WordNet Overview

Summary:

WordNet is a project from Princeton from the 1980s built to model and understand how people assign a hierarchy to organize concepts started by psychologist George Miller. WordNet acts as an organizational hierarchy of parts of speech (nouns, verbs, adjective, and adverbs) that lists definitions, synonym sets, use case examples, and word relations.

Noun Synonym Sets (Synsets):

Here the noun 'food' was chosen to have its synonym sets outputed using the synsets() function of WordNet.

```
from nltk.corpus import wordnet as wn
wn.synsets('food')
```

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

Definitions, Examples, and Lemmas:

Using the corresponding WordNet functions the definition, use example, and lemmas of the selected synset of the noun 'food' is outputted.

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

```
'any solid substance (as opposed to liquid) that is used as a source of
nourishment'
```

```
wn.synset('food.n.02').examples()
```

```
['food and drink']
```

```
wn.synset('food.n.02').lemmas()
```

```
[Lemma('food.n.02.food'), Lemma('food.n.02.solid_food')]
```

Traversing the Noun Hierarchy:

WordNet assigns a hierarchy to nouns where the root value (highest level) is that of 'entity' which is a broad definition for all nouns. As you move up the hierarchy the words get more and more broad until that root value is reached.

The code below traverses up the WordNet hierarchy for the word 'food'. As we can see it becomes broader the further up the hierarchy we move, going from 'food' to 'matter' all the way to 'entity'.

```
food = wn.synset('food.n.01')
root = food.root_hypernyms()
hyp = food.hypernyms()[0]

print(food)
while hyp:
    print(hyp)
    if hyp == root[0]:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('food.n.01')
Synset('substance.n.07')
Synset('matter.n.03')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Nyms of Every Shape and Size: Hyper, Hypo, Mero, Holo, and Antonyms

The code below outputs the hypernyms (higher in hierarchy), hyponyms (lower in hierarchy), meronyms (part-whole relation), holonyms (whole-part relation), and antonyms (opposite) of the word 'food' (or an empty list if none exist).

```
print("hypernyms:", food.hypernyms())
print("hyponyms:", food.hyponyms())
print("meronyms:", food.part_meronyms())
print("holonyms:", food.part_holonyms())
print("antonyms:", food.lemmas()[0].antonyms())
```

```
hypernyms: [Synset('substance.n.07')]
hyponyms: [Synset('beverage.n.01'), Synset('chyme.n.01'), Synset('comestible.n
meronyms: [Synset('food.n.02')]
holonyms: []
antonyms: []
```

Verb Synonym Sets (Synsets):

Here the verb 'reflect' was chosen to have its synonym sets outputed using the synsets() function of WordNet.

```
wn.synsets('reflect')
```

```
[Synset('reflect.v.01'),
 Synset('chew_over.v.01'),
 Synset('reflect.v.03'),
 Synset('reflect.v.04'),
 Synset('reflect.v.05'),
 Synset('reflect.v.06'),
 Synset('reflect.v.07')]
```

Definitions, Examples, and Lemmas:

Using the corresponding WordNet functions the definition, use example, and lemmas of the selected synset of the verb 'reflect' is outputted.

```
reflect = wn.synset('reflect.v.01')
reflect.definition()
```

```
'manifest or bring back'
```

```
reflect.examples()
```

```
['This action reflects his true beliefs']
```

```
reflect.lemmas()
```

```
[Lemma('reflect.v.01.reflect')]
```

Traversing the Verb Hierarchy:

Unlike nouns, verbs in WordNet do not have a top level synset. Each verb may connect to another through a higher level synset but not all verbs will be connected in the same way all nouns are bound by the root hypernym of 'entity'.

The code below traverses up the WordNet hierarchy for the verb 'reflect'.

```
root = reflect.root_hypernyms()
hyp = reflect.hypernyms()[0]

print(reflect)
while hyp:
    print(hyp)
    if hyp == root[0]:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

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

Morphy Word Forms:

The code below is my attempt at finding as many forms of the verb 'refelct' as I can. The chosen words are all varied forms in tense and case of 'reflect' and using the morph() function circle back to the original verb 'reflect'.

```
wn.morphy('reflected')
```

```
'reflect'
```

```
wn.morphy('reflecting')
```

```
'reflect'
```

```
wn.morphy('reflects')
```

```
'reflect'
```

Comparing an Apple and Orange: Similarity Metric and Lesk Algorithm

Wu-Palmer Metric

The words 'orange' and 'apple' were slected to have their similarity calculated based on the Wu-Palmer metric for similarity. In doing so we see that running the wup_similarity() function with the chosen words gives us a similarity metric of ~0.783, which compares the depth of the two senses in taxonomy and their most common ancestor. So I guess in this case it was fine to compare apples and oranges.

Lesk Algorithm

The Lesk Algorithm is implememted in nltk through the lesk() function which will return the synset with the highest number of overlapping words between the context sentence and the defiinition in each of the target words synsets.

For both 'apple' and 'orange' the Lesk Algorithm selected the first synsets which have the definitions of the words as fruits rather than in some other context.

```
wn.synsets('orange')
orange = wn.synset('orange.n.01')
orange.definition()
```

```
'round yellow to orange fruit of any of several citrus trees'
```

```
wn.synsets('apple')
apple = wn.synset('apple.n.01')
apple.definition()
```

```
'fruit with red or yellow or green skin and sweet to tart crisp whitish
flesh'
```

```
wn.wup_similarity(apple, orange)
```

```
0.782608695652174
```

```
from nltk.wsd import lesk
for ss in wn.synsets('orange'):
    print(ss, ss.definition())

print('\n')

for ss in wn.synsets('apple'):
    print(ss, ss.definition())
```

```
Synset('orange.n.01') round yellow to orange fruit of any of several citrus tı
Synset('orange.n.02') orange color or pigment; any of a range of colors betwe
Synset('orange.n.03') any citrus tree bearing oranges
Synset('orange.n.04') any pigment producing the orange color
Synset('orange.n.05') a river in South Africa that flows generally westward t
Synset('orange.s.01') of the color between red and yellow; similar to the col


Synset('apple.n.01') fruit with red or yellow or green skin and sweet to tart
Synset('apple.n.02') native Eurasian tree widely cultivated in many varieties
```

```
sent = ["I", "wish", "I", "could", "take", "a", "big", "bite", "of", "that", "orang
print(lesk(sent, 'orange'))
print(lesk(sent, "apple"))
```

```
Synset('orange.s.01')
Synset('apple.n.01')
```

How Does that Sound: SentiWordNet

SentiWordNet (swn) is resource that provides sentiment scores for synsets of words. The scores are in refereence to the positivity, negativity, and objectivity of words on a scale from 0.0 to 1.0 where the sum of teh scores will always be 1.0. It is a really cool resource for sentiment and opinion mining over sentences or bodies of text. SentWordNet could be used to accumulate opinion or rating information about a specific person or concept on a social media platform like twitter or reddit and fed into a machine learning system.

To test out the swn analysis the word 'murder' was chosen to observe the polarities, and the results were interesting. The swn ended up giving a mostly negativity score to only one of the synsets of 'murder' but unsurprisingly gave none of them a postivity score above 0.0.

```
import nltk
nltk.download('sentiwordnet')
```

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

```
from nltk.corpus import sentiwordnet as swn
senti_list = list(swn.senti_synsets('murder'))
for item in senti_list:
    print(item)
```

```
<murder.n.01: PosScore=0.0 NegScore=0.375>
<murder.v.01: PosScore=0.0 NegScore=0.625>
<mangle.v.03: PosScore=0.0 NegScore=0.125>
```

How Does that Sound: SentiWordNet with a Sentence

Using swn on the words in a sentence will give us polarity scores for each of the words that can have them (prepositions are seemingly excluded for obvious reasons). Using these polarity scores we can get an average or total polarity for the sentence and get a pretty basic understanding of whether a sentence is generally positive or negative in sentiment. This to me seems like it would be a rudamentarty way of going about it but it does provide a start to understanding the sentinemt of a sentence as a whole rather than its components without context.

```
text = "that was the best movie ever"
sent = text.split()

for item in sent:
    syn_list = list(swn.senti_synsets(item))
    print(item)
    if syn_list:
        syn = syn_list[0]
        print("negative:", syn.neg_score())
        print("positive:", syn.pos_score())
        print("objective:", syn.obj_score())
    else:
        print("no polarity score available")
```

```
that
no polarity score available
was
negative: 0.0
positive: 0.0
objective: 1.0
the
no polarity score available
best
negative: 0.0
positive: 0.25
objective: 0.75
movie
negative: 0.0
positive: 0.0
objective: 1.0
ever
negative: 0.0
positive: 0.0
objective: 1.0
```

The Key is Collocation, Collocation, Collocation: Collocation

Collocations are words that generally appear togther and form a greater meaning than the sum of their parts. The words mutually provide some context to one another and alter the meaing of the words if they were viewed without that altered context.

The code below uses the Inagural Address text from the nltk corpus to locate all of the collocations in the text. It then uses one of the collocations 'fellow Americans' to show off the mutual information formulas, which show how connected two words are, where a score of 0 pmi would show independence of two words, a negative score means two words aren't likely to a collocation, and a positive score means two words are likely a collocation. The scores are based on how often the two words appear together in the text and often they appear separately.

As seen below the proposed 'fellow Americans' collocation recieved a positive pmi of ~ 2.846 meaning the collocations() method correctly identified it as a collocation.

```
from nltk.book import *
text4
```

```
<Text: Inaugural Address Corpus>
```

```
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
```

```
text = ' '.join(text4.tokens)
text[:50]
```

```
'Fellow - Citizens of the Senate and of the House o'
```

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

```
p(fellow Americans) = 0.00199501246882793
p(fellow) = 0.013665835411471322
p(Americans) = 0.008478802992518703
pmi =  2.8459373434103195
```