WordNet

WordNet is a lexical database of the English language that groups words into sets of synonyms called synsets, and provides definitions and usage examples for each synset. It also includes relations between words such as hyponymy/hypernymy (subordinate/superordinate) and meronymy/holonymy (part/whole), as well as cross-references to other synsets. WordNet is widely used in natural language processing and computational linguistics applications, and has been the basis for the development of many other language resources and tools.

Imports

In [ ]:
```
import math

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

Noun

Selecting noun

In [ ]:
```
selected_noun = "hair"
```

All Synsets for the noun

In [ ]:
```
print(wn.synsets(selected_noun, pos=wn.NOUN))
```

```
[Synset('hair.n.01'), Synset('hair's-breadth.n.01'), Synset('hair.n.03'), S
ynset('hair.n.04'), Synset('haircloth.n.01'), Synset('hair.n.06')]
```

In [ ]:
```
selected_noun_synset = wn.synsets(selected_noun)[0]
```

Synset definition

In [ ]:
```
selected_noun_synset.definition()
```

Out[ ]:
```
'a covering for the body (or parts of it) consisting of a dense growth of t
hreadlike structures (as on the human head); helps to prevent heat loss'
```

Synset usage examples

In [ ]:
```
selected_noun_synset.examples()
```

Out[ ]:
```
['he combed his hair',
 'each hair consists of layers of dead keratinized cells']
```

Synset lemmas

In [ ]:
```
selected_noun_synset.lemmas()
```

Out[ ]:
```
[Lemma('hair.n.01.hair')]
```

Traversing up the WordNet hierarchy

In [ ]:
```
curr = selected_noun_synset
while curr:
    print(curr)
    if not curr.hypernyms():
        break
    curr = curr.hypernyms()[0]
```

```
Synset('hair.n.01')
Synset('body_covering.n.01')
Synset('covering.n.01')
Synset('natural_object.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

WordNet nouns if is an object converges to physical_entity or if it is an idea converges abstraction. Then they converge into entity.

Printing: hypernyms, hyponyms, meronyms, holonyms, antonym.

In [ ]:
```
print("hypernyms:", selected_noun_synset.hypernyms(), "\n")
print("hyponyms:", selected_noun_synset.hyponyms(), "\n")
print("meronym:", selected_noun_synset.part_meronyms(), "\n")
print("holonyms:", selected_noun_synset.part_holonyms(), "\n")

antonyms = []
for synset in wn.synsets(selected_noun):
    for word in synset.lemmas():
        for antonym in word.antonyms():
            antonyms.append(antonym.name())
print("antonym:", antonyms, "\n")
```

```
hypernyms: [Synset('body_covering.n.01')]

hyponyms: [Synset('beard.n.04'), Synset('body_hair.n.01'), Synset('coat.n.0
3'), Synset('cowlick.n.01'), Synset('down.n.05'), Synset('eyebrow.n.01'), S
ynset('eyelash.n.01'), Synset('facial_hair.n.01'), Synset('forelock.n.02'),
Synset('guard_hair.n.01'), Synset('hairdo.n.01'), Synset('lock.n.02'), Syns
et('mane.n.01'), Synset('mane.n.02'), Synset('pubic_hair.n.01')]

meronym: [Synset('hairline.n.02'), Synset('part.n.10')]

holonyms: [Synset('integumentary_system.n.01')]

antonym: []
```

Verb

Selecting verb

In [ ]:
```
selected_verb = "run"
```

All Synsets for the verb

In [ ]:
```
print(wn.synsets(selected_verb, pos=wn.VERB))
```

```
[Synset('run.v.01'), Synset('scat.v.01'), Synset('run.v.03'), Synset('opera
te.v.01'), Synset('run.v.05'), Synset('run.v.06'), Synset('function.v.01'),
Synset('range.v.01'), Synset('campaign.v.01'), Synset('play.v.18'), Synset(
'run.v.11'), Synset('tend.v.01'), Synset('run.v.13'), Synset('run.v.14'), S
ynset('run.v.15'), Synset('run.v.16'), Synset('prevail.v.03'), Synset('run.
v.18'), Synset('run.v.19'), Synset('carry.v.15'), Synset('run.v.21'), Synse
t('guide.v.05'), Synset('run.v.23'), Synset('run.v.24'), Synset('run.v.25')
, Synset('run.v.26'), Synset('run.v.27'), Synset('run.v.28'), Synset('run.v
.29'), Synset('run.v.30'), Synset('run.v.31'), Synset('run.v.32'), Synset('
run.v.33'), Synset('run.v.34'), Synset('ply.v.03'), Synset('hunt.v.01'), Sy
nset('race.v.02'), Synset('move.v.13'), Synset('melt.v.01'), Synset('ladder
.v.01'), Synset('run.v.41')]
```

In [ ]:
```
selected_verb_synset = wn.synsets(selected_verb)[0]
```

Synset definition

In [ ]:
```
selected_verb_synset.definition()
```

Out[ ]:
```
'a score in baseball made by a runner touching all four bases safely'
```

Synset usage examples

In [ ]:
```
selected_verb_synset.examples()
```

Out[ ]:
```
['the Yankees scored 3 runs in the bottom of the 9th',
 'their first tally came in the 3rd inning']
```

Synset lemmas

In [ ]:
```
selected_verb_synset.lemmas()
```

Out[ ]:
```
[Lemma('run.n.01.run'), Lemma('run.n.01.tally')]
```

Traversing up the WordNet hierarchy

In [ ]:
```
curr = selected_verb_synset
while curr:
    print(curr)
    if not curr.hypernyms():
        break
    curr = curr.hypernyms()[0]
```

```
Synset('run.n.01')
Synset('score.n.10')
Synset('success.n.02')
Synset('attainment.n.01')
Synset('accomplishment.n.01')
Synset('action.n.01')
Synset('act.n.02')
Synset('event.n.01')
Synset('psychological_feature.n.01')
Synset('abstraction.n.06')
Synset('entity.n.01')
```

WordNet verb if is an physical activity converges to entity or if it is an idea converges make.

Printing: hypernyms, hyponyms, meronyms, holonyms, antonym.

In [ ]:
```
print("hypernyms:", selected_verb_synset.hypernyms(), "\n")
print("hyponyms:", selected_verb_synset.hyponyms(), "\n")
print("meronym:", selected_verb_synset.part_meronyms(), "\n")
print("holonyms:", selected_verb_synset.part_holonyms(), "\n")

antonyms = []
for synset in wn.synsets(selected_verb):
    for word in synset.lemmas():
        for antonym in word.antonyms():
            antonyms.append(antonym.name())
print("antonym:", antonyms, "\n")
```

```
hypernyms: [Synset('score.n.10')]

hyponyms: [Synset('earned_run.n.01'), Synset('run_batted_in.n.01'), Synset(
'unearned_run.n.01')]

meronym: []

holonyms: []

antonym: ['malfunction', 'idle']
```

Morphy

In [ ]:
```
word = "patient"
```

Word as Noun

In [ ]:
```
print(wn.morphy(word, wn.NOUN))
```

```
patient
```

Word as Adjective

In [ ]:
```
print(wn.morphy(word, wn.ADJ))
```

```
patient
```

Word Similarity

Wu-Palmer similarity

In [ ]:
```
print(wn.synsets("teacher"))
print(wn.synsets("student"))
```

```
[Synset('teacher.n.01'), Synset('teacher.n.02')]
[Synset('student.n.01'), Synset('scholar.n.01')]
```

In [ ]:
```
teacher_synset = wn.synset('teacher.n.01')
student_synset = wn.synset('student.n.01')
teacher_synset.path_similarity(student_synset)
```

Out[ ]:
```
0.14285714285714285
```

The Wu-Palmer similarity metric is a measure of semantic relatedness between words that takes into account their distance in a semantic hierarchy. Even though the words should be closely related Wu-Palmer similarity metric gives less score.

Lesk algorithm

In [ ]:
```
lesk_word = "ring"
for ss in wn.synsets(lesk_word):
    print(ss, ss.definition())
```

```
Synset('ring.n.01') a characteristic sound
Synset('ring.n.02') a toroidal shape
Synset('hoop.n.02') a rigid circular band of metal or wood or other materia
l used for holding or fastening or hanging or pulling
Synset('closed_chain.n.01') (chemistry) a chain of atoms in a molecule that
forms a closed loop
Synset('gang.n.01') an association of criminals
Synset('ring.n.06') the sound of a bell ringing; ; ; --E. A. Poe
Synset('ring.n.07') a platform usually marked off by ropes in which contest
ants box or wrestle
Synset('ring.n.08') jewelry consisting of a circlet of precious metal (ofte
n set with jewels) worn on the finger
Synset('band.n.12') a strip of material attached to the leg of a bird to id
entify it (as in studies of bird migration)
Synset('ring.v.01') sound loudly and sonorously
Synset('resound.v.01') ring or echo with sound
Synset('ring.v.03') make (bells) ring, often for the purposes of musical ed
ification
Synset('call.v.03') get or try to get into communication (with someone) by
telephone
Synset('surround.v.01') extend on all sides of simultaneously; encircle
Synset('ring.v.06') attach a ring to the foot of, in order to identify
```

In [ ]:
```
sent = ['I', 'gave', 'her', 'a', 'ring', '.']
print(lesk(sent, 'ring', 'n'))
print(lesk(sent, 'ring'))
```

```
Synset('ring.n.08')
Synset('ring.v.06')
```

The Lesk algorithm could be applied to disambiguate the meaning of words in sentences

SentiWordNet

- SentiWordNet is a lexical resource that assigns sentiment scores to words in a natural language text. It is based on WordNet, a large lexical database of English words, and provides a way to determine the positivity, negativity, and neutrality of words based on their meanings.
- SentiWordNet can be used in NLP applications such as sentiment analysis, opinion mining, and text classification, where the goal is to identify the sentiment of text data.
- It can also be used in machine learning to enhance the accuracy of the models that rely on sentiment analysis for decision making.

In [ ]:
```
emotionally_charged_word = "serenity"
serenity_senti_synsets = swn.senti_synsets(emotionally_charged_word)

for senti_synset in serenity_senti_synsets:
  print(senti_synset)
  print("Positive score = ", senti_synset.pos_score())
  print("Negative score = ", senti_synset.neg_score())
  print("Objective score = ", senti_synset.obj_score())
  print()
```

```
<repose.n.03: PosScore=0.625 NegScore=0.0>
Positive score =  0.625
Negative score =  0.0
Objective score =  0.375

<peace.n.03: PosScore=0.125 NegScore=0.5>
Positive score =  0.125
Negative score =  0.5
Objective score =  0.375
```

In [ ]:
```
sentence = "The traffic on the way to work was terrible, but the coffee from

tokens = word_tokenize(sentence)
for token in tokens:
    senti_synsets = list(swn.senti_synsets(token))
    if senti_synsets:
        senti_synset = senti_synsets[0]
        pos_score = senti_synset.pos_score()
        neg_score = senti_synset.neg_score()
        obj_score = senti_synset.obj_score()
        print(f"{token:<9} Positive = {pos_score:<6} Negative = {neg_score:<
    else:
        print(f"{token:<9} No score found")
```

```
The       No score found
traffic   Positive = 0.0    Negative = 0.0    Objective = 1.0
on        Positive = 0.0    Negative = 0.0    Objective = 1.0
the       No score found
way       Positive = 0.0    Negative = 0.0    Objective = 1.0
to        No score found
work      Positive = 0.0    Negative = 0.0    Objective = 1.0
was       Positive = 0.0    Negative = 0.0    Objective = 1.0
terrible  Positive = 0.0    Negative = 0.625  Objective = 0.375
,         No score found
but       Positive = 0.0    Negative = 0.0    Objective = 1.0
the       No score found
coffee    Positive = 0.0    Negative = 0.0    Objective = 1.0
from      No score found
my        No score found
favorite  Positive = 0.25   Negative = 0.0    Objective = 0.75
shop      Positive = 0.0    Negative = 0.0    Objective = 1.0
helped    Positive = 0.0    Negative = 0.0    Objective = 1.0
brighten  Positive = 0.0    Negative = 0.0    Objective = 1.0
my        No score found
day       Positive = 0.0    Negative = 0.0    Objective = 1.0
.         No score found
```

Here, we see that the positive score for "favorite" is moderate, while the negative score for "terrible" is relatively high. This is consistent with our intuitive understanding of the sentence as being somewhat mixed or neutral in overall sentiment, but leaning slightly negative due to the negative sentiment associated with the traffic.

Knowing the sentiment scores for words in a sentence can be very useful in many NLP applications, such as sentiment analysis, opinion mining, and text classification. By analyzing the sentiment of words and phrases in a text, we can gain insight into the attitudes and opinions expressed in that text, and use that information to make predictions or recommendations about how to respond or act on that text. However, it's important to note that sentiment analysis is still a challenging problem, and the accuracy of sentiment scores can be affected by many factors, such as context, sarcasm, and the nuances of language.

Collocations

- In linguistics, a collocation is a sequence of words or terms that co-occur more often than would be expected by chance.
- Collocations can be made up of two or more words and are often idiomatic, meaning that their meaning cannot be inferred from the meanings of their individual words.
- Collocations are important in language learning and natural language processing, as they can provide insights into how words are used together in different contexts.

Collocations of nltk text4

In [ ]:
```
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
```

Mutual information

In [ ]:
```
text = ' '.join(text4.tokens)
vocab = len(set(text4))

gb = text.count('God bless')/vocab
print("probability(God bless) = ", gb)

g = text.count('God')/vocab
print("probability(God) = ", g)

b = text.count('bless')/vocab
print('probability(bless) = ', b)

pmi = math.log2(gb / (g * b))
print('pmi = ', pmi)
```

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

The mutual information score for the "God bless" collocation is 4.15, which is a high score. This suggests that "God" and "bless" are highly associated with each other and are very likely to form a collocation. As "God bless" is a common and frequently used expression in English, particularly in religious and patriotic contexts. The high mutual information score suggests that the "God bless" collocation could be a useful feature in NLP applications that deal with sentiment analysis, topic modeling, or other tasks that involve identifying and analyzing collocations in text.