WordNet

In [ ]:
```
# a simple sentence describing wordnet
text = "WordNet is a large lexical database of English. Nouns, verbs, adjectives an
```

Imports

In [ ]:
```
import nltk
from nltk.corpus import wordnet as wn
```

Nouns

In [ ]:
```
# select an example noun from the text
example_noun = 'database'
# get the synsets for the example noun
synsets = wn.synsets(example_noun)
synsets
```

Out[ ]:
```
[Synset('database.n.01')]
```

In [ ]:
```
# alright, maybe one with more synsets
example_noun = 'word'
synsets = wn.synsets(example_noun)
synsets
```

Out[ ]:
```
[Synset('word.n.01'),
 Synset('word.n.02'),
 Synset('news.n.01'),
 Synset('word.n.04'),
 Synset('discussion.n.02'),
 Synset('parole.n.01'),
 Synset('word.n.07'),
 Synset('son.n.02'),
 Synset('password.n.01'),
 Synset('bible.n.01'),
 Synset('give_voice.v.01')]
```

In [ ]:
```
# let's look at the second synset
synset = synsets[1]

# grab definition, usage examples, and lemmas
print(synset.definition())
print(synset.examples())
print(synset.lemma_names())
```

```
a brief statement
["he didn't say a word about it"]
['word']
```

Wordnet's noun organization is organized into synsets, which are groups of synonyms. Each synset has a unique name, and a unique offset. The offset is the synset's position in the WordNet database file. The name is a concatenation of the synset's offset, the part of speech, and the number of the synset.

Nouns are arranged in a hierarchy, with more general concepts at the top, and more specific concepts at the bottom. The top of the hierarchy is called the root, and is represented by the synset `entity.n.01`. The root is the most general concept, and all other concepts are more specific (you can see this in the example below).

In [ ]:
```
# progress up the hierarchy for our chosen synset
while synset.hypernyms():
    synset = synset.hypernyms()[0]
    print(synset.name())
```

```
statement.n.01
message.n.02
communication.n.02
abstraction.n.06
entity.n.01
```

In [ ]:
```
# reset our synset
synset = synsets[1]
# grab all the hyponyms, hypernyms, meronyms, antonyms, and holonyms
print(synset.hyponyms())
print(synset.hypernyms())
print(synset.part_meronyms())
print(synset.lemmas()[0].antonyms())
print(synset.member_holonyms())
```

```
[]
[Synset('statement.n.01')]
[]
[]
[]
```

In [ ]:
```
# not a great example, let's try a different example noun
example_noun = 'network'
synsets = wn.synsets(example_noun)
synsets
```

Out[ ]:
```
[Synset('network.n.01'),
 Synset('network.n.02'),
 Synset('net.n.06'),
 Synset('network.n.04'),
 Synset('network.n.05'),
 Synset('network.v.01')]
```

In [ ]:
```
synset = synsets[0]
print(synset.hyponyms())
print(synset.hypernyms())
print(synset.part_meronyms())
print(synset.lemmas()[0].antonyms())
print(synset.member_holonyms())
```

```
[Synset('espionage_network.n.01'), Synset('old_boy_network.n.01'), Synset('reticul
um.n.02'), Synset('support_system.n.01')]
[Synset('system.n.02')]
[]
[]
[]
```

Verbs

In [ ]:

```
# let's try a verb
example_verb = 'expressing'
synsets = wn.synsets(example_verb)
synsets
```

Out[ ]:

```
[Synset('express.v.01'),
 Synset('express.v.02'),
 Synset('carry.v.04'),
 Synset('express.v.04'),
 Synset('express.v.05'),
 Synset('press_out.v.03'),
 Synset('express.v.07')]
```

In [ ]:

```
# same as before, but with a verb
synset = synsets[1]

# grab definition, usage examples, and lemmas
print(synset.definition())
print(synset.examples())
print(synset.lemma_names())
```

```
articulate; either verbally or with a cry, shout, or noise
['She expressed her anger', 'He uttered a curse']
['express', 'verbalize', 'verbalise', 'utter', 'give_tongue_to']
```

WordNet's verb organization is similar to that of nouns, but with a few differences. First, verbs are not organized into a hierarchy. Instead, they are organized into a directed acyclic graph (DAG). This means that verbs can have multiple parents, and multiple children. Second, verbs are organized into frames, which are groups of verbs that share similar meanings. Each frame has a unique name, and a unique offset. The offset is the frame's position in the WordNet database file. The name is a concatenation of the frame's offset, and the number of the frame.

In [ ]:

```
# run the verb through morphy to get all possible verb forms
print(wn.morphy(example_verb, wn.NOUN))
print(wn.morphy(example_verb, wn.VERB))
print(wn.morphy(example_verb, wn.ADJ))
print(wn.morphy(example_verb, wn.ADV))
```

```
None
express
None
None
```

Similarity

In [ ]:
```python
# grab two words that are similar
word1 = 'interlinked'
word2 = 'network'

# get the synsets for each word
synsets1 = wn.synsets(word1)
synsets2 = wn.synsets(word2)

print(synsets1)
print(synsets2)
```

```
[Synset('complect.v.01'), Synset('interconnect.v.02')]
[Synset('network.n.01'), Synset('network.n.02'), Synset('net.n.06'), Synset('netwo
rk.n.04'), Synset('network.n.05'), Synset('network.v.01')]
```

In [ ]:
```python
#wu-palmer similarity
print(synsets1[1].wup_similarity(synsets2[0]))
print(synsets2[0].wup_similarity(synsets1[1]))

# lesk similarity
from nltk.wsd import lesk
print(lesk(text, word1))
print(lesk(text, word2))
```

```
0.2222222222222222
0.2222222222222222
Synset('interconnect.v.02')
Synset('network.v.01')
```

Observations:

Similarity in this case seems to be symmetric. If `a` is similar to `b`, then `b` is similar to `a`. This is not always the case, but it is in this case.

The lesk similarity was largely able to capture the meaning in this case.

Sentiwordnet

Sentiwordnet is a database of words and their associated sentiment. Each word is associated with a positive score, a negative score, and an objective score. The positive score is the probability that the word is positive. The negative score is the probability that the word is negative. The objective score is the probability that the word is objective.

In [ ]:
```python
from nltk.corpus import sentiwordnet as swn
```

```
In [ ]:
# an example of a word that is emotionally charged
word = 'hate'
synsets = wn.synsets(word)

# create a sentisynset for each synset
sentisysnsets = []
for synset in synsets:
    sentisysnsets.append(swn.senti_synset(synset.name()))

# print the positive and negative scores for each synset
for sentisynset in sentisysnsets:
    print(sentisynset)
    print(sentisynset.pos_score())
    print(sentisynset.neg_score())
    print(sentisynset.obj_score())
```

```
<hate.n.01: PosScore=0.125 NegScore=0.375>
0.125
0.375
0.5
<hate.v.01: PosScore=0.0 NegScore=0.75>
0.0
0.75
0.25
```

```
In [ ]:
example_sentence = 'You are a very nice person and I enjoy being around you.'

# tokenize the sentence
tokens = nltk.word_tokenize(example_sentence)

# find the polarity of each word in the sentence
print('Word\tPositive\tNegative\tObjective')
for token in tokens:
    synsets = wn.synsets(token)
    if synsets:
        sentisynset = swn.senti_synset(synsets[0].name())
        print(token + '\t' + str(sentisynset.pos_score()) + '\t' + str(sentisynset.
```

```
Word    Positive        Negative        Objective
are     0.0     0.0     1.0
a       0.0     0.0     1.0
very    0.5     0.0     0.5
nice    0.0     0.0     1.0
person  0.0     0.0     1.0
I       0.0     0.0     1.0
enjoy   0.375   0.0     0.625
being   0.0     0.0     1.0
around  0.0     0.0     1.0
```

These scores could be used for sentiment analysis of input text and, importantly, could also be used for tonal matching of response text. I expect these would be especially useful in stock market analysis, where the sentiment of a company's stock could be used to predict its future performance, and a trading bot could rapidly respond to changes in sentiment.

Collocations

A collocation is a sequence of words that occur together more often than would be expected by chance. Some examples of collocations are "New York", "United States", and "United Kingdom".

```
In [ ]: # import text4 from nltk book
        from nltk.book import text4

        # find collocations in text4
        collocations = text4.collocations()
        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
```

```
In [ ]: # calculate mutual information for the 'United States' collocation
        import math
        text = ' '.join(text4.tokens)
        vocab = len(set(text4))
        print(vocab)
        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)
```

```
10025
p(United States) =  0.015860349127182045
p(United) =  0.0170573566084788
p(States) =  0.03301745635910224
pmi =  4.815657649820885
```

A higher pmi score indicates a higher likelihood that the words are a collocation. What seems most interesting to myself though is that the "States" has a much higher frequency than "United", which indicates that "United" is frequently used with "States", but not the other way around. For the purposes of NLP, this could be useful for determining the most important words in a sentence, and for determining the most important words in a document.