WordNet Summary

Wordnet is a toolkit in python that acts as a big dictionary for the english language. Anything I could want about Parts of Speech, definitions, and root/unconjugated forms of words can be derived from functions of Word net. I can also use the Sysnet to get common synonyms and antonyms for words.

```
import nltk

nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')

nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('stopwords')

from nltk.book import *
text4
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import *
text4



# get all synsets of 'art' for my chosen noun
print('\n\n\n')
wn.synsets('art')
```

```
[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 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 sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[Synset('art.n.01'),
 Synset('art.n.02'),
 Synset('art.n.03'),
 Synset('artwork.n.01')]
```

```
# get a definition for the second noun synset, i like this definition more

print('Definition N:', wn.synset('art.n.02').definition(), '\n')

# extract examples
print('Examples:\n')
for i, example in enumerate(wn.synset('art.n.02').examples()):
  print(i, '.',example,'\n')
```

```
Definition N: the creation of beautiful or significant things

Examples:

0 . art does not need to be innovative to be good

1 . I was never any good at art

2 . he said that architecture is the art of wasting space beautifully
```

```
# extract lemmas
print('Lemmas:\n')
for i, lemma in enumerate(wn.synset('art.n.02').lemmas()):
  print(i, '.',lemma,'\n')
print(wn.synset('art.n.02').lemmas())
```

```
Lemmas:

0 . Lemma('art.n.02.art')

1 . Lemma('art.n.02.artistic_creation')

2 . Lemma('art.n.02.artistic_production')

[Lemma('art.n.02.art'), Lemma('art.n.02.artistic_creation'), Lemma('art.n.02.artistic_production')]
```

Hierarchy of nouns

It's cool seeing how the first definition of art (at the bottom) has less hypernyms because it includes 'works of art collectivley' and that definitions seems to lead to 'object'. But the definition of beauty and signifigance I chose lead to 'abstraction' before it got to entity. But the step by step of word hierarchy seems very logical even with a noun such as art.

```
art = wn.synset('art.n.02')
hyp = art.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('creation.n.01')
Synset('activity.n.01')
Synset('act.n.02')
Synset('event.n.01')
Synset('psychological_feature.n.01')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

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

```
Synset('creation.n.02')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

hypernyms, hyponyms, meronyms, holonyms, antonym below

```
art = wn.synset('art.n.02')
#for hyper in art.hypernyms():
print('hypernyms: ', art.hypernyms())
print('hyponyms: ', art.hyponyms())
print('meronyms: ', art.member_meronyms())
print('holonyms: ', art.member_holonyms())
print('Lemmas: ', art.lemmas())
print('Antonyms: ')
for x in art.lemmas():
  print(x.antonyms())
```

```
hypernyms:  [Synset('creation.n.01')]
hyponyms:  [Synset('arts_and_crafts.n.01'), Synset('ceramics.n.01'), Synset('decalcomania.n.02'), Synset('decoupage.n.02'), S
meronyms:  []
holonyms:  []
Lemmas:  [Lemma('art.n.02.art'), Lemma('art.n.02.artistic_creation'), Lemma('art.n.02.artistic_production')]
Antonyms:
[]
[]
[]
```

Verb

```
print(wn.synsets('love'))
love3 = wn.synset('love.v.03')
```

```
[Synset('love.n.01'), Synset('love.n.02'), Synset('beloved.n.01'), Synset('love.n.04'), Synset('love.n.05'), Synset('sexual_l
```

Definition, examples, lemmas

```
# get a definition for the third verb synset, i like this definition more

print('Definition N:', love3.definition(), '\n')

# extract examples
print('Examples:\n')
for i, example in enumerate(love3.examples()):
  print(i, '.',example,'\n')

  # extract lemmas
print('Lemmas:\n')
for i, lemma in enumerate(love3.lemmas()):
  print(i, '.',lemma,'\n')
print(love3.lemmas())
```

```
Definition N: have a great affection or liking for

Examples:

0 . I love French food

1 . She loves her boss and works hard for him

Lemmas:

0 . Lemma('love.v.01.love')

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

Only has 1 hypernym Weird!!

```
hyper = lambda s: s.hypernyms()
list(love3.closure(hyper))
love3.hypernyms()
```

```
[Synset('love.v.01')]
```

It seems the hierarchy doesn't really exist for verbs. They are the root word and mean what the action they are representing. The first definition of love didn't have any hypernyms at all so it just means what it means. The second definition does have a hypernym but of its own synset and definiton 3 point back to definition 1. Seem's like verbs hierarchy can point to other verbs of the same word but not go upo the tree the same way nouns do.

I think i chose a bad word

```
print(wn.morphy('love', wn.VERB))
```

```
love
```

```
glow = wn.synset('glow.v.03')

burn = wn.synset('burn.v.01')   #Change to v.02 for 100% similarity
print('glow3 ',glow.definition())
print('burn1 ',burn.definition())

print('path similarity', glow.path_similarity(burn))
print('Wu-Palmer similarity',wn.wup_similarity(glow, burn))
```

```
glow3  shine intensely, as if with heat
burn1  destroy by fire
path similarity 0.2
Wu-Palmer similarity 0.3333333333333333
```

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

```
print('\nlesk: ', lesk(burn.definition(), 'glow'))
print(wn.synset('radiance.n.01').definition())
```

```
Synset('freshness.n.03') an alert and refreshed state
Synset('luminescence.n.02') light from nonthermal sources
Synset('incandescence.n.01') the phenomenon of light emission by a body as its temperature is raised
Synset('glow.n.04') a feeling of considerable warmth
Synset('glow.n.05') a steady even light without flames
Synset('radiance.n.01') the amount of electromagnetic radiation leaving or arriving at a point on a surface
Synset('gleam.n.01') an appearance of reflected light
Synset('glow.v.01') emit a steady even light without flames
Synset('glow.v.02') have a complexion with a strong bright color, such as red or pink
Synset('burn.v.02') shine intensely, as if with heat
Synset('glow.v.04') be exuberant or high-spirited
Synset('glow.v.05') experience a feeling of well-being or happiness, as from good health or an intense emotion

lesk:  Synset('radiance.n.01')
the amount of electromagnetic radiation leaving or arriving at a point on a surface
```

Similarity

I really like what the lesk algorithm find for similarity. As seen above, radiance does tend to be used to describe a very shiny glow or a light from a burning flame. I think the Wu-Palmer works better for taxonomy of things like nouns better than ideas. Lesk seems to be more idea / concept focused and does a good job at matching that even if the definition of radiance isn't quite a perfect middle ground between the words, it's a good connecting idea.

```
love1 = wn.synset('love.n.01')
print(love1.definition())
```

```
love1 = swn.senti_synset('love.n.01')
print(love1)
print("Positive score = ", love1.pos_score())
print("Negative score = ", love1.neg_score())
print("Objective score = ", love1.obj_score())

loveV = wn.synset('love.v.01')
print('\n\nVerb:\n', loveV.definition())

loveV = swn.senti_synset('love.v.01')
print(loveV)
print("Positive score = ", loveV.pos_score())
print("Negative score = ", loveV.neg_score())
print("Objective score = ", loveV.obj_score())
```

```
    a strong positive emotion of regard and affection
    <love.n.01: PosScore=0.625 NegScore=0.0>
    Positive score =  0.625
    Negative score =  0.0
    Objective score =  0.375


    Verb:
     have a great affection or liking for
    <love.v.01: PosScore=0.5 NegScore=0.0>
    Positive score =  0.5
    Negative score =  0.0
    Objective score =  0.5
```

Show score for all synsets of love, only 1 negative at the bottom mostly positive, lots of object variance.

```
for ss in swn.senti_synsets('love'):
  print(ss, "Objective score = ", ss.obj_score())
```

```
    <love.n.01: PosScore=0.625 NegScore=0.0> Objective score =  0.375
    <love.n.02: PosScore=0.375 NegScore=0.0> Objective score =  0.625
    <beloved.n.01: PosScore=0.125 NegScore=0.0> Objective score =  0.875
    <love.n.04: PosScore=0.25 NegScore=0.0> Objective score =  0.75
    <love.n.05: PosScore=0.0 NegScore=0.0> Objective score =  1.0
    <sexual_love.n.02: PosScore=0.0 NegScore=0.0> Objective score =  1.0
    <love.v.01: PosScore=0.5 NegScore=0.0> Objective score =  0.5
    <love.v.02: PosScore=1.0 NegScore=0.0> Objective score =  0.0
    <love.v.03: PosScore=0.625 NegScore=0.0> Objective score =  0.375
    <sleep_together.v.01: PosScore=0.375 NegScore=0.125> Objective score =  0.5
```

```
lyric = 'I would rather feel your wrath than feel anothers passion'
tokens = lyric.split()

#taken from class github
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        print(token, 'neg: ', syn.neg_score())
        print(token, 'pos: ', syn.pos_score())
```

```
    I neg:  0.0
    I pos:  0.0
    rather neg:  0.5
    rather pos:  0.5
    feel neg:  0.0
    feel pos:  0.125
    wrath neg:  0.5
    wrath pos:  0.0
    feel neg:  0.0
    feel pos:  0.125
    passion neg:  0.0
    passion pos:  0.5
```

SentiWordNet

The functionality of SentiWordNet is to be able to analyze the sentiment behind words with three scores. Those scores being positive, negative, and an object score. It. can be used for analyzing reviews and if the customers said postive or negative things. I see this as the stepping stone

to being able to understand the subtext of a sentence and the meaning behind it.

Observations The sentence I created didn't have sentiment behind certain words such as 'would', 'your', and 'anothers'. These words describe possesion of the describing words but I guess aren't needed nor aid to sentiment in the eyes of this program. It does suck that the scores are perfect fractions for the words I chose but it is interesting to see which words are 50/50 on scores. I can also see passion isn't 50/50 but is 50% positive and 0% negative. It will be much easier to use and build sentiWordNet programmings understanding which words it finds positive and negative so I can avoid analyzing those words if I want them to take on a different meaning.

```
# Collocations
This is where we learn that a sentence is greater than the sum o
Mostly it seems like collocations recognizes bigrams and that mu
occurances of such bigrams has an influential or greater inportar
text. Even when I read poetry books which little reocurring words
will pair similar words in new ways to continue a theme. This su
reused words should be paid more attentnion to and cary more wei
those extra weighted bigrams to the front seems to be the goal a
case of collocations.
```

Collocations

This is where we learn that a sentence is greater than the sum of it's parts. Mostly it seems like collocations recognizes bigrams and that multiple occurances of such bigrams has an influential or greater inportance to the text. Even when I read poetry books which little reocurring words, some poets will pair similar words in new ways to continue a theme. This suggests those reused words should be paid more attentnion to and cary more weight. Bringing those extra weighted bigrams to the front seems to be the goal and best use case of collocations.

```
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
```

Choosing God bless to perform mutual information calculation on, seems half as likely as Old World. **This looks a lot like a bunch of weird decimals** But, I understand dividing by a tiny number is like multiplying and making the result larger. What I am curious about is how low of a percentage the initial collocations are and yet the formula produces a significant amount of occurances for the two words to be together.

```
#From the class github
text = ' '.join(text4.tokens)

import math
vocab = len(set(text4))
gb = text.count('God bless')/vocab
print("p(God bless) = ",gb )
g = text.count('God')/vocab
print('p(God) = ', g)
b = text.count('bless')/vocab
print("p(bless) = ", b)
pmi = math.log2(gb / (b * g))
print('pmi = ', pmi, '\n\n')

vocab = len(set(text4))
gb = text.count('Old World')/vocab
print("p(Old World) = ",gb )
g = text.count('Old')/vocab
print('p(Old) = ', g)
b = text.count('World')/vocab
print("p(World) = ", b)
pmi = math.log2(gb / (b * g))
print('pmi = ', pmi)
```

```
p(God bless) =  0.0016957605985037406
p(God) =  0.011172069825436408
p(bless) =  0.0085785536159601
pmi =  4.145157780720282


p(Old World) =  0.000997506234413965
p(Old) =  0.0010972568578553616
p(World) =  0.0017955112219451373
pmi =  8.983886091037398
```

✓ 0s completed at 6:40 PM