```
    []
    []
```

```
wn.synset('dogma.n.02').hyponyms()
```

```
    []
```

```
wn.synset('dogma.n.02').hypernyms()
```

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

```
wn.synset('dogma.n.02').part_meronyms()
```

```
    []
```

```
wn.synset('dogma.n.02').part_holonyms()
```

```
    []
```

```
synonyms = []
antonyms = []
```

```
for l in wn.synset('dogma.n.02').lemmas():
    synonyms.append(l.name())
    if l.antonyms():
        antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

```
    {'dogma'}
    set()
```

Dogma is a unique word which is why I chose it. It doesn't really have a synonym, or antonym. Its kinda of its own stand alone thing. The closest parralel is doctrine, but that strays into millitary and it is not held onto, well, dogmatically. It changes.

```
wn.synsets('proselytize')
```

```
    [Synset('proselytize.v.01')]
```

```
wn.synset('proselytize.v.01').definition()
```

```
    'convert to another faith or religion'
```

```
wn.synset('proselytize.v.01').examples()
```

```
[]
```

```
wn.synset('proselytize.v.01').lemmas()
```

```
[Lemma('proselytize.v.01.proselytize'), Lemma('proselytize.v.01.proselytise')]
```

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

```
Synset('convert.v.05')
Synset('change.v.01')
```

It seems that verbs don't go up to a single unifying thing (like entity, for nouns.) I tried running this on a few different verbs, like run, which went to travel. Proselytize goes to convert, then to change. I guess it's fair though, as I cannot think of a category more general than change that accurately includes the verb change. So unlike nouns, there is no single 'top' for verbs as I expected there would be.

```
wn.morphy('proselytize')
```

```
'proselytize'
```

```
print(wn.morphy('running'), wn.VERB)
```

```
running v
```

Comparison of Two Words

```
#Here we will compare the terms theory and hunch
```

```
wn.synsets('theory')
```

```
[Synset('theory.n.01'), Synset('hypothesis.n.02'), Synset('theory.n.03')]
```

```
wn.synset('theory.n.01').definition()
```

```
'a well-substantiated explanation of some aspect of the natural world; an organiz
ed system of accepted knowledge that applies in a variety of circumstances to exp
```

```
wn.synset('theory.n.03').definition()
```

```
'a belief that can guide behavior'
```

```
#Here we are going to go off synset 3 for theory
```

```
wn.synsets('hunch')
```

```
[Synset('intuition.n.02'), Synset('hunch.n.02'), Synset('hunch.v.01')]
```

```
wn.synset('intuition.n.02').definition()
```

```
'an impression that something might be the case'
```

```
wn.synset('intuition.n.02').wup_similarity(wn.synset('theory.n.03'))
```

```
0.6666666666666666
```

```
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize
nltk.download('all')
```

```
sent = "I have a hunch that man is not actually a doctor, and has never been to medical school."
temp = word_tokenize(sent)
print(lesk(temp, 'hunch').definition())
```

```
an impression that something might be the case
```

```
sent = "That man walks around with a hunch, like he lives in the bell tower of Notre Dame"
temp = word_tokenize(sent)
print(lesk(temp, 'hunch').definition())
```

```
the act of bending yourself into a humped position
```

```
sent = "I have to hunch down to get under the sign."
temp = word_tokenize(sent)
print(lesk(temp, 'hunch').definition())
```

```
an impression that something might be the case
```

```
#Apparently it's definitely not a perfect algorithm.
```

```
sent = "If my theory is correct, that man is not a doctor."
temp = word_tokenize(sent)
print(lesk(temp, 'theory').definition())
```

```
a tentative insight into the natural world; a concept that is not yet verified but that if true would explain certain facts or phenomena
```

```
sent = "The communist theory says that capitalism must be eliminated the world over for a utopia."
temp = word_tokenize(sent)
print(lesk(temp, 'theory').definition())
```

```
a well-substantiated explanation of some aspect of the natural world; an organized system of accepted knowledge that applies in a variet
```

```
#Theory has multiple similar definitions, unlike hunch. It makes more sense the algorithm would be confused here.
```

SentiWordNet

SentiWordNet is an extension of wordnet that assigns a score to terms based on their connotations. This is primarily for sentiment analysis- having a computer collect data and figure out how the posters fill on a certain subject.

It has three categories- positive, negative, and objective (eg. neutral).

Knowing these scores helps tune in marketing, by examining how customers react and review products.

```
#Our emotionally charged word is devastate
```

```
wn.synsets('devastate')
```

```
[Synset('lay_waste_to.v.01'), Synset('devastate.v.02')]
```

```
wn.synset('lay_waste_to.v.01').definition()
```

```
'cause extensive destruction or ruin utterly'
```

```
wn.synset('devastate.v.02').definition()
```

```
'overwhelm or overpower'
```

```
from nltk.corpus import sentiwordnet as swn
breakdown = swn.senti_synset('lay_waste_to.v.01')
```

```
print(breakdown)
```

```
<lay_waste_to.v.01: PosScore=0.0 NegScore=0.0>
```

```
#I think they forgot to give this word a score. lets try devastated
```

```
wn.synsets('devastated')
```

```
[Synset('lay_waste_to.v.01'), Synset('devastate.v.02')]
```

```
#Apparently the wordnet does not recognize this as an adjective. This qualifies as a hole in their coverage if you ask me.
```

```
sentence = "The mongols devastated the countryside, nothing was left in their wake but ash and ruin."
```

```
temp = word_tokenize(sentence)
neg = 0
pos = 0
for item in temp:
  synList = list(swn.senti_synsets(item))
  if synList:
        syn = synList[0]
        neg += syn.neg_score()
        pos += syn.pos_score()



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

```
neg     pos counts
0.75     0.25
```

```
sentence = "nothing was left but ash and ruin."
temp = word_tokenize(sentence)
neg = 0
pos = 0
for item in temp:
  synList = list(swn.senti_synsets(item))
  if synList:
        syn = synList[0]
        print(syn)
        neg += syn.neg_score()
        pos += syn.pos_score()
print("neg\tpos counts")
print(neg, '\t', pos)
```

```
<nothing.n.01: PosScore=0.25 NegScore=0.25>
<washington.n.02: PosScore=0.0 NegScore=0.0>
<left.n.01: PosScore=0.0 NegScore=0.0>
<merely.r.01: PosScore=0.0 NegScore=0.0>
<ash.n.01: PosScore=0.0 NegScore=0.0>
<ruin.n.01: PosScore=0.0 NegScore=0.5>
neg     pos counts
0.75     0.25
```

```
#Apparently the sentiwordnet only recognized nothing and ruin part as negative, no feelings about mongols or devastation.
```

```
sentence = "I was devastated."
temp = word_tokenize(sentence)
neg = 0
pos = 0
for item in temp:
  synList = list(swn.senti_synsets(item))
  if synList:
    syn = synList[0]
    print(syn)
    neg += syn.neg_score()
    pos += syn.pos_score()
print("neg\tpos counts")
print(neg, '\t', pos)
```

```
<iodine.n.01: PosScore=0.0 NegScore=0.0>
<washington.n.02: PosScore=0.0 NegScore=0.0>
<lay_waste_to.v.01: PosScore=0.0 NegScore=0.0>
neg     pos counts
0.0      0.0
```

```
sentence = "I'm sad, morose even. Really negative, down in the dumps, ruined, angry, furious and depressed that this stupid dumb idiot sentin
temp = word_tokenize(sentence)
neg = 0
pos = 0
for item in temp:
  synList = list(swn.senti_synsets(item))
  if synList:
    syn = synList[0]
    print(syn)
    neg += syn.neg_score()
    pos += syn.pos_score()
print("neg\tpos counts")
print(neg, '\t', pos)
```

```
<iodine.n.01: PosScore=0.0 NegScore=0.0>
<sad.a.01: PosScore=0.125 NegScore=0.75>
<dark.s.06: PosScore=0.25 NegScore=0.625>
<evening.n.01: PosScore=0.0 NegScore=0.0>
<truly.r.01: PosScore=0.625 NegScore=0.0>
<negative.n.01: PosScore=0.0 NegScore=0.25>
<down.n.01: PosScore=0.125 NegScore=0.0>
<inch.n.01: PosScore=0.0 NegScore=0.0>
<dumps.n.01: PosScore=0.0 NegScore=0.0>
<destroy.v.02: PosScore=0.0 NegScore=0.5>
<angry.a.01: PosScore=0.375 NegScore=0.375>
<ferocious.s.01: PosScore=0.25 NegScore=0.25>
<depress.v.01: PosScore=0.0 NegScore=0.125>
<stupid.n.01: PosScore=0.0 NegScore=0.125>
<dense.s.04: PosScore=0.0 NegScore=0.25>
<idiot.n.01: PosScore=0.0 NegScore=0.125>
<calcium.n.01: PosScore=0.0 NegScore=0.0>
<understand.v.01: PosScore=0.375 NegScore=0.0>
<iodine.n.01: PosScore=0.0 NegScore=0.0>
<feel.n.01: PosScore=0.125 NegScore=0.0>
neg     pos counts
3.375    2.25
```

```
#The fact this got positive points shows some serious flaws.
```

Sentiwordnet is neither complete nor accurate, which is a shame. The english language is simply too large and contextual to rely on sentiment analysis based on single words. For example, If you said "I was sad, then I realized that it wasn't so bad." Sentinet would coun't that as very negative. Connotation doesn't matter as much as context. So reflecting on my initial stance written earlier, sentinet isn't that useful in my opinion. Needs more work before it can be as useful as a human.

COLLOCATION

Collocation is when multiple words, when appearing in a certain grouping, carry a different definition then they would on their own. For exampl, a flash stick is another word for a USB drive, but individually the words mean a brief, bright light or quick period of time, and a thin, long, ridgid object. The full meaning cannot be grasped by looking at the words seperately, and instead you must analyze the collocation. I personally think when humans look at this we recognize it as one word.

```
from nltk.book import 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
```

```
import math
text = ' '.join(text4.tokens)
vocab = len(set(text4))
VP = text.count('Vice President')/vocab
print("p(Vice President) = ", VP )
v = text.count('Vice')/vocab
print("p(Vice) = ", v)
p = text.count('President')/vocab
print('p(President) = ', p)
pmi = math.log2(VP / (v * p))
print('pmi = ', pmi)
```

```
p(Vice President) =  0.0017955112219451373
p(Vice) =  0.0018952618453865336
p(President) =  0.010773067331670824
pmi =  6.458424602064904
```

As we can see, Vice President is a collocation. President appears much more often than vice- it is after all an annaugural address where someone is being sworn into that office.

But vice seldom appears unless it is vice president. Perhaps a few time it is used to refer to the moral pitfalls, but here you can tell that there is mutual information. When vice or president appears, the other is likely to be near.

Below we will attempt it with one that is not a meaningful collocation- 'of years' which can be understood just fine as individual words.

```
text = ' '.join(text4.tokens)
vocab = len(set(text4))
VP = text.count('of years')/vocab
print("p(of years) = ", VP )
v = text.count('of')/vocab
print("p(of) = ", v)
p = text.count('years')/vocab
print('p(years) = ', p)
pmi = math.log2(VP / (v * p))
print('pmi = ', pmi)
```

```
p(of years) =  0.00039900249376558606
p(of) =  0.7487281795511221
p(years) =  0.014264339152119701
pmi =  -4.74238529543572
```

```
#Definitely not a collocation.
```

✓ 0s completed at 9:00 PM