CS4395 Assignment 3 - WordNet

WordNet is a database featuring a highly complex and defined hierarchy of words in many different languages. It links together words into synsets (synonym sets), creating a network of words, connected by their semantic similarities. As an incredibly interconnected database, it exists to serve many natural language processing applications, such as spell/grammar checking, summarizing tools, and generally parsing through text.

Noun Synsets

In [18]:
```
from nltk.corpus import wordnet as wn

# get all sysnets of 'rose'
wn.synsets('rose')
```

Out[18]:
```
[Synset('rose.n.01'),
 Synset('blush_wine.n.01'),
 Synset('rose.n.03'),
 Synset('rise.v.01'),
 Synset('rise.v.02'),
 Synset('arise.v.03'),
 Synset('rise.v.04'),
 Synset('surface.v.01'),
 Synset('originate.v.01'),
 Synset('ascend.v.08'),
 Synset('wax.v.02'),
 Synset('heighten.v.01'),
 Synset('get_up.v.02'),
 Synset('rise.v.11'),
 Synset('rise.v.12'),
 Synset('rise.v.13'),
 Synset('rebel.v.01'),
 Synset('rise.v.15'),
 Synset('rise.v.16'),
 Synset('resurrect.v.03'),
 Synset('rose.s.01')]
```

In [42]:
```
# I will work with blush_wine.n.01 for the following

# definition of blush wine
wn.synset('blush_wine.n.01').definition()
```

Out[42]:
```
'pinkish table wine from red grapes whose skins were removed after fermentation began'
```

In [43]:
```
# example of blush wine
wn.synset('blush_wine.n.01').examples()
```

Out[43]:
```
[]
```

In [44]:
```
# lemmas of blush wine
wn.synset('blush_wine.n.01').lemmas()
```

Out[44]:
```
[Lemma('blush_wine.n.01.blush_wine'),
 Lemma('blush_wine.n.01.pink_wine'),
 Lemma('blush_wine.n.01.rose'),
 Lemma('blush_wine.n.01.rose_wine')]
```

In [45]:
```
# traverse up the blush wine sysnet hierarchy

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

Out[45]:
```
[Synset('wine.n.01'),
 Synset('alcohol.n.01'),
 Synset('beverage.n.01'),
 Synset('drug_of_abuse.n.01'),
 Synset('food.n.01'),
 Synset('liquid.n.01'),
 Synset('drug.n.01'),
 Synset('substance.n.07'),
 Synset('fluid.n.01'),
 Synset('agent.n.03'),
 Synset('matter.n.03'),
 Synset('substance.n.01'),
 Synset('causal_agent.n.01'),
 Synset('physical_entity.n.01'),
 Synset('part.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01'),
 Synset('relation.n.01'),
 Synset('entity.n.01'),
 Synset('abstraction.n.06')]
```

The traversal of the blush wine synset tree is really interesting to observe, particularly because I didn't expect blush wine when I chose the word 'rose'. It's clear to me up until beverage, but I think it's intriguing that its categorized as a drug of abuse. Not because it isn't, but because coloquially its seen as just a drink. It's also interesting that drug of abuse didn't follow alcohol--it followed beverage, which I think is quite strange. After that though, it descends into more abstract categorizations until it lands into abstraction, which I feel is interesting but fitting.

In [25]:
```
# get hypernyms of blush wine
wn.synset('blush_wine.n.01').hypernyms()
```

Out[25]:
```
[Synset('shrub.n.01')]
```

In [26]:
```
# get hyponyms of blush wine
wn.synset('blush_wine.n.01').hyponyms()
```

Out[26]:
```
[Synset('banksia_rose.n.01'),
 Synset('cherokee_rose.n.01'),
 Synset('china_rose.n.01'),
 Synset('damask_rose.n.01'),
 Synset('dog_rose.n.01'),
 Synset('ground_rose.n.01'),
 Synset('mountain_rose.n.01'),
 Synset('multiflora.n.01'),
 Synset('musk_rose.n.01'),
 Synset('sweetbrier.n.01'),
 Synset('tea_rose.n.01')]
```

In [30]:
```
# get meronyms of blush wine
wn.synset('blush_wine.n.01').part_meronyms()
```

Out[30]:
```
[Synset('hip.n.05')]
```

In [31]:
```
# get holonyms of blush wine
wn.synset('blush_wine.n.01').part_holonyms()
```

Out[31]:
```
[]
```

In [34]:
```
# get antonyms of blush wine
blush_wine = wn.synsets('blush_wine', pos=wn.ADJ)[0]
blush_wine.lemmas()[0].antonyms()
```

Out[34]:
```
[]
```

Verb Synsets

In [80]:
```
# get all synsets of glow

wn.synsets('glow')
```

Out[80]:
```
[Synset('freshness.n.03'),
 Synset('luminescence.n.02'),
 Synset('incandescence.n.01'),
 Synset('glow.n.04'),
 Synset('glow.n.05'),
 Synset('radiance.n.01'),
 Synset('gleam.n.01'),
 Synset('glow.v.01'),
 Synset('glow.v.02'),
 Synset('burn.v.02'),
 Synset('glow.v.04'),
 Synset('glow.v.05')]
```

In [81]:
```
# I will work with burn.v.02 for the following

# definition of burn
wn.synset('burn.v.02').definition()
```

Out[81]:
```
'shine intensely, as if with heat'
```

In [82]:
```
# definition of burn
wn.synset('burn.v.02').examples()
```

Out[82]:
```
['The coals were glowing in the dark', 'The candles were burning']
```

In [83]:
```
# definition of burn
wn.synset('burn.v.02').lemmas()
```

Out[83]:
```
[Lemma('burn.v.02.burn'), Lemma('burn.v.02.glow')]
```

In [84]:
```
# traverse up the blush wine synset hierarchy

burn = wn.synset('burn.v.02')
hyper = lambda s: s.hypernyms()
list(burn.closure(hyper))
```

Out[84]:
```
[Synset('shine.v.02')]
```

It seems as if the verb side of the hierarchy is either far less robust, or far less interconnected. This makes sense, though, as verbs are much more contextually defined and connected, unlike nouns, that can be categorized much more objectively. Still, it is a little disappointing to see only one word on the hierarchy.

In [92]:
```
# using morphy to find as many different forms of the word as possible

wn.morphy('burning')
```

Out[92]:
```
'burning'
```

In [87]:
```
wn.morphy('burned')
```

Out[87]:
```
'burn'
```

In [88]:
```
wn.morphy('burns')
```

Out[88]:
```
'burns'
```

In [90]:
```
wn.morphy('burned')
```

Out[90]:
```
'burn'
```

In [91]:
```
wn.morphy('burnt')
```

Out[91]:
```
'burn'
```

Word Comparision

In [95]:
```
# choosing the first word for similarity testing
wn.synsets('crown')
```

Out[95]:
```
[Synset('crown.n.01'),
 Synset('crown.n.02'),
 Synset('crown.n.03'),
 Synset('crown.n.04'),
 Synset('crown.n.05'),
 Synset('crown.n.06'),
 Synset('crown.n.07'),
 Synset('peak.n.04'),
 Synset('pennant.n.01'),
 Synset('pate.n.02'),
 Synset('crown.n.11'),
 Synset('crown.n.12'),
 Synset('crown.v.01'),
 Synset('crown.v.02'),
 Synset('crown.v.03'),
 Synset('crown.v.04')]
```

In [100]:
```
wn.synset('crown.n.01').definition()
```

Out[100]:
```
'the Crown (or the reigning monarch) as the symbol of the power and authority of a monarchy'
```

In [101]:
```
# choosing the second word for similarity testing
wn.synsets('tiara')
```

Out[101]:
```
[Synset('tiara.n.01')]
```

In [104]:
```
# using the wu palmer metric

crown = wn.synset('crown.n.01')
tiara = wn.synset('tiara.n.01')
wn.wup_similarity(crown, tiara)
```

Out[104]:
```
0.125
```

In [108]:
```
# using the lesk algorithm

from nltk.wsd import lesk

sent = ['The', 'royalty', 'wear', 'crowns', 'to', 'symolize', 'their', 'power', '.']
print(lesk(sent, 'crown'))
print(lesk(sent, 'tiara'))
```

```
Synset('pennant.n.01')
Synset('tiara.n.01')
```

It's interesting to see how crown and tiara, both fancy head ornaments, are determined as different by the wu palmer metric, and the synsets are derived differently from the same sentence using the lesk algorithm. It makes sense, as objectively, a crown is unisex and specifically for monarchy and power, and a tiara is more feminine and for smaller declarations of superiority-- such as prom, or a beauty pageant. Coloquially though, they're often associated together, and it's clear NLP tools cannot pick up on that colloquial connection.

SentiWordNet

SentiWordNet essentially defines the sentimental/emotional association to a word, breaking down the word into positive, negative, and neutral percentages. This can be used to parse through tone in a written text and suggest additions and changes based on the overall tone of the text, such as keyboard suggestions on IOS, or writing suggestions on Grammarly.

In [112]:
```
from nltk.corpus import sentiwordnet as swn

# Look for the right definition of an emotionally charged word

wn.synsets('panic')
```

Out[112]:
```
[Synset('panic.n.01'),
 Synset('panic.n.02'),
 Synset('panic.v.01'),
 Synset('panic.v.02')]
```

In [113]:
```
wn.synset('panic.n.01').definition()
```

Out[113]:
```
'an overwhelming feeling of fear and anxiety'
```

In [115]:
```python
# Found the word, lets get the sentisysnets now

senti_list = list(swn.senti_synsets('panic'))
for item in senti_list:
    print(item)
```

```
<panic.n.01: PosScore=0.25 NegScore=0.75>
<panic.n.02: PosScore=0.0 NegScore=0.0>
<panic.v.01: PosScore=0.0 NegScore=0.25>
<panic.v.02: PosScore=0.0 NegScore=0.0>
```

In [114]:
```python
# get the breakdowns of the polarity

panic = swn.senti_synset('panic.n.01')
print(panic)

print("Positive score = ", panic.pos_score())
print("Negative score = ", panic.neg_score())
print("Objective score = ", panic.obj_score())
```

```
<panic.n.01: PosScore=0.25 NegScore=0.75>
Positive score =  0.25
Negative score =  0.75
Objective score =  0.0
```

In [120]:
```python
# break down the sentisysnets in a sentence

sent2 = "I was terrified of losing you."
tokens = sent2.split()
for token in tokens:
    print(token)
    senti_token = list(swn.senti_synsets(token))
    if senti_token:
        print("Positive score = ", senti_token[0].pos_score())
        print("Negative score = ", senti_token[0].neg_score())
        print("Objective score = ", senti_token[0].obj_score())
    else:
        print("Word has no senti synsets.")
```

```
I
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
was
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
terrified
Positive score =  0.25
Negative score =  0.0
Objective score =  0.75
of
Word has no senti synsets.
losing
Positive score =  0.0
Negative score =  0.5
Objective score =  0.5
you.
Word has no senti synsets.
```

It's interesting observing the positive, negative, and neutral breakdowns of these words, especially in seeing "panic" and "terrified" have a 1/4th positive score--especially since I intended them as specifically negative words. I don't understand the positive associations with either of those words, however, I see the use in this in trying to parse through the tone of an overall piece of text, by tallying up the scores and guiding AI suggestions based on the calculated tone.

Collocations

A collocation is a set of words that compose a greater meaning altogether than the individual words do on their own. Additionally, either of the words in this set cannot be replaced with a synonym and still make sense. For example, "dead ahead" makes sense, "deceased ahead" does not.

In [124]:
```
from nltk.book import *
text4

# output collocations for text 4

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
```

In [126]:
```
#calculate mutual information

import math

text = ' '.join(text4.tokens)

vocab = len(set(text4))
ap = text.count('American people')/vocab
print("p(American people) = ",ap )
a = text.count('American')/vocab
print("p(American) = ", a)
p = text.count('people')/vocab
print('p(people) = ', p)
pmi = math.log2(ap / (a * p))
print('pmi = ', pmi)
```

```
p(American people) =  0.00399002493765586
p(American) =  0.025735660847880298
p(people) =  0.06264339152119701
pmi =  1.3073947068021263
```

The PMI score indicates that there is a likelihood of "American people" being a collocation, however, at least by the examples provided in the Github, it seems that it is not very definitive, as the Github had an example with a PMI greater than 5. Because of this I believe that it being a positive number greater than 1 indicates that it is very likely to be a collocation, but the PMI score is not large enough or small enough to determine so with absolute certainty.