Wordnet

Wordnet is a tool used to show the relations between words. It is like a thesaurus in that it groups words based on similar meanings, but the connections it shows are much more detailed and intricate. It can group words based on specificity, find indirect antonyms, make connections between words that are seen or used together, and even show inheritance between a parent and child word. Lets start by importing:

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('stopwords')
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import *
text4
!pip install nltk
```

Let's start by finding the synsets in the word "monkey".

```
wn.synsets('monkey')
```

```
[Synset('monkey.n.01'),
 Synset('imp.n.02'),
 Synset('tamper.v.01'),
 Synset('putter.v.02')]
```

Wordnet gives all the synsets, not just for the noun monkey, but also the verb. Since we are using the noun monkey, we should choose a noun synset also. Here we will look at the synset ('imp.n.02') and find its definition, examples, lemmas, and then traverse up the hierarchy as far up, while printing the synsets we pass through while going up.

```
imp = wn.synset('imp.n.02')
print("definition: ", imp.definition())
print("examples: ", " ".join(imp.examples()))
print("lemmas: ", ", ".join([lemma.name() for lemma in imp.lemmas()]))
hyper = lambda s: s.hypernyms()
list(imp.closure(hyper))
```

```
definition:  one who is playfully mischievous
examples:
lemmas:  imp, scamp, monkey, rascal, rapscallion, scalawag, scallywag
[Synset('child.n.01'),
 Synset('juvenile.n.01'),
 Synset('person.n.01'),
 Synset('causal_agent.n.01'),
 Synset('organism.n.01'),
 Synset('physical_entity.n.01'),
 Synset('living_thing.n.01'),
 Synset('entity.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01')]
```

It seems that WordNet organizes nouns in order of specificity, the more specific nouns are further down the tree, while the more general ones are closer to the root. The parent is always a more general version of the child. Additionally, WordNet groups the basic forms of similar words together, as we can see in the lemmas.

```
print("hypernyms: ", imp.hypernyms())
print("hyponyms: ", imp.hyponyms())
print("meronyms: ", imp.part_meronyms())
print("holonyms: ", imp.part_holonyms())
#print("antonym: ", imp.antonym())
```

```
hypernyms:  [Synset('child.n.01')]
hyponyms:  [Synset('terror.n.03')]
meronyms:  []
holonyms:  []
```

Looking at these results, it seems WordNet has hypernyms and hyponyms for imp, but no meronyms or holonyms.

Next, lets choose a verb to explore. How about "whisper"?

```
wn.synsets('whisper')
```

```
[Synset('whisper.n.01'), Synset('rustle.n.01'), Synset('whisper.v.01')]
```

Like for the noun, we will choose a synset listed here, and look for the definition, examples, lemmas, and then traverse up the hierarchy as far up, while printing the synsets we pass through while going up. This time let's look at Synset('whisper.v.01'),

```
whisper = wn.synset('whisper.v.01')
print("definition: ", whisper.definition())
print("examples: ", " ".join(whisper.examples()))
print("lemmas: ", ", ".join([lemma.name() for lemma in whisper.lemmas()]))
list(whisper.closure(hyper))
```

```
definition:  speak softly; in a low voice
examples:
lemmas:  whisper
[Synset('talk.v.02'),
 Synset('communicate.v.02'),
 Synset('interact.v.01'),
 Synset('act.v.01')]
```

Similar to the nouns, the hierarchy seems to be based on specificity, in this case whisper is similar to talk, but is more specific.

Next we will use morphy to find different forms of the word.

```
print(wn.morphy('whisper', wn.ADJ))
print(wn.morphy('whisper', wn.NOUN))
print(wn.morphy('whisper', wn.VERB))
print(wn.morphy('whisper', wn.ADV))
```

```
None
whisper
whisper
None
```

Next, lets use the Wu-Palmer similarity metric and the Lesk algorithm on two similar words, "hit" and "whack".

```
print((wn.synsets('hit')))
print(wn.synsets('whack'))
hit = wn.synset('hit.v.3')
whack = wn.synset('whack.v.01')
print(hit.definition())
print(whack.definition())
```

```
[Synset('hit.n.01'), Synset('hit.n.02'), Synset('hit.n.03'), Synset('collision.n.01'), Synset('hit.n.05'), Synset('hit.n.06'), Synset('h
[Synset('whack.n.01'), Synset('knock.n.05'), Synset('whack.v.01')]
deal a blow to, either with the hand or with an instrument
hit hard
```

Now that we have found the synsets we want to compare, let's run the metric and algorithm.

```
print("Wu-Palmer metric: ", wn.wup_similarity(hit, whack))
sentence = ['I', 'whacked', 'him', 'on', 'the', 'way', 'home', '.']
print("Lesk algorithm: ",  lesk(sentence,  'hit'))
print("Lesk algorithm: ",  lesk(sentence,  'whack'))
sentence = ['I', 'whacked', 'him', 'with', 'a', 'bat', 'on', 'the', 'way', 'home', '.']
print("Lesk algorithm: ",  lesk(sentence,  'hit'))
print("Lesk algorithm: ",  lesk(sentence,  'whack'))
```

```
Wu-Palmer metric:  0.8
Lesk algorithm:  Synset('hit.v.16')
```

```
Lesk algorithm:  Synset('whack.n.01')
Lesk algorithm:  Synset('hit.v.03')
Lesk algorithm:  Synset('whack.n.01')
```

Since the Wu-Palmer metric uses a scale of 0-1, with 1 being the most similar, and 0 being the least, the metric shows that the similarity is at .8, meaning these words are very similar. The Lesk algorithm looks at the sentence and decides which definition of the word would fit the sentence. In this case, it seems that the usage of "whacked" in this sentence is closest to a different definition of "hit" than the one I chose, however the "whack" I chose was the most similar here. Interestingly, when adding a few words for extra context, the "hit" I chose became the most similar one.

SentiWordNet

SentiWordNet is a useful tool that analyzes words and even sentences to determine sentiment. It looks at words and assigns sentiment scores for positivity, negativity, and objectivity. It can be used for sentiment analysis to observe how people think of certain topics, items, or ideas. Analyzing the sentiment behind a sentence can also help an AI better understand the sentence and a deeper understanding can help the AI come to a solution that is more relevant.

To test the functionality of this tool, I am finding all the senti-synsets and polarity scores for the word "tragic", because this is a very emotionally charged word. Afterwards, I will be running an analysis on a sentence and obeserving the scores for each word and the whole sentence.

```
senti_list = list(swn.senti_synsets('tragic', "a"))
for item in senti_list:
    print(item)
sent = 'The food we ate today was tasty'
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]
        print("word: ", token)
        print("positive score: ", syn.pos_score())
        print("negative score: ", syn.neg_score())
        neg += syn.neg_score()
        pos += syn.pos_score()

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

```
 <tragic.s.01: PosScore=0.0 NegScore=0.625>
 <tragic.a.02: PosScore=0.0 NegScore=0.125>
 word:  food
 positive score:  0.0
 negative score:  0.0
 word:  ate
 positive score:  0.0
 negative score:  0.0
 word:  today
 positive score:  0.125
 negative score:  0.0
 word:  was
 positive score:  0.0
 negative score:  0.0
 word:  tasty
 positive score:  0.625
 negative score:  0.25
 neg     pos
 0.25     0.75
```

Not surprisingly, the polarity for the word "tragic" was very negative. The sentence that I intended to seem positive did score mostly positive, however I was surprised that tasty was given a .25 for the negative score. These scores would definitely help a computer understand the sentiment behind sentences, and helps a company know how customers feel about their products. It could also be used for helpbots, which would benefit from knowing when a customer is frustrated.

Collocations

A collocation is two or more words that can be seen together, but create a meaning different from the simple combination of the definitions of both words. For example, "fast" and "food" apart from each other mean something very different compared to when they are together. Here, I

```
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
```

For our collocation, we will choose United States. I will calculate the mutual information between these two words. Using the formula pmi = log_2(P(x,y) / [P(x) * P(y)]), we will calculate the pmi of United States.

```
import math

text = ' '.join(text4.tokens)
text[:50]
vocab = len(set(text4))
hg = text.count('United States')/vocab
print("p(United States) = ",hg )
h = text.count('United')/vocab
print("p(United) = ", h)
g = text.count('States')/vocab
print('p(States) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)

hg = text.count('of the')/vocab
print("\np(of the) = ",hg )
o = text.count('of')/vocab
print("p(of) = ", o)
t = text.count('the ')/vocab
print('p(the) = ', t)
pmi = math.log2(hg / (o * t))
print('pmi = ', pmi)
```

```
p(United States) =  0.015860349127182045
p(United) =  0.0170573566084788
p(States) =  0.03301745635910224
pmi =  4.815657649820885

p(of the) =  0.20089775561097256
p(of) =  0.7487281795511221
p(the) =  0.9533167082294264
pmi =  -1.8290080938996587
```

The code calculates the probability of the words appearing, and the probability of the words appearing together. In this case, the phrase "United States" appears around .016 of the text, and the word "United" appears around .017, meaning that most times that United appears, it is seen with States. Additionally, the pmi is around 4.82, meaning that the phrase is highly likely to be a collocation. When compared to the phrase "of the", which has a negative pmi, this is made more clearly.

✓ 0s completed at 5:38 PM