WordNet Notebook

Richard Padilla | Dr. Mazidi | CS 6301.M02 Spring 2023

1. Give a summary of WordNet

WordNet is a large lexical database. It is organized using synsets or synonym sets to group words based on their semantic meaning. WordNet also tags words with their part of speech, provides definitions, examples, and lemmas for the words in the database. WordNet also categorizes its words into hypernyms, holonyms, meronyms, hyponyms, and antonyms. WordNet is very useful for NLP processing and analysis.

1. Select a noun. Output all synsets.

In [ ]:
```
from nltk.corpus import wordnet as wn

wn.synsets("brass")
```

Out[ ]:
```
[Synset('brass.n.01'),
 Synset('brass.n.02'),
 Synset('administration.n.02'),
 Synset('boldness.n.02'),
 Synset('brass.n.05'),
 Synset('brass_section.n.01'),
 Synset('brass.n.07')]
```

1. Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas.

In [ ]:
```
syn_noun = "brass.n.02"
```

In [ ]:
```
wn.synset(syn_noun).definition()
```

Out[ ]:
```
'a wind instrument that consists of a brass tube (usually of variable length)
that is blown by means of a cup-shaped or funnel-shaped mouthpiece'
```

In [ ]:
```
wn.synset(syn_noun).examples()
```

Out[ ]:
```
[]
```

In [ ]:
```
wn.synset(syn_noun).lemmas()
```

Out[ ]:
```
[Lemma('brass.n.02.brass'), Lemma('brass.n.02.brass_instrument')]
```

From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

In [ ]:
```
# From your selected synset, traverse up the WordNet hierarchy as far as you ca
# synsets as you go.

brass = wn.synset(syn_noun)
hyp = brass.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('wind_instrument.n.01')
Synset('musical_instrument.n.01')
Synset('device.n.01')
Synset('instrumentality.n.03')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Write a couple of sentences observing the way that WordNet is organized for nouns.

WordNet categorizes nouns really well! It appears to categorize nouns in a heirarchical manner from the most specific instance of the noun up to the most abstract version, an entity. I did not think of 'artifact' or 'physical entity' when I think of a brass instrument, but it is indeed both of these.

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

In [ ]:
```
brass.hypernyms()
```

Out[ ]:
```
[Synset('wind_instrument.n.01')]
```

In [ ]:
```
brass.hyponyms()
```

Out[ ]:
```
[Synset('baritone.n.03'),
 Synset('bass_horn.n.01'),
 Synset('bugle.n.01'),
 Synset('clarion.n.01'),
 Synset('cornet.n.01'),
 Synset('flugelhorn.n.01'),
 Synset('french_horn.n.01'),
 Synset('saxhorn.n.01'),
 Synset('trombone.n.01')]
```

In [ ]:
```
brass.part_meronyms()
```

Out[ ]:
```
[Synset('valve.n.02')]
```

In [ ]:
```
brass.part_holonyms()
```

Out[ ]:
```
[]
```

In [ ]:
```
brass.lemmas()[0].antonyms()
```

Out[ ]:
```
[]
```

1. Select a verb. Output all synsets

In [ ]:
```
wn.synsets("coding")
```

Out[ ]:
```
[Synset('cryptography.n.02'),
 Synset('code.v.01'),
 Synset('code.v.02'),
 Synset('gull.v.02'),
 Synset('tease.v.02')]
```

1. Select one synset from the list of synsets.

In [ ]:
```
syn_verb = "code.v.02"
```

Extract its definition, usage examples, and lemmas.

In [ ]:
```
wn.synset(syn_verb).definition()
```

Out[ ]:
```
'convert ordinary language into code'
```

In [ ]:
```
wn.synset(syn_verb).examples()
```

Out[ ]:
```
['We should encode the message for security reasons']
```

In [ ]:
```
wn.synset(syn_verb).lemmas()
```

Out[ ]:
```
[Lemma('code.v.02.code'),
 Lemma('code.v.02.encipher'),
 Lemma('code.v.02.cipher'),
 Lemma('code.v.02.cypher'),
 Lemma('code.v.02.encrypt'),
 Lemma('code.v.02.inscribe'),
 Lemma('code.v.02.write_in_code')]
```

From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

In [ ]:
```
code = wn.synset(syn_verb)
hyper = lambda s: s.hypernyms()
list(code.closure(hyper))
```

Out[ ]:
```
[Synset('encode.v.01'), Synset('convert.v.02'), Synset('change.v.01')]
```

Write a couple of sentences observing the way that WordNet is organized for verbs.

WordNet appears to organize verbs in nearly the same way that it organizes nouns. WordNet lists verbs from the most specific instance of the action to the most broad or

abstract version of that action. It makes sense that coding can be thought of as encoding (`encode.v.01`) logic in a language, converting (`convert.v.02`) human language to machine readable language, or enacting some change (`change.v.01`) in the machine.

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

In [ ]:
```
wn.morphy("coding", wn.VERB)
```

Out[ ]:
```
'code'
```

1. Select two words that you think might be similar.

In [ ]:
```
word_one = "whale"
word_two = "fish"
```

Find the specific synsets you are interested in.

In [ ]:
```
syn_one = wn.synsets(word_one)
syn_two = wn.synsets(word_two)
print(word_one + " synsets are:", syn_one)
print("\n" + word_two + " synsets are:", syn_two)
```

```
whale synsets are: [Synset('giant.n.04'), Synset('whale.n.02'), Synset('whale.
v.01')]

fish synsets are: [Synset('fish.n.01'), Synset('fish.n.02'), Synset('pisces.n.
02'), Synset('pisces.n.01'), Synset('fish.v.01'), Synset('fish.v.02')]
```

Run the Wu-Palmer similarity metric

In [ ]:
```
fish = syn_one[1]
whale = syn_two[0]

print("fish is:", fish.definition())
print("\nwhale is:", whale.definition())

fish.wup_similarity(whale)
```

```
fish is: any of the larger cetacean mammals having a streamlined body and brea
thing through a blowhole on the head

whale is: any of various mostly cold-blooded aquatic vertebrates usually havin
g scales and breathing through gills
```

Out[ ]:
```
0.72
```

Run the Lesk algorithm.

In [ ]:
```
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize

context_sentence = word_tokenize("that whale is swimming in the ocean.")
ambiguous = "whale"
lesk(context_sentence,ambiguous, "n")
```

Out[ ]:
```
Synset('whale.n.02')
```

In [ ]:
```
context_sentence = word_tokenize("I guess you just have to fish around for the
ambiguous = "fish"
lesk(context_sentence, ambiguous, 'v')
```

Out[ ]:
```
Synset('fish.v.02')
```

Write a couple of sentences with your observations.

Using whale and fish to compare with wu palmer similarity, it makes sense to see that they had a similarity of `0.72`. Since they are both creatures that swim in the ocean they would likely have a common noun ancestor close to each other, despite one being a mammal.

It is cool to see that running the lesk algorithm on whale and fish produced the correct versions of the word (noun and verb respectively) based on their context in the sentence.

1. Write a couple of sentences about SentiWordNet, describing its functionality and possible use cases.

SentiWordNet adds functionality to WordNet by assigning positive, negative, and neutral (objective) sentiment values to each word. Using SentiWordNet we can analyze reviews for a product or try to measure sentiment from tweets regarding a political or news event.

Select an emotionally charged word.

In [ ]:
```
emotional_word = "terrified"
wn.synsets(emotional_word)
```

Out[ ]:
```
[Synset('terrify.v.01'), Synset('panicky.s.01')]
```

Find its senti-synsets and output the polarity scores for each word.

In [ ]:
```
from nltk.corpus import sentiwordnet as swn

ews = swn.senti_synset('terrify.v.01')
print("Positive score = ", ews.pos_score())
print("Negative score = ", ews.neg_score())
print("Objective score = ", ews.obj_score())
```

```
Positive score =  0.25
Negative score =  0.0
Objective score =  0.75
```

Make up a sentence.

In [ ]:
```
made_up_sentence = "the new game terror from the deep is really great!"
```

Output the polarity for each word in the sentence.

In [ ]:
```
neg = 0
pos = 0
```

```
tokens = made_up_sentence.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(syn.synset.name() + " polarities:\nPositive: " + str(syn.pos_scor

print("\n-----------------------------------------\ntotal neg counts\ttotal
print(neg, '\t\t\t', pos)
```

```
new.a.01 polarities:
Positive: 0.375 Negative: 0.0

game.n.01 polarities:
Positive: 0.0   Negative: 0.0

panic.n.01 polarities:
Positive: 0.25  Negative: 0.75

deep.n.01 polarities:
Positive: 0.0   Negative: 0.0

be.v.01 polarities:
Positive: 0.25  Negative: 0.125

truly.r.01 polarities:
Positive: 0.625 Negative: 0.0


-----------------------------------------
total neg counts        total pos counts
0.875                    1.5
```

Write a couple of sentences about your observations of the scores and the utility of knowing these scores in an NLP application.

Honestly, some of the scores assigned to words in SentiWordNet don't seem right to me. I guess that is part of the problem with sentiment analysis, in that the severity of "positivity" and "negativity" of a word may differ from person to person. It is a bit subjective. However, knowing these score in an NLP application can be useful when you want to capture how the speaker or author is feeling.

1. Write a couple of sentences about what a collocation is.

A collocation is a gropuing of words that when put together, have a meaning that is useful and greater than the sum of their individual parts. An example of a collocation would be vice president. Both of these words have independent meaning, but when used in this order they have a combined meaning that refers to the 2nd in command of the United States of America.

Output collocations for text4, the Inaugural corpus.

In [ ]:
```
from nltk.book import *
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
```

Select one of the collocations identified by NLTK. Calculate mutual information.

In [ ]:
```
import math

colloc = "fellow citizens"
text = ' '.join(text4.tokens)
vocab = len(set(text4))
fc = text.count(colloc)/vocab
print("p(fellow citizens) = ",fc )
f = text.count('fellow')/vocab
print("p(fellow) = ", f)
c = text.count('citizens')/vocab
print('p(citizens) = ', c)
pmi = math.log2(fc / (f * c))
print('pmi = ', pmi)
```

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

Write commentary on the results of the mutual information formula and your interpretation:

The results of the mutual information formula make complete sense to me. 'fellow citizens' has much more meaning together than fellow, or citizens by themselves, and a score of 4.04 appears to confirm this. My interpretation is that the probability of fellow and citizens are both higher than fellow citizens, so each word is likely to appear on its own. However, the meaning of 'fellow citizens' is very important when used together.