WordNet is a hirerarchical organization of nouns, verbs adjectives, and adverbs including their glossary, sets of synonyms, or related words, known as synsets and provides a brief definition and usage examples for each word. It also organizes words into hierarchies based on their meanings and relationships. WordNet is widely used by linguists, lexicographers, and computer scientists to help them build natural language processing systems and applications.

First, we need to import wordnet

```
from nltk.corpus import wordnet as wn


# get all synsets of 'man'

wn.synsets('man')
```

```
[Synset('man.n.01'),
 Synset('serviceman.n.01'),
 Synset('man.n.03'),
 Synset('homo.n.02'),
 Synset('man.n.05'),
 Synset('man.n.06'),
 Synset('valet.n.01'),
 Synset('man.n.08'),
 Synset('man.n.09'),
 Synset('man.n.10'),
 Synset('world.n.08'),
 Synset('man.v.01'),
 Synset('man.v.02')]
```

From a given synset, I used 'man.n.01', we can extract definition, usage example, and lemmas.

```
# get a definition for the 'man.n.01' synset

wn.synset('man.n.01').definition()
```

```
'an adult person who is male (as opposed to a woman)'
```

```
# extract examples for 'man.n.01' synset

wn.synset('man.n.01').examples()
```

```
['there were two women and six men on the bus']
```

```
# extract lemmas for 'man.n.01' synset

wn.synset('man.n.01').lemmas()
```

```
[Lemma('man.n.01.man'), Lemma('man.n.01.adult_male')]
```

```
# Traversing the hierarchy

man = wn.synset('man.n.01')

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

```
Synset('adult.n.01')
Synset('person.n.01')
Synset('causal_agent.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

WordNet is organized for nouns in a hierarchical structure, with each noun represented by a set of related words and concepts. Nouns have the top set to 'entity' which makes it possible to traverse the hierarchy of the synsets. one way the travers up of the hierarchy is to recursivly call hypernyms until we reach the top entity.

```
# Output for hypernyms

man.hypernyms()
```

```
[Synset('adult.n.01'), Synset('male.n.02')]
```

```
# Output for hyponyms

man.hyponyms()
```

```
[Synset('adonis.n.01'),
 Synset('babu.n.01'),
 Synset('bachelor.n.01'),
 Synset('bey.n.01'),
 Synset('black_man.n.01'),
 Synset('boy.n.02'),
 Synset('boyfriend.n.01'),
 Synset('bull.n.02'),
 Synset('dandy.n.01'),
 Synset('ejaculator.n.01'),
 Synset('esquire.n.02'),
 Synset('eunuch.n.01'),
 Synset('ex-boyfriend.n.01'),
 Synset('ex-husband.n.01'),
 Synset('father-figure.n.01'),
 Synset('father_figure.n.01'),
 Synset('fellow.n.06'),
 Synset('galoot.n.01'),
 Synset('geezer.n.01'),
 Synset('gentleman.n.01'),
 Synset('grass_widower.n.01'),
 Synset('guy.n.01'),
 Synset('herr.n.01'),
 Synset('hooray_henry.n.01'),
 Synset('housefather.n.01'),
 Synset('hunk.n.01'),
 Synset('inamorato.n.01'),
 Synset('iron_man.n.01'),
 Synset('ironside.n.01'),
 Synset('middle-aged_man.n.01'),
 Synset('monsieur.n.01'),
 Synset('old_boy.n.01'),
 Synset('old_man.n.01'),
 Synset('patriarch.n.02'),
 Synset('peter_pan.n.01'),
 Synset('ponce.n.01'),
 Synset('posseman.n.01'),
 Synset('senhor.n.01'),
 Synset('shaver.n.01'),
 Synset('signor.n.01'),
 Synset('signore.n.01'),
 Synset('sir.n.01'),
 Synset('stiff.n.01'),
 Synset('stud.n.01'),
 Synset('tarzan.n.01'),
 Synset('white_man.n.01'),
 Synset('widower.n.01'),
 Synset('womanizer.n.01'),
 Synset('wonder_boy.n.01'),
 Synset('yellow_man.n.01'),
 Synset('young_buck.n.01')]
```

```
# Output the meronyms

print('meronyms: ' ,man.part_meronyms())
```

```
meronyms:  [Synset('adult_male_body.n.01')]
```

```
# Output the holonyms

print('holonyms: ', man.part_holonyms())
```

```
holonyms:  []
```

```
# Output the antonym

print('antonym: ', man.lemmas()[0].antonyms())
```

```
antonym:  [Lemma('woman.n.01.woman')]
```

Select a verb 'talk'

```
# get all synsets of 'talk'
wn.synsets('talk')
```

```
[Synset('talk.n.01'),
 Synset('talk.n.02'),
 Synset('talk.n.03'),
 Synset('lecture.n.01'),
 Synset('talk.n.05'),
 Synset('talk.v.01'),
 Synset('talk.v.02'),
 Synset('speak.v.03'),
 Synset('spill.v.05'),
 Synset('spill_the_beans.v.01'),
 Synset('lecture.v.01')]
```

From a given synset, I used 'talk.n.01', we can extract definition, usage example, and lemmas.

```
# get a definition for the 'talk.v.01' synset

wn.synset('talk.v.01').definition()
```

```
'exchange thoughts; talk with'
```

```
# extract examples for 'talk.v.01' synset

wn.synset('talk.v.01').examples()
```

```
['We often talk business', 'Actions talk louder than words']
```

```
# extract lemmas for 'talk.v.01' synset

wn.synset('talk.v.01').lemmas()
```

```
[Lemma('talk.v.01.talk'), Lemma('talk.v.01.speak')]
```

```
# Traversing the hierarchy

talk = wn.synset('talk.v.01')

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

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

WordNet is organized in a hierarchical manner for verbs, with hypernym/hyponym relationship. unlike nouns, they do not have a top so it is difficult to traverse in the same way as nouns. A better way to approach is using closure method provided by nltk to pass in a lambda function.

```
# use morphy to find different forms of the word

morphy_verb = wn.morphy('talk')
wn.synsets(morphy_verb)
```

```
[Synset('talk.n.01'),
 Synset('talk.n.02'),
 Synset('talk.n.03'),
 Synset('lecture.n.01'),
 Synset('talk.n.05'),
 Synset('talk.v.01'),
 Synset('talk.v.02'),
 Synset('speak.v.03'),
 Synset('spill.v.05'),
 Synset('spill_the_beans.v.01'),
 Synset('lecture.v.01')]
```

```
# select two words
word1 = 'cost'
word2 = 'price'
```

```
# get all synsets of 'cost'
wn.synsets('cost')
```

```
[Synset('cost.n.01'),
 Synset('monetary_value.n.01'),
 Synset('price.n.03'),
 Synset('cost.v.01'),
 Synset('cost.v.02')]
```

```
# get all synsets of 'price'
wn.synsets('price')
```

```
[Synset('monetary_value.n.01'),
 Synset('price.n.02'),
 Synset('price.n.03'),
 Synset('price.n.04'),
 Synset('price.n.05'),
 Synset('price.n.06'),
 Synset('price.n.07'),
 Synset('price.v.01'),
 Synset('price.v.02')]
```

```
# check similarity compare to Wu-palmer
cost = wn.synset('cost.v.01')
price = wn.synset('price.v.01')

print('wn_palmer Similarity: ', wn.wup_similarity(cost, price))
```

```
wn_palmer Similarity:  0.3333333333333333
```

```
# check similarity compare to Wu-palmer
cost = wn.synset('cost.n.01')
price = wn.synset('price.n.02')

print('wn_palmer Similarity: ', wn.wup_similarity(cost, price))
```

```
wn_palmer Similarity:  0.9333333333333333
```

The first similarity comparison between 'cost.v.01' and 'price.v.01' using the Wu-Palmer algorithm has a low similarity score of 0.333333, indicating that these two synsets have little overlap in their semantic meaning. On the other hand, the second comparison between 'cost.n.01' and 'price.n.02' has a high similarity score of 0.933333, suggesting that these two synsets are closely related in meaning, likely because they both belong to the same semantic domain of economics.

The SemtiWordNet is used to analyze bodies of text in order to determine if it positive/negative or subjective. it also provides information on the degree in which a certain word displays the aforementioned characteristics. it has various practical applications such as analyzing journals to analyze the participant emotions, to flag inappropriate and vulgar text on social media website automatically and so on.

```
from nltk.corpus import sentiwordnet as swn
```

```
wn.synsets('smile')
```

```
[Synset('smile.n.01'), Synset('smile.v.01'), Synset('smile.v.02')]
```

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

```
<smile.n.01: PosScore=0.125 NegScore=0.0>
Positive score =  0.125
Negative score =  0.0
Objective score =  0.875
```

```
# look up sentisynsets
word = 'smile'
syn_list = list(swn.senti_synsets(word))
for item in syn_list:
  print(item)
```

```
<smile.n.01: PosScore=0.125 NegScore=0.0>
<smile.v.01: PosScore=0.0 NegScore=0.0>
<smile.v.02: PosScore=0.0 NegScore=0.0>
```

```
# make a sentence and output the polarity for each work in the sentence
sent = 'I am very happy'
tokens = sent.split()
for token in tokens:
  print('For the Word: ', token)
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    for item in syn_list:
      print(item)
    print()
```

```
For the Word:  I
<iodine.n.01: PosScore=0.0 NegScore=0.0>
<one.n.01: PosScore=0.0 NegScore=0.0>
<i.n.03: PosScore=0.0 NegScore=0.0>
<one.s.01: PosScore=0.0 NegScore=0.25>

For the Word:  am
<americium.n.01: PosScore=0.0 NegScore=0.0>
<master_of_arts.n.01: PosScore=0.0 NegScore=0.125>
<amplitude_modulation.n.01: PosScore=0.0 NegScore=0.0>
<be.v.01: PosScore=0.25 NegScore=0.125>
<be.v.02: PosScore=0.0 NegScore=0.0>
<be.v.03: PosScore=0.0 NegScore=0.0>
<exist.v.01: PosScore=0.0 NegScore=0.0>
<be.v.05: PosScore=0.0 NegScore=0.0>
<equal.v.01: PosScore=0.125 NegScore=0.125>
<constitute.v.01: PosScore=0.0 NegScore=0.0>
<be.v.08: PosScore=0.0 NegScore=0.0>
<embody.v.02: PosScore=0.0 NegScore=0.0>
<be.v.10: PosScore=0.0 NegScore=0.0>
<be.v.11: PosScore=0.0 NegScore=0.0>
<be.v.12: PosScore=0.0 NegScore=0.0>
<cost.v.01: PosScore=0.0 NegScore=0.0>

For the Word:  very
<very.s.01: PosScore=0.5 NegScore=0.0>
<identical.s.02: PosScore=0.5 NegScore=0.125>
<very.r.01: PosScore=0.25 NegScore=0.25>
<very.r.02: PosScore=0.25 NegScore=0.0>

For the Word:  happy
<happy.a.01: PosScore=0.875 NegScore=0.0>
<felicitous.s.02: PosScore=0.75 NegScore=0.0>
<glad.s.02: PosScore=0.5 NegScore=0.0>
<happy.s.04: PosScore=0.125 NegScore=0.0>
```

The scores here show whether the words have a positive or negative score and in addition shows the degree to which they are negative or positive. This can have real life applications especially if we are trying to automatically flag profance twwets or other social media post by seeing the ratio between positive to negative words, the fraction of negative words in the text, and so on to make an informed decision to estimate if a text body is malicious or not.

A collocation is a combination of two or more words that often go together to form a phrase that cannot be substituted by a single wor to yield the same meaning, such as "strong coffee" or "make a decision." Collocations are often used in everyday language and can help give a sentence more natural-sounding rhythm.

```
import nltk
from nltk.book import *
text4
```

```
*** 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
<Text: Inaugural Address Corpus>
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
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
```

```
# calculate mutual information

text = ' '.join(text4.tokens)
text[:50]
```

```
'Fellow - Citizens of the Senate and of the House o'
```

```
import math
vocab = len(set(text4))
hg = text.count('United States')/vocab
print("p(United States) = ",hg )
h = text.count('United')/vocab
print("p(United) = ", h)
g = text.count('States')/vocab
print('p(States) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(United States) =  0.015860349127182045
p(United) =  0.0170573566084788
p(States) =  0.03301745635910224
pmi =  4.815657649820885
```

The results show a positive pmi and rather high positive pmi which suggests that "United States" is most likely a collocation.