What WordNet means to Me

Founded during the mid 1980s, Wordnet is an hierarchical structure of each of different parts of speech the English language has. This hierarchy uses the word's glosser, Synsets, example usage, and other related words as a catalyst for calculation of that word's placement in the organization.

#More on Synsets The use of synset in WordNet is important to the semantic relation that is placed in the hierarcy. There are 5 relations that WordNet uses for Synsets:

1. Hypernym - Words in this level are a broader term for a group of words below it. For instance, Feline is a hypernym for cat
2. Hyponym - Words at this level are common words that describe/identify a specific object or action. For example, dogs is a hyponym to the word canine
3. Meronym - Theses words are usually a small instrument to a larger system. Think of words such as a nut, screw, or axles.
4. Holonym - Theses words are the big picture of what a group of Meronym form together. Some common Holonyms include car, skyscraper, computer, and bicycle.
5. Troponym - Words in this category are reserved for narrow specified actions from a broad branch of a common action. some of these specific actions include: whisper, sprint, uppercut, and axel kick.

```
#Needed imports for the program
import nltk
#nltk.download('all')
nltk.download('wordnet')

#Variable for WordNet
from nltk.corpus import wordnet as WN

#Our choosen Noun wil be Gun
nounWord = "Gun"

print("Here are all the synsets of \'Gun\'.")
WN.synsets(nounWord)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Here are all the synsets of 'Gun'.
[Synset('gun.n.01'),
 Synset('artillery.n.01'),
 Synset('gunman.n.02'),
 Synset('gunman.n.01'),
 Synset('grease-gun.n.01'),
```

```
 Synset('accelerator.n.01'),
 Synset('gun.n.07'),
 Synset('gun.v.01')]
```

Exploring a Synset

From the synsets of 'Gun', lets explore the word 'grease-gun'.

We will try to find the definition, exmples of practice, and the lemmas of 'grease-gun'. Furthermore, we will climb up the hierarchy of 'grease-gun'

```
#simplifing the process synsets
artillery = WN.synset('artillery.n.01')

#Getting the definition of the synsets
print("There is the definition of artillery: ",artillery.definition())

#Getting the examples of the synsets
print("\nThere is sentence using of artillery: ",artillery.examples())

#Getting the lemmas of the synsets
print("\nThere is the lemmas of artillery: ",artillery.lemmas(), "\n")

#Traversing the hierarchy of grease-gun
baseHypo = artillery.hypernyms()[0]
top = WN.synset('entity.n.01')

print("\nLets go through the rabbit hole of artillery hierarchy.")
#The start of the traversal
while baseHypo:
  print("We are looking at ", baseHypo)

  if baseHypo == top:
    break

  else:
    baseHypo = baseHypo.hypernyms()[0]
```

```
 There is the definition of artillery:  large but transportable armament

 There is sentence using of artillery:  []

 There is the lemmas of artillery:  [Lemma('artillery.n.01.artillery'), Lemma('artillery


 Lets go through the rabbit hole of artillery hierarchy.
 We are looking at  Synset('armament.n.01')
 We are looking at  Synset('weaponry.n.01')
 We are looking at  Synset('instrumentality.n.03')
```

```
We are looking at  Synset('artifact.n.01')
We are looking at  Synset('whole.n.02')
We are looking at  Synset('object.n.01')
We are looking at  Synset('physical_entity.n.01')
We are looking at  Synset('entity.n.01')
```

Thought on the Outcome

There are a goodamount of responce that wordNet has rovided to us from the word artillery. I am a bit shock that there was no mention of an sentence using our noun dispite it can be easily done. Our herarchy order, at the most part, we most people would expect. Armament and weaponry are great examples of artillery's hypernyms and it can be safe to say that the last four hypernyms are normally at the top in other nouns.

```
#Looking at the hypernyms of greasegun
print("Here are some hypernyms of artillery ", artillery.hypernyms())

#Looking at the hyponyms of greasegun
print("Here are some hyponyms of artillery ",artillery.hyponyms())

#Looking at the meronyms of greasegun
print("Here are some meronyms of artillery ",artillery.member_meronyms())

#Looking at the holonyms of greasegun
print("Here are some holonyms of artillery ",artillery.member_holonyms())

#Looking at the antonyms of greasegun
print("Here are some antonyms of artillery ",artillery.lemmas()[0].antonyms())
```

```
Here are some hypernyms of artillery  [Synset('armament.n.01')]
Here are some hyponyms of artillery  [Synset('cannon.n.01'), Synset('field_artillery.n.0
Here are some meronyms of artillery  []
Here are some holonyms of artillery  []
Here are some antonyms of artillery  []
```

Exlporing a Verb

Now that we have seen what WordNet can do to a noun, lets see the differences of what a verb produces

```
#Sorting our verb
verbWord = "drift"

#Showing our verb to the audience
print("Here are all the synsets of \'Drift\'.")
WN.synsets(verbWord)
```

```
Here are all the synsets of 'Drift'.
[Synset('drift.n.01'),
 Synset('drift.n.02'),
 Synset('drift.n.03'),
 Synset('drift.n.04'),
 Synset('drift.n.05'),
 Synset('drift.n.06'),
 Synset('drift.n.07'),
 Synset('float.v.01'),
 Synset('stray.v.02'),
 Synset('roll.v.12'),
 Synset('drift.v.04'),
 Synset('freewheel.v.01'),
 Synset('drift.v.06'),
 Synset('drift.v.07'),
 Synset('drift.v.08'),
 Synset('drift.v.09'),
 Synset('drift.v.10')]
```

Exploring a Synset of a Verb

From the synsets of 'Drift', lets explore the word 'freewheel.v.01'.

We will try to find the definition, exmples of practice, and the lemmas of 'freewheel.v.01'.

Furthermore, we will climb up the hierarchy of 'freewheel.v.01'

```
#simplifing the process synsets
freeWheel = WN.synset('freewheel.v.01')

#Getting the definition of the synsets
print("There is the definition of freewheel: ",freeWheel.definition())

#Getting the examples of the synsets
print("\nThere is a sentence of freewheel: ",freeWheel.examples())

#Getting the lemmas of the synsets
print("\nThere is the lemmas of freewheel: ",freeWheel.lemmas(), "\n")

#Wxploring the hierarchy of freewheel
hyper = lambda s: s.hypernyms()

print("\nLets go through the rabbit hole of freewheel hierarchy.")
list(freeWheel.closure(hyper))
```

```
There is the definition of freewheel:  live unhurriedly, irresponsibly, or freely

There is a sentence of freewheel:  ['My son drifted around for years in California befor

There is the lemmas of freewheel:  [Lemma('freewheel.v.01.freewheel'), Lemma('freewheel


Lets go through the rabbit hole of freewheel hierarchy.
[Synset('exist.v.02')]
```

Thoughts on the Outcome

in comparison to nouns, wordNet did not deliver much hypernyms to travers in the gierarchy of 'Freewheel'. It is a bit weird that the example usage of freewheel has no mention of the actual word, but it an example for an simular word. A more astounding result from wordNet is that our verb had more substance to work with than our noun.

The Morphy of Freewheel

Now that we saw everything that our synset word has to offer, lets explore it's morphy.

```
# get the adjective lemma of ’friendlier’
freewhel_ADJ = WN.morphy('freewheel', WN.ADJ)
print("Here is the adjective of freewheel: ", freewhel_ADJ)

# get the verb lemma of ’exercised’
freewhel_V = WN.morphy('freewheel', WN.VERB)
print("Here is the verb of freewheel: ", freewhel_V)

# get the noun lemma of ’exercises’
freewhel_N = WN.morphy('freewheel', WN.NOUN)
print("Here is the noun of freewheel: ", freewhel_N)
```

```
Here is the adjective of freewheel:  None
Here is the verb of freewheel:  freewheel
Here is the noun of freewheel:  freewheel
```

Word Simularity

In this section of the program, we will run the Wu-Palmer siliarity metric as well as the Lesk algorthim on Turtle and Tortoise

---

```
#Exploring the synsets of turtles
print("Here are all the synsets of \'Turtles\'.\n", WN.synsets('turtle'))

#Exploring the synsets of turtles
print("\nHere are all the synsets of \'tortoise\'.\n", WN.synsets('tortoise')
)
```

```
Here are all the synsets of 'Turtles'.
 [Synset('turtleneck.n.01'), Synset('turtle.n.02'), Synset('capsize.v.01'), Synset('turt

Here are all the synsets of 'tortoise'.
 [Synset('tortoise.n.01')]
```

```
from nltk.wsd import lesk

Turt = WN.synset('turtle.n.02')
Tort = WN.synset('tortoise.n.01')

#Using the Wu-Palmer algorithm
Wu_PaNUM = WN.path_similarity(Turt, Tort)
Wu_PaNUM *= 100.0
print("According to the Wu-Palmer algorithm, the words turtle and tortoise are", Wu_PaNUM, "%

#Using the Lesk algorithm
sent1 = ['I', 'loving', 'watching', 'baby', 'turtles', 'crawling', 'to', 'the', 'ocean', '.']

sent2 = ['A', 'tortoise', 'can', 'live', 'up', 'to', 'one', 'hundred', 'years', '!']

#seeing the resulting for turtle and sent1
print(lesk( sent1, 'turtles', 'n' ))

print('\n', lesk( sent2, 'tortoise', 'n' ))
```

```
According to the Wu-Palmer algorithm, the words turtle and tortoise are 50.0 % alike.
Synset('turtleneck.n.01')

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

Observations

Truthfully, I was expecting a larger percentage from Wu-Palmer metric since the two animals are in the same order of the taxonomy.

As for the Lesk algorithm, I am certain that there is more work to be needed as the result refers to a clothing component.

SentiWordNet

Another special lexical resource that WordNet provides is SentiWordNet. This class from WordNet calculates a given corpus's positive, negative, and objective synset. Such a function can help a writer determine if their text portrays a certain opinion. A good example would be to determine if a food Yelper/critique is speaking in a subjective and negative manner or is complementing the food from an objective stance.

```
from nltk.corpus import sentiwordnet as SWN

#Getting our strong emotional word's synset
emotionalWord = "distraught"
WN.synsets(emotionalWord)

#setting our veriable
EmoWord = SWN.senti_synset('distraught.s.01')
print("The senti-synsets of Distraught is: ", EmoWord)

print("\nPositve score is: ", EmoWord.pos_score())
print("\nNegative score is: ", EmoWord.neg_score())
print("\nObjective score is: ", EmoWord.obj_score())

#Forming a sentences
sentence = "I love turtles so much than I would be distraught to see one on its back."

#counters
neg = pos = 0

#Tokenizing our sentence
tokens = sentence.split()

#looping throught our sentence
for token in tokens:
  syn_list = list(SWN.senti_synsets(token))

  if syn_list:
    syn = syn_list[0]
    neg += syn.neg_score()
    pos += syn.pos_score()

#Shownig our results
print("\n\nAfter reading through this sentence \'", sentence, "\'\nI have concluded this sent
print("\nneg \t pos counts")
print(neg, '\t', pos)
```

```
The senti-synsets of Distraught is:  <distraught.s.01: PosScore=0.0 NegScore=0.125>
```

```
Positve score is:  0.0

Negative score is:  0.125

Objective score is:  0.875


After reading through this sentence ' I love turtles so much than I would be distraught
I have concluded this sentiment analysis:

neg      pos counts
0.25     1.25
```

Collocation and its Utility

Using collocation is like attempting to predict the next 7 words from a person's thoughts. Collocation zone in on a word's neighbor and attempts finds a pattern between neighboring words. This is all in effort to comprehend the true meaning of a sentence rather than understanding the phrase at face value.

```
import nltk
nltk.download('all')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('Text')
nltk.download('webtext')

from nltk.book import text4


#Getting the collocations for text4
text4.collocations()

#Storing our identified collocation
wordPairs = 'Indian tribes'

print("\nWe will do a further investigation on the pair word ", wordPairs)
#spacing
print("\n\n")

################################################################################
##########      Calculating mutal information     ##############################
################################################################################
import math

#tokenizing text4 while adding spaces
```

```
#tokenizing text1 while adding spaces
text = " ".join(text4.tokens)

#calculating the mutual information of our collocation pair
text4_length = len(set(text4))

#our P(x,y)
collocatWord_Num = text.count(wordPairs) / text4_length

#our P(x)
IndianCount = text.count('Indian') / text4_length

#our P(y)
TribeCount = text.count('tribes') / text4_length

#our P(x) * P(y)
Denomin = IndianCount * TribeCount

#our P(x,y) / (P(x) * P(y))
fraction = collocatWord_Num / Denomin

#Final product
PMI = math.log2(fraction)

#Our results
print("  P({})        \t{}\n-------------------    = \t-------------------\nP(Indian) * P(tri
      .format(wordPairs, collocatWord_Num, IndianCount, TribeCount))

print("\nThis results in: ", fraction)
print("To conclude, the log base 2 of {} is {}".format(fraction, 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

We will do a further investigation on the pair word  Indian tribes



  P(Indian tribes)              0.000598503740648379
-------------------    =       -------------------
P(Indian) * P(tribes)   0.0010972568578553616*0.000598503740648379

This results in:  911.3636363636364
To conclude, the log base 2 of 911.3636363636364 is 9.831882997592349
```

Thoughts on my Findings

Assuming my calculatios are correct, it seems that the word pairs 'Indian tribes' appears just shy under 10% of the entire text4.

While my implementation was maticulus, it esures that the calculation will not mistaken by PEMDAS ruling.