Homework4_pbp180000

September 25, 2022

WordNet is a database containing hierarchical relationships between words that are nouns, verbs, adjectives, and adverbs. WordNet can give the user a short definition of a word or even relations to other words like synsets where you can get the set on synonyms and their part of speech.

In [22]:
```
#import library fo wordnet and get synsets of the noun bed
from nltk.corpus import wordnet as wn
wn.synsets("bed")
```

Out[22]:
```
[Synset('bed.n.01'),
 Synset('bed.n.02'),
 Synset('bed.n.03'),
 Synset('bed.n.04'),
 Synset('seam.n.03'),
 Synset('layer.n.01'),
 Synset('bed.n.07'),
 Synset('bed.n.08'),
 Synset('bed.v.01'),
 Synset('bed.v.02'),
 Synset('bed.v.03'),
 Synset('sleep_together.v.01'),
 Synset('go_to_bed.v.01')]
```

In [26]:
```
#print the definition, usage example, and the lemmas of the noun bed
print(wn.synset('bed.n.03').definition())
print(wn.synset('bed.n.03').examples())
print(wn.synset('bed.n.03').lemmas())

#traverse the hierarchy and print out the values in the hierarchy
bed = wn.synset('bed.n.03')
hypernym = bed.hypernyms()[0]
top = wn.synset('entity.n.01')
while hypernym:
    print(hypernym)
    if hypernym == top:
        break
    if hypernym.hypernyms():
        hypernym = hypernym.hypernyms()[0]
```

```
a depression forming the ground under a body of water
['he searched for treasure on the ocean bed']
[Lemma('bed.n.03.bed'), Lemma('bed.n.03.bottom')]
Synset('natural_depression.n.01')
Synset('geological_formation.n.01')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Wordnet is organized very hierarchically for nouns. Since each noun has a hypernym, Wordnet can easily organize these nouns into a hierarchy. We find that the hierarchy generally becomes vague and ends when it hits object, physical_entity, and entity. It is very easy to traverse the hierarchy because we know what the top value ends up being.

```
In [44]: #if bed contains hypernyms, hyponyms, meronyms, holonyms, or even antonyms print them
         # otherwise print an empty list
         if bed.hypernyms():
             print(bed.hypernyms())
         if bed.hyponyms():
             print(bed.hyponyms())
         if bed.part_meronyms():
             print(bed.part_meronyms())
         else:
             print("[]")
         if bed.part_holonyms():
             print(bed.part_holonyms())
         else:
             print("[]")

         antonyms = []
         for syn in wn.synsets("bed.n.03"):
             for i in syn.lemmas():
                  if i.antonyms():
                       antonyms.append(i.antonyms()[0].name())
         print(antonyms)
```

```
[Synset('natural_depression.n.01')]
[Synset('lake_bed.n.01'), Synset('ocean_floor.n.01'), Synset('riverbed.n.01'), Synset('streambe
[]
[]
[]
```

```
In [45]: #get synsets of the verb drive
         wn.synsets("drive")
```

```
Out[45]: [Synset('drive.n.01'),
          Synset('drive.n.02'),
```

```
          Synset('campaign.n.02'),
          Synset('driveway.n.01'),
          Synset('drive.n.05'),
          Synset('drive.n.06'),
          Synset('drive.n.07'),
          Synset('drive.n.08'),
          Synset('drive.n.09'),
          Synset('drive.n.10'),
          Synset('drive.n.11'),
          Synset('drive.n.12'),
          Synset('drive.v.01'),
          Synset('drive.v.02'),
          Synset('drive.v.03'),
          Synset('force.v.06'),
          Synset('drive.v.05'),
          Synset('repel.v.01'),
          Synset('drive.v.07'),
          Synset('drive.v.08'),
          Synset('drive.v.09'),
          Synset('tug.v.02'),
          Synset('drive.v.11'),
          Synset('drive.v.12'),
          Synset('drive.v.13'),
          Synset('drive.v.14'),
          Synset('drive.v.15'),
          Synset('drive.v.16'),
          Synset('drive.v.17'),
          Synset('drive.v.18'),
          Synset('drive.v.19'),
          Synset('drive.v.20'),
          Synset('drive.v.21'),
          Synset('drive.v.22')]
```

```
In [109]: #print the definition, usage example, and the lemmas of the word drive
          print(wn.synset('drive.v.05').definition())
          print(wn.synset('drive.v.05').examples())
          print(wn.synset('drive.v.05').lemmas())

          #traverse the hierarchy and print the values
          drive = wn.synset('drive.v.05')
          hypernym = lambda x: x.hypernyms()
          list(drive.closure(hypernym))
```

```
to compel or force or urge relentlessly or exert coercive pressure on, or motivate strongly
['She is driven by her passion']
[Lemma('drive.v.05.drive')]
```

```
Out[109]: [Synset('coerce.v.01'), Synset('compel.v.01'), Synset('induce.v.02')]
```

Wordnet seems to not have as much of a hierarchy for verbs. When trying to traverse the hierarchy using the same method I used for nouns, I found that the list was infinite and that the value would not change. Using the closure method allowed me to get a hierarchy that was not infinite and seems to be more of a list of synonyms than hierarchy.

In [64]:
```
#get as many forms of drive possible
wn.morphy('drive')
```

Out[64]: 'drive'

In [112]:
```
#import lesk library
from nltk.wsd import lesk

#get the specific synset for tiger and jaguar
tiger = wn.synset('tiger.n.02')
jaguar = wn.synset('jaguar.n.01')

#perform the Wu-Palmer Similarity and the Lesk Algorithm
print(wn.wup_similarity(tiger, jaguar))
print(lesk(['I', 'went', 'to', 'the', 'safari', 'to', 'see', 'the', 'tiger', '.'],'ti
print(lesk(['I', 'went', 'to', 'the', 'zoo', 'to', 'see', 'the', 'jaguar', 'in', 'my
```

```
0.9333333333333333
Synset('tiger.n.02')
Synset('jaguar.n.01')
```

I found that the Wu-Palmer similarity to be accurate because I chose tigers and jaguars. I found that it predicted the similarity as 0.9333 which is accurate because tigers and jaguars, while they are not the same, they are both predatory feline animals. They are also large in size and are both found in the wild generally. Thus the Wu-Palmer similarity seemed to capture that accurately. I also thought the lesk algorithm seemed to accurately predict the specific tiger/jaguar word I was looking for.

In [79]:
```
import nltk
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
```

```
[nltk_data] Downloading package sentiwordnet to
[nltk_data]     C:\Users\Prachi\AppData\Roaming\nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
```

SentiWordNet is built on top of WordNet, but it also assigns a positive, negative,and objective score to the word. This can be used to do sentiment analysis, as we can figure out the sentiment of a word, so we can figure out the sentiment of sentences and even documents. Since words can have multiple meanings, using the context of the sentence, we can figure out the sentiment behind the document.

```
In [90]: #find the senti synsets for the word sob
         sentlist = list(swn.senti_synsets('sob','v'))
         for item in sentlist:
             print(item)
```

```
<sob.v.01: PosScore=0.0 NegScore=0.25>
```

```
In [87]: # get the positive, negative, and objective scores for the word sob.
         sob = swn.senti_synset('sob.v.01')
         print(sob)
         print("Positive score = ", sob.pos_score())
         print("Negative score = ", sob.neg_score())
         print("Objective score = ", sob.obj_score())
```

```
<sob.v.01: PosScore=0.0 NegScore=0.25>
Positive score =  0.0
Negative score =  0.25
Objective score =  0.75
```

```
In [115]: #create a sentence and split into tokens
          sent = 'I love eating pizza with olives but I hate pineapples on my pizza.'

          tokens = sent.split()

          #for each token get the sentiment and print it
          for t in tokens:
              syn = list(swn.senti_synsets(t))
              if syn:
                  syn = syn[0]
                  print(syn)
                  print(syn.neg_score(), ", ",syn.pos_score())
```

```
<iodine.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<love.n.01: PosScore=0.625 NegScore=0.0>
0.0 ,  0.625
<eating.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<pizza.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<olive.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<merely.r.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<iodine.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<hate.n.01: PosScore=0.125 NegScore=0.375>
```

```
0.375 ,  0.125
<pineapple.n.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
<on.a.01: PosScore=0.0 NegScore=0.0>
0.0 ,  0.0
neg        pos
0.375          0.75
```

I found that when I found the positive, objective, and negative scores of the singular word, the score was more objective than negative I thought it would be. It was accurate in that the word was not given a positive score, but I thought the negative score was lower than it should have been. I felt that when I looked at the sentiment of the sentence, the combinations of all the words allowed the total positive and negative score to be more accurate. In NLP, knowing these scores would allow for us to be able to use these sentiments to figure out maybe whether a customer is happy with a service/product. Or even one could figure out public sentiment towards a certain political figure during election season. The use of these scores would allow for an understanding for the computer in order to make further computations using these sentiment scores.

```
In [93]: import nltk.book
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

A collocation is a pair or group of words that are usually seen together. An example of a collocation would be something like, “For example”, or even “right now”. The examples I gave only have a pair of words, but a collocation is something you would see grouped together more often than not.

```
In [100]: #get the text 4 collocations
          text4 = nltk.book.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
```

```
In [108]: #import libraries and get vocab length, while creating a text string
          import math
          vocablen = len(set(text4))
          text = ' '.join(text4.tokens)

          # get the percentage of times the phrase/word appears in the text and print the resu
          fn = text.count('foreign nations')/vocablen
          print("p(foreign nations) = ",fn )
          f = text.count('foreign')/vocablen
          print("p(foreign) = ", f)
          n = text.count('nations')/vocablen
          print('p(nations) = ', n)

          #calculate the mutual and print it
          calc = (fn/(f*n))
          mutual = math.log2(calc)
          print('Mutual = ', mutual)
```

```
p(foreign nations) =  0.0014962593516209476
p(foreign) =  0.01027431421446384
p(nations) =  0.020448877805486283
Mutual =  2.8322245851494996
```

I felt that the mutal score was high at a 2.83. I also thought that the score was accurate, because foreign nations is a collocation that is used not just in the Inaugural Address, but in many political documents. My interpretation was that the mutual score was able to accurately say whether or not a collocation was actually a collocation

```
In [ ]:
```