Portfolio Component 3 - WordNet

Aleena Syed

CS 4395.001

Summary of Wordnet

WordNet is a lexical database for the English language and is part of the NLTK corpus (https://pythonprogramming.net/wordnet-nltk-tutorial/). It categorizes words into synonyms (synsets), which can then be defined or modified into various types, such as hyponyms or hypernyms, among others.

In [58]:
```
# select noun - output all synsets
import nltk
from nltk.corpus import wordnet as wn
wn.synsets('table')
synset_arr = wn.synsets('table')
print(synset_arr)
```

```
[Synset('table.n.01'), Synset('table.n.02'), Synset('table.n.03'), Synset('mesa.n.01'), Synset('table.n.05'), Synset('board.n.04'), Synset('postpone.v.01'), Synset('table.v.02')]
```

In [59]:
```
# Select one synset from the list of synsets.
#Extract its definition, usage examples, and lemmas.

print(synset_arr[1].definition())
print(synset_arr[1].examples())
print(synset_arr[1].lemmas())

# traverse hierarchy of noun
hyper = lambda s: s.hypernyms()
list(synset_arr[1].closure(hyper))
```

```
a piece of furniture having a smooth flat top that is usually supported by one or more vertical legs
['it was a sturdy table']
[Lemma('table.n.02.table')]
```

Out[59]:
```
[Synset('furniture.n.01'),
 Synset('furnishing.n.02'),
 Synset('instrumentality.n.03'),
 Synset('artifact.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

Wordnet Noun Hierarchy

Synsets are organized in hierarchical relation, including:

- hypernyms
- hyponyms
- meronyms
- holonyms

with 'entity' at the top of this hierarchy.

In [60]:
```
# Hypernym, hyponym, meronym, and holonyms of table
print(synset_arr[1].hypernyms())
print(synset_arr[1].hyponyms())
print(synset_arr[1].part_meronyms())
print(synset_arr[1].part_holonyms())
# no antonym for 'table'
for lemma in synset_arr[1].lemmas():
    if lemma.antonyms():
        lemma()[0].name()
```

```
[Synset('furniture.n.01')]
[Synset('altar.n.01'), Synset('booth.n.01'), Synset('breakfast_table.n.01'), Synset('card_table.n.01'), Synset('card_table.n.02'), Synset('coffee_table.n.01'), Synset('conference_table.n.01'), Synset('console_table.n.01'), Synset('counter.n.01'), Synset('desk.n.01'), Synset('dressing_table.n.01'), Synset('drop-leaf_table.n.01'), Synset('gaming_table.n.01'), Synset('gueridon.n.01'), Synset('kitchen_table.n.01'), Synset('operating_table.n.01'), Synset('parsons_table.n.01'), Synset('pedestal_table.n.01'), Synset('pier_table.n.01'), Synset('plate.n.01'), Synset('pool_table.n.01'), Synset('stand.n.04'), Synset('table-tennis_table.n.01'), Synset('tea_table.n.01'), Synset('trestle_table.n.01'), Synset('worktable.n.01')]
[Synset('leg.n.03'), Synset('tabletop.n.01'), Synset('tableware.n.01')]
[]
```

In [61]:
```
# Select verb - output all synsets
wn.synsets('run')
synset_arr = wn.synsets('run', pos=wn.VERB)
print(synset_arr)
```

```
[Synset('run.v.01'), Synset('scat.v.01'), Synset('run.v.03'), Synset('operate.v.01'), Synset('run.v.05'), Synset('run.v.06'), Synset('function.v.01'), Synset('range.v.01'), Synset('campaign.v.01'), Synset('play.v.18'), Synset('run.v.11'), Synset('tend.v.01'), Synset('run.v.13'), Synset('run.v.14'), Synset('run.v.15'), Synset('run.v.16'), Synset('prevail.v.03'), Synset('run.v.18'), Synset('run.v.19'), Synset('carry.v.15'), Synset('run.v.21'), Synset('guide.v.05'), Synset('run.v.23'), Synset('run.v.24'), Synset('run.v.25'), Synset('run.v.26'), Synset('run.v.27'), Synset('run.v.28'), Synset('run.v.29'), Synset('run.v.30'), Synset('run.v.31'), Synset('operate.v.03'), Synset('run.v.33'), Synset('run.v.34'), Synset('ply.v.03'), Synset('hunt.v.01'), Synset('race.v.02'), Synset('move.v.13'), Synset('melt.v.01'), Synset('ladder.v.01'), Synset('run.v.41')]
```

In [62]:
```
# print examples and defintion for verb of choice
print(synset_arr[1].examples())
print(synset_arr[1].lemmas())
```

```
['If you see this man, run!', 'The burglars escaped before the police showed up']
[Lemma('scat.v.01.scat'), Lemma('scat.v.01.run'), Lemma('scat.v.01.scarper'), Lemma('scat.v.01.turn_tail'), Lemma('scat.v.01.lam'), Lemma('scat.v.01.run_away'), Lemma('scat.v.01.hightail_it'), Lemma('scat.v.01.bunk'), Lemma('scat.v.01.head_for_the_hills'), Lemma('scat.v.01.take_to_the_woods'), Lemma('scat.v.01.escape'), Lemma('scat.v.01.fly_the_coop'), Lemma('scat.v.01.break_away')]
```

In [63]:
```
# traverse up hierarchy
hyper = lambda s: s.hypernyms()
list(synset_arr[1].closure(hyper))
```

Out[63]:
```
[Synset('leave.v.01')]
```

WordNet Verb Hierarchy

Verbs do not have a top-level synset, unlike nouns.

In [64]:
```
# use morphy to find as many different form of 'run'
wn.morphy('run', wn.VERB)
```

Out[64]:
```
'run'
```

In [65]:
```
# select two similar words and output Wu-Palmer similarity metric
dog = wn.synset('dog.n.01')
wolf = wn.synset('wolf.n.01')

wn.wup_similarity(dog, wolf)
```

Out[65]:
```
0.9285714285714286
```

In [66]:
```
# Run Lesk algorithm
from nltk.wsd import lesk
sent = ['I', 'saw', 'a', 'dog', 'and', 'it', 'looked', 'like', 'a', 'wolf', '.']
print(lesk(sent, 'dog'))
print(lesk(sent, 'wolf'))
```

```
Synset('pawl.n.01')
Synset('wolf.n.04')
```

Observations

Dog and wolf have a high level of similarity, which makes sense since they are from the same family. The output 'pawl' from the Lesk algorithm is strange for 'dog', since the context of 'dog' in this case is just the animal. The output for 'wolf' makes sense.

SentiWordNet

SentiWordNet is a lexical resource supporting sentiment classification and opinion mining. It assigns to each synset three sentiment scores, positivity, negativity, and objectivity.

In [67]:
```
# Select emotionally charged word and find its senti-synsets and polarity scores
from nltk.corpus import sentiwordnet as swn
synset_arr = wn.synsets('depress')
print(synset_arr[0].definition())

depressed = swn.senti_synset('depress.v.01')
print(depressed)
print("Positive score = ", depressed.pos_score())
print("Negative score = ", depressed.neg_score())
print("Objective score = ", depressed.obj_score())

p = list(swn.senti_synsets('depress'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score())
```

```
lower someone's spirits; make downhearted
<depress.v.01: PosScore=0.0 NegScore=0.125>
Positive score =  0.0
Negative score =  0.125
Objective score =  0.875
negative:  0.125
positive:  0.0
objective:  0.875
```

In [68]:
```
# Output polarity of each word in sentence
sent = "I was super happy yesterday after dinner."
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()

print("neg\tpos counts")
print(neg, '\t', pos)
```

```
neg     pos counts
0.0      0.875
```

SentiWordNet Observations

For the word 'depress', which in this case, is the verb to lower someone's spirits; make downhearted. This has a negative connotation, which is why the positive score was 0. However, I thought that the negative score would be higher than the objective score. These scores are useful in analyzing the sentiment of larger texts, allowing us to determine labels for models that can extend to other texts.

Collocations

Collocations are words that appear together to form a meaning greater than the sum of their parts.

In [69]:
```
from nltk.book import *
text4
```

Out[69]:
```
<Text: Inaugural Address Corpus>
```

In [70]:
```
# get collocations
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 [71]:
```
text = ' '.join(text4.tokens)
text[:100]
```

Out[71]:
```
'Fellow - Citizens of the Senate and of the House of Representatives : Among the vicissitudes inciden'
```

In [72]:
```
# select 'fellow citizens' as collocation and calculate mutual information
import math
vocab = len(set(text4))
hg = text4.count('fellow citizens')/vocab
print("p(fellow citizens) = ",hg )
h = text4.count('fellow')/vocab
print("p(fellow) = ", h)
g = text4.count('citizens')/vocab
print('p(citizens) = ', g)
# Math domain error when calculating pmi
#pmi = math.log2(hg / (h * g))
#print('pmi = ', pmi)
```

```
p(fellow citizens) =  0.0
p(fellow) =  0.013366583541147132
p(citizens) =  0.024039900249376557
```

Collocations Observations

Citizens scores higher than fellow with this formula. When attempting to calculate pmi score, I was getting a math domain error, which I am not sure why I am getting.