CS 4395.001 - Assignment 4: Wordnet

Michael Menjivar

1. WordNet is a tool often used in Natural Language Processing (NLP). It is a database of words grouped together using sets of synonyms (synsets). WordNet can be used to accomplish many tasks, such as Word Sense Disambiguation (WSD), sentiment analysis, collocations, and more.

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
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
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
from nltk.probability import FreqDist
```

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

2. Select noun and output all sysnsets.

```
wn.synsets('book')
```

```
[Synset('book.n.01'),
 Synset('book.n.02'),
 Synset('record.n.05'),
 Synset('script.n.01'),
 Synset('ledger.n.01'),
 Synset('book.n.06'),
 Synset('book.n.07'),
 Synset('koran.n.01'),
 Synset('bible.n.01'),
 Synset('book.n.10'),
 Synset('book.n.11'),
 Synset('book.v.01'),
 Synset('reserve.v.04'),
 Synset('book.v.03'),
 Synset('book.v.04')]
```

3. Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas. From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting thesynsets as you go. Write a couple of sentences observing the way that WordNet is organized for nouns.

For nouns, when traversing up through the hierarchy using hypernyms, the nouns become less and less specific. The nouns become more general and conceptual compared to the specific physical object the traversal began with (in this case, a book).

```
print(wn.synset('book.n.02').definition())
print(wn.synset('book.n.02').examples())
print(str(wn.synset('book.n.02').lemmas()) + "\n")

book = wn.synset('book.n.02')
book_hypernyms = book.hypernyms()[0]
book_top = wn.synset('entity.n.01')
```

```
while book_hypernyms:
  print(book_hypernyms)
  if book_hypernyms == book_top:
    break
  if book_hypernyms.hypernyms():
    book_hypernyms = book_hypernyms.hypernyms()[0]
```

```
physical objects consisting of a number of pages bound together
['he used a large book as a doorstop']
[Lemma('book.n.02.book'), Lemma('book.n.02.volume')]

Synset('product.n.02')
Synset('creation.n.02')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

4. Output the following (or an empty list if none exist): hypernyms, hyponyms, meronyms, holonyms, antonym.

```
print(wn.synset('book.n.02').hypernyms())
print(wn.synset('book.n.02').hyponyms())
print(wn.synset('book.n.02').part_meronyms())
print(wn.synset('book.n.02').part_holonyms())
```

```
[Synset('product.n.02')]
[Synset('album.n.02'), Synset('coffee-table_book.n.01'), Synset('folio.n.03'), Synset
[Synset('binding.n.05'), Synset('fore_edge.n.01'), Synset('spine.n.04')]
[]
```

5. Select a verb. Output all synsets

```
wn.synsets('flop')
```

```
[Synset('floating-point_operation.n.01'),
 Synset('flop.n.02'),
 Synset('flop.n.03'),
 Synset('flop.n.04'),
 Synset('flop.v.01'),
 Synset('flop.v.02'),
 Synset('fall_through.v.01'),
 Synset('flop.r.01'),
 Synset('right.r.03')]
```

6. Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas. From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go. Write a couple of sentences observing the way that WordNet is organized for verbs.

For the verb chosen (flop), when travelling up the hierarchy the hypernyms aren't as clearly related they are for the nouns. It makes sense that there is no top level verb like there is for nouns, as seen by the short hierarchy tree height. The logical progression from "flop" to "descend" to "travel" makes it seem as if the a verb's hypernyms are only tangentially related.

```
print(wn.synset('flop.v.02').definition())
print(wn.synset('flop.v.02').examples())
print(str(wn.synset('flop.v.02').lemmas()) + "\n")

flop = wn.synset('flop.v.02')
print(flop.hypernyms()[0])
print(flop.hypernyms()[0].hypernyms()[0])
print(flop.hypernyms()[0].hypernyms()[0])
```

```
     fall suddenly and abruptly
     []
     [Lemma('flop.v.02.flop')]

     Synset('descend.v.01')
     Synset('travel.v.01')
     Synset('travel.v.01')
```

7. Use morphy to find as many different forms of the word as you can.

```
print(wn.morphy('slayed', wn.VERB))
```

```
     slay
```

8. Select two words that you think might be similar. Find the specific synsets you are interested in. Run the Wu-Palmer similarity metric and the Lesk algorithm. Write a couple of sentences with

metric and the Lesk algorithm. Write a couple of sentences with your observations.

I think it's interesting how 'slay' and 'assassinate', which both are fancy words for 'kill', are only given a similarity rating of 0.3333 when using the Wu-Palmer similarity metric. However, given that they are verbs and the way that the hypernym hierarchy tree is set up for WordNet, it isn't too surprising. As for the Lesk algorithm, it's interesting how for the word 'slay', the synset returned was for the word 'murder', but for 'assassinate' the synset returned was also for the word 'assassinate'.

```
from nltk.wsd import lesk
slay = wn.synset('slay.v.01')
assassinate = wn.synset('assassinate.v.01')
print(wn.wup_similarity(slay, assassinate))
sent_1 = ["I", "will", "slay", "the", "king", "in", "his", "sleep", "."]
sent_2 = ["The", "attempt", "to", "assassinate", "the", "president", "was", "unsuccessful",
print(lesk(sent_1, 'slay', 'v'))
print(lesk(sent_2, 'assassinate','v'))
```

```
0.3333333333333333
Synset('murder.v.01')
Synset('assassinate.v.02')
```

9. Write a couple of sentences about SentiWordNet, describing its functionality and possible use cases. Select an emotionally charged word. Find its senti-synsets and output the polarity scores for each word. Make up a sentence. Output the polarity for each word in the sentence. Write a couple of sentences about your observations of the scores and the utility of knowing these scores in an NLP application.

SentiWordNet is a tool used for analyzing the overall sentiment of a user's messages, whether they be reviews or texts. It's possible use cases include analyzing feelings through product reviews or searching for and filtering out toxic comments under videos.

It's interesting how the score for the word 'embarrassment' was more positive than negative. I'm not sure why that's the case since for me personally, the word 'embarrassment' feels clearly negative. As for the sentence, seeing that most of the words in said sentence got a mostly positive score is useful since the sentence overall evokes a feeling of disappointment (in my opinion), since I wrote it). For a more straightforward sentence, I'm sure the scores for each individual word would match up with the overall sentiment of said sentence. This

sure the scores for each individual word would match up with the overall sentiment of said sentence. This could be useful for quickly analyzing comments to see if there are any words that are overly negative (toxic) and potentially filter them out or prevent users from posting said comment.

```
# print(swn.senti_synset('embarrassment.n.01'))
embarrassment = swn.senti_synset('embarrassment.n.01')
print(embarrassment)
print("Positive score = ", embarrassment.pos_score())
print("Negative score = ", embarrassment.neg_score())
print("Objective score = ", embarrassment.obj_score())

sent = 'honestly the food was okay but it could definitely have been better'
n_score = 0
p_score = 0
tokens = sent.split()
for token in tokens:
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    n_score = syn.neg_score()
    p_score = syn.pos_score()
    print("\n" + token + ":\tNeg = " + str(syn.neg_score()) + "\tPos = " + str(syn.pos_scor
```

```
 <embarrassment.n.01: PosScore=0.25 NegScore=0.125>
 Positive score =  0.25
 Negative score =  0.125
 Objective score =  0.625

 honestly:       Neg = 0.0       Pos = 0.375

 food:   Neg = 0.0       Pos = 0.0

 was:    Neg = 0.0       Pos = 0.0

 okay:   Neg = 0.125     Pos = 0.125

 but:    Neg = 0.0       Pos = 0.0

 it:     Neg = 0.0       Pos = 0.0

 definitely:     Neg = 0.0       Pos = 0.25

 have:   Neg = 0.0       Pos = 0.0

 been:   Neg = 0.125     Pos = 0.25

 better: Neg = 0.0       Pos = 0.5
```

10. Write a couple of sentences about what a collocation is. Output

collocations for text4, the Inaugural corpus. Select one of the collocations identified by NLTK. Calculate mutual information. Write commentary on the results of the mutual information formula and your interpretation.

A collocation is when a group of specific words are used together to form some meaning, and none of the words in the collocation can be changed without changing the meaning. For example, interchanging the word 'tired' with the word 'exhausted' in the collocation 'sick and tired' to form 'sick and exhausted' changes the meaning of the phrase. The original phrase means being overwhelmed and upset about certain circumstances, but the modified phrase simply means being physically sick and exhausted.

My interpretation for the results of calculating the PMI score for the phrase 'Almighty God' is that this phrase is most likely a collocation. If the PMI score is a positive number, it means that the phrase is more likely to be a collocation based solely on a given corpus and the formula used to predict whether said phrase is a collocation or not. If the score is negative, it is less likely to be a collocation. Based on the fact that the phrase got a score of approximately 9.5 (well above zero), it is very likely to be a collocation.

```
import math
text4.collocations()

fdist = FreqDist(text4)

print("\nTotal tokens in text4 = " + str(fdist.N()))
print("Appearances of word 'Almighty' in text4 = " + str(fdist['Almighty']))
print("Appearances of word 'God' in text4 = " + str(fdist['God']))
print("Appearances of 'Almighty God' in text4 = " + str(len(text4.concordance_list(['Almigh

x = fdist['Almighty']
y = fdist['God']
xy = len(text4.concordance_list(['Almighty','God']))
totalTokens = fdist.N()
p_x = x / totalTokens
p_y = y / totalTokens
p_xy = xy / totalTokens
pmi = math.log2((p_xy)/(p_x*p_y))

print("\nPMI Score for the phrase 'Almighty God' = " + str(pmi))
```

```
     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

     Total tokens in text4 = 152901
```

```
Appearances of word 'Almighty' in text4 = 28
Appearances of word 'God' in text4 = 111
Appearances of 'Almighty God' in text4 = 15

PMI Score for the phrase 'Almighty God' = 9.527358123851858
```