Homework3_bjl190000

February 25, 2023

1 HLT Assignment 3: WordNet

WordNet is a module in the NLTK library which allows programmers to access information about a given word such as all the definitions and synonyms of a word. It also organizes words into a hierarchy which can be traversed to see how words are related to each other.

[1]:
```
#imports
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.book import *
from nltk.corpus import sentiwordnet as swn

import math
```

```
*** 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
```

1.1 Nouns

[2]:
```
noun = "fish"
word_synsets = wn.synsets(noun)
print("Synsets: \n", word_synsets)
```

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

[3]:
```
synset_sample = word_synsets[0]
synset_sample_name = synset_sample.name()
print(wn.synset(synset_sample_name).definition())
```

```
any of various mostly cold-blooded aquatic vertebrates usually having scales and
breathing through gills
```

1.1.1 Traversal up the Noun Hierarchy

WordNet's nouns have a tall and well-connected hierarchy. Especially because I chose a biological word, which has hierarchical nouns to describe life anyways.

[4]:
```
hypernym = synset_sample.hypernyms()[0]
top = wn.synset('entity.n.01')
while hypernym:
   print(hypernym)
   if hypernym == top:
      break
   if hypernym.hypernyms():
      hypernym = hypernym.hypernyms()[0]
```

```
Synset('aquatic_vertebrate.n.01')
Synset('vertebrate.n.01')
Synset('chordate.n.01')
Synset('animal.n.01')
Synset('organism.n.01')
Synset('living_thing.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

[5]:
```
print("Hypernyms:\n",synset_sample.hypernyms())
print("Hyponyms:\n", synset_sample.hyponyms())
print("Meronyms:\n",synset_sample.part_meronyms())
print("Holonyms:\n", synset_sample.part_holonyms())
print("Antonyms:\n", synset_sample.lemmas()[0].antonyms())
```

```
Hypernyms:
 [Synset('aquatic_vertebrate.n.01')]
Hyponyms:
 [Synset('bony_fish.n.01'), Synset('bottom-feeder.n.02'),
Synset('bottom_lurkers.n.01'), Synset('cartilaginous_fish.n.01'),
Synset('climbing_perch.n.01'), Synset('fingerling.n.01'),
Synset('food_fish.n.01'), Synset('game_fish.n.01'), Synset('mouthbreeder.n.01'),
Synset('northern_snakehead.n.01'), Synset('rough_fish.n.01'),
Synset('spawner.n.01'), Synset('young_fish.n.01')]
Meronyms:
 [Synset('fin.n.06'), Synset('fish_scale.n.01'), Synset('fishbone.n.01'),
```

```
Synset('lateral_line.n.01'), Synset('milt.n.02'), Synset('roe.n.02'),
Synset('tail_fin.n.03')]
Holonyms:
 []
Antonyms:
 []
```

[6]:
```
verb = "examine"
verb_synsets = wn.synsets(verb)
print(verb_synsets)
```

```
[Synset('analyze.v.01'), Synset('examine.v.02'), Synset('probe.v.01'),
Synset('examine.v.04'), Synset('test.v.01')]
```

[7]:
```
verb_synset_sample = verb_synsets[2]
verb_synset_name = verb_synset_sample.name()
print("Definition:\n", wn.synset(verb_synset_name).definition())
print("Examples:\n", wn.synset(verb_synset_name).examples())
print("Lemmas:\n", wn.synset(verb_synset_name).lemmas())
```

```
Definition:
 question or examine thoroughly and closely
Examples:
 []
Lemmas:
 [Lemma('probe.v.01.probe'), Lemma('probe.v.01.examine')]
```

1.1.2 Verb Traversal

The verb hierarchy seems to be very flat and divided. It does not have a top level synset unlike the noun hierarchy. This is likely because verbs cannot be connected easily using synonyms. For example, look and run are both bodily movements, but there's no verb to describe the movement of a body part.

[8]:
```
print("Traversal:")
hypernym = verb_synset_sample.hypernyms()[0]
previous_hypernym = wn.synset('analyze.v.01')
while hypernym:
   print(hypernym, previous_hypernym)
   if hypernym == previous_hypernym:
      break
   if hypernym.hypernyms():
      hypernym = hypernym.hypernyms()[0]

   previous_hypernym = hypernym
```

```
Traversal:
Synset('investigate.v.01') Synset('analyze.v.01')
Synset('analyze.v.01') Synset('analyze.v.01')
```

1.2 Morphy

[9]:
```
#show all forms of the verb
print(wn.morphy("examining", wn.VERB))
print(wn.morphy("examined"))
```

```
examine
examine
```

[10]:
```
mouse = wn.synset("mouse.n.01")
hamster = wn.synset("hamster.n.01")
wn.path_similarity(mouse, hamster)
```

[10]:
```
0.3333333333333333
```

1.3 Lesk Algorithm

[11]:
```
lesk_word = "bottle"

for ss in wn.synsets(lesk_word):
   print(ss, ss.definition())
```

```
Synset('bottle.n.01') a glass or plastic vessel used for storing drinks or other
liquids; typically cylindrical without handles and with a narrow neck that can
be plugged or capped
Synset('bottle.n.02') the quantity contained in a bottle
Synset('bottle.n.03') a vessel fitted with a flexible teat and filled with milk
or formula; used as a substitute for breast feeding infants and very young
children
Synset('bottle.v.01') store (liquids or gases) in bottles
Synset('bottle.v.02') put into bottles
```

[12]:
```
test_sentence = "I want to bottle the water in that lake."

print(lesk(test_sentence, lesk_word))
```

```
Synset('bottle.n.03')
```

1.4 SentiWordNet

SentiWordNet is a modification of WordNet, which adds sentiment scores to each word reflecting how positive or negative the word usually is depending on the definition.

The scores given are fairly accurate and there are some synsets where the word has both a positive and a negative score. For example, “fury” can be be used positively like “I worked myself up into a fury”. SentiWordNet seems to struggle with modifiers like ‘not’ or ‘never’. It should modify the word after it to be the opposite, but since SWN is just for tokens, it just assigns them as negative.

While knowing the strength of how positive/negative any word is is useful, it clearly requires a good model to utilize it. For example, if you were to simply add all the scores of the example sentence

below, you would get a negative score. However, the 'not' in the sentence makes the sentence neutral at worst.

[13]:
```
emotional_word = "very"
emot_word_list = swn.senti_synsets(emotional_word)
for ss in emot_word_list:
   print(ss, "ObjScore:", ss.obj_score())
```

```
<very.s.01: PosScore=0.5 NegScore=0.0> ObjScore: 0.5
<identical.s.02: PosScore=0.5 NegScore=0.125> ObjScore: 0.375
<very.r.01: PosScore=0.25 NegScore=0.25> ObjScore: 0.5
<very.r.02: PosScore=0.25 NegScore=0.0> ObjScore: 0.75
```

[14]:
```
test_sentence = "His question was not terribly bad."

print(test_sentence.split())
for token in test_sentence.split():
   syn_list = list(swn.senti_synsets(token))
   if (syn_list):
      syn = syn_list[0]
      print(syn)
```

```
['His', 'question', 'was', 'not', 'terribly', 'bad.']
<question.n.01: PosScore=0.125 NegScore=0.25>
<washington.n.02: PosScore=0.0 NegScore=0.0>
<not.r.01: PosScore=0.0 NegScore=0.625>
<terribly.r.01: PosScore=0.25 NegScore=0.0>
```

1.5 Collocations

Collocations are sequences of words which are frequently used together and have greater rhetorical effect. They usually occur because they sounds more natural to native speakers. The below code finds collocations in U.S. Inaugural Addresses and finds the Point-Wise Mutual Information (PMI) to find how strong the association between the two words.

[15]:
```
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
```

[16]:
```
text = " ".join(text4.tokens)
print(text[:50])

num_words = len(set(text4.tokens))
print(num_words)
```

```
def find_pmi(text, num_words, word1, word2):
    word1_prob = text.count(word1)/num_words
    word2_prob = text.count(word2)/num_words
    both_words = word1 + " " + word2
    both_words_prob =  text.count(both_words)/num_words

    print("Word '%s' probability: %.3f" % (word1, word1_prob))
    print("Word '%s' probability: %.3f" % (word2, word2_prob))
    print("Collocation '%s' probability: %.3f" % (both_words, both_words_prob))

    pmi = math.log2(both_words_prob/(word1_prob * word2_prob))
    return pmi

pmi = find_pmi(text, num_words, "United", "States")
print(pmi)
```

```
Fellow - Citizens of the Senate and of the House o
10025
Word 'United' probability: 0.017
Word 'States' probability: 0.033
Collocation 'United States' probability: 0.016
4.815657649820885
```