WordNet

September 24, 2022

1 WordNet

1.1 By: Bridgette Bryant (NetID: BMB180001), for CS 4395.001 Human Language Technologies

1.1.1 WordNet:

WordNet was started/organized as a project by George Miller at Princeton University. It's goal was to support theories of human semantic memory which suggested that people organize concepts mentally in some sort of hierarchy. WordNet is now a lexical database of nouns, verbs, adjectives, and adverbs which provides short definitions (called glosses) and examples. WordNet uses synsets, which are synonym sets for each word. Theses synsets have the following relations:

- hypernym (higher): a plant is a hypernym of tree
- hyponym (lower): a tree is a hyponym of plant
- meronym (part of): graphics card is a meronym of computer
- holonym (whole): computer is a holonym of graphics card
- troponym (more specific action): stealthy is a troponym of walk

Keep in mind not all synset lemmas will have an entry for every relation above, the nouns are the most connected compared to other types of words.

1.1.2 Importing Wordnet

[1]:
```python
# Import the wordnet library from nltk corpus
from nltk.corpus import wordnet as wn
# Import the lesk algorithm from nltk wsd
from nltk.wsd import lesk
# Import the SentiWordNet library from nltk corpus
from nltk.corpus import sentiwordnet as swn
# Import the text4 from the nltk book library
from nltk.book import text4
# Import the library math from python
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
```

1.1.3 Displaying Synsets of a Noun

[2]:
```python
# Below is a an example of selecting a noun and displaying all synsets.
# I chose turtle.
wn.synsets('turtle')
```

[2]:
```
[Synset('turtleneck.n.01'),
 Synset('turtle.n.02'),
 Synset('capsize.v.01'),
 Synset('turtle.v.02')]
```

1.1.4 Noun Synset's Information, Hierarchy, and Relationships

Displaying Information

[3]:
```python
# Display the turtle synset's definition
wn.synset('turtle.n.02').definition()
```

[3]:
```
'any of various aquatic and land reptiles having a bony shell and flipper-like
limbs for swimming'
```

[4]:
```python
# Display the turtle synset's examples
wn.synset('turtle.n.02').examples()
```

[4]:
```
[]
```

[5]:
```python
# Display the turtle synset's lemmas
wn.synset('turtle.n.02').lemmas()
```

[5]:
```
[Lemma('turtle.n.02.turtle')]
```

Hierarchy

[6]:
```python
# Traverse through the Hierarchy from synset
snake = wn.synset('turtle.n.02')
hyper = lambda s: s.hypernyms()
list(snake.closure(hyper))
```

[6]:
```
[Synset('chelonian.n.01'),
 Synset('anapsid.n.01'),
 Synset('reptile.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')]
```

Wordnet Hierarchy Organization The wordnet hierarchy seems to move up one classification or umbrella term. For example looking at the turtle noun synset hierarchy above, we go from turtle to anapsid, to reptile, to vertebrate, and eventually all the away to entity. I think it could be a good way of getting umbrella and word classification for words as it is very organized and simple to traverse through.

Relationships

[7]:
```
# To output hypernyms for the turtle synset
wn.synset('turtle.n.02').hypernyms()
```

[7]:
```
[Synset('chelonian.n.01')]
```

[8]:
```
# To output hyponyms for the turtle synset
wn.synset('turtle.n.02').hyponyms()
```

[8]:
```
[Synset('box_turtle.n.01'),
 Synset('cooter.n.01'),
 Synset('mud_turtle.n.01'),
 Synset('painted_turtle.n.01'),
 Synset('red-bellied_terrapin.n.01'),
 Synset('sea_turtle.n.01'),
 Synset('slider.n.03'),
 Synset('snapping_turtle.n.01'),
 Synset('soft-shelled_turtle.n.01'),
 Synset('terrapin.n.01'),
 Synset('tortoise.n.01')]
```

[9]:
```
# To output meronyms for the turtle synset
wn.synset('turtle.n.02').part_meronyms()
```

[9]:
```
[Synset('carapace.n.01'), Synset('plastron.n.05')]
```

[10]:
```
# To output holonyms for the turtle synset
wn.synset('turtle.n.02').part_holonyms()
```

[10]:
```
[]
```

[11]:
```python
# To output antonyms for the snake synset
wn.synset('turtle.n.02').lemmas()[0].antonyms()
```

[11]:
```
[]
```

1.1.5 Displaying Synsets of a Verb

1.1.6 Verb Synset's Information and Hierarchy

Displaying Information

[12]:
```python
# Below is a an example of selecting a verb and displaying all synsets.
# I chose walk.
wn.synsets('walk')
```

[12]:
```
[Synset('walk.n.01'),
 Synset('base_on_balls.n.01'),
 Synset('walk.n.03'),
 Synset('walk.n.04'),
 Synset('walk.n.05'),
 Synset('walk.n.06'),
 Synset('walk_of_life.n.01'),
 Synset('walk.v.01'),
 Synset('walk.v.02'),
 Synset('walk.v.03'),
 Synset('walk.v.04'),
 Synset('walk.v.05'),
 Synset('walk.v.06'),
 Synset('walk.v.07'),
 Synset('walk.v.08'),
 Synset('walk.v.09'),
 Synset('walk.v.10')]
```

[13]:
```python
# Display the walk synset's definition
wn.synset('walk.v.01').definition()
```

[13]:
```
"use one's feet to advance; advance by steps"
```

[14]:
```python
# Display the walk synset's examples
wn.synset('walk.v.01').examples()
```

[14]:
```
["Walk, don't run!",
 'We walked instead of driving',
 'She walks with a slight limp',
 'The patient cannot walk yet',
 'Walk over to the cabinet']
```

[15]:
```python
# Display the walk synset's examples
wn.synset('walk.v.01').lemmas()
```

[15]: [Lemma('walk.v.01.walk')]

[16]:
```python
# Traverse through the Hierarchy from synset
walk = wn.synset('walk.v.01')
hyper = lambda s: s.hypernyms()
list(walk.closure(hyper))
```

[16]: [Synset('travel.v.01')]

[17]:
```python
# Here is what happends when we traverse through travel
walk = wn.synset('travel.v.01')
hyper = lambda s: s.hypernyms()
list(walk.closure(hyper))
```

[17]: []

[18]:
```python
# Here is traversing the Hierarchy from synset for wave as a verb to get more␣
↪output
wave = wn.synset('wave.v.01')
hyper = lambda s: s.hypernyms()
list(wave.closure(hyper))
```

[18]:
```
[Synset('gesticulate.v.01'),
 Synset('communicate.v.02'),
 Synset('interact.v.01'),
 Synset('act.v.01')]
```

Wordnet Hierarchy Organization The wordnet hierarchy for verbs is not as robust as nouns and don't have a top level synset (like entity.n.01). For example looking at the wave verb synset hierarchy above, we go from wave to gesticulate, to communicate, to interact, and eventually all the away to act. Whereas the walk verb synset only had travel. I think it could be a good way of getting umbrella and word classification for words as it is very organized and simple to traverse through. But it is much more limited that than the nouns and that should be accounted for that not all verbs will neccessarily have a hierarchy.

1.1.7 Morphy

[19]:
```python
# Using morphy to find as many different forms of the word walk as possible
wn.morphy('walk', wn.NOUN)
```

[19]: 'walk'

[20]:
```python
wn.morphy('walks', wn.NOUN)
```

[20]: 'walk'

[21]:
```python
wn.morphy('walk', wn.VERB)
```

[21]: 'walk'

[22]:
```
wn.morphy('walks', wn.VERB)
```

[22]: 'walk'

[23]:
```
wn.morphy('walked', wn.VERB)
```

[23]: 'walk'

[24]:
```
wn.morphy('walking', wn.VERB)
```

[24]: 'walk'

1.2 Similarities Between Words

I have choosen words turtle and snake, which are very similar. Below I have choosen the animal-based synsets. Run the Wu-Palmer similarity metric and Lesk algorithm. Write a couple sentences with your observations.

[25]:
```
# Getting the synset for turtle
wn.synsets("turtle")
```

[25]:
```
[Synset('turtleneck.n.01'),
 Synset('turtle.n.02'),
 Synset('capsize.v.01'),
 Synset('turtle.v.02')]
```

[26]:
```
# Choosing the sysnset 'turtle.n.02'
wn.synset('turtle.n.02').definition()
```

[26]:
```
'any of various aquatic and land reptiles having a bony shell and flipper-like
limbs for swimming'
```

[27]:
```
# Getting the synset for snake
wn.synsets("snake")
```

[27]:
```
[Synset('snake.n.01'),
 Synset('snake.n.02'),
 Synset('snake.n.03'),
 Synset('hydra.n.02'),
 Synset('snake.n.05'),
 Synset('snake.v.01'),
 Synset('snake.v.02'),
 Synset('snake.v.03')]
```

[28]:
```
# Choosing the sysnset 'snake.n.01'
wn.synset('snake.n.01').definition()
```

[28]: 'limbless scaly elongate reptile; some are venomous'

[29]:
```python
# Using the WordNet similarity to see how similar the two synsets are
wn.path_similarity(wn.synset('turtle.n.02'),wn.synset('snake.n.01'))
```

[29]: 0.16666666666666666

1.2.1 Wu-Palmer Similarity Metric

Running the Wu-Palmer similarity metric on the turtle and tortoise synsets.

[30]:
```python
# Using the WordNet Wu-Palmer simularity metric to see how similar the two␣
↪synsets are
wn.wup_similarity(wn.synset('turtle.n.02'),wn.synset('snake.n.01'))
```

[30]: 0.8

1.2.2 Lesk Algorithm

Running the Lesk Algorithm on the turtle and snake synsets

[31]:
```python
# Here is a function which takes a string and prints all the definitions
# For all the found synsets in WordNet
def getDef(s):
    for syn in wn.synsets(str(s)):
        print(syn, syn.definition())
```

[32]:
```python
# We will look at the definitions for turtle using our function above
getDef("turtle")
```

```
Synset('turtleneck.n.01') a sweater or jersey with a high close-fitting collar
Synset('turtle.n.02') any of various aquatic and land reptiles having a bony
shell and flipper-like limbs for swimming
Synset('capsize.v.01') overturn accidentally
Synset('turtle.v.02') hunt for turtles, especially as an occupation
```

[33]:
```python
# We will look at the definitions for snake using our function above
getDef("snake")
```

```
Synset('snake.n.01') limbless scaly elongate reptile; some are venomous
Synset('snake.n.02') a deceitful or treacherous person
Synset('snake.n.03') a tributary of the Columbia River that rises in Wyoming and
flows westward; discovered in 1805 by the Lewis and Clark Expedition
Synset('hydra.n.02') a long faint constellation in the southern hemisphere near
the equator stretching between Virgo and Cancer
Synset('snake.n.05') something long, thin, and flexible that resembles a snake
Synset('snake.v.01') move smoothly and sinuously, like a snake
Synset('snake.v.02') form a snake-like pattern
Synset('snake.v.03') move along a winding path
```

[34]:
```python
# Creating a few sentences to run the algorithm on
sent1 = ['There', 'is', 'a', 'wrinkle', 'in', 'my', 'turtleneck', '.']
sent2 = ['There', 'is', 'a', 'turtle', 'in', 'my', 'yard', '.']
sent3 = ['I', 'turtled', 'over', 'my', 'car', '.']
sent4 = ['Those', 'crazy', 'poachers', 'keep', 'going', 'turtling', 'in',␣
    ↪'our', 'pond', '!']
sent5 = ['There', 'is', 'a', 'snake', 'in', 'my', 'yard', '.']
sent6 = ['That', 'snake', 'stole', 'some', 'money', 'again', '!']
sent7 = ['Do', 'you', 'know', 'about', 'the', 'snake', 'river', '?']
sent8 = ['Can', 'you', 'see', 'the', 'snake', 'in', 'the', 'stars', '?']
sent9 = ['I', 'bought', 'this', 'new', 'drain', 'snake', 'I', 'saw', 'on',␣
    ↪'TV', '.']
sent10 = ['He', 'can', 'easily', 'snake', 'through', 'the', 'muddy', 'water', '.
    ↪']
sent11 = ['They', 'snake', 'around', 'the', 'yard', '.']
sent12 = ['The', 'hiking', 'trail', 'snakes', 'along', '.']
```

[35]:
```python
# Testing the Lesk Algorithm on the different sentences
print(lesk(sent1, 'turtle'))
```

```
Synset('turtleneck.n.01')
```

[36]:
```python
print(lesk(sent2, 'turtle'))
```

```
Synset('turtleneck.n.01')
```

[37]:
```python
print(lesk(sent3, 'turtle'))
```

```
Synset('turtleneck.n.01')
```

[38]:
```python
print(lesk(sent4, 'turtle'))
```

```
Synset('turtleneck.n.01')
```

[39]:
```python
print(lesk(sent5, 'snake'))
```

```
Synset('snake.v.01')
```

[40]:
```python
print(lesk(sent6, 'snake'))
```

```
Synset('snake.v.01')
```

[41]:
```python
print(lesk(sent7, 'snake'))
```

```
Synset('snake.v.01')
```

[42]:
```python
print(lesk(sent8, 'snake'))
```

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

[43]:
```python
print(lesk(sent9, 'snake'))
```

```
Synset('snake.v.01')
```

[44]:
```
print(lesk(sent10, 'snake'))
```

```
Synset('snake.v.01')
```

[45]:
```
print(lesk(sent11, 'snake'))
```

```
Synset('snake.v.01')
```

[46]:
```
print(lesk(sent12, 'snake'))
```

```
Synset('snake.v.03')
```

1.2.3 Observations:

The Wu-Palmer algorithm was very effective at seeing the similarity between turtle and snake. Especially when compared to the path similarity.

The Lesk algorithm performed very horribly on the turtle synset, as it only got 1 out of 4 sentences correct. It also performed terribly on the snake synset as it got 1 out of 8 sentences correct. Personally, I believe the correct ones were by chance and the algorithm really seemed to struggle with the difference in nouns. I found this quite interesting and humorous, if I added more parameters it might have helped. I suppose the synsets snaked their way through the algorithm.

1.3 SentiWordNet

SentiWordNet builds upon WordNet's synset resources and adds opinion attibutes to words/sentences. They all have 3 sentiment scores for each synset, positivity, negativity, and objectivity. The sum of all 3 scores will always equate to 1, and they can only be imbetween values 0 and 1 (just like a percentage). SentiWordNet could be used to attempt to read tone of the text's emotions, detect abuse/bullying/toxic behavior, or possibly spam (which is usually filled with emotionally charged and urgent words).

1.3.1 SentiWordNet with Words

[47]:
```
# I have written a function to print polarity scores of a given sent_synset
def printScores(sent_syn):
    try:
        print(sent_syn)
        print("Positive Score = ", sent_syn.pos_score())
        print("Negative Score = ", sent_syn.neg_score())
        print("Objective Score = ", sent_syn.obj_score())
    except:
        print("Not a sent_synset!")
```

[48]:
```
# I have written a function to get scores of a given synset
def getScores(w):
    try:
        word = swn.senti_synset(w)
```

```
        # Calling the printScores function
        printScores(word)
    except:
        print("Not a synset!")
```

[49]:
```
# I have selected the emotionally charged word 'hate'
# Using my getScores function from above
getScores("hate.v.01")
```

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

[50]:
```
# I have selected the emotionally charged word 'hate'
# Using my getScores function from above
getScores("love.v.01")
```

```
<love.v.01: PosScore=0.5 NegScore=0.0>
Positive Score =  0.5
Negative Score =  0.0
Objective Score =  0.5
```

[51]:
```
# I have selected the emotionally charged word 'hate'
# Using my getScores function from above
getScores("passionate.a.01")
```

```
<passionate.a.01: PosScore=0.375 NegScore=0.375>
Positive Score =  0.375
Negative Score =  0.375
Objective Score =  0.25
```

1.3.2 SentiWordNet with Sentences

[52]:
```
# I wrote some sentences to try with SentiWordNet
sent1 = "I hated that terrible car it was always breaking down!"
sent2 = "That book had so much love in it I just had to keep reading."
sent3 = "He was very passionate about his job, but I do not think he loved it."
sent4 = "People are always so quick to hate a new good thing."
```

[53]:
```
# This is a function to output the polarity for each word given a sentence
def sentPolarity(sent):
    try:
        # Split the sentence in to word tokens
        tokens = sent.split()
        # For each word token in the tokens list
        for token in tokens:
            # Create list of possible matching synsets for the current token
            syn_list = list(swn.senti_synsets(token))
```

```
            if syn_list:
                # Print the name and score of the current token
                print(token)
                printScores(syn_list[0])
                print()


    except:
        print("Something went wrong, with sentPolarity function!")
```

[54]:
```
# This is a function to calculate the polarity sum given a sentence
def sentPolaritySum(sent):
    negSum = 0
    posSum = 0
    try:
        # Split the sentence in to word tokens
        tokens = sent.split()
        # For each word token in the tokens list
        for token in tokens:
            # Create list of possible matching synsets for the current token
            syn_list = list(swn.senti_synsets(token))
            if syn_list:
                # For the current senti_syn
                cur_syn = syn_list[0]
                # Add the negative to the sum
                negSum += cur_syn.neg_score()
                # Add the positive to the sum
                posSum += cur_syn.pos_score()

        print("Sentence:\n", sent)
        print("Negative Score = ", negSum)
        print("Positive Score = ", posSum)



    except:
        print("Something went wrong, with sentPolaritySum function!")
```

Sentence 1:

[55]:
```
# Use the function to get the scores of each word in our sentence
sentPolarity(sent1)
```

```
I
<iodine.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0
```

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

terrible
<awful.s.02: PosScore=0.0 NegScore=0.625>
Positive Score =  0.0
Negative Score =  0.625
Objective Score =  0.375

car
<car.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

it
<information_technology.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

was
<washington.n.02: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

always
<always.r.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

breaking
<breakage.n.03: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0
```

[56]:

```
# Use the function to get the sum of the scores for each word in our sentence
sentPolaritySum(sent1)
```

```
Sentence:
 I hated that terrible car it was always breaking down!
```

```
Negative Score =  1.375
Positive Score =  0.0
```

Sentence 2:

[57]:
```python
# Use the function to get the scores of each word in our sentence
sentPolarity(sent2)
```

```
book
<book.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

had
<have.v.01: PosScore=0.25 NegScore=0.0>
Positive Score =  0.25
Negative Score =  0.0
Objective Score =  0.75

so
<sol.n.03: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

much
<much.n.01: PosScore=0.125 NegScore=0.125>
Positive Score =  0.125
Negative Score =  0.125
Objective Score =  0.75

love
<love.n.01: PosScore=0.625 NegScore=0.0>
Positive Score =  0.625
Negative Score =  0.0
Objective Score =  0.375

in
<inch.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

it
<information_technology.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0
```

```
I
<iodine.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

just
<just.a.01: PosScore=0.625 NegScore=0.0>
Positive Score =  0.625
Negative Score =  0.0
Objective Score =  0.375

had
<have.v.01: PosScore=0.25 NegScore=0.0>
Positive Score =  0.25
Negative Score =  0.0
Objective Score =  0.75

keep
<support.n.06: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0
```

[58]:
```
# Use the function to get the sum of the scores for each word in our sentence
sentPolaritySum(sent2)
```

```
Sentence:
 That book had so much love in it I just had to keep reading.
Negative Score =  0.125
Positive Score =  1.875
```

Sentence 3:

[59]:
```
# Use the function to get the scores of each word in our sentence
sentPolarity(sent3)
```

```
He
<helium.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

was
<washington.n.02: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
```

```
Objective Score =  1.0

very
<very.s.01: PosScore=0.5 NegScore=0.0>
Positive Score =  0.5
Negative Score =  0.0
Objective Score =  0.5

passionate
<passionate.a.01: PosScore=0.375 NegScore=0.375>
Positive Score =  0.375
Negative Score =  0.375
Objective Score =  0.25

about
<about.s.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

but
<merely.r.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

I
<iodine.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

do
<bash.n.02: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

not
<not.r.01: PosScore=0.0 NegScore=0.625>
Positive Score =  0.0
Negative Score =  0.625
Objective Score =  0.375

think
<think.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
```

```
Objective Score =  1.0

he
<helium.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

loved
<love.v.01: PosScore=0.5 NegScore=0.0>
Positive Score =  0.5
Negative Score =  0.0
Objective Score =  0.5
```

[60]:
```
# Use the function to get the sum of the scores for each word in our sentence
sentPolaritySum(sent3)
```

```
Sentence:
 He was very passionate about his job, but I do not think he loved it.
Negative Score =  1.0
Positive Score =  1.375
```

Sentence 4:

[61]:
```
# Use the function to get the scores of each word in our sentence
sentPolarity(sent4)
```

```
People
<people.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

are
<are.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

always
<always.r.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

so
<sol.n.03: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
```

```
Negative Score =  0.0
Objective Score =  1.0

quick
<quick.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

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

a
<angstrom.n.01: PosScore=0.0 NegScore=0.0>
Positive Score =  0.0
Negative Score =  0.0
Objective Score =  1.0

new
<new.a.01: PosScore=0.375 NegScore=0.0>
Positive Score =  0.375
Negative Score =  0.0
Objective Score =  0.625

good
<good.n.01: PosScore=0.5 NegScore=0.0>
Positive Score =  0.5
Negative Score =  0.0
Objective Score =  0.5
```

[62]:
```python
# Use the function to get the sum of the scores for each word in our sentence
sentPolaritySum(sent4)
```

```
Sentence:
 People are always so quick to hate a new good thing.
Negative Score =  0.375
Positive Score =  1.0
```

1.3.3 Observations

Overall for how simple the processing is I think for most uses SentWordNet does a good job of labeling words/phrases as positive and negative. However, it does have it's limitations and doesn't handle context heavy sentences very well. As far as uses it could easily be used to detect spam or toxic/bullying in chats, emails, and other text based messaging systems. It could possibly detect extreme reviews as well (good or bad) such as for Google/Amazon product reviews to filter out automated or exaggerated responses.

1.4 Collocation

Collocations are words which appear together forming a greater meaning than by themselves. As an example the phrase 'dead person' has much more meaning than either words 'dead' or 'person' by themselves. It is common for phrases and titles to be collocations.

[63]:
```
# Here is the output of collocations of text4 from nltk.book
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
```

1.4.1 Mutual Information

[64]:
```
# I have selected 'Inidian tribes' to calculate mutual information on
# Holds the text4 tokens
text = ' '.join(text4.tokens)
# Holds the total number of vocabulary in text4
vocab = len(set(text4))
# Holds the p(Inidian Tribes) (the count/vocab of the collocation)
indTrb = text.count('Indian tribes')/vocab
# Holds the p(Indian) (the count/vocab of just Indian)
ind = text.count('Indian')/vocab
# Holds the p(tribes) (the count/vocab of just tribes
trb = text.count('tribes')/vocab
# Calculates the pmi ( log ( (P(x,y)) / (P(x) * P(y)) ) )
pmi = math.log2(indTrb / (ind * trb))
```

[65]:
```
# Prints our mutual calculation results
print("p('Indian tribes') = ", indTrb)
print("p('Indian') = ", ind)
print("p('tribes') = ", trb)
print("pmi = ", pmi)
```

```
p('Indian tribes') =  0.000598503740648379
p('Indian') =  0.0010972568578553616
p('tribes') =  0.000598503740648379
pmi =  9.831882997592349
```

The results above show a PMI of 9 which is fairly high and positive, this implies the that 'Indian tribes' is likely to be a collocation. I agree with this output as 'Indian tribes' is a collocation as 'Indian' and 'tribes' by themeselves don't give the same/as meaningful meaning as 'Indian tribes'. If the result was close to 0 it would imply they are independent, if it was negative it would imply they are not a collocation.