CS4395_HW3

February 25, 2023

0.0.1 David Nguyen

0.0.2 dxn180015

0.0.3 Dr. Karen Mazidi

0.0.4 CS 4395.001

#

HW 3 - WordNet

---

1 1. What is WordNet?

WordNet is a lexicle database of English words. Words are organized and linked by their lexical & semantic relations or in other words concept and meaning. This database is viewable on a web browser. However, WordNet is more designed to be used by computers for automatic text analysis and artificial intelligence applications.

Nouns, verbs, adjectives and adverbs are grouped into sets (synsets) that are each expressing a distinct concept. And synsets are interlinked by semantic and lexical relations. This network of words can then be navigated.

2 2. Select a noun (game). Output all synsets of that noun/word

[181]:
```
# Import WordNet and NLTK dependencies
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn
```

```
[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!
```

[182]:
```
#  Get all synsets of the noun 'game' and output all synsets of that noun/word
wn.synsets('game')
```

[182]:
```
[Synset('game.n.01'),
 Synset('game.n.02'),
 Synset('game.n.03'),
 Synset('game.n.04'),
 Synset('game.n.05'),
 Synset('game.n.06'),
 Synset('game.n.07'),
 Synset('plot.n.01'),
 Synset('game.n.09'),
 Synset('game.n.10'),
 Synset('game.n.11'),
 Synset('bet_on.v.01'),
 Synset('crippled.s.01'),
 Synset('game.s.02')]
```

3 3.a. Select one synset (game.n.02) from the list of synsets. Extract its definition, usage examples, and lemmas.

[183]:
```
# Extract its definition
wn.synset('game.n.01').definition()
```

[183]:
```
'a contest with rules to determine a winner'
```

[184]:
```
# Extract its usage examples
wn.synset('game.n.01').examples()
```

[184]:
```
['you need four people to play this game']
```

[185]:
```
# Extract its lemmas
wn.synset('game.n.01').lemmas()
```

[185]:
```
[Lemma('game.n.01.game')]
```

4 3.b. Traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

[186]:
```
# iterate over synsets
game_synsets = wn.synsets('game', pos=wn.NOUN)
for sense in game_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:
 ↪" + str(lemmas))
```

```
Synset: game.n.01(a contest with rules to determine a winner)
         Lemmas:['game']
Synset: game.n.02(a single play of a sport or other contest)
```

```
         Lemmas:['game']
Synset: game.n.03(an amusement or pastime)
         Lemmas:['game']
Synset: game.n.04(animal hunted for food or sport)
         Lemmas:['game']
Synset: game.n.05((tennis) a division of play during which one player serves)
         Lemmas:['game']
Synset: game.n.06((games) the score at a particular point or the score needed to
win)
         Lemmas:['game']
Synset: game.n.07(the flesh of wild animals that is used for food)
         Lemmas:['game']
Synset: plot.n.01(a secret scheme to do something (especially something
underhand or illegal))
         Lemmas:['plot', 'secret_plan', 'game']
Synset: game.n.09(the game equipment needed in order to play a particular game)
         Lemmas:['game']
Synset: game.n.10(your occupation or line of work)
         Lemmas:['game', 'biz']
Synset: game.n.11(frivolous or trifling behavior)
         Lemmas:['game']
```

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

WordNet organizes nouns by having the most general concept of that noun/synset at the top of the list. And as you go down the list, the word/noun becomes more specific. And it is also a subset of that concept.

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

[187]:
```
# Output all hypernyms of 'game.n.01'
noun = wn.synset('game.n.01')
noun.hypernyms()
```

[187]:
```
[Synset('activity.n.01')]
```

[188]:
```
# Output all hyponyms of 'game.n.01'
noun.hyponyms()
```

[188]:
```
[Synset('athletic_game.n.01'),
 Synset('bowling.n.01'),
 Synset('card_game.n.01'),
 Synset('child's_game.n.01'),
 Synset('curling.n.01'),
```

```
 Synset('game_of_chance.n.01'),
 Synset('pall-mall.n.01'),
 Synset('parlor_game.n.01'),
 Synset('table_game.n.01'),
 Synset('zero-sum_game.n.01')]
```

[189]:
```
# Output all meronyms of 'game.n.01'
noun.part_meronyms()
```

[189]:
```
[]
```

[190]:
```
# Output all holonyms of 'game.n.01'
noun.part_holonyms()
```

[190]:
```
[]
```

[191]:
```
# Output all antonym of 'game.n.01'
noun.lemmas()[0].antonyms()
```

[191]:
```
[]
```

7 5. Select a verb. Output all synsets.

[192]:
```
#  Get all synsets of the verb 'stretch' and output all synsets of that verb/
 ↪word
verb = wn.synsets('stretch')
verb
```

[192]:
```
[Synset('stretch.n.01'),
 Synset('reach.n.03'),
 Synset('stretch.n.03'),
 Synset('stretch.n.04'),
 Synset('stretch.n.05'),
 Synset('stretch.n.06'),
 Synset('stretch.n.07'),
 Synset('stretch.v.01'),
 Synset('stretch.v.02'),
 Synset('unfold.v.03'),
 Synset('stretch.v.04'),
 Synset('elongate.v.01'),
 Synset('stretch.v.06'),
 Synset('stretch.v.07'),
 Synset('stretch.v.08'),
 Synset('load.v.05'),
 Synset('extend.v.17'),
 Synset('stretch.v.11'),
 Synset('stretch.s.01'),
```

```
 Synset('stretch.s.02')]
```

8 6. Select one synset (game.n.02) from the list of synsets. Extract its definition, usage examples, and lemmas.

[193]:
```
# Extract its definition
wn.synset('stretch.v.01').definition()
```

[193]:
```
'occupy a large, elongated area'
```

[194]:
```
# Extract its usage examples
wn.synset('stretch.v.01').examples()
```

[194]:
```
['The park stretched beneath the train line']
```

[195]:
```
# Extract its lemmas
wn.synset('stretch.v.01').lemmas()
```

[195]:
```
[Lemma('stretch.v.01.stretch'), Lemma('stretch.v.01.stretch_along')]
```

9 6.b. Traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

[196]:
```
# iterate over synsets
stretch_synsets = wn.synsets('stretch', pos=wn.VERB)
for sense in stretch_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:
 ↪" + str(lemmas))
```

```
Synset: stretch.v.01(occupy a large, elongated area)
         Lemmas:['stretch', 'stretch_along']
Synset: stretch.v.02(extend one's limbs or muscles, or the entire body)
         Lemmas:['stretch', 'extend']
Synset: unfold.v.03(extend or stretch out to a greater or the full length)
         Lemmas:['unfold', 'stretch', 'stretch_out', 'extend']
Synset: stretch.v.04(become longer by being stretched and pulled)
         Lemmas:['stretch']
Synset: elongate.v.01(make long or longer by pulling and stretching)
         Lemmas:['elongate', 'stretch']
Synset: stretch.v.06(lie down comfortably)
         Lemmas:['stretch', 'stretch_out']
Synset: stretch.v.07(pull in opposite directions)
         Lemmas:['stretch']
Synset: stretch.v.08(extend the scope or meaning of; often unduly)
         Lemmas:['stretch']
```

```
Synset: load.v.05(corrupt, debase, or make impure by adding a foreign or
inferior substance; often by replacing valuable ingredients with inferior ones)
         Lemmas:['load', 'adulterate', 'stretch', 'dilute', 'debase']
Synset: extend.v.17(increase in quantity or bulk by adding a cheaper substance)
         Lemmas:['extend', 'stretch']
Synset: stretch.v.11(extend one's body or limbs)
         Lemmas:['stretch', 'stretch_out']
```

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

Just like in 3c or for nouns, WordNet organizes verbs by having the most general concept of that noun/synset at the top of the list. And as you go down the list, the word/noun becomes more specific. And it is also a subset of that concept.

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

[197]:
```
wn.morphy('stretch',wn.VERB)
```

[197]:
```
'stretch'
```

12 8.a. Select two words that you think might be similar. Find the specific synsets you are interested in.

[198]:
```
# Find 2 similar words
word1 = wn.synsets('rest')
word2 = wn.synsets('relax')
print(word1)
print()
print(word2)

# Find 2 specific synsets
word1_synset = wn.synset('rest.v.01')
word2_synset = wn.synset('relax.v.01')
print()
print('rest.v.01')
print('relax.v.01')
```

```
[Synset('remainder.n.01'), Synset('rest.n.02'), Synset('respite.n.04'),
Synset('rest.n.04'), Synset('rest.n.05'), Synset('rest.n.06'),
Synset('rest.n.07'), Synset('rest.v.01'), Synset('rest.v.02'),
Synset('rest.v.03'), Synset('lie.v.06'), Synset('rest.v.05'),
Synset('stay.v.01'), Synset('rest.v.07'), Synset('rest.v.08'),
Synset('perch.v.01'), Synset('pillow.v.01'), Synset('rest.v.11')]
```

```
[Synset('relax.v.01'), Synset('relax.v.02'), Synset('loosen.v.07'),
Synset('relax.v.04'), Synset('relax.v.05'), Synset('relax.v.06'),
Synset('relax.v.07'), Synset('slack.v.04')]

rest.v.01
relax.v.01
```

13 8.b. Run the Wu-Palmer similarity metric and the Lesk algorithm.

[199]:
```
# Run Wu-Palmer similarity metric
wn.wup_similarity(word1_synset, word2_synset)
```

[199]:
```
0.2857142857142857
```

[200]:
```
# Import the Lesk algorithm
from nltk.wsd import lesk

# Look at the definitions for 'rest'
for ss in wn.synsets('rest'):
    print(ss, ss.definition())
```

```
Synset('remainder.n.01') something left after other parts have been taken away
Synset('rest.n.02') freedom from activity (work or strain or responsibility)
Synset('respite.n.04') a pause for relaxation
Synset('rest.n.04') a state of inaction
Synset('rest.n.05') euphemisms for death (based on an analogy between lying in a
bed and in a tomb)
Synset('rest.n.06') a support on which things can be put
Synset('rest.n.07') a musical notation indicating a silence of a specified
duration
Synset('rest.v.01') not move; be in a resting position
Synset('rest.v.02') take a short break from one's activities in order to relax
Synset('rest.v.03') give a rest to
Synset('lie.v.06') have a place in relation to something else
Synset('rest.v.05') be at rest
Synset('stay.v.01') stay the same; remain in a certain state
Synset('rest.v.07') be inherent or innate in
Synset('rest.v.08') put something in a resting position, as for support or
steadying
Synset('perch.v.01') sit, as on a branch
Synset('pillow.v.01') rest on or as if on a pillow
Synset('rest.v.11') be inactive, refrain from acting
```

[201]:
```
# Run the Lesk algorithm
print('The ball was at rest.')
```

```
sent = ['The', 'ball', 'was', 'at', 'rest', '.']
print(lesk(sent, 'rest', 'v'), '\n')

print("The game was put to rest on my bed.")
sent = ['The', 'game', 'was', 'put', 'to', 'rest', 'on', 'my', 'bed', '.']
print(lesk(sent, 'rest', 'v'))
```

```
The ball was at rest.
Synset('rest.v.05')

The game was put to rest on my bed.
Synset('rest.v.03')
```

14 8.c. Write a couple of sentences with your observations.

The Wu-Palmer similarity metric is a very cool way to see how similar 2 words are to each other. I was suprised how rest and relax had a low score.

And the Lesk algorithm was also interresting to use to see which definition a word was used in a sentence. I used multiple forms of rest in a sentence and the lesk alforithmn gave me the correct definition.

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

SentiWordNet is a lexical resource that's is based on WordNet. It's function is to calculae and assign each synset 3 "sentiment scores". They are positivity, negativity, and objectivity.

Thus, this resource has many possible use cases with sentiment analysis on subjective text and see if people are viewing somethign positively, negatively, or neutrally. This can be useful for search engines or summarizing/analyzing product reviews, blog posts, books, tweets, and forum posts.

16 9.b. Select an emotionally charged word. Find its senti-synsets and output the polarity scores for each word.

[202]:
```
# Import SentiWordNet
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn

# Select emotionally charged word
print(wn.synsets('hate'))
print()

# iterate over synsets
hate_synsets = wn.synsets('hate')
for sense in hate_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
```

```
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:
  ↪" + str(lemmas))
print()

# Find its senti-synsets and output the polarity scores for each word
senti_synsets = list(swn.senti_synsets('hate'))

for i in senti_synsets:
  print(i)
  print("Positive score = ", i.pos_score())
  print("Negative score = ", i.neg_score())
  print("Objective score = ", i.obj_score())
  print()
```

```
[Synset('hate.n.01'), Synset('hate.v.01')]

Synset: hate.n.01(the emotion of intense dislike; a feeling of dislike so strong
that it demands action)
         Lemmas:['hate', 'hatred']
Synset: hate.v.01(dislike intensely; feel antipathy or aversion towards)
         Lemmas:['hate', 'detest']

<hate.n.01: PosScore=0.125 NegScore=0.375>
Positive score =  0.125
Negative score =  0.375
Objective score =  0.5

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


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

17 9.c. Make up a sentence. Output the polarity for each word in the sentence.

[203]:
```
# Make up a sentence
sent = 'HP has the worst customer service I have ever seen'
print(sent + "\n")

neg = 0
pos = 0
tokens = sent.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("neg\tpos counts")
print(neg, '\t', pos)
```

```
HP has the worst customer service I have ever seen

neg     pos counts
1.0      0.0
```

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

The score shows that my sentence has a 100% negative sentiment, which is correct as that sentence is meant to be from an angry customer of HP.

This can be helpful very in NLP applications where one wants to analyze the sentiments from the comments/posts of users (be it an comment, review, feedback, etc.) from a forum, form, book, article, etc. With this analysis, one can determine how a user or post may be emotionally negative, positive, helpful, dangerous, or even harmful. And the results can then be used to possibly improve your product based on feedback, implement changes, ban/delete/report users/posts, etc.

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

A collocation is a set of 2 or more words that are frequently used together to form a meaning of a thing or concept. When used together, those words means differently or something greater than the meaning of each indiviual word. Examples of collocations are 'a quick meal', 'hard disk', 'a quick shower'. While a set of words like 'a fast shower' sounds unnatural and is not collocation.

20 10.b. Output collocations for text4, the Inaugural corpus.

[204]:
```
# Import text4
import nltk
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 *
```

```
text4

# Output collocations for text4, the Inaugural corpus
print()
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
```

```
[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!
```

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

[205]:
```
# Import math
import math

# Print section of text4
text = ' '.join(text4.tokens)
print(text[:150], '...\n')

# Select a collocations identified by NLTK
# and calculate Mutual info - 'Federal Government'
import math
vocab = len(set(text6))
hg = text.count('Federal Government')/vocab
print("p(Federal Government) = ",hg )

h = text.count('Federal')/vocab
print("p(Federal) = ", h)
```

```
g = text.count('Government')/vocab
print('p(Government) = ', g)

pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
print()

# Select a collocations identified by NLTK
# and calculate Mutual info - 'foreign nations'
import math
vocab = len(set(text6))
hg = text.count('foreign nations')/vocab
print("p(foreign nations) = ",hg )

h = text.count('foreign')/vocab
print("p(foreign) = ", h)

g = text.count('nations')/vocab
print('p(nations) = ', g)

pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
Fellow - Citizens of the Senate and of the House of Representatives : Among the
vicissitudes incident to life no event could have filled me with great …

p(Federal Government) =  0.014773776546629732
p(Federal) =  0.030009233610341645
p(Government) =  0.15604801477377656
pmi =  1.6575702782976882

p(foreign nations) =  0.006925207756232687
p(foreign) =  0.04755309325946445
p(nations) =  0.09464450600184672
pmi =  0.6217274965281813
```

22 10.d. Write commentary on the results of the mutual information formula and your interpretation.

When calculating the PMI or Pointwise Mutual Information for “Federal Government” and “foreign nations”, the results were 1.6575702782976882 and 0.6217274965281813 respectively. Thus, “Federal Government” is more likey to be a collocation than “foreign nations”.

We can see that “Government” occurs more frequently than “Federal” or “Federal Government”. While “Federal Government” occurs more than “Federal”. We can also see that “nations” occurs more frequently than “foreign” or “foreign nations”. While “foreign” occurs more than “foreign nations”.