Portfolio Component 3: WordNet

Wyatt Kirby (IRK180000)

CS 4395.001

WordNet was created to prove the theory that people organize concepts in a mental hierarchy. It stores a miriad of words, a short definition, and examples of how the words are used.

```
#Importing what is needed
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
import re
import math
```

Noun

```
#Getting all of the synsets for cake
wn.synsets('cake')
```

```
[Synset('cake.n.01'),
 Synset('patty.n.01'),
 Synset('cake.n.03'),
 Synset('coat.v.03')]
```

Getting the definition, examples of use, and lemmas of coat.

```
#Getting information on one of the syset (attic)
wn.synset('coat.v.03').definition()
```

```
'form a coat over'
```

```
wn.synset('coat.v.03').examples()
```

```
['Dirt had coated her face']
```

```
wn.synset('coat.v.03').lemmas()
```

```
[Lemma('coat.v.03.coat'), Lemma('coat.v.03.cake')]
```

Now I'm traversing the heirarchy as high as I can go using the synset "coat".

```
slather = wn.synset('coat.v.03')
up = lambda x: x.hypernyms()
list(slather.closure(up))
```

```
[Synset('cover.v.02'), Synset('touch.v.05')]
```

It seems that for nouns, the higher up they are in the heirarchy, the broader the definition of the word is. We went from a fairly specific definition to a very broad definition the further up the tree we went. When you coat something, you do touch it but it specifies *how* an object is being touched with another.

Getting the hypernyms, hyponyms, meronyms, holonyms, and antonym of coat.

```
wn.synset('coat.v.03').hypernyms()
```

```
[Synset('cover.v.02')]
```

```
wn.synset('coat.v.03').hyponyms()
```

```
[]
```

```
wn.synset('coat.v.03').part_meronyms()

     []
```

```
wn.synset('coat.v.03').part_holonyms()

     []
```

```
wn.synset('coat.v.03').lemmas()[0].antonyms()

     []
```

It seems like the noun I picked was a little boring :(

Verb

```
#Getting all the sysnets for confuse
wn.synsets('confuse')

     [Synset('confuse.v.01'),
      Synset('confuse.v.02'),
      Synset('confuse.v.03'),
      Synset('jumble.v.02'),
      Synset('confuse.v.05')]
```

Getting the definition, examples of use, and lemmas of jumble.

```
wn.synset('jumble.v.02').definition()

     'assemble without order or sense'
```

```
wn.synset('jumble.v.02').examples()

     ['She jumbles the words when she is supposed to write a sentence']
```

```
wn.synset('jumble.v.02').lemmas()

     [Lemma('jumble.v.02.jumble'),
      Lemma('jumble.v.02.confuse'),
      Lemma('jumble.v.02.mix_up')]
```

Now I'm traversing the heirarchy as high as I can go using jumble.

```
baffle = wn.synset('jumble.v.02')
up = lambda x: x.hypernyms()
list(baffle.closure(up))

     [Synset('assemble.v.01'),
      Synset('join.v.02'),
      Synset('make.v.03'),
      Synset('connect.v.01')]
```

It seems like it performed similarly to how traversing up their hierarchy did for nouns. The words given keep becoming more and more generalized the higher up the tree we get.

Morphy

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

     'confuse'
```

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

```
     'confuse'
```

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

```
     'confuse'
```

Two Similar Words

For my two words I chose bare and bear as they sound similarly but have different meaning. I'm thinking that the similarity will be low.

```
bare = wn.synset('bare.v.01')
bear = wn.synset('bear.n.01')
print(bare.path_similarity(bear))
```

```
     0.0625
```

Now for Wu-Palmer

```
wn.wup_similarity(bare,bear)
```

```
     0.11764705882352941
```

Lusk Algorithm time

```
s1 = ['I', 'have', 'seen', 'a', 'bear', 'in ', 'the', 'yard','!']
s2 = ['Sharol', 'try', 'and', 'bare', 'with', 'it', '.']
s3 = ['Jeff', 'has', 'bare', 'arms', '.']
s4 = ['Sharol', 'has', 'bear', 'arms', '.']
print(lesk(s1, 'bear'))
```

```
     Synset('wear.v.02')
```

```
print(lesk(s2, 'bare'))
```

```
     Synset('denude.v.01')
```

```
print(lesk(s3, 'bare'))
```

```
     Synset('denude.v.01')
```

```
print(lesk(s4, 'bear'))
```

```
     Synset('yield.v.10')
```

I think the words I picked weren't the best suited for this algorithm so I'm going to try again with another set of words.

```
s1 = ['I', 'burned', 'my', 'toast', '.']
s2 = ['John', 'saw', 'the', 'stars', 'last', 'night', '.']
print(lesk(s1, 'burned'))
```

```
     Synset('sunburn.v.01')
```

```
print(lesk(s2, 'stars'))
```

```
     Synset('star.n.03')
```

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

```
     'any celestial body visible (as a point of light) from the Earth at night'
```

I got slightly better outputs, but I wonder why it picked the words it picked.

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

```
     'lay bare'
```

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

```
'lay bare'
```

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

```
'massive plantigrade carnivorous or omnivorous mammals with long shaggy coats and
strong claws'
```

```
wn.synset('wear.v.02').definition()
```

```
'have on one's person'
```

```
wn.synset('yield.v.10').definition()
```

```
'bring in'
```

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

```
'destroy by fire'
```

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

```
'get a sunburn by overexposure to the sun'
```

It seems like when it tries the best it can to find words with similar meanings. The sentences I used were tricky for the algorithm, but it managed to find a couple synonyms for some of the words or just words that are related were chosen. Still very impressive.

SentiWordNet

SentiWordNet functions similarly to WordNet, but gives words scores based on how positive, negative, or objective the word is. This could be used to sift through articles to see if they are objective or use emotional language and show bias in the article (could help spot propoganda or fake news). It could also be used for teaching people languages, ensuring that people aren't accidentally using words that have the undesirable connotations in the language.

```
def sents(s):
    tokens = s.split()
    for t in tokens:
      slist = list(swn.senti_synsets(t))
      if slist:
        print(t)
        print("Positive Score: ", slist[0].pos_score())
        print("Negative Score: ", slist[0].neg_score())
        print("Objective Score: ", slist[0].obj_score())
```

```
sents('awful')
```

```
awful
Positive Score:  0.0
Negative Score:  0.875
Objective Score:  0.125
```

```
sent = "I loved that movie"
sents(sent)
```

```
I
Positive Score:  0.0
Negative Score:  0.0
Objective Score:  1.0
loved
Positive Score:  0.5
Negative Score:  0.0
Objective Score:  0.5
movie
Positive Score:  0.0
Negative Score:  0.0
Objective Score:  1.0
```

It seems like SentiWordNet is fairly good at catagorizing words as long as it has the exact definition that an individual wants to use. For 'awful' it seems to have used version 1 of the definition of awful instead of the second definition (which had a score of 62.5% negative). This would be very useful so that a chat bot can distinguish what a person is asking for or needs from a given interaction. If a user seems to be using a lot of negative words, the chatbot could alter its word choice to better suit the users mental state. It could be used to make sure that a given sentence makes sense and isn't giving mixed messages before sending to a user.

Collocation

Collocation is when two words come together to create a specific meaning. This means that a synonyms of the words couldn't be substituted for one of the words (or both) and still have the same meaning. Examples of this include "big sister" as big couldn't be substituted for large and have the same meaning.

```
text4.collocations()
sText4 = ' '.join(text4.tokens)
```

```
    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
```

```
def prob(word):
  nums = text4.count(word)
  num = nums/149797
  return num
```

```
def prob1(word):
  #I used the text$ that had been joined together into a string because its simpler to find a phrase in a string than it is in a list where e
  #word is seperated
  nums = sText4.count(word)
  num = nums/149796
  return num
```

```
def wordLink(phrase):
  words = phrase.split()
  num = (prob1(phrase)) / ((prob(words[0]) * prob(words[1])))
  num = math.log(num, 2)
  return num
```

I used the examples from the textbook so I can verify the PMI

```
print("PMI is: ", wordLink('fellow citizens'))
```

```
    PMI is:  8.144417647428305
```

```
print("PMI is: ", wordLink('the citizens'))
```

```
    PMI is:  -0.4828390389043306
```

My methods seem to be working as they produced the same numbers that were present in the text book. For the first one, it shows that there is a highlikelihood that 'fellow citizens' is a collocation as the probability of them appearing together as opposed to the probability of them appearing seperately shows that they are *very* likely to be used together. The opposite is true for 'the citizens', as the -.48 suggests that the words were more likely to appear seperately than together in the text.

Other examples:

```
print("PMI is: ", wordLink('Chief Magistrate'))
```

```
    PMI is:  12.247800390745615
```

```
print("PMI is: ", wordLink('years ago'))
```

```
    PMI is:  9.328243383300935
```

"Chief Magistrate" seems to be a collocation as the PMI is *very* high. This means that the two words are far more likely to be used together, meaning its likely that those two words together have a specific meaning. Oddly enough, 'years ago' has a high PMI score despite not being a true collocation. You could substitute ago with before and get the same meaning, but it is still more common to say years ago as opposed to years before. This would cause the PMI to be high despite not being a true collocation.

✓ 0s completed at 9:15 PM