Reg Gonzalez | CS 4395.001

Assignment 4: Exploring WordNet and SentiWordNet

Step 1: Summary of WordNet

WordNet is a database that specifically includes a variety of words and parts of specech, including nouns, adjectives, verbs, and adverbs. The WordNet database is specifically used for natural language processing and is able to group words into sets, which are called *synsets*. WordNet has a variety of functions that users can call upon to perform natural language processing, such as .synsets(), which finds the synsets for a particular word; .definition(), which provides the definition of a particular word; .examples(), which gives different use cases of a word; etc.

Step 2: Output synsets of a noun

For this step, I'm outputting the synset of the noun "torture".

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn

wn.synsets('torture')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[Synset('anguish.n.01'),
 Synset('torture.n.02'),
 Synset('agony.n.01'),
 Synset('distortion.n.05'),
 Synset('torture.n.05'),
 Synset('torment.v.01'),
 Synset('torture.v.02')]
```

Step 3: Using .definition(), .examples(), and .lemmas() and traversing the word hierarchy (for a noun)

In Step 3 of this assignment, I first used the .definition(), .examples(), and .lemmas() functions for the word "agony". This was one of the synsets for the word "torture" in the previous step. The functions give the definition of the word, use cases for the word, and a list of lemmas related to that word, respectively.

After that, I traversed the WordNet hierarchy for "agony". One thing to note about this hierarhy is that all the words are nouns, which makes sense because the word we're traversing the hierarhy of is a noun. As you go down the list, you can see that each word is progressively more broad (e.g., "suffering" is broader than "agony", "pain" is broader than "suffering", "feeling" is broader than "pain", etc., etc.). Again, this makes sense because of the hierarhical structure. For nouns, it's important to note that the top hypernym is always "entity", which shouldn't be too surprising since pretty much

```
# Use the .definition(), .examples(), and .lemmas() functions on the word 'agony'—one of the
print('Agony definition: ', wn.synset('agony.n.01').definition())
print("Examples of using the word 'agony': ", wn.synset('agony.n.01').examples())
print("Lemmas of the word 'agony': ", wn.synset('agony.n.01').lemmas())

# Traverse hierarchy of 'agony'
agony = wn.synset('agony.n.01')
hypernym = agony.hypernyms()[0]     # stores the hypernym
top_hyp = wn.synset('entity.n.01')  # 'entity' is the top hypernym

# While there are hypernyms, print them out.
# If you reach the top hypernym, break out of the loop.
# If there are more hypernyms, get them so we can continue to print them out.
print()
print("Traversing the hierarchy of 'agony':")
while hypernym:
  print(hypernym)
  if hypernym == top_hyp:
    break
  if hypernym.hypernyms():
    hypernym = hypernym.hypernyms()[0]
```

```
Agony definition:  intense feelings of suffering; acute mental or physical pain
Examples of using the word 'agony':  ['an agony of doubt', 'the torments of the damned'
Lemmas of the word 'agony':  [Lemma('agony.n.01.agony'), Lemma('agony.n.01.torment'), Le

Traversing the hierarchy of 'agony':
Synset('suffering.n.04')
Synset('pain.n.02')
Synset('feeling.n.01')
Synset('state.n.02')
Synset('attribute.n.02')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

Step 4: Outputting hypernyms, hyponyms, meronyms, holonyms, and antonyms

For Step 4 of this assignment, I printed out the hypernyms, hyponyms, meronym, holonyms, and antonyms of the word "agony"; however, there were only hypernyms available for the word, which is why the other four were not printed out.

```
# Output the hypernyms, hyponyms, meronyms, holonyms, and antonyms of 'agony'
print("Hypernyms of agony: ", wn.synset('agony.n.01').hypernyms())
print("Hyponyms of agony: ", wn.synset('agony.n.01').hyponyms())
print("Meronyms of agony: ", wn.synset('agony.n.01').part_meronyms())
print("Holonyms of agony: ", wn.synset('agony.n.01').part_holonyms())

agony = wn.synsets('agony', pos=wn.NOUN)[0]
print("Antonyms of agony: ", agony.lemmas()[0].antonyms())
```

```
     Hypernyms of agony:  [Synset('suffering.n.04')]
     Hyponyms of agony:  []
     Meronyms of agony:  []
     Holonyms of agony:  []
     Antonyms of agony:  []
```

Step 5: Output synsets of a verb

Here, we are simply outputting the synsets of a verb. The verb I've chosen is "paint" (i.e., to paint something).

```
wn.synsets("paint")
```

```
     [Synset('paint.n.01'),
      Synset('key.n.09'),
      Synset('rouge.n.01'),
      Synset('paint.v.01'),
      Synset('paint.v.02'),
      Synset('paint.v.03'),
      Synset('paint.v.04')]
```

Step 6: Using .definition(), .examples(), and .lemmas() and traversing the word hierarchy (for a verb)

The process here is very similar to what we did in Step 3. We did the same kind of processing/method calling as we did in Step 3 for "agony" by calling the .definition(), .examples(), and .lemmas() methods.

The big difference here is how we traversed through the hierarchy. In part 3, we traversed through the hierarchy by using various if statements—it was a more brute force way of approaching the problem. Here, we used NLTK's closure method, which is a much simpler and more elegant way of traversing the hierarchy. The "hyper" variable gets the hypernyms of a synset.

Similar to the hierarchy in step 3, we can see that the words that come sequentially in the hierarchy are broader than the words prior. In this case however, we only have 2 words in the hierarchy. But still, the pattern remains: "act" is broader than "create". Here, it seems like "act" is the top-level

```
# Use the .definition(), .examples(), and .lemmas() functions on the word 'paint'—one of the
print('Paint definition: ', wn.synset('paint.v.01').definition())
print("Examples of using the word 'paint': ", wn.synset('paint.v.01').examples())
print("Lemmas of the word 'paint': ", wn.synset('paint.v.01').lemmas())
print()

paint = wn.synset('paint.v.01')
hyper = lambda s: s.hypernyms()
print("Traversing the hierarchy of 'paint':", list(paint.closure(hyper)))
```

```
    Paint definition:  make a painting
    Examples of using the word 'paint':  ['he painted all day in the garden', 'He painted a
    Lemmas of the word 'paint':  [Lemma('paint.v.01.paint')]

    Traversing the hierarchy of 'paint': [Synset('create.v.03'), Synset('act.v.01')]
```

Step 7: Using .morphy()

For this very small code block, I used the .morphy() function to get the verb lemma of the verb "painted." I wanted to use "painted" instead of "paint" like I did in the previous couple of steps so that it'd be easier to see the difference between the lemma and the word in the function argument. If I used "paint" as argument instead of "painted", it also would've printed out "paint".

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

```
    paint
```

Step 8: Wu-Palmer similarity metric & the Lesk algorithm

For this step in the assignment, I chose to use the words "octopus" and "serpent". More specifically, I chose to use the synsets "octopus.n.01" and "serpent.n.02". When I applied the Wu-Palmer metric on them, I for a similar of ~0.353, which is pretty much what I expected. Serpents and octopuses look vaguely similar, both can be found in the water, both are animals, etc.

After I used the Wu-Palmer metric, I used the Lesk algorithm, giving it two different sentences for octopus and serprent. The sentence I used for octopus was "I went to the zoo and saw an octopus in the aquarium." and the sentence I used for serpent was "There is a giant serpent hidden beneath the ground." The answer I got for octopus was what I expected, which was "octopus.n.01", the synset I used for the Wu-Palmer metric. However, what surprised me was the answer I got for serpent. Instead of getting "serpent.n.02", I got "serpent.n.03". I checked up on the definitions of serpent.n.02 and serpent.n.03 to see why the Lesk algorithm classified "serpent" as n.03 and not n.02. It turns out that the n.02's definition of the word isn't the *animal*, but instead a word to describe a firework. This means that for the Wu-Palmer metric, I wasn't comparing two animals, but an animal and a firework. Now that puts my similarity metric into a different perspective. I now think the number is high considering we're now finding the similarities between an animal and a firework instead of two animals.

```
# Print out synsets of the words "octopus" and "serpent"
print("Synsets of octopus: ", wn.synsets('octopus'))
print("Synsets of serpent: ", wn.synsets('serpent'))
print()

# Get specific synsets to use for the Wu-Palmer similarity metric
octopus = wn.synset('octopus.n.01')
serpent = wn.synset('serpent.n.02')

print("Wu-Palmer similiarty metric between 'octopus' and 'serpent': ", wn.wup_similarity(octo
print()

# Use the Lesk algorithm for the words "octopus" and "serpent"
from nltk.wsd import lesk

sent_octopus = ['I', 'went', 'to', 'the', 'zoo', 'and', 'saw', 'an', 'octopus', 'in', 'the',
print("Lesk algorithm output for octopus: ", lesk(sent_octopus, 'octopus', 'n'))

sent_serpent = ['There', 'is', 'a', 'giant', 'serpent', 'hidden', 'beneath', 'the', 'ground',
print("Lesk algorithm output for serpent: ", lesk(sent_serpent, 'serpent', 'n'))
print()

# Check for the different definitions of serpent
print("Definition of serpent.n.02: ", wn.synset('serpent.n.02').definition())
print("Definition of serpent.n.03: ", wn.synset('serpent.n.03').definition())
```

```
Synsets of octopus:  [Synset('octopus.n.01'), Synset('octopus.n.02')]
Synsets of serpent:  [Synset('snake.n.01'), Synset('serpent.n.02'), Synset('serpent.n.0

Wu-Palmer similiarty metric between 'octopus' and 'serpent':  0.35294117647058826

Lesk algorithm output for octopus:  Synset('octopus.n.01')
Lesk algorithm output for serpent:  Synset('serpent.n.03')

Definition of serpent.n.02:  a firework that moves in serpentine manner when ignited
Definition of serpent.n.03:  an obsolete bass cornet; resembles a snake
```

Step 9: SentiWordNet

SentiWordNet is a lexical resource that works in conjunction with WordNet in order to perform sentiment analysis. In SentiWordNet, there are three different sentiment scores, that being positive, negative, and objective. You can find the sentiment scores of a particular word or a sentence.

In this section, the emotionally-charged word I chose was "provoke". I'm specifically using the definition of the word that means to elicit emotions out of people. First, I outputted the sentiment/polarity scores for "provoke" by itself and I got a positive score of 0.25, a negative score of 0.25, and an objective score of 0.5. The range of polarity scores is from [0.0, 1.0]. Here, the positive and negative scores are the same, which I suppose makes sense because you can provoke a lot of different kinds of emotions out of someone. It's not all entirely positive or negative.

After I outputted the polarity scores of the word itself, I made up the sentence "Her comments provoked feelings of anger within him, causing him to storm out of the room." I then used a [naive] approach to sentiment analysis, which was just adding up the positive, negative, and objective scores of each synset in the sentence. I also outputted the positive, negative, and objective scores of each synset. In all, the positive score of the sentence was 1.125, the negative score was 0.625, and the objective score was 6.25.

Overall, the sentence was a lot more objective than I thought it'd be. However, by looking at the scores for the individual synsets, we can see why that is. Words like "remark", "storm", "inside", and "out" all had objective scores of 1.0 and positive/negative scores of 0.0. This greatly inflated the objective score. It's also important to point out that some of the words weren't classified correctly. For example, the word "storm" was classified as a noun when I used it as a verb in the sentence. This also impacts the final score. The sentence also had a higher positive score than negative, which I found to be fairly surprising. The culprit of this is that words like "anger" or "feelings" had a higher positive score than negative score, which boosted the overall positive score. This was somewhat of a surprise to me, especially with the word "anger", which normally has a more negative connotation than a positive one.

```
from nltk.corpus import sentiwordnet as swn
nltk.download('sentiwordnet')

# Choose an emotionally-charged word: provoke
provoke = swn.senti_synset('provoke.v.01')

# Find senti-synsets and output the polarity scores
print("Polarity scores for 'provoke': ", provoke)
print("\tPositive score = ", provoke.pos_score())
print("\tNegative score = ", provoke.neg_score())
print("\tObjective score = ", provoke.obj_score())
print()

# Make up a sentence and output the polarity of each word for that sentence.
sentence = "Her comments provoked feelings of anger within him, causing him to storm out of t
positive = 0
negative = 0
objective = 0
tokens = sentence.split()

for t in tokens:
  synsets_list = list(swn.senti_synsets(t))
  if synsets_list:
    synset = synsets_list[0]
    print("Polarity scores for ", synset, ": ", "Pos =", synset.pos_score(), " Neg =", synset
    positive += + synset.pos_score()
    negative += synset.neg_score()
    objective += synset.obj_score()

print()
print("[Sentiment analysis] Positive score: ", positive)
print("[Sentiment analysis] Negative score: ", negative)
print("[Sentiment analysis] Objective score: ", objective)
print()
```

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
Polarity scores for 'provoke':  <arouse.v.01: PosScore=0.25 NegScore=0.25>
        Positive score =  0.25
        Negative score =  0.25
        Objective score =  0.5

Polarity scores for  <remark.n.01: PosScore=0.0 NegScore=0.0> :  Pos = 0.0  Neg = 0.0  (
Polarity scores for  <arouse.v.01: PosScore=0.25 NegScore=0.25> :  Pos = 0.25  Neg = 0.2
Polarity scores for  <feelings.n.01: PosScore=0.5 NegScore=0.125> :  Pos = 0.5  Neg = 0
Polarity scores for  <anger.n.01: PosScore=0.375 NegScore=0.0> :  Pos = 0.375  Neg = 0.6
Polarity scores for  <inside.r.02: PosScore=0.0 NegScore=0.0> :  Pos = 0.0  Neg = 0.0  (
Polarity scores for  <causing.n.01: PosScore=0.0 NegScore=0.25> :  Pos = 0.0  Neg = 0.25
Polarity scores for  <storm.n.01: PosScore=0.0 NegScore=0.0> :  Pos = 0.0  Neg = 0.0  Ob
Polarity scores for  <out.n.01: PosScore=0.0 NegScore=0.0> :  Pos = 0.0  Neg = 0.0  Obj

[Sentiment analysis] Positive score:  1.125
[Sentiment analysis] Negative score:  0.625
[Sentiment analysis] Objective score:  6.25
```

Step 10: Collocations

A collocation is a term that describes multiple words that occur in succession that appear more frequently than chance would suggest. For example, some collocations are "fast food," "pay attention," "saving time," etc. These are all phrases that have multiple words in them that appear more frequently than just by chance.

In this final step of the assignment, I had to output the collocations for text4 and then calculate the PMI score (the point-wise mutual information score). The equation of this score is described as: log2(P(x, y) / (P(x) * P(y))). P(x, y) will represent the collocation phrase we want to calculate (e.g., "foreign nations") divided by the total number of tokens. P(x) will represent the first part of the collocation ("foreign") divided by the total number of tokens. P(y) will represent the second part of the collocation ("nations") divided by the total number of tokens. After doing all the work to calculate the PMI, I got a score of ~6.918. A positive PMI score indicates that the phrase that we're looking at is more than likely a collocation. That is, for the phrase "foreign nations" in text4, it's probably a collocation

```
from nltk.tokenize import word_tokenize
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('stopwords')
from nltk.book import *
from nltk import word_tokenize
from nltk.util import ngrams
nltk.download('punkt')
import pickle
import math

# Output collocations for text4
print()
text4.collocations()
print()

# Create a bigram and a bigram dictionary, then get the count of how many times
# the phrase "foreign nations" appears in the text
tokens_text4 = text4.tokens
bigrams = list(ngrams(tokens_text4, 2))
bigram_dict = {b:bigrams.count(b) for b in set(bigrams)}
foreign_nations_count = bigram_dict[("foreign", "nations")]

# Calculate and display the pmi score of the collocation "foreign nations"
foreign_count = text4.count("foreign")
nations_count = text4.count("nations")
tokens_count = len(tokens_text4)
numerator = foreign_nations_count / tokens_count
denominator = (foreign_count / tokens_count) * (nations_count / tokens_count)

pmi = math.log2(numerator / denominator)
print("PMI score for 'foreign nations' in text4: ", pmi)
```

```
    print()
```

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

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

PMI score for 'foreign nations' in text4:  6.917544759180042
```