Homework 3

Nina Rao

WordNet is a hierarchical database of nouns, verbs, adjectives, and adverbs that includes glosses, synsets, use examples, and relations to other words. It is used to analyze semantic relations between words. Glosses are the definitions of words. Synsets are sets of synonyms and are connected to other synsets through these semantic relations. These include hypernym, hyponym, meronym, holonym, and troponym.

```
In [2]: from nltk.corpus import wordnet as wn
```

Nouns

```
In [3]: # NOUN : dream
        # get all synsets of 'dream'
        wn.synsets('blood')

Out[3]: [Synset('blood.n.01'),
         Synset('blood.n.02'),
         Synset('rake.n.01'),
         Synset('lineage.n.01'),
         Synset('blood.n.05'),
         Synset('blood.v.01')]
```

```
In [4]: # definition
        wn.synset('blood.n.01').definition()

Out[4]: 'the fluid (red in vertebrates) that is pumped through the body by the heart
        and contains plasma, blood cells, and platelets'
```

```
In [5]: # extract examples
        wn.synset('blood.n.01').examples()

Out[5]: ['blood carries oxygen and nutrients to the tissues and carries away waste pr
        oducts',
         'the ancients believed that blood was the seat of the emotions']
```

```
In [6]: # extract lemmas
        wn.synset('blood.n.01').lemmas()

Out[6]: [Lemma('blood.n.01.blood')]
```

In [7]:
```
# traversing hierarchy
blood = wn.synset('blood.n.01')
hyp = blood.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('liquid_body_substance.n.01')
Synset('body_substance.n.01')
Synset('substance.n.01')
Synset('matter.n.03')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Wordnet organizes nouns by their relations to other nouns hierarchically. These nouns are linked by varying degrees of broadness upward and specificity downward. As you move up, you could consider the word to be a generalization of the former word. We stop once we reach entity because that is the most generalized form of the word.

Verbs

In [8]:
```
blood_lemma = wn.lemma('blood.n.01.blood')

# hypernyms
print(blood.hypernyms())

# hyponyms
print(blood.hyponyms())

# meronyms
print(blood.part_meronyms())

# holonyms
print(blood.part_holonyms())

# antonyms
print(blood_lemma.antonyms())
```

```
[Synset('liquid_body_substance.n.01')]
[Synset('arterial_blood.n.01'), Synset('blood_clot.n.01'), Synset('blood_grou
p.n.01'), Synset('bloodstream.n.01'), Synset('cord_blood.n.01'), Synset('gor
e.n.02'), Synset('lifeblood.n.01'), Synset('menorrhea.n.01'), Synset('venous_
blood.n.01'), Synset('whole_blood.n.01')]
[Synset('blood_cell.n.01')]
[]
[]
```

In [9]:
```
# VERB : play
# get all synsets of play
wn.synsets('play')
```

Out[9]:
```
[Synset('play.n.01'),
 Synset('play.n.02'),
 Synset('play.n.03'),
 Synset('maneuver.n.03'),
 Synset('play.n.05'),
 Synset('play.n.06'),
 Synset('bid.n.02'),
 Synset('play.n.08'),
 Synset('playing_period.n.01'),
 Synset('free_rein.n.01'),
 Synset('shimmer.n.01'),
 Synset('fun.n.02'),
 Synset('looseness.n.05'),
 Synset('play.n.14'),
 Synset('turn.n.03'),
 Synset('gambling.n.01'),
 Synset('play.n.17'),
 Synset('play.v.01'),
 Synset('play.v.02'),
 Synset('play.v.03'),
 Synset('act.v.03'),
 Synset('play.v.05'),
 Synset('play.v.06'),
 Synset('play.v.07'),
 Synset('act.v.05'),
 Synset('play.v.09'),
 Synset('play.v.10'),
 Synset('play.v.11'),
 Synset('play.v.12'),
 Synset('play.v.13'),
 Synset('play.v.14'),
 Synset('play.v.15'),
 Synset('play.v.16'),
 Synset('play.v.17'),
 Synset('play.v.18'),
 Synset('toy.v.02'),
 Synset('play.v.20'),
 Synset('dally.v.04'),
 Synset('play.v.22'),
 Synset('dally.v.01'),
 Synset('play.v.24'),
 Synset('act.v.10'),
 Synset('play.v.26'),
 Synset('bring.v.03'),
 Synset('play.v.28'),
 Synset('play.v.29'),
 Synset('bet.v.02'),
 Synset('play.v.31'),
 Synset('play.v.32'),
 Synset('play.v.33'),
 Synset('meet.v.10'),
 Synset('play.v.35')]
```

In [10]:
```
# definition
wn.synset('play.v.03').definition()
```
Out[10]:
```
'play on an instrument'
```

In [11]:
```
# extract examples
wn.synset('play.v.03').examples()
```
Out[11]:
```
['The band played all night long']
```

In [12]:
```
# extract lemmas
wn.synset('play.v.03').lemmas()
```
Out[12]:
```
[Lemma('play.v.03.play')]
```

In [13]:
```
play = wn.synset('play.v.03')
hyper = lambda s: s.hypernyms()
list(play.closure(hyper))
```
Out[13]:
```
[Synset('perform.v.03'), Synset('re-create.v.01'), Synset('make.v.03')]
```

Wordnet also organizes verbs by their hierarchical relation to other verbs. For the verbs, we used the closure method defined in nltk in order to display the hierarchy. I noticed that play is a word with many meanings and has a generalized use case as well, so its closure is composed more so of synonyms.

Using Morphy

In [14]:
```
wn.morphy('godliest', wn.ADJ)
```
Out[14]:
```
'godly'
```

Finding the similarity between words

In [15]:
```
# desert vs. tundra
# a tundra gets very minimal precipitation annually, technically classifying
# it as a desert. However deserts
print(wn.synsets('desert'))
print(wn.synset('desert.n.01').definition(), "\n")
desert = wn.synset('desert.n.01')

print(wn.synsets('tundra'))
print(wn.synset('tundra.n.01').definition())
tundra = wn.synset('tundra.n.01')
```

```
[Synset('desert.n.01'), Synset('abandon.v.05'), Synset('defect.v.01'), Synset
('desert.v.03')]
arid land with little or no vegetation

[Synset('tundra.n.01')]
a vast treeless plain in the Arctic regions where the subsoil is permanently
frozen
```

In [16]:
```
# Wu-Palmer similarity metric
wn.wup_similarity(desert, tundra)
```

Out[16]:
```
0.42857142857142855
```

In [39]:
```
# Lesk algorithm
from nltk.wsd import lesk
# the Arctic has tundra and desert like qualities
sent1 = ['The', 'Artic', 'is', 'a', 'rugged', 'biome', 'with', 'little', 'biod
iversity', ',', 'the', 'flora', 'and', 'fauna', 'must', 'endure', 'brutal', 'c
old', 'and', 'winds', '.']
print(lesk(sent1, 'desert'))
print(lesk(sent1, 'tundra'))
```

```
Synset('desert.n.01')
Synset('tundra.n.01')
```

It is interesting to see that the Wu-Palmer similarity metric gave these two words an index of 0.43 out of 1. It was able to recognize the similarities of these words by their definitions (both being biome types having little precipitation, little to no vegetation, and extreme temperatures) but also see major differences, which I believe is reflected in the score. Meanwhile, the Lesk algorithm uses word overlap count by looking at context and comparing to glosses. It correctly placed both definitions, however both these words synsets were very small so that should be taken into account.

SentiWordNet is an extension of WordNet and deals with the emotional connotations of words. It is a type of sentiment analyzer that assigns scores to words. There are 3 types of scores: positive, negative, and objective. This can be used to identify sentiment in text and has applications in detecting hate speech on social media.

In [18]:
```
from nltk.corpus import sentiwordnet as swn
print("euphoria:", wn.synsets('euphoria'))
euphoric = swn.senti_synset('euphoria.n.01')
print(wn.synset('euphoria.n.01').definition(), "\n")
print(euphoric)
print("Positive score = ", euphoric.pos_score())
print("Negative score = ", euphoric.neg_score())
print("Objective score = ", euphoric.obj_score())
```

```
euphoria: [Synset('euphoria.n.01')]
a feeling of great (usually exaggerated) elation

<euphoria.n.01: PosScore=0.125 NegScore=0.5>
Positive score =  0.125
Negative score =  0.5
Objective score =  0.375
```

In [25]:
```
sent = "Eating spicy food hurts so good"
words = sent.split()
for w in words:
    word = list(swn.senti_synsets(w))[0]
    print(word)
    print("Positive score = ", word.pos_score())
    print("Negative score = ", word.neg_score())
    print("Objective score = ", word.obj_score())
```

```
<eating.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
<piquant.s.01: PosScore=0.375 NegScore=0.5>
Positive score =  0.375
Negative score =  0.5
Objective score =  0.125
<food.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
<injury.n.01: PosScore=0.0 NegScore=0.625>
Positive score =  0.0
Negative score =  0.625
Objective score =  0.375
<sol.n.03: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
<good.n.01: PosScore=0.5 NegScore=0.0>
Positive score =  0.5
Negative score =  0.0
Objective score =  0.5
```

I intentionally picked a contradictory sentence with language that a native English speaker would immediately understand the sentiment of, but a text processor may not grasp that "hurts so good" means that this is a type of pain that one enjoys. SentiWordNet gave spicy a high negative score which is interesting as spicy is a subjective term for many. Overall, SentiWordNet is a useful tool for assessing sentiment in a large corpus of text but for sentences like these with ambiguity and idioms, it may not be entirely accurate.

A collocation is 2+ words that appear so often together that they can be considered a unique phrase that will not hold the same meaning if a word is replaced. Words in a collocation are typically recognized because they are combined more than probabilistically expected.

In [28]:
```
from nltk.book import *
text4
```

Out[28]:
```
<Text: Inaugural Address Corpus>
```

In [30]:
```
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 [32]:
```
# calculate mutual information
text = ' '.join(text4.tokens)
text[:50]
```

Out[32]:
```
'Fellow - Citizens of the Senate and of the House o'
```

In [42]:
```
import math
vocab = len(set(text6))
fc = text.count('Fellow Citizens')/vocab
print("p(Fellow Citizens) = ", fc)
f = text.count('Fellow')/vocab
print("p(Fellow) = ", f)
c = text.count('Citizens')/vocab
print("p(Citizens) = ", c)
pmi = math.log2(fc / (f * c))
print("pmi = ", pmi)
```

```
p(Fellow Citizens) =  0.0004616805170821791
p(Fellow) =  0.0110803324099723
p(Citizens) =  0.003231763619575254
pmi =  3.688500104829567
```

It can be observed that in text4 of nltk, 'Fellow Citizens' has a relatively mutual information score. Mutual information is calculated by the log of the probability of P(x,y)/P(x) * P(y) where P is the probability of the text appearing in the vocabulary of the corpus.