Portfolio Assignment WordNet

By Jibin Prince

```
import nltk
#nltk.download('all')
from nltk.corpus import wordnet as wn, sentiwordnet as swn
from nltk.wsd import lesk
from nltk import word_tokenize
from nltk.book import *
import math
```

WordNet Summary (Instruction 1)

WordNet is a large lexical database of English words and their sematic relations to each other. These semantic relations form a unique network to help NLP processes find relationship between words. In WordNet, nouns, verbs, adjectives, and adverbs are grouped into sets of synonyms, which are known as synsets.

Reference: L. Williams, "WordNet: A Lexical Taxonomy of English Words," Medium, Jan. 27, 2021. https://towardsdatascience.com/%EF%B8%8Fwordnet-a-lexical-taxonomy-of-english-words-4373b541cfff

```
# Instruction 2
wn.synsets('person')
```

```
[Synset('person.n.01'), Synset('person.n.02'), Synset('person.n.03')]
```

```
# Instruction 3
print("Synset: person.n.02")
person = wn.synset('person.n.02')
print("Definition: " + person.definition())
print("Usage Examples: ")
print(person.examples())
print("Lemmas: ")
print(person.lemmas())
print()
print("Synset: person.n.02 Hierarchy")
hyper = lambda s: s.hypernyms()
list(person.closure(hyper))
```

```
Synset: person.n.02
Definition: a human body (usually including the clothing)
Usage Examples:
['a weapon was hidden on his person']
```

```
    Synset: person.n.02 Hierarchy
    [Synset('human_body.n.01'),
     Synset('body.n.01'),
     Synset('natural_object.n.01'),
     Synset('whole.n.02'),
     Synset('object.n.01'),
     Synset('physical_entity.n.01'),
     Synset('entity.n.01')]
```

For nouns, it looks like there is a hierarchy of hypernyms that a noun can reach with entity.n.01 being the top-most level of the noun hierarchy. It is interesting that entity is above object and it makes sense to me since an entity of an object means the object has an independent existence.

```
# Instruction 4
print("Synset: person.n.02")
print("Hypernyms: ")
print(person.hypernyms())
print("Hyponyms: ")
print(person.hyponyms())
print("Meronyms: ")
print(person.part_meronyms() + person.substance_meronyms())
print("Holonyms: ")
print(person.part_holonyms() + person.substance_holonyms())
print("Antonyms: ")
print(person.lemmas()[0].antonyms())
```

```
    Synset: person.n.02
    Hypernyms:
    [Synset('human_body.n.01')]
    Hyponyms:
    []
    Meronyms:
    []
    Holonyms:
    []
    Antonyms:
    []
```

```
# Instruction 5
wn.synsets("try")
```

```
    [Synset('attempt.n.01'),
     Synset('try.v.01'),
     Synset('test.v.01'),
     Synset('judge.v.05'),
     Synset('sample.v.01'),
     Synset('hear.v.03'),
     Synset('try.v.06'),
     Synset('try.v.06'),
     Synset('try.v.07'),
     Synset('try.v.08'),
     Synset('try_on.v.01')]
```

```
# Instruction 6
print("Synset: try.v.01")
tryVerb = wn.synset('try.v.01')
print("Definition: " + tryVerb.definition())
print("Usage Examples: ")
print(tryVerb.examples())
print("Lemmas: ")
print(tryVerb.lemmas())
print()
print("Synset: try.v.01 Hierarchy")
hyper = lambda s: s.hypernyms()
list(tryVerb.closure(hyper))
```

```
    Synset: try.v.01
    Definition: make an effort or attempt
    Usage Examples:
    ['He tried to shake off his fears', 'The infant had essayed a few wobbly steps', 'The
    Lemmas:
    [Lemma('try.v.01.try'), Lemma('try.v.01.seek'), Lemma('try.v.01.attempt'), Lemma('try

    Synset: try.v.01 Hierarchy
    [Synset('act.v.01')]
```

Unlike nouns, for verbs, there is no similar top-level hierarchy. For verbs, the top-most verb in the hierarchy can be different for each verb.

```
# Instruction 7
print("Adjective: ")
print(wn.morphy("outstanding", wn.ADJ))
print("Verb: ")
print(wn.morphy("outstanding", wn.VERB))
print("Noun: ")
print(wn.morphy("outstanding", wn.NOUN))
print("Adverb: ")
print(wn.morphy("outstanding", wn.ADV))
```

```
    Adjective:
    outstanding
    Verb:
    None
    Noun:
    None
    Adverb:
    None
```

```
# Instruction 8
print(wn.synsets("travel"))
print(wn.synsets("walk"))
print()
travel = wn.synset("travel.v.01")
print("Travel def: " + travel.definition())
walk = wn.synset("walk.v.01")
print("Walk def: "+ walk.definition())
print()
print("Wu-Palmer Similarity Metric: " + str(wn.wup_similarity(travel, walk)))
print()
print("Lesk Algorithm: ")
travel_tokens = word_tokenize("I traveled to the bank.")
walk_tokens = word_tokenize("I walked to the bank.")
print(lesk(travel_tokens, "travel", "v"))
print(lesk(walk_tokens, "walk", "v"))
print()
print("Travel.v.06 Def: " + wn.synset('travel.v.06').definition())
print("Walk.v.05 Def: " + wn.synset('walk.v.05').definition())
```

```
[Synset('travel.n.01'), Synset('change_of_location.n.01'), Synset('locomotion.n.02'),
[Synset('walk.n.01'), Synset('base_on_balls.n.01'), Synset('walk.n.03'), Synset('walk

Travel def: change location; move, travel, or proceed, also metaphorically
Walk def: use one's feet to advance; advance by steps

Wu-Palmer Similarity Metric: 0.6666666666666666

Lesk Algorithm:
Synset('travel.v.06')
Synset('walk.v.05')

Travel.v.06 Def: travel from place to place, as for the purpose of finding work, prea
Walk.v.05 Def: give a base on balls to
```

Wu-Palmer similarity metric did a fair good job of judging how similar travel.v.01 and walk.v.01, since both definitions are fairly similar in that they mean to move forward, but not exactly the same since travel.v.01 means to change location while walk.v.01 means to advance by steps. Lesk algorithm did worse than Wu-Palmer when using the travel and walk verbs since when they were used interchangeably in the same sentence, the algorithm outputted synset of both verbs with completely and unrelated (in the case of walk) meanings.

Instruction 9

SentiWordNet is built on top on WordNet that gives 3 sentiment scores for each synset: positivity, negativity, objectivity. Possible use cases of this resource can be to help determine what the person saying the word feels, or to help generate an appropriate response based on the emotional state of the conversation.

```
exacerbate = swn.senti_synset('exacerbate.v.02')
print("Def: " + wn.synset('exacerbate.v.02').definition())
print("Senti-synsets:")
print(exacerbate)
print("Positive score: " + str(exacerbate.pos_score()))
print("Negative score: " + str(exacerbate.neg_score()))
print("Objective score: " + str(exacerbate.obj_score()))
print()
print("Polarity of each word in sentence:")
sentence = "the high cost of living only exacerbated the problem"
print(sentence)
tokens = sentence.split()
for token in tokens:
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    print(token + ": " + "Positive score = " + str(syn.pos_score()) + ", Negative score = '
```

```
    Def: exasperate or irritate
    Senti-synsets:
    <exacerbate.v.02: PosScore=0.0 NegScore=0.0>
    Positive score: 0.0
    Negative score: 0.0
    Objective score: 1.0

    Polarity of each word in sentence:
    the high cost of living only exacerbated the problem
    high: Positive score = 0.125, Negative score = 0.0, Objective score = 0.875
    cost: Positive score = 0.0, Negative score = 0.0, Objective score = 1.0
    living: Positive score = 0.0, Negative score = 0.0, Objective score = 1.0
    only: Positive score = 0.0, Negative score = 0.0, Objective score = 1.0
    exacerbated: Positive score = 0.0, Negative score = 0.25, Objective score = 0.75
    problem: Positive score = 0.0, Negative score = 0.625, Objective score = 0.375
```

Looks like most of the words in this sentence is deemed highly objective. Once exacerbate was made into past tense, the negativity score increased. A possible utility of knowing these scores in an NLP application could be to have a simple determination of the polarity of a sentence.

Instruction 10

Basically, a collocation is a sequence of words that appear together more often than what would be expected. An indication is that you cannot substitute synonyms in a collocation.

```
print("Text4 Collocations:")
text4.collocations()
print()
print("Mutual information of 'United States'")
vocab = len(set(text4))
text = ' '.join(text4.tokens)
us = text.count('United States') / vocab
print("p(United States) = " + str(us))
u = text.count('United') / vocab
print("p(United) = " + str(u))
s = text.count('States') / vocab
print("p(States) = " + str(s))
pmi = math.log2(us / (u * s))
print("pmi of United States = " + str(pmi))
```

```
    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 of 'United States'
    p(United States) = 0.015860349127182045
    p(United) = 0.0170573566084788
    p(States) = 0.03301745635910224
    pmi of United States = 4.815657649820885
```

The result of the mutual information formula for 'United States' is around 4.8. This indicates that 'United States' is likely a collocation since the result is positive. Before attempting this calculation, I was initially think that the result would be higher since according to the chapter notes 'fellow citizens' got a pmi of 8.2 and text4 is the inaugural corpus and the phrase 'United States' would be used very frequently.