WordNet

WordNet can be thought of as a large dictionary containing numerous relationshps between english words. A word in this database contains mant synonyms (synsets) which are comprised of verbs, nouns, adjectives etc. The most important distinction and advantage of WordNet over a regular thesaurus s that WordNet creates and labels the sematic relationship between word. This make WordNet is a very useful tool for natural language processing, and also makes it attractive for quick searchups in terms of language processing.

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

```
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data]   Package wordnet is already up-to-date!
    [nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Nouns

```
wn.synsets('doctor')
```

```
    [Synset('doctor.n.01'),
     Synset('doctor_of_the_church.n.01'),
     Synset('doctor.n.03'),
     Synset('doctor.n.04'),
     Synset('sophisticate.v.03'),
     Synset('doctor.v.02'),
     Synset('repair.v.01')]
```

```
wn.synset('doctor.n.04').definition()
```

```
    'a person who holds Ph.D. degree (or the equivalent) from an academic institution'
```

```
wn.synset('doctor.n.04').examples()
```

```
    ['she is a doctor of philosophy in physics']
```

```
wn.synset('doctor.n.04').lemmas()
```

```
    [Lemma('doctor.n.04.doctor'), Lemma('doctor.n.04.Dr.')]
```

```
doctor = wn.synset('doctor.n.04')
print(doctor)
doctor_hypernym = doctor.hypernyms()[0]
while doctor_hypernym:
  print(doctor_hypernym)
  if doctor_hypernym == wn.synset('entity.n.01'):
    break
  if doctor_hypernym.hypernyms():
    doctor_hypernym = doctor_hypernym.hypernyms()[0]
```

```
    Synset('doctor.n.04')
    Synset('scholar.n.01')
    Synset('intellectual.n.01')
    Synset('person.n.01')
    Synset('causal_agent.n.01')
    Synset('physical_entity.n.01')
    Synset('entity.n.01')
```

Nouns, WordNet

It seems that nouns are classified as object in WordNet. As you go up the hierarchy, WordNet tries to put the noun into a classification that changes it into an object. For example, a doctor turns into a 'physical entity' at some point in time up the hierarchy. Based on this logic, a 'pen' might also eventually join the same common acenstral node: 'physical_entity'. Something that is not physical, such as a 'soul' might be only an 'entity' as it has no physical form.

```
print(doctor.hypernyms())
print(doctor.hyponyms())
print(doctor.part_meronyms())
print(doctor.substance_meronyms())
print(doctor.member_meronyms())
```

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

```
print(doctor.part_holonyms())
print(doctor.member_holonyms())
print(doctor.substance_holonyms())
```

```
[]
[]
[]
```

```
doctor.lemmas()[0].antonyms()
```

```
[]
```

Verbs

```
wn.synsets('write')
```

```
[Synset('write.v.01'),
 Synset('write.v.02'),
 Synset('publish.v.03'),
 Synset('write.v.04'),
 Synset('write.v.05'),
 Synset('compose.v.02'),
 Synset('write.v.07'),
 Synset('write.v.08'),
 Synset('spell.v.03'),
 Synset('write.v.10')]
```

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

```
'communicate or express by writing'
```

```
wn.synset('write.v.02').examples()
```

```
['Please write to me every week']
```

```
wn.synset('write.v.02').lemmas()
```

```
[Lemma('write.v.02.write')]
```

```
verb = wn.synset('write.v.02')
print(verb)
verb_hypernym = verb.hypernyms()[0]
while verb_hypernym:
  print(verb_hypernym)
  if verb_hypernym == wn.synset('act.v.01'):
    break
  if verb_hypernym.hypernyms():
    verb_hypernym = verb_hypernym.hypernyms()[0]
```

```
Synset('write.v.02')
Synset('communicate.v.02')
Synset('interact.v.01')
Synset('act.v.01')
```

Verbs, WordNet

From the above results, it seems that WordNet classifies most verbs as an action performed. This makes sense as the root of most verbs is to perform some action. In the example above, the verb write is translated as a action that enables interaction between people, thus translating to 'interact higher up the order. WE can usually classify all verbs to be some sort of act, thus this fits perfectly with the hierarchy in WordNet

```
print(wn.morphy('writing', wn.VERB))
print(wn.morphy('written',wn.VERB))
print(wn.morphy('wrote',wn.VERB))
print(wn.morphy('writes',wn.VERB))
print(wn.morphy('write',wn.VERB))
```

```
write
write
write
write
write
```

Wu-Palmer, Lesk Algorithm

```
wn.synsets('drive')
```

```
[Synset('drive.n.01'),
 Synset('drive.n.02'),
 Synset('campaign.n.02'),
 Synset('driveway.n.01'),
 Synset('drive.n.05'),
 Synset('drive.n.06'),
 Synset('drive.n.07'),
 Synset('drive.n.08'),
 Synset('drive.n.09'),
 Synset('drive.n.10'),
 Synset('drive.n.11'),
 Synset('drive.n.12'),
 Synset('drive.v.01'),
 Synset('drive.v.02'),
 Synset('drive.v.03'),
 Synset('force.v.06'),
 Synset('drive.v.05'),
 Synset('repel.v.01'),
 Synset('drive.v.07'),
 Synset('drive.v.08'),
 Synset('drive.v.09'),
 Synset('tug.v.02'),
 Synset('drive.v.11'),
 Synset('drive.v.12'),
 Synset('drive.v.13'),
 Synset('drive.v.14'),
 Synset('drive.v.15'),
 Synset('drive.v.16'),
 Synset('drive.v.17'),
 Synset('drive.v.18'),
 Synset('drive.v.19'),
 Synset('drive.v.20'),
 Synset('drive.v.21'),
 Synset('drive.v.22')]
```

```
drive = wn.synset('drive.v.01')
```

```
ride = wn.synset('ride.v.01')
```

```
wn.wup_similarity(drive,ride)
```

```
0.2222222222222222
```

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

```
tiger = wn.synset('tiger.n.02')
```

```
wn.wup_similarity(lion,tiger)
```

```
0.9333333333333333
```

```
from nltk.wsd import lesk
```

```
context = ['The','computer-animated','version','looked', 'very','similar','to','a','real','lion','.']
lesk(context,'lion','n')
```

```
Synset('lion.n.02')
```

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

```
'a celebrity who is lionized (much sought after)'
```

Wu-Palmer Algorithm

From the above, results it seems that the Wu-Palmer similarity algorithms mainly assesses two words based on thier respective taxonomies. Thus, even though 'drive' and 'ride' seem to be a lot more similar than 'lion' and 'tiger', theh have a lower similarity rating. This is due to the fact that fundamentally, driving and riding are performing two different actions, thus will only have a common ancestor much higher up the hierarchy in comparision to lions or tigers.

Lesk Algorithm

In terms of the lesk algorithm, the similarity criteria seems to be primarily based on the number of hits in which both words share similar contextual meaning is the highest. This means that the contextual meaning is more important and that is what determines similarity and the most similar word. For the example above, the lion clearly is an animal, but the lesk algorithm seems to think it is a celebrity, probably due to the similarity of the context to other sentences in the database and how they are used in conjunction with 'lion.n.02'.

SentiWordNet

SentiWordNet is another resources built off of WordNet, which primarily focusses on obtaining the sentiments of words and sentences based in WordNet. This is usually a very useful tool to many industries, which utilize sentiment analysis on some level to make business decisions. For examnple, a high frequency trading company would find this sentiment based information about a certain stock in the market very useful.

```
from nltk.corpus import sentiwordnet as swn
nltk.download('sentiwordnet')
```

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
True
```

```
l = list(swn.senti_synsets('nasty'))
```

```
for word in l:
  print(word)
  print('objective score: ',word.obj_score())
```

```
<nasty.a.01: PosScore=0.0 NegScore=0.875>
objective score:  0.125
<nasty.s.02: PosScore=0.0 NegScore=0.75>
objective score:  0.25
<cruddy.s.01: PosScore=0.125 NegScore=0.75>
objective score:  0.125
<filthy.s.01: PosScore=0.0 NegScore=0.875>
objective score:  0.125
```

```
sen = 'Sherlock Holmes is an intelligent detective'
s = sen.split(' ')
for i in s:
  print(i)
  temp_list = list(swn.senti_synsets(i))
  print(temp_list[0])
```

```
Sherlock
<private_detective.n.01: PosScore=0.0 NegScore=0.0>
Holmes
<sherlock_holmes.n.01: PosScore=0.0 NegScore=0.0>
is
<be.v.01: PosScore=0.25 NegScore=0.125>
an
<associate_in_nursing.n.01: PosScore=0.0 NegScore=0.125>
intelligent
<intelligent.a.01: PosScore=0.25 NegScore=0.0>
```

```
detective
<detective.n.01: PosScore=0.0 NegScore=0.0>
```

Base on the results, we can tell that the polarity is highly based off of the meaning of the particular synset that is assumed by SentiWordNet. In the real word, knowing the sentiments of key words is extremely beneficial. While this approach is very accurate to guess the polarity of individual words, it might be a little more complex to use this ti guess the sentiment of a statement. For example: 'My friend is terrible at being a bad person" is actually a compliment but this is a lot harder to interpret with just the polarity of individual words.

Collocations

A collocation is essentially the grouping of two words together a lot more often than any other group of words, relatively. For example, "Fellow Americans" is used by most politicians and used mostly after each other, thus forming a collocation. Something that would not be a collocation is 'every person'. This is because the word person seldom is preceded by 'every' in the english language.

```
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
from nltk.book import *
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package genesis to /root/nltk_data...
[nltk_data]   Package genesis is already up-to-date!
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
[nltk_data] Downloading package webtext to /root/nltk_data...
[nltk_data]   Package webtext is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
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
```

```
passage = ' '.join(text4.tokens)
p_x_y = passage.count('United States')/len(set(text4))
p_x = passage.count('United')/len(set(text4))
p_y = passage.count('States')/len(set(text4))
mutual_info = math.log2(p_x_y/(p_x*p_y))
mutual_info
```

```
4.815657649820885
```

The mutual information formula in my opinion, accurately describes the probability of a collocation being a possibility. If we take a closer look at the formula, we know that the the numerator considers the probability that both words occur together. The higher this happens, the more likely they are to be a collocation. The denominator is telling us that the combined probability is monitored by the individual probailibities. This means that even if the collcoation has a high occurence in the text, this doesnt mean anything if the individual words occur exponentially more often, and vice versa. The final logarithmic component, helps containg the numbers and they might reach high numbers, this helps a ton with visualizign the potential of a collocation.