CS4395 Portfolio WordNet Assignment

Shane Arwood

September 25, 2022

This notebook aims to demonstrate basic skills using WordNet and SentiWordNet, as well as to learn how to identify collocations.

WordNet Summary:

WordNet is a project that aims to understand how humans organize language not just through parts of speech like nouns and verbs, but through hierarchies within those parts of speech. It was started by George Miller, a psychologist, who wanted to investigate how humans organize concepts in language, and is now a database of synsets that are connected via lexical and conceptual links. It is an effective way to visualize how different words are closely and distantly related to one another.

Princeton University "About WordNet." WordNet. Princeton University. 2010.

```
import nltk
import math
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.book import *
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.corpus import sentiwordnet as swn
nltk.download('book')
nltk.download('sentiwordnet')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
```

```
[nltk_data]    |   Package timit 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 toolbox to /root/nltk_data...
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package udhr to /root/nltk_data...
[nltk_data]    |   Package udhr is already up-to-date!
[nltk_data]    | Downloading package udhr2 to /root/nltk_data...
[nltk_data]    |   Package udhr2 is already up-to-date!
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package unicode_samples 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 wordnet to /root/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_treebank_pos_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package maxent_treebank_pos_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package universal_tagset to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package universal_tagset is already up-to-date!
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package tagsets to /root/nltk_data...
[nltk_data]    |   Package tagsets is already up-to-date!
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package panlex_swadesh is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection book
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[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 punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

WordNet Nouns

```
# Get all synsets of the noun "music"
wn.synsets('music')
```

```
[Synset('music.n.01'),
 Synset('music.n.02'),
 Synset('music.n.03'),
 Synset('music.n.04'),
 Synset('music.n.05')]
```

```
# Selecting one synset: 'music.n.02'
# Extract its definition, usage examples, and lemmas.

wn.synset('music.n.02').definition()
```

```
     'any agreeable (pleasing and harmonious) sounds'
```

```
wn.synset('music.n.02').examples()
```

```
     ['he fell asleep to the music of the wind chimes']
```

```
wn.synset('music.n.02').lemmas()
```

```
     [Lemma('music.n.02.music'), Lemma('music.n.02.euphony')]
```

```
# Traverse up the WordNet hierarchy using closure()
music_synsets = wn.synset('music.n.02')
hypernyms = lambda s: s.hypernyms()
list(music_synsets.closure(hypernyms))
```

```
     [Synset('sound.n.02'),
      Synset('sensation.n.01'),
      Synset('perception.n.03'),
      Synset('basic_cognitive_process.n.01'),
      Synset('process.n.02'),
      Synset('cognition.n.01'),
      Synset('psychological_feature.n.01'),
      Synset('abstraction.n.06'),
      Synset('entity.n.01')]
```

Observations: WordNet Organization for nouns

For nouns, wordnet is organized based on broader and broader definition scopes for each hypernym up the hierarchy. Even the most specific noun is contained within another noun, and they all eventually end up being connected to the top noun, entity. It is interesting that the nouns aren't all necessarily tangible even if the starting noun is; for example, music (noun 02) is a sound, but it eventually is traced to abstraction.

```
# hypernyms, hyponyms, meronyms, holonyms, antonyms of "music.n.02".
nusic_lemmas = music_synsets.lemmas()
print(music_synsets.hypernyms())
print(music_synsets.hyponyms())
print(wn.synset('music.n.02').part_meronyms())
print(wn.synset('music.n.02').part_holonyms())
print(nusic_lemmas[0].antonyms())

     [Synset('sound.n.02')]
     [Synset('music_of_the_spheres.n.01')]
     []
     []
     []
```

WordNet Verbs

```
# Get all synsets of the verb "knit"
wn.synsets('knit')

     [Synset('knit.n.01'),
      Synset('knit.n.02'),
      Synset('knit.n.03'),
      Synset('knit.v.01'),
      Synset('knit.v.02'),
      Synset('pucker.v.01')]
```

```
# Selecting one synset: 'knit.v.01'
# Extract its definition, usage examples, and lemmas.

wn.synset('knit.v.01').definition()

     'make (textiles) by knitting'
```

```
wn.synset('knit.v.01').examples()

     ['knit a scarf']
```

```
wn.synset('knit.v.01').lemmas()

     [Lemma('knit.v.01.knit')]
```

```
# Traverse up the WordNet hierarchy using closure()
knit_synsets = wn.synset('knit.v.01')
hypernyms = lambda s: s.hypernyms()
list(knit_synsets.closure(hypernyms))

     [Synset('create_from_raw_material.v.01'), Synset('make.v.03')]
```

Observations: WordNet Organization for verbs

For verbs, wordnet is organized based on less and less specific verbs for each hypernym up the hierarchy. For a very specific verb, the hierarchy is not nearly as long as for a very specific noun. In WordNet, verbs don't have a top verb, hence the hierarchy just ends when a gerneral verb is reached. It is curious that the more broad verb, create, was present within one of the synsets, but not by itself. Make was a synset instead; this could be a limitation of the WordNet hierarchy.

Morphy

```
# Using morphy to find different forms of 'knit'. The only form that doesn't
# end up with knit as the base word is "knitting".
print(wn.morphy('knitted'))
print(wn.morphy('knitting'))
print(wn.morphy('knit'))
print(wn.morphy('knits'))
```

```
knit
knitting
knit
knit
```

Similarity

```
# Using the two words that I believe to be similar: kind and nice.
# Finding the appropriate synsets.
print(wn.synsets('kind'))
print(wn.synset('kind.a.01').definition())
```

```
[Synset('kind.n.01'), Synset('kind.a.01'), Synset('kind.s.02'), Synset('kind.s.0
having or showing a tender and considerate and helpful nature; used especially o
```

```
print(wn.synsets('nice'))
print(wn.synset('nice.a.01').definition())
```

```
[Synset('nice.n.01'), Synset('nice.a.01'), Synset('decent.s.01'), Synset('nice.s
pleasant or pleasing or agreeable in nature or appearance; - George Meredith
```

```
# Run the Wu-Palmer similarity metric
kind = wn.synset('kind.a.01')
nice = wn.synset('nice.a.01')
wn.wup_similarity(kind, nice)
```

```
    0.5
```

```
# Run the Lesk algorithm
sent = word_tokenize("The volunteer was very kind and caring.")
print(lesk(sent, 'kind'))
```

```
    Synset('kind.s.03')
```

```
wn.synset('kind.s.03').definition()
```

```
    'tolerant and forgiving under provocation'
```

```
sent = word_tokenize("The volunteer was very nice and caring.")
print(lesk(sent, 'nice'))
```

```
    Synset('nice.s.03')
```

```
wn.synset('nice.s.03').definition()
```

```
    'done with delicacy and skill'
```

Observations: WordNet Similarity

The results of NLTK's similarity functionality were surprising to me. Two words that I believed to be very similar, kind and nice, scored only 0.5 on the Wu-Palmer similarity metric. 0.5 is halfway between 0 (little similarity) and 1 (high similarity), indicating the two words are rather neutral in relation to one another; not very similar and not very different. Perhaps it is an indication of the importance of context, or that the synsets I believed were similar were not that similar after all.

When the Lesk algorithm was used, the two words were used in the same place in the same sentence to understand the interpreted meaning of each word within the same context. Unexpectedly, the synsets chosen by the Lesk algorithm were different than the ones I chose initially. The Lesk algorithm's results chose the meanings the dealt more with a way of responding to things rather than a state of being/character. This would likely change if the context changed.

SentiWordNet

SentiWordNet is a functionality within WordNet that assigns scores to WordNet synsets for negativity, positivity, and objectivity. There is a sentisynset functionality that finds words with similar scores. It allows analysis of the sentiments associated with different words. It is useful because so far, WordNet's methods have only been used to identify connections between synsets, not insight into specific synsets other than definition and examples. It seems that SentiWordNet adds a more human element by measuring connotation as well. Possible applications could be detecting hateful speech in social media posts or mesuring general response to a certain post/product via analyzing words in the comments/reviews.

```
# Sentiwordnet analysis of an emotionally charged word: devastating
sentisynsets = list(swn.senti_synsets('devastating'))
print('Sentisynsets of devastating:')
for synset in sentisynsets:
  print(synset)
  print('Positive: ', synset.pos_score(), ', Negative: ', synset.neg_score(),
        ', Objectivity: ', synset.obj_score())
  print('\n')
```

```
Sentisynsets of devastating:
<lay_waste_to.v.01: PosScore=0.0 NegScore=0.0>
Positive:  0.0 , Negative:  0.0 , Objectivity:  1.0


<devastate.v.02: PosScore=0.125 NegScore=0.0>
Positive:  0.125 , Negative:  0.0 , Objectivity:  0.875


<annihilating.s.02: PosScore=0.0 NegScore=0.0>
Positive:  0.0 , Negative:  0.0 , Objectivity:  1.0


<annihilative.s.01: PosScore=0.5 NegScore=0.125>
Positive:  0.5 , Negative:  0.125 , Objectivity:  0.375


<crushing.s.01: PosScore=0.0 NegScore=0.75>
Positive:  0.0 , Negative:  0.75 , Objectivity:  0.25
```

```
# Make up a sentence and output polarity for each word in the sentence:
# The scary ghost lurked in the graveyard.
sentence = 'The scary ghost lurked in the graveyard.'
tokens = word_tokenize(sentence)
for token in tokens:
  token_synsets = list(swn.senti_synsets(token))
  if token_synsets:
    print(token_synsets[0], 'Positive: ', token_synsets[0].pos_score(),
          ', Negative: ', token_synsets[0].neg_score(),
          ', Objectivity: ', token_synsets[0].obj_score())
```

```
<chilling.s.01: PosScore=0.0 NegScore=0.75> Positive:  0.0 , Negative:  0.75 , Ol
<ghost.n.01: PosScore=0.0 NegScore=0.0> Positive:  0.0 , Negative:  0.0 , Object
```

```
<lurk.v.01: PosScore=0.125 NegScore=0.0> Positive:  0.125 , Negative:  0.0 , Obj
<inch.n.01: PosScore=0.0 NegScore=0.0> Positive:  0.0 , Negative:  0.0 , Objecti
<cemetery.n.01: PosScore=0.0 NegScore=0.0> Positive:  0.0 , Negative:  0.0 , Obj
```

Observations: SentiWordNet Scores

When comparing the sentisynsets of an emotionally charged word, most of them had similar low-levels of objectivity. However, some of the sentisynsets had an opposite high positive/negative score from the original word, even though I would consider them to be synonyms or at least very similar in connotation. It is curious which synset of the original word the sentisynsets are based off of.

Looking at the sentence, I came up with one that has a relatively negative meaning. However, looking at the synsets for each of the words, that negative meaning would not be assumed. All of the words had high objectivity scores save for the one adjective in the sentence, which was the only one whose score matched my perceived perception of the whole sentence.

In an NLP application, these scores could be useful for moderating the detection of emotionally-charged language. Because sentiwordnet only considers a word by itself, not context, it doesn't detect as sensitively as a human might. However, if such sensitivity was required, sentiwordnet scores in conjuection with other NLP tools for context could be used.

Collocations

A collocation is a series of words that common occur together, such as 'crystal clear' or 'neat and tidy'. The combined words have a specific meaning and connotation, almost as if they were a completely separate word. This specific meaning means that in a collocation, you cannot replace either of the words with a synonym and get the same meaning. In context, the words in a collocation are more likely to occur together than would be expected.

```
# Collocations for 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
```

```
# Mutual information for 'Vice President' identified by NLTK.
text = ' '.join(text4.tokens)
```

```
# probability of vice president
vocab = len(set(text4))
vice_president = text.count('Vice President') / vocab
print('Probability of vice president: ', vice_president)
vice = text.count('Vice') / vocab
print('Probability of vice: ', vice)
president = text.count('President') / vocab
print('Probability of president: ', president)
pmi = math.log2(vice_president / (vice * president))
print('PMI: ', pmi)
```

```
    Probability of vice president:  0.0017955112219451373
    Probability of vice:  0.0018952618453865336
    Probability of president:  0.010773067331670824
    PMI:  6.458424602064904
```

Observation: results of the mutual information formula and interpretation

The mutual information formula combines the probabilities of each part of the collocation individually as well as the probability of the collocation itself. In calculating the formula, insight is gained into each of the probabilities. It was interesting to see that for the detected collocation 'vice president', the probabilities of it and 'vice' were very similar, suggesting that vice occurred mostly in the context of 'vice president'.'President' had a higher probability than the other two, likely because that is a common single standing word. Ultimately, the PMI was positive, indicating that 'vice president' is a collocation.