Arjun Balasubramanian axb200075

In [1]:

```
import nltk
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.collocations import *
import math
from nltk.book import text4
#nltk.download('sentiwordnet')
```

```
*** 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

Wordnet Summary: Wordnet is database which holds relations between words. The relations tell us how words relate to each other with them being synonyms, hyponyms, and more. In addition each word has details reguarding defnition, part of speech, and other details.

In [2]:

```
#2
word = "rabbit" ## select noun
synsets = wn.synsets(word) ## get the synsets
print(synsets) ## output the synsets
```

```
[Synset('rabbit.n.01'), Synset('lapin.n.01'), Synset('rabbit.n.03'), Synset('rabbit.v.0
1')]
```

In [3]:

```
#3
synset = synsets[0] ## choose a synset

print(word + " : "+ synset.definition()) ## output the definition
print("\n")

print("examples" + " :")
for ex in synset.examples(): ## iterate through the examples
    print(ex) ## output the example
print("\n")

print("lemmas" + " : ")
for lem in synset.lemmas(): ## iterate through the lemmas
    print(lem) ## output the lemma
print("\n")

print("Traversing up we get :")
```

```
while len(synset.hypernyms()) > 0: ## terminating condition is when the hypernym list h
    print (synset.hypernyms()[0]) ## print the hypernym
    synset = synset.hypernyms()[0] ## set the current noun to the hypernym to keep iter
```

```
rabbit : any of various burrowing animals of the family Leporidae having long ears and s
hort tails; some domesticated and raised for pets or food


examples :


lemmas :
Lemma('rabbit.n.01.rabbit')
Lemma('rabbit.n.01.coney')
Lemma('rabbit.n.01.cony')


Traversing up we get :
Synset('leporid.n.01')
Synset('lagomorph.n.01')
Synset('placental.n.01')
Synset('mammal.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')
```

# 3

The way that the nouns are organized is that all nouns branch out from the all encompassing noun, entity. They are all branched in a way such that the upper branches are hypernyms of the lower branches and the lower branches are hyponyms of the upper branches.

In [11]:

```
#4
synset = synsets[0] ## choose one synset

print("hypernyms : \n")
print(synset.hypernyms()) ## output the hypernyms
print()

print("hyponyms : \n")
print(synset.hyponyms()) ## output the hyponyms
print()

print("meronyms : \n")
print(synset.part_meronyms()) ## output the meronyms
print()

print("holonyms : \n")
print(synset.part_holonyms()) ## output the holonyms
print()

print("antonym : \n")
```

```
for ant in synset.lemmas(): ## get the Lemmas
    print(ant.antonyms()) ## get the antonym from the Lemmas
```

```
hypernyms :

[Synset('increase.n.02')]

hyponyms :

[Synset('quantum_leap.n.01')]

meronyms :

[]

holonyms :

[]

antonym :

[]
[]
```

In [5]:

```
#5
word = "jump" ## select the verb
synsets = wn.synsets(word) ## get all synsets
print(synsets) ## print all synsets
```

```
[Synset('jump.n.01'), Synset('leap.n.02'), Synset('jump.n.03'), Synset('startle.n.01'),
Synset('jump.n.05'), Synset('jump.n.06'), Synset('jump.v.01'), Synset('startle.v.02'), S
ynset('jump.v.03'), Synset('jump.v.04'), Synset('leap_out.v.01'), Synset('jump.v.06'), S
ynset('rise.v.11'), Synset('jump.v.08'), Synset('derail.v.02'), Synset('chute.v.01'), Sy
nset('jump.v.11'), Synset('jumpstart.v.01'), Synset('jump.v.13'), Synset('leap.v.02'), S
ynset('alternate.v.01')]
```

In [6]:

```
#6
synset = synsets[7] ## choose a synset

print(word + " : "+ synset.definition()) ## output the definiton
print("\n")

print("examples" + " :")
for ex in synset.examples(): ## iterate through the examples
    print(ex) ## output the example
print("\n")

print("lemmas" + " : ")
for lem in synset.lemmas(): ## iterate through the Lemmas
    print(lem) ## output the Lemma
print("\n")

print("Traversing up we get :")
while len(synset.hypernyms()) > 0: ## terminating condition when the hypernym list has
    print (synset.hypernyms()[0]) ## output the hypernym
    synset = synset.hypernyms()[0] ## traverse up by setting the current noun to the hy
```

```
jump : move or jump suddenly, as if in surprise or alarm


examples :
```

```
She startled when I walked into the room


lemmas :
Lemma('startle.v.02.startle')
Lemma('startle.v.02.jump')
Lemma('startle.v.02.start')


Traversing up we get :
Synset('move.v.03')
```

# 6

Verbs in Wordnet are organized into separate hierarchies. The way these work is that there is a general verb term at the top and more specific as they get down with the upper branches being hypernyms of the lower branches.

In [7]:

```
#7
wn.morphy(word,wn.VERB) ## get the different forms
```

Out[7]:

```
'jump'
```

In [8]:

```
#8
word1 = 'lunch' ## Word one
word2 = 'dinner' ## Word two
synset1 = wn.synsets(word1)[0] ## get the first synset
synset2 = wn.synsets(word2)[0] ## get the second synset

wuPalmer = synset1.wup_similarity(synset2) ## get the Wu Palmer metric
print("The Wu-Palmer similarity between " + word1 + " and " + word2 + " is " + str(wuPa

sent = "I like to iron my clothes after washing them" ## set a sentence
sent_token = sent.split(' ') ## split it into the tokens
lesk_val = lesk(sent, 'iron', 'n') ## gets the correct form of iron, the verb form
print("The lesk choice for the sentenct : ")
print(sent)
print("is " + str(lesk_val))
print("which has a defintion of : ")
print(lesk_val.definition())
```

```
The Wu-Palmer similarity between lunch and dinner is 0.875
The lesk choice for the sentenct :
I like to iron my clothes after washing them
is Synset('iron.n.04')
which has a defintion of :
home appliance consisting of a flat metal base that is heated and used to smooth cloth
```

# 8

The Wu-Palmer similarity between lunch and dinner should be pretty similar as they are similar words since they are both meals. The .875 makes sense as a value of 1 would mean they are the same. The lesk algorithm gives us the correct iron as it is the one that appears in the sentence.

In [9]:

```
#9
charged_word = 'hate' ## set a charged word
senti_synsets = list(swn.senti_synsets(charged_word)) ## get the senti-synsets
for senti_synset in senti_synsets: ## iterate through the senti-synsets
    print("The synset " + str(senti_synset.synset) +
          " has a positive score of " + str(senti_synset.pos_score()) +
          " has a negative score of " + str(senti_synset.neg_score()) +
          " has an objective score of " + str(senti_synset.obj_score())
         ) ## output the scores of the synsets
    print()

sent = 'I love learning about natural language processing' ## make a sentence
for word in sent.split(' '): ## iterate through the words in the sentence
    sentinet = list(swn.senti_synsets(word)) ## convert the output to a list
    if(len(sentinet) > 0): ## if the list has elements
        sentinet = sentinet[0] ## get the first sentinet
        print(word +
              ' : pos = ' + str(sentinet.pos_score()) +
              ', neg = ' + str(sentinet.neg_score()) +
              ', obj = ' + str(sentinet.obj_score())
             ) ## output the sentiment of the word
    else:
        print(word + ' does not have a SentiWordNet') ## if there is no sentinet for th
```

```
The synset Synset('hate.n.01') has a positive score of 0.125 has a negative score of 0.3
75 has an objective score of 0.5

The synset Synset('hate.v.01') has a positive score of 0.0 has a negative score of 0.75
has an objective score of 0.25

I : pos = 0.0, neg = 0.0, obj = 1.0
love : pos = 0.625, neg = 0.0, obj = 0.375
learning : pos = 0.0, neg = 0.0, obj = 1.0
about : pos = 0.0, neg = 0.0, obj = 1.0
natural : pos = 0.0, neg = 0.0, obj = 1.0
language : pos = 0.0, neg = 0.0, obj = 1.0
processing : pos = 0.25, neg = 0.0, obj = 0.75
```

# 9

These scores correlate quite well with the words as the more emotionally charged words have higher sentiment scores. One thing I found odd was that love didn't have a higer positive sentiment. These would be helpful in many applications as it can tell you how the writer of the text feels about the subject of the sentence. For example you could use the average sentiment of the sentence to figure out whether reviews are positive or not.

In [10]:

```
#10
text4.collocations() ## output the collocations
coallocation = 'fellow citizens' ## choose one collocation
num_bigrams = len(text4.tokens)-1 ## get the number of bigrams
pxy = " ".join(text4.tokens).count(coallocation) / num_bigrams ## get p(x,y)
px = text4.count(coallocation.split(' ')[0]) / len(text4.tokens) ## get p(x)
py = text4.count(coallocation.split(' ')[1]) / len(text4.tokens) ## get p(y)
pmi = math.log(pxy / (px * py) , 2) ## get the point-wise mutual information
print()
print("The point-wise mutual information value is " + str(pmi))
```

```
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

The point-wise mutual information value is 8.174006563041724
```

# 10

Collocations are two words that have a higher chance of occurring together than most other words. They are sort of outliers in the sense they have a greater chance of them occurring out of all the possible combinations.

The higher the pmi is between two words, the stronger the collocation is. This collocation between fellow citizens is quite strong as the frequently occur together as there are 61 occurrences of them occurring together.

```
In [ ]:
```