WordNet

Alekhya Pinnamaneni

What is WordNet?

WordNet is essentially a database representing the English language and the complex relationships between English words. For each word, WordNet keeps a record of the word's various definitions (glosses), use examples, synonyms (synsets), and its relation to other words. WordNet hierarchically organizes synsets using hierarchical relations, including hypernyms, hyponyms, meronyms, holonyms, and troponyms.

In [1]:
```
import nltk
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Out[1]:
```
True
```

Nouns

In [2]:
```
wn.synsets('tulip')
```

Out[2]:
```
[Synset('tulip.n.01')]
```

In [3]:
```
tulip = wn.synset('tulip.n.01')
tulip.definition()
```

Out[3]:
```
'any of numerous perennial bulbous herbs having linear or broadly la
nceolate leaves and usually a single showy flower'
```

In [4]:
```
tulip.examples()
```

Out[4]:
```
[]
```

In [5]:
```
tulip.lemmas()
```

Out[5]:
```
[Lemma('tulip.n.01.tulip')]
```

In [6]:
```
hyp = tulip.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('liliaceous_plant.n.01')
Synset('bulbous_plant.n.01')
Synset('vascular_plant.n.01')
Synset('plant.n.02')
Synset('organism.n.01')
Synset('living_thing.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

WordNet is organized for nouns so that all nouns fall under one big umbrella synset: 'entity'. Starting from 'entity', nouns are organized into smaller and more specific synsets. Entities are divided into physical or abstract entities. Physical entities are dividied into living or non-living things. The synsets slowly get more and more specific until the target synset is reached. Since all nouns originate from the same large 'entity' synset, it takes more iterations to traverse up the WordNet hierarchy from the original synset.

In [7]:
```
tulip.hypernyms()
```

Out[7]:
```
[Synset('liliaceous_plant.n.01')]
```

In [8]:
```
tulip.hyponyms()
```

Out[8]:
```
[Synset('cottage_tulip.n.01'),
 Synset('darwin_tulip.n.01'),
 Synset('dwarf_tulip.n.01'),
 Synset('lady_tulip.n.01'),
 Synset('tulipa_gesneriana.n.01')]
```

In [9]:
```
tulip.part_meronyms()
```

Out[9]:
```
[]
```

In [10]:
```
tulip.part_holonyms()
```

Out[10]:
```
[]
```

In [11]:
```
tulip.lemmas()[0].antonyms()
```

Out[11]:
```
[]
```

Verbs

In [12]:
```
wn.synsets('bake')
```

Out[12]:
```
[Synset('bake.v.01'),
 Synset('bake.v.02'),
 Synset('broil.v.02'),
 Synset('bake.v.04')]
```

In [13]:
```
bake = wn.synset('bake.v.01')
bake.definition()
```

Out[13]:
```
'cook and make edible by putting in a hot oven'
```

In [14]:
```
bake.examples()
```

Out[14]:
```
['bake the potatoes']
```

In [15]:
```
bake.lemmas()
```

Out[15]:
```
[Lemma('bake.v.01.bake')]
```

In [16]:
```
wn.synsets('bake', 'v')[0].root_hypernyms()
```

Out[16]:
```
[Synset('change.v.02')]
```

In [17]:
```
hyp = bake.hypernyms()[0]
top = wn.synset('change.v.02')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('cook.v.03')
Synset('change_integrity.v.01')
Synset('change.v.02')
```

WordNet does not organize verbs in the same way it organizes nouns. All nouns eventually lead to one synset (the 'entity' synset), but verbs eventually lead to different synsets. The word 'bake' eventually leads to the 'change.v.02' synset, but a different verb will lead to a different root synset. Therefore, it takes fewer iterations to traverse up the WordNet hierarchy for verbs.

Morphy

In [37]:
```
wn.morphy('baking', wn.VERB)
```

Out[37]:
```
'bake'
```

In [45]:
```
wn.morphy('baker')
```

Out[45]:
```
'baker'
```

In [46]:
```
wn.morphy('baked')
```

Out[46]:
```
'bake'
```

Similarity

In [20]:
```
yell = wn.synset('yell.v.02')
shout = wn.synset('shout.v.01')
print('yell: ', yell.definition())
print('shout: ', shout.definition())
```

```
yell:  utter or declare in a very loud voice
shout:  utter in a loud voice; talk in a loud voice (usually denotin
g characteristic manner of speaking)
```

Wu-Palmer Similarity Metric

In [21]:
```
wn.wup_similarity(yell, shout)
```

Out[21]:
```
0.9090909090909091
```

Lesk algorithm

In [8]:
```
from nltk.wsd import lesk
sentence = "I am so angry at my cat for spilling the milk that I could
yell at her."
yell2 = lesk(sentence, 'shout', pos='v')
print(yell2, yell2.definition())
```

```
Synset('shout.v.02') utter a sudden loud cry
```

In [9]:
```
sentence = "You can scream and shout all you want, but it will not get
you what you want."
shout2 = lesk(sentence, 'yell', pos='v')
print(shout2, shout2.definition())
```

```
Synset('yell.v.02') utter or declare in a very loud voice
```

Running the Wu-Palmer algorithm resulting in a similary of approximately 0.9, which I expected since 'yell' and 'shout' have very similar meanings. I also ran the Lesk algorithm to find the synset for 'shout' that had the most overlap with the cotext sentence containing 'yell in it, and it returned a different synset of 'shout' than the one I expected. This could be because the context sentence implied a different meaning of 'shout. I also tried the Lesk algorithm to find a synset of 'yell' with a context sentence with 'shout' in it, and it returned the expected synset.

SentiWordNet

SentiWordNet is built on top of WordNet and is used to extract the sentiment of text. SentiWordNet assigns 3 sentiment scores for each synset from WordNet: positivity, negativity, objectivity. SentiWordNet could be useful for sentiment analysis to analyze product reviews, customer feedback, survey responses, and more.

In [11]:
```
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.
```

Out[11]:
```
True
```

In [10]:
```
destroy_list = list(swn.senti_synsets('destroy'))
for word in destroy_list:
    print(word)
```

```
<destroy.v.01: PosScore=0.0 NegScore=0.0>
<destroy.v.02: PosScore=0.0 NegScore=0.5>
<demolish.v.03: PosScore=0.125 NegScore=0.0>
<destroy.v.04: PosScore=0.0 NegScore=0.25>
```

In [17]:
```
sentence = 'My whole family hated this movie'
words = sentence.split()
for word in words:
    syn_list = list(swn.senti_synsets(word))
    if syn_list:
      print(syn_list[0])
```

```
<whole.n.01: PosScore=0.0 NegScore=0.0>
<family.n.01: PosScore=0.0 NegScore=0.0>
<hate.v.01: PosScore=0.0 NegScore=0.75>
<movie.n.01: PosScore=0.0 NegScore=0.0>
```

The polarity scores for the 'destroy' synsets surprised me because I expected the results to be much more negative than they were. None of the synsets had a negative score of more than 0.5. In fact, 'demolish.v.03' even had a positive score instead of negative.

I was not surprised, however, by the polarity of the words in the sentence. All of the words were neutral besides 'hate', which had a negativity score of 0.75 as I expected.

Knowing the polarity scores of words would be useful in NLP applications for deciphering and processing real text/speech to understand the sentiment and intent of the text/speech. It would also be useful to generate appropriate responses.

Collocations

A collocation is a phrase made up of multiple words. These words have different meanings when used individually. When these words are put together in a certain order to make a collocation, the collocation has a different and greater meaning. For example, 'tip toes' has a different meaning than 'tip' and 'toes' individually.

In [26]:
```
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
from nltk.book import text4
```

```
[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
```

In [29]:
```
nltk.download('stopwords')
text4.collocations()
```

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

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; Indi
an
tribes; public debt; foreign nations
```

Mutual Information

In [30]:
```
import math
text = ' '.join(text4.tokens)
total = len(set(text4))
hg = text.count('God bless') / total
print("p(God bless) = ",hg )
h = text.count('God') / total
print("p(God) = ", h)
g = text.count('bless') / total
print('p(bless) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(God bless) =  0.0016957605985037406
p(God) =  0.011172069825436408
p(bless) =  0.0085785536159601
pmi =  4.145157780720282
```

The mutual information for 'God bless' is approximately 4.15. This score is fairly high, meaning that 'God bless' is fairly likely to be a collocation. This can mean that the words 'God' and 'bless' are usually not put together unless it is to mean 'God bless' as a greater meaning.