Assignment 4: WordNet

Summary of WordNet

WordNet is a lexical database of semantic relationships between words. It organizes nouns, verbs, adjectives, and adverbs into a hierarchy, which are grouped into sets of synonyms called synsets. Synsets may be used to provide short definitions (known as glosses), usage examples, lemmas, hypernyms, hyponyms, meronyms, and holonyms for different words.

Importing NLTK

In [ ]:
```
import nltk
```

Synsets and Relations of Nouns

The code below imports WordNet from NLTK and displays all synsets of the noun "image".

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

wn.synsets('image')
```

Out[ ]:
```
[Synset('image.n.01'),
 Synset('persona.n.02'),
 Synset('picture.n.01'),
 Synset('prototype.n.01'),
 Synset('trope.n.01'),
 Synset('double.n.03'),
 Synset('image.n.07'),
 Synset('image.n.08'),
 Synset('effigy.n.01'),
 Synset('image.v.01'),
 Synset('visualize.v.01')]
```

The code below extracts the definition, usage examples, and lemmas of the 'image.n.08' synset of the noun "image".

In [ ]:
```
image = wn.synset('image.n.08')
print("definition:", image.definition())
print("usage examples:", image.examples())
print("lemmas:", image.lemmas())
```

```
definition: the general impression that something (a person or organization or produc
t) presents to the public
usage examples: ['although her popular image was contrived it served to inspire music
and pageantry', 'the company tried to project an altruistic image']
lemmas: [Lemma('image.n.08.image')]
```

The code below traverses up the WordNet hierarchy from the 'image.n.08' synset of the noun

"image". It uses the function 'hyper' and the NLTK method 'closure()' to return the hypernym of the synset.

WordNet organizes nouns such that there is a constant top-level synset for all nouns. The top level of the hierarchy for nouns has the noun "entity", which the hierarchy traverses up to from a given noun synset.

In [ ]:
```
hyper = lambda s: s.hypernyms()
list(image.closure(hyper))
```

Out[ ]:
```
[Synset('impression.n.02'),
 Synset('appearance.n.01'),
 Synset('quality.n.01'),
 Synset('attribute.n.02'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

The code below displays the hypernyms (higher), hyponyms (lower), meronyms (part of), holonyms (whole), and antonyms (opposite) of the 'image.n.08' synset, if they exist. The synset's lemma is used to find antonyms.

In [ ]:
```
print("hypernyms:", image.hypernyms())
print("hyponyms:", image.hyponyms())
print("meronyms:", image.part_meronyms())
print("holonyms:", image.part_holonyms())
print("antonyms:", image.lemmas()[0].antonyms())
```

```
hypernyms: [Synset('impression.n.02')]
hyponyms: []
meronyms: []
holonyms: []
antonyms: []
```

Synsets and Relations of Verbs

The code below displays all synsets of the verb "consume".

In [ ]:
```
wn.synsets('consume')
```

Out[ ]:
```
[Synset('devour.v.03'),
 Synset('consume.v.02'),
 Synset('consume.v.03'),
 Synset('consume.v.04'),
 Synset('consume.v.05'),
 Synset('consume.v.06')]
```

The code below extracts the definition, usage examples, and lemmas of the 'consume.v.05' synset of the verb "consume".

In [ ]:
```
consume = wn.synset('consume.v.05')
print("definition:", consume.definition())
print("usage examples:", consume.examples())
print("lemmas:", consume.lemmas())
```

```
definition: use up (resources or materials)
usage examples: ['this car consumes a lot of gas', 'We exhausted our savings', 'They
run through 20 bottles of wine a week']
lemmas: [Lemma('consume.v.05.consume'), Lemma('consume.v.05.eat_up'), Lemma('consume.
v.05.use_up'), Lemma('consume.v.05.eat'), Lemma('consume.v.05.deplete'), Lemma('consu
me.v.05.exhaust'), Lemma('consume.v.05.run_through'), Lemma('consume.v.05.wipe_out')]
```

The code below traverses up the WordNet hierarchy from the 'consume.v.05' synset of the verb "consume". It uses the function 'hyper' and the NLTK method 'closure()' to return the hypernym of the synset.

WordNet organizes verbs such that there is no constant top-level synset for all verbs. Tracing the synset for the verb "consume" gives "spend" as the hypernym, the verb "spend" gives "pay" as the hypernym, and so on. The verbs "spend" and "pay" are not very similar semantically.

In [ ]:
```
hyper = lambda s: s.hypernyms()
list(consume.closure(hyper))
```

Out[ ]:
```
[Synset('spend.v.02'),
 Synset('pay.v.01'),
 Synset('give.v.03'),
 Synset('transfer.v.05')]
```

The code below uses the 'morphy()' function to find different forms of the verb "consume".

In [ ]:
```
wn.morphy('consume')
```

Out[ ]:
```
'consume'
```

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

```
None
```

In [ ]:
```
wn.morphy('consume', wn.VERB)
```

Out[ ]:
```
'consume'
```

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

```
None
```

In [ ]:
```
print(wn.morphy('consume', wn.ADV))
```

```
None
```

Word Similarity and Comparison

The code below finds and defines the synsets of two similar words.

In [ ]:
```
print("synset of 'thief':", wn.synsets('thief'))
print("definition of 'thief':", wn.synset('thief.n.01').definition())
print()
print("synset of 'robber':", wn.synsets('robber'))
print("definition of 'robber':", wn.synset('robber.n.01').definition())
```

```
synset of 'thief': [Synset('thief.n.01')]
definition of 'thief': a criminal who takes property belonging to someone else with t
he intention of keeping it or selling it

synset of 'robber': [Synset('robber.n.01')]
definition of 'robber': a thief who steals from someone by threatening violence
```

The code below runs the Wu-Palmer similarity metric to measure the similarity between the words "thief" and "robber". It uses the depth of the words and common ancestor words.

In [ ]:
```
thief = wn.synset('thief.n.01')
robber = wn.synset('robber.n.01')
wn.wup_similarity(thief, robber)
```

Out[ ]:
```
0.96
```

The code below runs the Lesk algorithm using the words "thief" and "robber". This algorithm looks at context words for each word using the given sentences and compares them to words in dictionary glosses to return a synset that has the highest count of overlapping words.

In [ ]:
```
from nltk.wsd import lesk

thief_sent = ['The', 'thief', 'was', 'caught', 'and', 'went', 'to', 'jail', '.']
print("output for 'thief' after algorithm:", lesk(thief_sent, 'thief'))

robber_sent = ['The', 'robber', 'was', 'caught', 'and', 'went', 'to', 'jail', '.']
print("output for 'robber' after algorithm:", lesk(robber_sent, 'robber'))
```

```
output for 'thief' after algorithm: Synset('thief.n.01')
output for 'robber' after algorithm: Synset('robber.n.01')
```

SentiWordNet

SentiWordNet is a lexical resource that is built on top of WordNet. It assigns 3 sentiment scores between 0.0 and 1.0, with the scores adding up to 1.0, for each synset: positivity, negativity, and objectivity. SentiWordNet can be useful for sentiment analysis or opinion mining.

The code below downloads and imports SentiWordNet from NLTK.

In [ ]:
```
#nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
```

The code below finds senti-synsets for the word "move" and displays the polarity scores for each word.

In [ ]:
```
move = swn.senti_synsets('move')
for item in move:
    print(item)
```

```
<move.n.01: PosScore=0.0 NegScore=0.0>
<move.n.02: PosScore=0.0 NegScore=0.0>
<motion.n.03: PosScore=0.0 NegScore=0.125>
<motion.n.06: PosScore=0.0 NegScore=0.0>
<move.n.05: PosScore=0.0 NegScore=0.0>
<travel.v.01: PosScore=0.0 NegScore=0.0>
<move.v.02: PosScore=0.25 NegScore=0.0>
<move.v.03: PosScore=0.0 NegScore=0.0>
<move.v.04: PosScore=0.0 NegScore=0.0>
<go.v.02: PosScore=0.0 NegScore=0.0>
<be_active.v.01: PosScore=0.0 NegScore=0.0>
<move.v.07: PosScore=0.0 NegScore=0.0>
<act.v.01: PosScore=0.0 NegScore=0.0>
<affect.v.05: PosScore=0.125 NegScore=0.125>
<motivate.v.01: PosScore=0.0 NegScore=0.0>
<move.v.11: PosScore=0.375 NegScore=0.25>
<move.v.12: PosScore=0.0 NegScore=0.0>
<move.v.13: PosScore=0.0 NegScore=0.0>
<move.v.14: PosScore=0.0 NegScore=0.0>
<move.v.15: PosScore=0.0 NegScore=0.0>
<move.v.16: PosScore=0.0 NegScore=0.0>
```

The code below displays the polarity for each word in a given sentence, after tokenization.

The scores display how positive or negative a word may be, where many words in the sentence have a neutral polarity. Knowing polarity scores can be useful in NLP applications, such as for text processing by analyzing the sentiment of text to see if it is positive or negative and objective or subjective.

In [ ]:
```python
sent = 'Cats are fun'
tokens = sent.split()
for token in tokens:
    print("word:", token)
    syn_list = swn.senti_synsets(token)
    for item in syn_list:
        print("polarity for", item)
    print()
```

```
word: Cats
polarity for <cat.n.01: PosScore=0.0 NegScore=0.0>
polarity for <guy.n.01: PosScore=0.0 NegScore=0.0>
polarity for <cat.n.03: PosScore=0.0 NegScore=0.125>
polarity for <kat.n.01: PosScore=0.0 NegScore=0.0>
polarity for <cat-o'-nine-tails.n.01: PosScore=0.0 NegScore=0.0>
polarity for <caterpillar.n.02: PosScore=0.0 NegScore=0.0>
polarity for <big_cat.n.01: PosScore=0.0 NegScore=0.0>
polarity for <computerized_tomography.n.01: PosScore=0.0 NegScore=0.0>
polarity for <cat.v.01: PosScore=0.0 NegScore=0.0>
polarity for <vomit.v.01: PosScore=0.0 NegScore=0.0>

word: are
polarity for <are.n.01: PosScore=0.0 NegScore=0.0>
polarity for <be.v.01: PosScore=0.25 NegScore=0.125>
polarity for <be.v.02: PosScore=0.0 NegScore=0.0>
polarity for <be.v.03: PosScore=0.0 NegScore=0.0>
polarity for <exist.v.01: PosScore=0.0 NegScore=0.0>
polarity for <be.v.05: PosScore=0.0 NegScore=0.0>
polarity for <equal.v.01: PosScore=0.125 NegScore=0.125>
polarity for <constitute.v.01: PosScore=0.0 NegScore=0.0>
polarity for <be.v.08: PosScore=0.0 NegScore=0.0>
polarity for <embody.v.02: PosScore=0.0 NegScore=0.0>
polarity for <be.v.10: PosScore=0.0 NegScore=0.0>
polarity for <be.v.11: PosScore=0.0 NegScore=0.0>
polarity for <be.v.12: PosScore=0.0 NegScore=0.0>
polarity for <cost.v.01: PosScore=0.0 NegScore=0.0>

word: fun
polarity for <fun.n.01: PosScore=0.375 NegScore=0.0>
polarity for <fun.n.02: PosScore=0.0 NegScore=0.5>
polarity for <fun.n.03: PosScore=0.125 NegScore=0.125>
polarity for <playfulness.n.02: PosScore=0.0 NegScore=0.0>
```

Collocations and Mutual Information

A collocation is when two or more words occur next to each other more often than expected by chance. A collocation of two words cannot be replaced by synonyms as the semantic of the words is different when they exist together than when they exist as separate words.

The code below imports NLTK Text objects from the book collection.

In [ ]:
```
from nltk.book import *
```

```
*** 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
```

The code below displays all collocations for the 'text4' Text object using the 'collocations()' method.

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
```

The code below calculates mutual information for the collocation 'United States'. The point-wise mutual information (pmi) formula shows that the log of the probability of "United" and "States" occuring together divided by the probability of them occuring separately results in a higher value compared to the count of each of the two word from the total number of words in the Text object. As the pmi for "United States" is positive, the words "United" and "States" appear together more frequently than expected chance and can thus be a collocation.

In [ ]:
```
import math

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

vocab = len(set(text4))
us = text.count('United States')/vocab
print("p(United States) = ", us)
u = text.count('United')/vocab
print("p(United) = ", u)
s = text.count('States')/vocab
print('p(States) = ', s)
pmi = math.log2(us / (u * s))
print('pmi = ', pmi)
```

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