Homework 3 Supraj Punnam CS 4395.001

WordNet is a large lexical database of the english language. Nouns, adjectives, verbs, and adverbs are grouped into synsets each of which represents a different concept. These synsets are in a hierarchical organization.

```
import nltk
import math
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('stopwords')
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
```

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

```
# get all synsets of 'snake', my chosen noun

wn.synsets('cloud')
```

```
[Synset('cloud.n.01'),
 Synset('cloud.n.02'),
 Synset('cloud.n.03'),
 Synset('cloud.n.04'),
 Synset('cloud.n.05'),
 Synset('swarm.n.02'),
 Synset('overcast.v.01'),
 Synset('obscure.v.01'),
 Synset('cloud.v.03'),
 Synset('cloud.v.04'),
 Synset('defile.v.01'),
 Synset('cloud.v.06'),
 Synset('mottle.v.02'),
 Synset('cloud.v.08')]
```

```
# get a definition for the third noun synset

wn.synset('cloud.n.03').definition()
```

```
'out of touch with reality'
```

```
# get usage examples

wn.synset('cloud.n.03').examples()
```

```
['his head was in the clouds']
```

```
# get lemmas

wn.synset('cloud.n.03').lemmas()
```

```
[Lemma('cloud.n.03.cloud')]
```

```
# traverse up hierarchy for chosen synset
cloud = wn.synset('cloud.n.03')
hyper = lambda s: s.hypernyms()
list(cloud.closure(hyper))
```

```
[Synset('unreality.n.02'),
 Synset('nonexistence.n.01'),
 Synset('nonbeing.n.01'),
 Synset('state.n.02'),
 Synset('attribute.n.02'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

As you traverse up the hierarchy for nouns, the words start to get more and more conceptual and broad. The starting noun, 'cloud', was a term that wasn't really that conceptual or broad. As I traversed through the hierarchy, I saw words such as 'nonexistence', 'state', and 'entity' which all were more conseptual terms that encompassed the terms before it.

```
# get hypernyms, hyponyms, meronyms, holonyms, and antonyms if they exist
print('hypernyms: ', cloud.hypernyms())
print('hyponyms: ', cloud.hyponyms())
print('meronyms: ', cloud.part_meronyms())
print('holonyms: ', cloud.part_holonyms())
print('antonyms: ', cloud.lemmas()[0].antonyms())
```

```
hypernyms:  [Synset('unreality.n.02')]
hyponyms:  []
meronyms:  []
holonyms:  []
antonyms:  []
```

```
# get all synsets of 'dance', my chosen verb

wn.synsets('dance')
```

```
[Synset('dance.n.01'),
 Synset('dance.n.02'),
 Synset('dancing.n.01'),
 Synset('dance.n.04'),
 Synset('dance.v.01'),
 Synset('dance.v.02'),
 Synset('dance.v.03')]
```

```
# get a definition for the first vowel synset

wn.synset('dance.v.01').definition()
```

```
'move in a graceful and rhythmical way'
```

```
# get usage examples

wn.synset('dance.v.01').examples()
```

```
['The young girl danced into the room']
```

```
# get lemmas

wn.synset('dance.v.01').lemmas()
```

```
[Lemma('dance.v.01.dance')]
```

```
dance = wn.synset('dance.v.01')
hyper = lambda s: s.hypernyms()
list(dance.closure(hyper))
```

```
[Synset('move.v.03')]
```

The hierarchy for verbs seems to be less connected than the one for nouns. This can be seen from the fact that there is no top level word connecting all verbs like 'entity' does for nouns. For my verb, 'dance', there is only one word above it in the hierarchy. The verb 'move' doesn't serve as a connection to all other verbs like 'entity' does for nouns.

```
wn.morphy('danced', wn.VERB)
```

```
'dance'
```

```
wn.morphy('dancing', wn.VERB)
```

```
'dance'
```

```
wn.morphy('danced')
```

```
'dance'
```

```
# get all synsets of 'triangle', my first chosen word

wn.synsets('triangle')
```

```
[Synset('triangle.n.01'),
 Synset('triangle.n.02'),
 Synset('triangulum.n.01'),
 Synset('triangle.n.04'),
 Synset('triangle.n.05')]
```

```
# get all synsets of 'circle', my second chosen verb

wn.synsets('circle')
```

```
[Synset('circle.n.01'),
 Synset('set.n.05'),
 Synset('circle.n.03'),
 Synset('lap.n.05'),
 Synset('traffic_circle.n.01'),
 Synset('r-2.n.01'),
 Synset('circle.n.07'),
 Synset('circle.n.08'),
 Synset('circle.v.01'),
 Synset('circle.v.02'),
 Synset('encircle.v.01')]
```

```
wn.synset('triangle.n.01').definition()
```

```
'a three-sided polygon'
```

```
wn.synset('circle.n.01').definition()
```

```
'ellipse in which the two axes are of equal length; a plane curve generated by one point moving at a constant distance from
a fixed point'
```

```
# choose the synsets I want to work with for the similarities
triangle = wn.synset('triangle.n.01')
circle = wn.synset('circle.n.01')
```

```
# Wu-Palmer Similarity Metric
wn.wup_similarity(triangle, circle)
```

```
0.7058823529411765
```

```
# Lesk algorithm
sent = ['There', 'are', 'many', 'different', 'kinds', 'of', 'shapes', '.']
print(lesk(sent, 'triangle'))
print(lesk(sent, 'circle'))
```

```
Synset('triangle.n.05')
Synset('circle.n.01')
```

```
# get the definition for the synset of triangle we don't know yet
wn.synset('triangle.n.05').definition()
```

```
'a percussion instrument consisting of a metal bar bent in the shape of an open triangle'
```

It makes sense that the nouns 'triangle' and 'circle' are very similar to each other as they are both shapes. The word 'shape' would probabley serve as their common ancestor node. I thought it was weird though that, with the Lesk Algorithm, the synset returned for triangle was not one related to it being a shape. It kind of makes sense with the way the algorithm works though. The defintion for the instrument triangle includes the word 'shape' which is also inlcuded in the sentence.

SentiWordNet is built on top of WordNet. It assigns 3 separate scores to each word: positive, negative, and objective. It has many possible real world use cases. For example, one way that it can be used is as part of a comment filtering system. A system can use SentiWordNet to filter overly toxic or negative comments. Another use is that it would be possible for a company or users to analyze feelings through product reviews without having to actually read through all of the reviews. Essentially, it can be used whenever knowing the senitments behind text would be useful.

```
# get senti-synsets for the word 'kill'
senti_list = list(swn.senti_synsets('kill'))
for item in senti_list:
    print(item)
```

```
<killing.n.02: PosScore=0.0 NegScore=0.0>
<kill.n.02: PosScore=0.0 NegScore=0.0>
<kill.v.01: PosScore=0.0 NegScore=0.5>
<kill.v.02: PosScore=0.0 NegScore=0.0>
<stamp_out.v.01: PosScore=0.0 NegScore=0.0>
<kill.v.04: PosScore=0.0 NegScore=0.125>
<kill.v.05: PosScore=0.25 NegScore=0.375>
<kill.v.06: PosScore=0.75 NegScore=0.125>
<kill.v.07: PosScore=0.0 NegScore=0.0>
<kill.v.08: PosScore=0.0 NegScore=0.0>
<kill.v.09: PosScore=0.0 NegScore=0.0>
<kill.v.10: PosScore=0.0 NegScore=0.0>
<toss_off.v.02: PosScore=0.375 NegScore=0.0>
<kill.v.12: PosScore=0.0 NegScore=0.0>
<kill.v.13: PosScore=0.25 NegScore=0.75>
<kill.v.14: PosScore=0.0 NegScore=0.0>
<kill.v.15: PosScore=0.5 NegScore=0.25>
```

```
# output polarity for each word in the following sentence
sent = 'the food was bland and cold'
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("\n" + token + ":\tNeg = " + str(syn.neg_score()) + "\tPos = " + str(syn.pos_score()))

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

```

food:   Neg = 0.0       Pos = 0.0

was:    Neg = 0.0       Pos = 0.0

bland:  Neg = 0.375     Pos = 0.0

cold:   Neg = 0.125     Pos = 0.0

Neg     Pos counts
0.5      0.0
```

It makes sense that words such as 'food' and and 'was' are neutral because they aren't typically emotionally charged words. I thought it was interesting how 'bland'was so much more negative than 'cold'. It makes sense after thinking about it though as 'cold' has some neutral applications such as when talking about temperature. I also thought it was interesting that not all applications of the word 'kill' are seen as negative.

Collocations are combinations of two words that combine more often than expected based on random chance. The meaning of the phrase is usually attached to the two words specifically so substituting synonyms changes the meaning of the collocation.

```
# check text4
text4
```

```
    <Text: Inaugural Address Corpus>
```

```
# get collocations for text4
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
```

```
# calculate mutual information on the selected collocation, 'fellow citizens'
text = ' '.join(text4.tokens)
text[:50]
vocab = len(set(text4))
hg = text.count('fellow citizens')/vocab
print("p(fellow citizens) = ",hg )
h = text.count('fellow')/vocab
print("p(fellow) = ", h)
g = text.count('citizens')/vocab
print('p(citizens) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
    p(fellow citizens) =  0.006084788029925187
    p(fellow) =  0.013665835411471322
    p(citizens) =  0.026932668329177057
    pmi =  4.0472042737811735
```

The result of the mutual information formula is in line with what we already know. We know that 'fellow citizens' is a collocation and this is supported by the fact that the mutual information formula gave us a number greater than 0. A result greater than 0 means that the two words show up next to each other more often than expected based on the probability of occurrence of each word.

✓ 0s completed at 6:29 PM