WordNet

Summary:

WordNet is a lexical database that links words together based on semantic relations. It was started as a project at Princeton University where the goal was to support the idea that humans remember things more easily if they organize concepts in some sort of hierarchy. WordNet has an interface for use with NLTK and NLP projects.

Import

First, import NLTK and WordNet

```
import nltk
import math
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('book')
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn
from nltk.tokenize import word_tokenize
from nltk.wsd import lesk
from nltk.util import ngrams
from nltk.book import *
```

Nouns in WordNet

Synsets

Synsets are sets of synonyms for a word. The following code prints all the synsets for the noun form of the word 'project'.

```
wn.synsets('project')
```

```
[Synset('undertaking.n.01'),
 Synset('project.n.02'),
 Synset('project.v.01'),
```

```
 Synset('stick_out.v.01'),
 Synset('project.v.03'),
 Synset('project.v.04'),
 Synset('project.v.05'),
 Synset('project.v.06'),
 Synset('plan.v.03'),
 Synset('project.v.08'),
 Synset('visualize.v.01'),
 Synset('project.v.10'),
 Synset('project.v.11'),
 Synset('project.v.12')]
```

Synset Features

Each synset for 'project' has a definition, usage examples, and lemmas. 'Undertaking' was chosen from the synsets above to explore these features since it is also a noun.

Definition

```
1 wn.synset('undertaking.n.01').definition()
```

```
'any piece of work that is undertaken or attempted'
```

Usage

```
1 wn.synset('undertaking.n.01').examples()
```

```
['he prepared for great undertakings']
```

Lemmas

```
1 wn.synset('undertaking.n.01').lemmas()
```

```
[Lemma('undertaking.n.01.undertaking'),
 Lemma('undertaking.n.01.project'),
 Lemma('undertaking.n.01.task'),
 Lemma('undertaking.n.01.labor')]
```

WordNet Hierarchy

Here the WordNet hierarchy from 'undertaking' is explored from the bottom up using hypernyms. Hypernyms are higher level words. The synsets are displayed for each level.

WordNet has 'entity' as the top level for all nouns. So it appears that 'undertaking' is just 7 levels below the upper-most level. The first few hypernyms seem to make sense, but it starts getting a little weird at 'psychological_feature'.

```
undertaking = wn.synset('undertaking.n.01')
hyper = lambda s: s.hypernyms()
list(undertaking.closure(hyper))
```

```
[Synset('work.n.01'),
 Synset('activity.n.01'),
 Synset('act.n.02'),
 Synset('event.n.01'),
 Synset('psychological_feature.n.01'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

Synset Relations

Noun synsets have 5 semantic relations which are used to connect them to other synsets.

Hypernyms

Higher level words.

```
wn.synset('undertaking.n.01').hypernyms()
```

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

Hyponyms

Lower level words.

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

```
[Synset('adventure.n.01'),
 Synset('assignment.n.05'),
 Synset('baby.n.07'),
 Synset('cinch.n.01'),
 Synset('enterprise.n.01'),
 Synset('labor_of_love.n.01'),
 Synset('marathon.n.01'),
 Synset('no-brainer.n.01'),
 Synset('proposition.n.05'),
 Synset('tall_order.n.01'),
 Synset('venture.n.01')]
```

Meronyms

Words that are part of another. 'Undertaking' doesn't have any meronyms.

```
wn.synset('undertaking.n.01').part_meronyms()
```

```
[]
```

Holonyms

Words that are the whole of another. 'Undertaking' doesn't have any holonyms.

```
wn.synset('undertaking.n.01').part_holonyms()
```

```
[]
```

Antonyms

Words with very differnt meaning than another. This is only provided for the lemma of a synset. The lemma for 'undertaking' doesn't have an antonym.

```
wn.synset('undertaking.n.01').lemmas()[0].antonyms()
```

```
[]
```

Verbs in WordNet

Synsets

Synsets are sets of synonyms for a word. The following code prints all the synsets for the verb form of the word 'project'. I thought it might be interesting to use the same word.

```
wn.synsets('project')
```

```
[Synset('undertaking.n.01'),
 Synset('project.n.02'),
 Synset('project.v.01'),
 Synset('stick_out.v.01'),
 Synset('project.v.03'),
 Synset('project.v.04'),
```

```
 Synset('project.v.05'),
 Synset('project.v.06'),
 Synset('plan.v.03'),
 Synset('project.v.08'),
 Synset('visualize.v.01'),
 Synset('project.v.10'),
 Synset('project.v.11'),
 Synset('project.v.12')]
```

Synset Features

Each synset for 'project' has a definition, usage examples, and lemmas. 'Project' was chosen from the synsets above to explore these features since it is also a verb and probably most closely related to the way I imagine the word.

Definition

```
1 wn.synset('project.v.01').definition()
```

```
'communicate vividly'
```

Usage

```
1 wn.synset('project.v.01').examples()
```

```
['He projected his feelings']
```

Lemmas

```
1 wn.synset('project.v.01').lemmas()
```

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

WordNet Hierarchy

Here the WordNet hierarchy from 'project' is explored from the bottom up using hypernyms. Hypernyms are higher level words. The synsets are displayed for each level.

WordNet does not have a top level for verbs like it does nouns, but I feel 'act' is a pretty good top level. 'Project' is only 4 levels from the top.

```
project = wn.synset('project.v.01')
hyper = lambda s: s.hypernyms()
list(project.closure(hyper))
```

```
[Synset('communicate.v.02'), Synset('interact.v.01'), Synset('act.v.01')]
```

Morphy

Here, morphy() is used to find as many different forms of the verb as possible with very sad results.

```
print(wn.morphy('project'))
print(wn.morphy('project',wn.NOUN))
print(wn.morphy('project',wn.VERB))
```

```
project
project
project
```

Similarity

I want to explore how similar the words 'data' and 'information' are. I'll start by finding the specific synsets that I want to use and then performing 2 different similarity metics on them.

```
wn.synsets('data')
```

```
[Synset('data.n.01'), Synset('datum.n.01')]
```

```
wn.synsets('information')
```

```
[Synset('information.n.01'),
 Synset('information.n.02'),
 Synset('information.n.03'),
 Synset('data.n.01'),
 Synset('information.n.05')]
```

Wu-Palmer Similarity Metric

The Wu-Palmer similarity metric is based on taxonomy and shared ancestor nodes. I am pretty surprised by the dissimilarity of the words 'data' and 'information' for the first comparison. I tried comparing 'data' to all the synset of 'information', but nothing appeared to be very similar.

```
data = wn.synset('data.n.01')
```

```
info = wn.synset('information.n.01')
wn.path_similarity(data, info)
```

```
0.14285714285714285
```

```
data = wn.synset('data.n.01')
info = wn.synset('information.n.05')
wn.path_similarity(data, info)
```

```
0.125
```

```
data = wn.synset('data.n.01')
info = wn.synset('information.n.02')
wn.path_similarity(data, info)
```

```
0.14285714285714285
```

```
data = wn.synset('data.n.01')
info = wn.synset('information.n.03')
wn.path_similarity(data, info)
```

```
0.09090909090909091
```

Lesk Algorithm

I originally thought that the Lesk algorithm returned the synset of 'information' that was most similar to 'data', but after running the Wu-Palmer metric on all of the synset options, I seee that isn't true. I compared the definitions of the synset returned and the synset deemed most similar by the Wu-Palmer metric.

```
lesk('data', 'information')
```

```
Synset('information.n.05')
```

```
wn.synset('information.n.05').definition()
```

```
'(communication theory) a numerical measure of the uncertainty of an outcome'
```

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

```
'a message received and understood'
```

SentiWordNet

Summary:

SentiWordNet is a sentiment analysis tool built on top of WordNet that gives words or sentences positive, negative, and objective scores that add up to 100%. It can be used to determine whether a movie review is good or bad, or it could be used to see what consumer's general feelings are about a product or company.

For this part of the notebook, I have chosen to use the word stress to explore SentiWordNet.

```
synsets = list(swn.senti_synsets('stress'))
print(synsets)

```

```
[SentiSynset('stress.n.01'), SentiSynset('tension.n.01'), SentiSynset('stress.n.03'), S
```

Printing the senti-synsets for synsets of 'stress'.

```
for i in range(len(synsets)-1):
    stress = synsets[i]
    print("\nWord: ", stress.synset)
    print("Positive score = ", stress.pos_score())
    print("Negative score = ", stress.neg_score())
    print("Objective score = ", stress.obj_score())
```

```
Word:  Synset('stress.n.01')
Positive score =  0.125
Negative score =  0.0
Objective score =  0.875

Word:  Synset('tension.n.01')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('stress.n.03')
Positive score =  0.125
Negative score =  0.0
Objective score =  0.875

Word:  Synset('stress.n.04')
Positive score =  0.0
Negative score =  0.375
Objective score =  0.625
```

```
Word:  Synset('stress.n.05')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('stress.v.01')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('stress.v.02')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
```

Here, I am doing the same thing as above for the first synset of each word in a made up sentence.

Looking at the output, it doesn't seem that I have gained much from learning the polarity of each word in the sentence. Most words were rated as objective. I also had to change "I've" to "I have" so that SentiWordNet would pick up on the first 2 words. While I didn't learn much on this particular sentence, I could see how the polarity of individual words might prove useful in giving a naive sentiment rating.

```
sent = "I have got a golden ticket."
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        stress = syn_list[0]
        print("\nWord: ", stress.synset)
        print("Positive score = ", stress.pos_score())
        print("Negative score = ", stress.neg_score())
        print("Objective score = ", stress.obj_score())


```

```
Word:  Synset('iodine.n.01')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('rich_person.n.01')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('get.v.01')
Positive score =  0.125
Negative score =  0.0
```

```
Objective score =  0.875

Word:  Synset('angstrom.n.01')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

Word:  Synset('aureate.s.02')
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
```

Collocation

A collocation is a combination of 2 or more words that occur together more often than is indicated by chance. It is similar to a cliche. A key indication that something is a collocation is that it won't make sense if any of the words are substituted by a synonym.

Here, I explore collocations using NLTK's Book library and text4 on inauguration.

```
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
```

Next, I am going to calculate mutual information for "one another". Mutual information tells us if this is actually a collocation. If the PMI is negative, then the words are likely not a collocation.

```
bigrams = list(ngrams(text4.tokens,2))
bigram_dict = {b:bigrams.count(b) for b in set(bigrams)}
```

```
total_words = len(text4.tokens)
total_bigrams = len(bigrams)
total_one = text4.count('one')
total_another = text4.count('another')
total_oneanother = bigram_dict[('one','another')]
```

```
152900
```

The equation for collocation is the log of:

```
(# of times the specific bigram appears/# of bigrams in the text)
                        divided by
     (# first word/total words)*(# second word/total words)
```

This equation is a probablistic equation that uses the probability of the bigram and the probability of the 2 words individually to determine whether or not the bigram is actually a collocation. The PMI = 7.6 which is a strong indicator that "one another" is a collocation.

```
1 probs = (total_oneanother/total_bigrams)/((total_one/total_words)*(total_another/total_wor
2 min_information·=·math.log2(probs)
3 print(min_information)
```

```
7.632596605651786
```