Assignment 3: WordNet

Demonstrating basic skills using WordNet and SentiWordNet

This notebook examines basic techniques for using WordNet and SentiWordNet in Python using JupyterLab

What is WordNet?

WordNet is a lexical database of nouns, verbs, adjectives, and adverbs tied to short definitions of these words (called glosses) alongside examples. WordNet groups words into synonym sets called synsets.

2. Select a noun and output all synsets.

In [221…
```
# Import wordnet
import nltk
from nltk.corpus import wordnet as wn
```

In [222…
```
# Gets all synsets of the noun 'capital'
wn.synsets('capital')
```

Out[222]:
```
[Synset('capital.n.01'),
 Synset('capital.n.02'),
 Synset('capital.n.03'),
 Synset('capital.n.04'),
 Synset('capital.n.05'),
 Synset('capital.n.06'),
 Synset('das_kapital.n.01'),
 Synset('capital.n.08'),
 Synset('capital.s.01'),
 Synset('capital.s.02'),
 Synset('capital.s.03')]
```

3. Select a synset from the list of synsets and output definition, usage examples, and lemmas. Also, print out WordNet hierarchy as far as possible.

In [223…
```
# Extract synset
synEx = wn.synset('capital.n.04')

# Display definition
print("Definition:", synEx.definition(), "\n")
# Display usage examples
print("Examples:", synEx.examples(), "\n")
# Display lemmas
print("Lemmas:", synEx.lemmas(), "\n")
```

```
# Traverse up WordNet hierarchy as far as possible, outputting synsets as it goes
# iterate over synsets
synIterate = wn.synsets('capital', pos = wn.NOUN)
for sense in synIterate:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" +
```

```
Definition: one of the large alphabetic characters used as the first letter in wri
ting or printing proper names and sometimes for emphasis

Examples: ['printers once kept the type for capitals and for small letters in sepa
rate cases; capitals were kept in the upper half of the type case and so became kn
own as upper-case letters']

Lemmas: [Lemma('capital.n.04.capital'), Lemma('capital.n.04.capital_letter'), Lemm
a('capital.n.04.uppercase'), Lemma('capital.n.04.upper-case_letter'), Lemma('capit
al.n.04.majuscule')]

Synset: capital.n.01(assets available for use in the production of further assets)
         Lemmas:['capital', 'working_capital']
Synset: capital.n.02(wealth in the form of money or property owned by a person or
business and human resources of economic value)
         Lemmas:['capital']
Synset: capital.n.03(a seat of government)
         Lemmas:['capital']
Synset: capital.n.04(one of the large alphabetic characters used as the first lett
er in writing or printing proper names and sometimes for emphasis)
         Lemmas:['capital', 'capital_letter', 'uppercase', 'upper-case_letter', 'm
ajuscule']
Synset: capital.n.05(a center that is associated more than any other with some act
ivity or product)
         Lemmas:['capital']
Synset: capital.n.06(the federal government of the United States)
         Lemmas:['Capital', 'Washington']
Synset: das_kapital.n.01(a book written by Karl Marx (1867) describing his economi
c theories)
         Lemmas:['Das_Kapital', 'Capital']
Synset: capital.n.08(the upper part of a column that supports the entablature)
         Lemmas:['capital', 'chapiter', 'cap']
```

WordNet organizes nouns by their connections to other synsets through hierarchical relations between hyponyms and hypernyms, called a hyponym-hypernym relations. Hypo, meaning under, and hyper, meaning over, describes the positions of these words in the hierarchy. Synsets also have other relations, such as meronym (part of), holonym (whole), and troponym (more specific action).

4. Output hypernyms, hyponyms, meronyms, and holonyms.

In [224…

```
# Re-initialize capital synset
capital = wn.synset('capital.n.04')
# Print hypernyms, hyponyms, meronyms, and holonyms
print("Hypernyms:", capital.hypernyms())
print("Hyponyms:", capital.hyponyms())
print("Meronyms:", capital.part_meronyms())
print("Holonyms:", capital.part_holonyms())

# Print all antonyms for each lemma
antList = []
for lemma in capital.lemmas():
    if lemma.antonyms():
        antList.append(lemma.antonyms())
print("Antonyms:", antList)
```

```
Hypernyms: [Synset('character.n.08')]
Hyponyms: [Synset('small_capital.n.01')]
Meronyms: []
Holonyms: []
Antonyms: [[Lemma('small_letter.n.01.lowercase')]]
```

5. Select a verb and output all synsets.

In [225…

```
# Get dance synsets
danceSyn = wn.synsets('dance', pos = wn.VERB)
print(danceSyn)
```

```
[Synset('dance.v.01'), Synset('dance.v.02'), Synset('dance.v.03')]
```

6. Select a synset from the list of synsets and output definition, usage examples, and lemmas. Also, print out WordNet hierarchy as far as possible.

In [226…

```
# Extract synset
dance = wn.synset('dance.v.01')

# Display definition
print("Definition:", dance.definition(), "\n")
# Display usage examples
print("Examples:", dance.examples(), "\n")
# Display lemmas
print("Lemmas:", dance.lemmas(), "\n")

# Traverse up WordNet hierarchy as far as possible, outputting synsets as it goes
# iterate over synsets
synIterate = wn.synsets('dance', pos = wn.VERB)
for sense in synIterate:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" +
Definition: move in a graceful and rhythmical way

Examples: ['The young girl danced into the room']

Lemmas: [Lemma('dance.v.01.dance')]

Synset: dance.v.01(move in a graceful and rhythmical way)
         Lemmas:['dance']
Synset: dance.v.02(move in a pattern; usually to musical accompaniment; do or perf
orm a dance)
         Lemmas:['dance', 'trip_the_light_fantastic', 'trip_the_light_fantastic_to
e']
Synset: dance.v.03(skip, leap, or move up and down or sideways)
         Lemmas:['dance']
```

WordNet also can organize verbs by their connections to other synsets through hierarchical relations between hyponyms and hypernyms, called a hyponym-hypernym relations. Hypo, meaning under, and hyper, meaning over, describes the positions of these words in the hierarchy. Synsets also have other relations, such as meronym (part of), holonym (whole), and troponym (more specific action).

7. Use morphy to find as many different forms of the word dance as possible.

In [227…

```
# Get morphy for the verb 'dance'
morph = wn.morphy('dance', wn.VERB)
print(morph)
```

```
dance
```

8. Select two words that I think might be similar and run the Wu-Palmer similarity metric and the Lesk algorithm.

In [228…

```
rob = wn.synset('rob.v.01')
steal = wn.synset('steal.v.01')

# Calculate Wu-Palmer similarity metric
wn.wup_similarity(rob, steal)
```

Out[228]:

```
0.8
```

In [229…

```
# Look at all definitions for 'rob'
for ss in wn.synsets('rob'):
    print(ss, ss.definition())
```

```
Synset('rob.v.01') take something away by force or without the consent of the owne
r
Synset('overcharge.v.01') rip off; ask an unreasonable price
```

In [230…

```
sent = ['the', 'thief', 'took', 'my', 'watch', 'without', 'my', 'consent']

from nltk.wsd import lesk

# Use Lesk algorithm to find most similar synset based on context sentence
```

```
print(lesk(sent, 'rob', 'v'))
```

```
Synset('rob.v.01')
```

In [231…

```
# Look at all definitions for 'rob'
for ss in wn.synsets('steal'):
    print(ss, ss.definition())
```

```
Synset('bargain.n.02') an advantageous purchase
Synset('steal.n.02') a stolen base; an instance in which a base runner advances sa
fely during the delivery of a pitch (without the help of a hit or walk or passed b
all or wild pitch)
Synset('steal.v.01') take without the owner's consent
Synset('steal.v.02') move stealthily
Synset('steal.v.03') steal a base
```

In [232…

```
sent = ['the', 'robber', 'took', 'my', 'watch', 'without', 'my', 'consent']

# Use Lesk algorithm to find most similar synset based on context sentence

print(lesk(sent, 'steal', 'v'))
```

```
Synset('steal.v.01')
```

The Wu-Palmer algorithm performs well on words with fewer synsets because there is less room for abiguity between the words. When there are fewer synsets, it often accurately measures the similarity between two words.

The Lesk algorithm accurately found the most similar synset for the sentence I provided. Even when switching some words around, it still produced accurate results.

9. Describe SentiWordNet, its functionality, and possible use cases. Find senti-synsets of an emotionally charged word and output the polarity scores. Output the polarity scores of each word in a sentence.

SentiWordNet is a lexical resource designed around WordNet that calculates 3 sentiment scores (positive, negative, objective) for each synset. This can be a powerful tool for sentiment analysis, such as auto scanning reviews for a product to get an estimate on its reception.

In [233…

```
from nltk.corpus import sentiwordnet as swn


# Choose emotionally charged word
passion = swn.senti_synset('passion.n.01')
print(passion)
print("Positive score =", passion.pos_score())
print("Negative score =", passion.neg_score())
print("Objective score =", passion.obj_score())
print()

# Print out senti-synsets
senti_list = list(swn.senti_synsets('passion'))
```

```
for item in senti_list:
    print(item)
```

```
<passion.n.01: PosScore=0.5 NegScore=0.0>
Positive score = 0.5
Negative score = 0.0
Objective score = 0.5

<passion.n.01: PosScore=0.5 NegScore=0.0>
<heat.n.04: PosScore=0.5 NegScore=0.125>
<rage.n.03: PosScore=0.625 NegScore=0.0>
<mania.n.01: PosScore=0.0 NegScore=0.125>
<passion.n.05: PosScore=0.625 NegScore=0.0>
<love.n.02: PosScore=0.375 NegScore=0.0>
<passion.n.07: PosScore=0.0 NegScore=0.5>
```

In [234…

```
# Find the polarity scores of all words in a sentence
sent = "I really don't like how things are being run around here"
# Split sentence into tokens
tokens = sent.split()
for token in tokens:
    neg = 0
    pos = 0
    obj = 0
    # Average senti scores of each synset for each word
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()
        obj += syn.obj_score()
    print("Word = {:s};   Pos = {:.2f}, Neg = {:.2f}, Obj = {:.2f}\n".format(token,
Word = I;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = really;   Pos = 0.62, Neg = 0.00, Obj = 0.38

Word = don't;   Pos = 0.00, Neg = 0.00, Obj = 0.00

Word = like;   Pos = 0.12, Neg = 0.00, Obj = 0.88

Word = how;   Pos = 0.00, Neg = 0.00, Obj = 0.00

Word = things;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = are;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = being;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = run;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = around;   Pos = 0.00, Neg = 0.00, Obj = 1.00

Word = here;   Pos = 0.00, Neg = 0.00, Obj = 1.00
```

The scores for each of the words tends to vary widely depending on with synset is used. The current system is rudimentary, so it picks the top synset returned by swn.senti_synsets(), which can sometimes give inaccurate results. Because of this, it may be better to implement a system which finds the most accurate synset given the context of the sentence. Generally speaking, the scores perform well and a similar system could be used in an NLP application as described above, where a developer needs to perform sentiment analysis on a large databank of reviews for a product.

10. Describe collocation. Output collocations for text4, the Inaugural corpus. Select one and calculate its mutual information.

A collocation is when two or more words occur together more often than chance would suggest. A collocation can be found if the substitution of a word in a set of words would distort its meaning. For example, the word 'United' in the collocation 'United States' could not be substituted for another.

In [235…

```python
# Import text4 from nltk.book
import nltk
from nltk.book import *
text4

# Print text4 collocations
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
```

In [236…

```python
import math

# Store text4 as 1 string
text = ' '.join(text4.tokens)
# Calculate vocabulary of text4
vocab = len(set(text4))
# Calculate Mutual Information for "United States"
US = text.count('United States')/vocab
U = text.count('United')/vocab
S = text.count('States')/vocab

# Calculate pmi
pmi = math.log2(US / (U * S))

# Print Mutual Information
print('p(United States) =', US)
print('p(United) =', U)
print('p(States) =', S)
print('pmi =', pmi)
```

```
p(United States) = 0.015860349127182045
p(United) = 0.0170573566084788
p(States) = 0.033017456359102244
pmi = 4.815657649820885
```

In [237…

```python
hg = text.count('of the')/vocab
print("p(of the) = ",hg )
o = text.count('of')/vocab
print("p(of) = ", o)
t = text.count('the ')/vocab # space so it doesn't capture 'their' etc.
print('p(the) = ', t)
pmi = math.log2(hg / (o * t))
print('pmi = ', pmi)
```

```
p(of the) =  0.20089775561097256
p(of) =  0.7487281795511221
p(the) =  0.9533167082294264
pmi =  -1.8290080938996587
```

The mutual information for 'United States' is higher than that of 'of the', meaning it is very likely that 'United States' is a collocation