```
import nltk
import math
nltk.download('popular')
nltk.download('sentiwordnet')
nltk.download('nps_chat')
nltk.download('webtext')


from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import *
text4
'''
TO DO:
- write 2-3 sentence summary of wordnet
- select noun, output all synsets
  - select one synset from list of synsets
    - extract definition, usage examples, lemmas
    - from selected synset, traverse up wordnet hierarchy and output as you go
    - write a couple of sentences of observations about how wordnet organizes nouns
  - output the following (or empty list):
    - hypernyms
    - hyponyms
    - meronyms
    - holonyms
    - antonyms
- select verb, output all synsets
  - extract definition, usage examples, lemmas
    - from selected synset, traverse up wordnet hierarchy and output as you go
    - write a couple of sentences of observations about how wordnet organizes verbs
    - use morphy to find as many different forms of the word as you can
- select two words you think might be similar
  - find specific synsets you are interested in
  - run wu-palmer similarity metric and lesk algorithm
  - write a couple of sentences with your observations
- write a couple of sentences about sentiwordnet describing
  - functionality
  - possible use cases
- select emotionally charged word
  - find sentisynsets
  - output each polarity score
- make up a sentence
  - output polarity for each word in the sentence
  - write a couple of sentences about your observations of the scores and the
    utility of knowing these scores in an NLP application
- write a couple of sentences about what a collocation is
  - output collocations for text4 (inaugural corpus) -> part of nltk
  - select a collocation identified by nltk
  - calculate mutual information
  - commentate on the results of the mutual info formula and your interpretation
'''
```

QUESTION 1 :)

Write 2-3 sentence summary of wordnet:

- WordNet groups words together based on their meanings and each synset represents a distinct concept. Words in this lexical database are related with synonymy/antonymy, hyperonomy/hyponymy, and meronymy/holonymy which makes semantic analysis easier due to words in close proximity to each other in the "word net" being made unambiguous.

QUESTION 2, 3, 4 :)

```
'''
[X]- select noun, output all synsets
  [X]- select one synset from list of synsets
    [X]- extract definition, usage examples, lemmas
    [X]- from selected synset, traverse up wordnet hierarchy and output as you go
    [X]- write a couple of sentences of observations about how wordnet organizes nouns
  [X]- output the following (or empty list):
    [X]- hypernyms
    [X]- hyponyms
    [X]- meronyms
    [X]- holonyms
    [X]- antonyms
'''
# run cell 2 (from nltk.corpus import wordnet as wn)

noun = "car"
noun_synset = wn.synsets(noun)
noun_definition = wn.synset('car.n.02').definition()
noun_example = wn.synset('car.n.02').examples()
noun_lemma = wn.synset('car.n.02').lemmas()

print(f"Noun: {noun}")
print(f"Synset: {noun_synset}")
print(f"Definition: {noun_definition}")
print(f"Example: {noun_example}")
print(f"Lemmas: {noun_lemma}")
# a hyponym of a given word has an IS A relationship with it
# meronym is the part of something
# holonym is the whole
# traverse up wordnet hierarchy and output as you go
car = wn.synset('car.n.02')
hyper = lambda s: s.hypernyms()
hypo = lambda s: s.hyponyms()
mero = lambda s: s.member_meronyms()
holo = lambda s: s.member_holonyms()
ant = noun_lemma[0].antonyms()
print(f"Car Hypernyms: {list(car.closure(hyper))}")
print(f"Car Hyponyms: {list(car.closure(hypo))}")
print(f"Car Meronyms: {list(car.closure(mero))}")
print(f"Car Holonyms: {list(car.closure(holo))}")
print(f"Car Antonyms: {ant}")
```

```
Noun: car
Synset: [Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'), Synset('cable_car.n.01')]
Definition: a wheeled vehicle adapted to the rails of railroad
Example: ['three cars had jumped the rails']
Lemmas: [Lemma('car.n.02.car'), Lemma('car.n.02.railcar'), Lemma('car.n.02.railway_car'), Lemma('car.n.02.railroad_car')]
Car Hypernyms: [Synset('wheeled_vehicle.n.01'), Synset('container.n.01'), Synset('vehicle.n.01'), Synset('instrumentality.n.03'), Synset
Car Hyponyms: [Synset('baggage_car.n.01'), Synset('cabin_car.n.01'), Synset('club_car.n.01'), Synset('freight_car.n.01'), Synset('guard
Car Meronyms: []
Car Holonyms: [Synset('train.n.01')]
Car Antonyms: []
```

Write a couple of sentences of observations about how wordnet organizes nouns:

- In WordNet, all nouns can be traced back to the same parent noun: entity. All the hyponyms of a given noun are more specific instances of that noun and the given noun itself is a more specific instance of its hypernyms. Nouns are more likely to have meronyms and holonyms than antonyms.

QUESTION 5, 6, 7 :)

```
'''
[X]- select verb, output all synsets
  [X]- extract definition, usage examples, lemmas
    [X]- from selected synset, traverse up wordnet hierarchy and output as you go
    [X]- write a couple of sentences of observations about how wordnet organizes verbs
    [X]- use morphy to find as many different forms of the word as you can
'''

# run cell 2 (from nltk.corpus import wordnet as wn)

verb = "see"
verb_synset = wn.synsets(verb)
verb_definition = wn.synset('see.v.01').definition()
verb_example = wn.synset('see.v.01').examples()
verb_lemma = wn.synset('see.v.01').lemmas()

print(f"Verb: {verb}")
print(f"Synset: {verb_synset}")
print(f"Definition: {verb_definition}")
print(f"Example: {verb_example}")
print(f"Lemmas: {verb_lemma}")

# traverse up wordnet hierarchy and output as you go
see = wn.synset('see.v.01')
hyper = lambda s: s.hypernyms()
hypo = lambda s: s.hyponyms()
mero = lambda s: s.member_meronyms()
holo = lambda s: s.member_holonyms()
ant = verb_lemma[0].antonyms()
print(f"Car Hypernyms: {list(see.closure(hyper))}")
print(f"Car Hyponyms: {list(see.closure(hypo))}")
print(f"Car Meronyms: {list(see.closure(mero))}")
print(f"Car Holonyms: {list(see.closure(holo))}")
print(f"Car Antonyms: {ant}")

print(f"As an adjective w/ Morphy: {wn.morphy('see', wn.ADJ)}")
print(f"As a noun w/ Morphy: {wn.morphy('see', wn.NOUN)}")
print(f"As a verb w/ Morphy: {wn.morphy('see', wn.VERB)}")
print(f"As an adverb w/ Morphy: {wn.morphy('see', wn.ADV)}")
```

```
Verb: see
Synset: [Synset('see.n.01'), Synset('see.v.01'), Synset('understand.v.02'), Synset('witness.v.02'), Synset('visualize.v.01'), Synset('se
Definition: perceive by sight or have the power to perceive by sight
Example: ['You have to be a good observer to see all the details', 'Can you see the bird in that tree?', 'He is blind--he cannot see']
Lemmas: [Lemma('see.v.01.see')]
Car Hypernyms: [Synset('perceive.v.01')]
Car Hyponyms: [Synset('behold.v.01'), Synset('catch_sight.v.01'), Synset('glimpse.v.01'), Synset('see.v.17')]
Car Meronyms: []
Car Holonyms: []
Car Antonyms: []
As an adjective w/ Morphy: None
As a noun w/ Morphy: see
As a verb w/ Morphy: see
As an adverb w/ Morphy: None
```

Write a couple of sentences of observations about how wordnet organizes verbs:

- It seems that WordNet organizes verbs differently than nouns because not all verbs have the same common parent. Hypernyms seem to be broader concepts of the verb and hyponyms are more specific concepts of the verb. Both categories seem to be also synonyms of whatever verb that is chosen for analysis.

QUESTION 8 :)

```
'''
[X]- select two words you think might be similar
  [X]- find specific synsets you are interested in
  [x]- run wu-palmer similarity metric and lesk algorithm
  [X]- write a couple of sentences with your observations
'''
# run cell 2 (from nltk.wsd import lesk)

bottle = wn.synset('bottle.n.01')
cup = wn.synset('cup.n.01')
#run = wn.synset('run.v.01')
#catch = wn.synset('catch.v.01')
#wn.path_similarity(bottle, cup)

# based on depth of two senses in taxonomy and of most specific common ancestor node
wup_sim = wn.wup_similarity(bottle, cup)
#wup_sim1 = wn.wup_similarity(run, catch)
print(f"Wu-Palmer Similarity: {wup_sim}")
#print(f"Wu-Palmer Similarity: {wup_sim1}")

# returns synset w/ highest number of overlapping words btwn context sentence
# and definitions in each synset for the target word
sent = ['Give', 'the', 'baby', 'her', 'bottle', '!']
print(f"Lesk Algorithm (bottle): {lesk(sent, 'bottle', pos='n')}")
```

```
Wu-Palmer Similarity: 0.8235294117647058
Lesk Algorithm (bottle): Synset('bottle.n.02')
```

Write a couple of sentences with your observations:

- It seems that the Wu-Palmer similarity metric is higher when the words have more similar meanings or are related under a topic like sports activities. The Lesk algorithm uses sentence analysis to find a synset of a given word that has the most similar sentences in its synset as the sentence chosen by the programmer. With both of these, we can disambiguate the meaning of words in a sentence more easily by choosing the definition of words that score highest with these two methods instead of solely relying on our own contextual analysis.

QUESTION 9 :)

Write a couple of sentences about sentiwordnet describing: functionality, possible use cases

- SentiWordNet is an added functionality to WordNet that gives positive, negative, and objective scores to every word. It has uses in the field of opinion mining, which is important for analysis of public opinions about important entities such as political candidates or businesses. It can also help anyone who feels that they struggle to accurately discern the meanings of subjective sentences.

```
'''
[X]- write a couple of sentences about sentiwordnet describing
  [X]- functionality
  [X]- possible use cases
[X]- select emotionally charged word
  [X]- find sentisynsets
  [X]- output each polarity score
[X]- make up a sentence
  [X]- output polarity for each word in the sentence
  - write a couple of sentences about your observations of the scores and the
    utility of knowing these scores in an NLP application
'''
# run cell 2 (from nltk.corpus import sentiwordnet as swn)

senti_list = list(swn.senti_synsets('satisfied'))
for item in senti_list:
  print(item)
  print("Positive score = ", item.pos_score())
  print("Negative score = ", item.neg_score())
  print("Objective score = ", item.obj_score())

print("\n")
#sent = ['Have', 'a', 'great', 'day', ',', 'honey', '!']
sent = 'have yourself a merry little christmas'
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
  print(token)
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    neg += syn.neg_score()
    pos += syn.pos_score()
    #print(syn)
    print("Positive score = ", syn.pos_score())
    print("Negative score = ", syn.neg_score())
    print("Objective score = ", syn.obj_score())
print("neg\tpos counts")
print(neg, '\t', pos)
```

```
<satisfy.v.01: PosScore=0.5 NegScore=0.0>
Positive score =  0.5
Negative score =  0.0
Objective score =  0.5
<satisfy.v.02: PosScore=0.375 NegScore=0.0>
Positive score =  0.375
Negative score =  0.0
Objective score =  0.625
<meet.v.04: PosScore=0.125 NegScore=0.0>
Positive score =  0.125
```

```
Negative score =  0.0
Objective score =  0.875
<satisfied.s.01: PosScore=0.375 NegScore=0.0>
Positive score =  0.375
Negative score =  0.0
Objective score =  0.625
<quenched.s.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0


have
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
yourself
a
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
merry
Positive score =  0.75
Negative score =  0.125
Objective score =  0.125
little
Positive score =  0.0
Negative score =  0.25
Objective score =  0.75
christmas
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
neg     pos counts
0.375    0.75
```

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

- Some of the scores given to the words I chose for the sentence and for the "emotionally charged" word surprised me because I was expecting either a higher positive score or a higher negative score. I realize that taking into account such a large dataset gives a broader sense of a word than my own singular perceptions of that word, so I believe that it is very useful to have a tool like this in an NLP application. Using something like SentiWordNet to know the scores of certain words would be helpful in a business or marketing field along with NLP because it would assist with market research, reputation management, and overall enhancement of customer experiences.

QUESTION 10 :)

Write a couple of sentences about what a collocation is:

- A collocation is a combination of words that cannot convey a certain concept separately. Because of this, collocations can be found by analyzing the frequency at which two or more words are found together in a certain corpus, even if the words themselves are infrequent.

```
'''
[X]- write a couple of sentences about what a collocation is
  [X]- output collocations for text4 (inaugural corpus) -> part of nltk
  [X]- select a collocation identified by nltk
  [X]- calculate mutual information
  - commentate on the results of the mutual info formula and your interpretation
'''
# run cell 1 and 2
#import math
#from nltk.book import *
#text4
text4.collocations()
colloc = 'oneanother'
# collocations: two words have a meaning greater than what the words can convey alone
# mutual information: P(x,y) / [P(x) * P(y)]
text = "".join(text4.tokens)

vocab = len(set(text4))
oa = text.count(colloc)/vocab
print("p(one another) = ", oa)
o = text.count('one')/vocab
print("p(one) = ", o)
```

```
a = text.count('another')/vocab
print('p(another) = ', a)
mut_info = math.log2(oa / (o * a))
print('mutual information = ', mut_info)
```

```
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
p(one another) =  0.0021945137157107233
p(one) =  0.06733167082294264
p(another) =  0.006683291770573566
mutual information =  2.2859133524709767
```

Commentate on the results of the mutual info formula and your interpretation:

- The mutual information formula is useful for guessing if a group of words is a collocation or not because it takes into account the possible infrequency of the collocation itself. The algorithm shows how dependent two variables are on each other, so if two words together are a collocation, they are likely to score as highly dependent on each other.

✓ 0s completed at 5:26 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.