Assignment 3: Wordnet

Abdullah Hasani - AHH190004

About Wordnet

Wordnet is an English language database that is used in natural language processing and is provided through the NLTK library. Wordnet's database contains relationships between words, such as a word's synonyms, antonyms, hypernyms and hyponyms, meronyms, and holonyms. It also has methods that can be applied to synsets, such as getting the gloss (definition) of a word, getting use case examples, and more.

```
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk

# Select a noun
noun = 'card'

# Output all synsets
synsets = wn.synsets(noun)
print(f"All synsets for {noun}:")
for synset in synsets:
    print(synset)

# Select a synset
selected = synsets[0]
print(f"\nSelected: {selected}")

# Synset definition
print(f"Definition: {selected.definition()}")

# Synset usage examples
print(f"Usage examples: {selected.examples()}")

# Synset lemmas
print(f"Lemmas: {[lemma.name() for lemma in selected.lemmas()]}")

# Traversal over WordNet heirarchy
hypernyms = lambda s: s.hypernyms()
hyper_list = list(selected.closure(hypernyms))

# Outputting the synsets
print("\nTraversing up the hierarchy:")
for word in hyper_list:
    print(word)

# Outputting the hypernyms
```

```
print(f'\nHypernyms: {selected.hypernyms()}')

# Outputting the hyponyms
print(f'\nHyponyms: {selected.hyponyms()}')

# Outputting the meronyms
print(f'\nMeronyms: {selected.part_meronyms()}')

# Outputting the holonyms
print(f'\nHolonyms: {selected.part_holonyms()}')

# Outputting the antonyms
print(f'\nAntonyms: {selected.lemmas()[0].antonyms()}')
```

```
/Users/hasani/opt/anaconda3/lib/python3.9/site-packages/scipy/
__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is
required for this version of SciPy (detected version 1.24.2
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"

All synsets for card:
Synset('card.n.01')
Synset('card.n.02')
Synset('card.n.03')
Synset('card.n.04')
Synset('wag.n.01')
Synset('poster.n.01')
Synset('calling_card.n.02')
Synset('card.n.08')
Synset('menu.n.01')
Synset('batting_order.n.01')
Synset('circuit_board.n.01')
Synset('tease.v.07')
Synset('card.v.02')

Selected: Synset('card.n.01')
Definition: one of a set of small pieces of stiff paper marked in
various ways and used for playing games or for telling fortunes
Usage examples: ['he collected cards and traded them with the other
boys']
Lemmas: ['card']

Traversing up the hierarchy:
Synset('paper.n.01')
Synset('material.n.01')
Synset('substance.n.01')
Synset('matter.n.03')
Synset('part.n.01')
Synset('physical_entity.n.01')
Synset('relation.n.01')
```

```
Synset('entity.n.01')
Synset('abstraction.n.06')

Hypernyms: [Synset('paper.n.01')]

Hyponyms: [Synset('playing_card.n.01'), Synset('punched_card.n.01'),
Synset('tarot_card.n.01'), Synset('trading_card.n.01')]

Meronyms: []

Holonyms: []

Antonyms: []
```

```
/Users/hasani/.local/lib/python3.9/site-packages/nltk/corpus/reader/
wordnet.py:604: UserWarning: Discarded redundant search for
Synset('entity.n.01') at depth 7
  for synset in acyclic_breadth_first(self, rel, depth):
```

WordNet Organization for Nouns

WordNet is organizes nouns into "synsets", or sets of synonyms, which describes a word and the words that are closely related in definition to it. It then also has antonym relationships with other words to represent the opposite of a word, if it exists. It also has part-whole relationships, known as meronyms and holonyms, where a meronym is a part of of another thing and a holonym is the thing a word is a part of. It also has hypernyms and hyponyms, where a hypernym is a higher version of the word and a hyponym is a lower version of the word. These organized relationships exist to allow for ease of use and access when being used in the context of natural language processing.

```
# Select a verb
verb = 'meandered'

# Output all synsets
synsets = wn.synsets(verb, pos=wn.VERB)
print(f"All synsets for {verb}:")
for synset in synsets:
    print(synset)

# Select a synset
selected = synsets[0]
print(f"\nSelected: {selected}")

# Synset definition
print(f"Definition: {selected.definition()}")

# Synset usage examples
print(f"Usage examples: {selected.examples()}")
```

```
# Synset lemmas
print(f"Lemmas: {[lemma.name() for lemma in selected.lemmas()]}")

# Traversal over WordNet heirarchy
hypernyms = lambda s: s.hypernyms()
hyper_list = list(selected.closure(hypernyms))

# Outputting the synsets
print("\nTraversing up the hierarchy:")
for word in hyper_list:
    print(word)
```

```
All synsets for meandered:
Synset('weave.v.04')

Selected: Synset('weave.v.04')
Definition: to move or cause to move in a sinuous, spiral, or circular
course
Usage examples: ['the river winds through the hills', 'the path
meanders through the vineyards', 'sometimes, the gout wanders through
the entire body']
Lemmas: ['weave', 'wind', 'thread', 'meander', 'wander']

Traversing up the hierarchy:
Synset('travel.v.01')
```

WordNet Organization for Verbs

Much like nouns, verbs are organized based on their relationships to other words and to words similar in meaning to them. Each version of a word can be represented by a synset, which can be related to other synsets through semantic relationships such as hypernyms and hyponyms, meronyms and holonyms, and antonyms. These relationships exist to allow for ease of use and access when being used in the context of natural language processing.

```
# Use morphy to find different forms of the word
morphy_output = set()
for synset in wn.synsets(verb):
    for lemma in synset.lemmas():
        morphy_output.add(wn.morphy(lemma.name()))
print(f'Output of all forms of the word \'{verb}\' using morphy:\
n{morphy_output}')

# Select two words that might be similar
word1 = 'run'
word2 = 'sprint'

# Find the specific synsets you are interested in
run = wn.synsets(word1)
sprint = wn.synsets(word2)
```

```python
# Output word1 definitions
print(f"\nAll synsets for {word1}:")
for synset in run:
    print(f'{synset} - {synset.definition()}')

# Output word2 definitions
print(f"\nAll synsets for {word2}:")
for synset in sprint:
    print(f'{synset} - {synset.definition()}')

chosen_word_1 = run[6]
chosen_word_2 = sprint[1]

# Wu-Palmer similarity metric
print(f'\nWu-Palmer similarity metric between {chosen_word_1} and
{chosen_word_2}: {wn.wup_similarity(chosen_word_1, chosen_word_2)}')

# Lesk algorithm for word 1
sentence1 = 'He broke into a run'
sentence1_list = sentence1.split(' ')
lesk1 = lesk(sentence1_list, word1, 'v')
print(f'\nLesk Algorithm for {chosen_word_1} on \'{sentence1}\':
{lesk1}')

# Lesk algorithm for word 2
sentence2 = 'Joe sprinted across the finish line'
sentence2_list = sentence2.split(' ')
lesk2 = lesk(sentence2_list, word2, 'v')
print(f'\nLesk Algorithm for {chosen_word_2} on \'{sentence2}\':
{lesk2}')
```

```
Output of all forms of the word 'meandered' using morphy:
{'thread', 'wander', 'meander', 'wind', 'weave'}

All synsets for run:
Synset('run.n.01') - a score in baseball made by a runner touching all
four bases safely
Synset('test.n.05') - the act of testing something
Synset('footrace.n.01') - a race run on foot
Synset('streak.n.01') - an unbroken series of events
Synset('run.n.05') - (American football) a play in which a player
attempts to carry the ball through or past the opposing team
Synset('run.n.06') - a regular trip
Synset('run.n.07') - the act of running; traveling on foot at a fast
pace
Synset('run.n.08') - the continuous period of time during which
something (a machine or a factory) operates or continues in operation
Synset('run.n.09') - unrestricted freedom to use
Synset('run.n.10') - the production achieved during a continuous
period of operation (of a machine or factory etc.)
```

```
Synset('rivulet.n.01') - a small stream
Synset('political_campaign.n.01') - a race between candidates for
elective office
Synset('run.n.13') - a row of unravelled stitches
Synset('discharge.n.06') - the pouring forth of a fluid
Synset('run.n.15') - an unbroken chronological sequence
Synset('run.n.16') - a short trip
Synset('run.v.01') - move fast by using one's feet, with one foot off
the ground at any given time
Synset('scat.v.01') - flee; take to one's heels; cut and run
Synset('run.v.03') - stretch out over a distance, space, time, or
scope; run or extend between two points or beyond a certain point
Synset('operate.v.01') - direct or control; projects, businesses, etc.
Synset('run.v.05') - have a particular form
Synset('run.v.06') - move along, of liquids
Synset('function.v.01') - perform as expected when applied
Synset('range.v.01') - change or be different within limits
Synset('campaign.v.01') - run, stand, or compete for an office or a
position
Synset('play.v.18') - cause to emit recorded audio or video
Synset('run.v.11') - move about freely and without restraint, or act
as if running around in an uncontrolled way
Synset('tend.v.01') - have a tendency or disposition to do or be
something; be inclined
Synset('run.v.13') - be operating, running or functioning
Synset('run.v.14') - change from one state to another
Synset('run.v.15') - cause to perform
Synset('run.v.16') - be affected by; be subjected to
Synset('prevail.v.03') - continue to exist
Synset('run.v.18') - occur persistently
Synset('run.v.19') - carry out a process or program, as on a computer
or a machine
Synset('carry.v.15') - include as the content; broadcast or publicize
Synset('run.v.21') - carry out
Synset('guide.v.05') - pass over, across, or through
Synset('run.v.23') - cause something to pass or lead somewhere
Synset('run.v.24') - make without a miss
Synset('run.v.25') - deal in illegally, such as arms or liquor
Synset('run.v.26') - cause an animal to move fast
Synset('run.v.27') - be diffused
Synset('run.v.28') - sail before the wind
Synset('run.v.29') - cover by running; run a certain distance
Synset('run.v.30') - extend or continue for a certain period of time
Synset('run.v.31') - set animals loose to graze
Synset('run.v.32') - keep company
Synset('run.v.33') - run with the ball; in such sports as football
Synset('run.v.34') - travel rapidly, by any (unspecified) means
Synset('ply.v.03') - travel a route regularly
Synset('hunt.v.01') - pursue for food or sport (as of wild animals)
Synset('race.v.02') - compete in a race
```

```
Synset('move.v.13') - progress by being changed
Synset('melt.v.01') - reduce or cause to be reduced from a solid to a
liquid state, usually by heating
Synset('ladder.v.01') - come unraveled or undone as if by snagging
Synset('run.v.41') - become undone

All synsets for sprint:
Synset('dash.n.02') - a quick run
Synset('sprint.v.01') - run very fast, usually for a short distance

Wu-Palmer similarity metric between Synset('run.n.07') and
Synset('sprint.v.01'): 0.125

Lesk Algorithm for Synset('run.n.07') on 'He broke into a run':
Synset('run.v.29')

Lesk Algorithm for Synset('sprint.v.01') on 'Joe sprinted across the
finish line': Synset('sprint.v.01')
```

Observations about the Wu-Palmer similarity metric and the Lesk algorithm

Both the Wu-Palmer similarity metric and the Lesk algorithm are ways oof determining the similarity between two words. The Wu-Palmer similarity metric is a better way of measuring the similarity between two words, and does so by calculating their distance from the closest common hypernym. The Lesk algorithm, on the other hand, determines the most common senses of a word in a context. They are not suitable for 1 to 1 comparisions, and should be applied differently depending on the needs of the program.

About SentiWordNet

SentiWordNet is a tool used in natural language processing to conduct sentiment analysis on a text. This tool can analyze a text to see if it is positive or negative and how objective it is, and does so by breaking a text down to the words it is made up of and looking up the corresponding synsets. Each synset has a sentiment score, and SentiWordNet combines all the synset scores to get the overall sentiment score.

Some use cases for SentiWordNet can be in figuring out how well-liked a product or movie is by looking at reviews. A score can then be assigned as to how generally liked the product is, and this can be compared to other products to see which is the better one. Other use cases may be for ad targeting, where based on a customer's thoughts on different topics ads can be targeted towards them, or for guessing how a stock price will do based on seniment towards that company on social media.

```
import nltk
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
from nltk.tokenize import word_tokenize
```

```python
# Select an emotionally charged word
charged_word = 'desperate'

# Find its senti-synsets
synsets = swn.senti_synsets(charged_word)

# Output the polarity scores for each word
for synset in synsets:
    print(f'\nName: {synset.synset.name()}')
    print(f'Positive Score: {synset.pos_score()}')
    print(f'Negative Score: {synset.neg_score()}')
    print(f'Objective Score: {synset.obj_score()}')

# Make up a sentence
sentence = 'Bob was desperate after he lost his job due to the mass
layoffs in the tech industry.'
sentence = sentence.split(' ')

# Output the polarity for each word in the sentence
print('\n\nPolarity for each word in the sentence:')
for word in sentence:
    try:
        synset = list(swn.senti_synsets(word))[0]
        print(f'Word: {word}')
        print(f'Polarity: {synset.pos_score() - synset.neg_score()}\
n')
    except:
        print(f'\'{word}\' has no synset\n')
```

```
[nltk_data] Downloading package sentiwordnet to
[nltk_data]     /Users/hasani/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!


Name: desperate.n.01
Positive Score: 0.0
Negative Score: 0.25
Objective Score: 0.75

Name: despairing.s.01
Positive Score: 0.0
Negative Score: 0.5
Objective Score: 0.5

Name: desperate.s.02
Positive Score: 0.125
Negative Score: 0.25
Objective Score: 0.625

Name: desperate.s.03
```

```
Positive Score: 0.125
Negative Score: 0.625
Objective Score: 0.25

Name: desperate.s.04
Positive Score: 0.5
Negative Score: 0.125
Objective Score: 0.375

Name: desperate.s.05
Positive Score: 0.5
Negative Score: 0.125
Objective Score: 0.375

Name: desperate.s.06
Positive Score: 0.0
Negative Score: 0.5
Objective Score: 0.5


Polarity for each word in the sentence:
Word: Bob
Polarity: 0.0

Word: was
Polarity: 0.0

Word: desperate
Polarity: -0.25

Word: after
Polarity: 0.0

Word: he
Polarity: 0.0

Word: lost
Polarity: 0.0

'his' has no synset

Word: job
Polarity: 0.0

Word: due
Polarity: 0.375

'to' has no synset
```

```
'the' has no synset

Word: mass
Polarity: 0.0

Word: layoffs
Polarity: 0.0

Word: in
Polarity: 0.0

'the' has no synset

Word: tech
Polarity: 0.0

'industry.' has no synset
```

Observations of the Scores and the Utility of Knowing these Scores in an NLP Application

From looking at the scores outputted, one observation that can be made is that the context in which a word is used can change the sentiment value of the word, and that different uses of the word can have different polarities. In an NLP application, knowing these scores can be useful because one can use the sentiment of the word for tasks such as sentiment analysis on a text, among other projects. See the section 'About SentiWordNet' for more details.

Collocations

A collocation is when two or more words occur together in a language as a saying, and where substituting one word for any synonym of the word would result in a sentence that does not make sense. Collocations can be used in the context of NLP to help identify patterns in usage for various words and phrases.

For example, a "strong coffee" is not the same thing as a "muscular coffee", and the phrase "muscular coffee" does not make sense.

```
import math
nltk.download('inaugural')
from nltk.corpus import inaugural
from nltk.text import Text


# Load text4 with inaugural corpus
text4 = Text(inaugural.words())

# Output collocations for text4
print(text4.collocations())
```

```python
# Select one of the collocations identified by NLTK
selected_collocation = list(text4.collocation_list())[1]
print(f'\nSelected Collocation: {selected_collocation}')


# Calculate mutual information using equation:
LOG2(P(x,y)/[P(x)*P(y)])

# Load up raw text4 data and tokenize
text4 = inaugural.raw()
tokens = nltk.word_tokenize(text4)

# Calculate number of tokens
text_len = len(tokens)
print(f'Number of tokens: {text_len}')

# Stringify selected collocation and tokens
word = ' '.join(selected_collocation)
text = ' '.join(tokens)

# Number of times selected collocation occurs divided by text len
sc = text.count(word)/text_len
print(f'P({word}) = {sc}')

# Number of times first word in selected collocation occurs divided by
text len
first_word = text.count(selected_collocation[0])/text_len
print(f'P({selected_collocation[0]}) = {first_word}')

# Number of times second word in selected collocation occurs divided
by text len
second_word = text.count(selected_collocation[1])/text_len
print(f'P({selected_collocation[1]}) = {second_word}')

# PMI score logic: log2(P(both words)/[P(first word)*P(second word)])
pmi = math.log2(sc / (first_word * second_word))
print(f'PMI = {pmi}')
```

```
[nltk_data] Downloading package inaugural to
[nltk_data]     /Users/hasani/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!

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
None
```

```
Selected Collocation: ('fellow', 'citizens')
Number of tokens: 152019
P(fellow citizens) = 0.0004012656312697755
P(fellow) = 0.009901203139081299
P(citizens) = 0.0017760937777514653
PMI = 7.969781781267196
```

Commentary on the Results of the Mutual Information Formula

The mutual information formula were found using point-wise mutual information. The results of the formula show that the because the PMI score is a positive number, then the selected phrase is likely to be a collocation. We can also input a generic phrase and calculate the mutual information score of the phrase, and will see that the phrase has a significantly lower, if not negative, PMI score, leading to the conclusion that the selected phrase is a collocation.