WordNet

Assignment 4
CS 4395.001: Human Language Technologies
Hannah Valena - HCV180000

WordNet Summary

WordNet is a database for the English language that is used to help with natural language processing. It hierarchically organizes nouns, verbs, adjectives, and adverbs, listing for each word:

- glosses, which are short definitions of a word
- synsets, which are synonym sets for a word
- usage examples of a word
- relations to other words

```
# imports to use nltk's wordnet
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Organization of Nouns

The top-level synset of nouns in WordNet is 'entity.n.01', as seen in the below output. As you traverse to higher levels, the higher level is either a hypernym (higher) or holonym (whole) of the level below. Conversely, the lower level is a hyponym (lower) or meronym (part of) of the level above.

```
# output all synsets of noun 'dragon'
print(wn.synsets('dragon'))

# selecting one synset of dragon
dragon_synset = wn.synset('dragon.n.01')

# extract synset definition, usage examples, and lemmas
print('\ndragon.n.01:')
print(f'\ndefinition: {dragon_synset.definition()}')
```

```
print(f'examples: {dragon_synset.examples()}')
print(f'lemmas: {dragon_synset.lemmas()}')

# traverse the hierarchy
print('\nTraversing the hierarchy:')
hyp = dragon_synset.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
  print(hyp)
  if hyp == top:
    break;
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]

# hypernyms of dragon.n.01
print(f'\nHypernyms: {dragon_synset.hypernyms()}')
print(f'Hyponyms: {dragon_synset.hyponyms()}')
print(f'Meronyms {dragon_synset.part_meronyms()}')
print(f'Holonyms: {dragon_synset.member_holonyms()}')
print(f'Antonyms: {dragon_synset.lemmas()[0].antonyms()}')
```

```
[Synset('dragon.n.01'), Synset('dragon.n.02'), Synset('draco.n.02'), Synset('dra

dragon.n.01:

definition: a creature of Teutonic mythology; usually represented as breathing f
examples: []
lemmas: [Lemma('dragon.n.01.dragon'), Lemma('dragon.n.01.firedrake')]

Traversing the hierarchy:
Synset('mythical_monster.n.01')
Synset('monster.n.01')
Synset('imaginary_being.n.01')
Synset('imagination.n.01')
Synset('creativity.n.01')
Synset('ability.n.02')
Synset('cognition.n.01')
Synset('psychological_feature.n.01')
Synset('abstraction.n.06')
Synset('entity.n.01')

Hypernyms: [Synset('mythical_monster.n.01')]
Hyponyms: [Synset('wyvern.n.01')]
Meronyms []
Holonyms: []
Antonyms: []
```

Organization of Verbs

Unlike nouns, verbs do not have a common top-level synset in WordNet. As seen in the code below, the top level for 'cook' is 'make.v.03', while the top level for 'swim' is 'travel.v.01'

```
# output all synsets of verb 'cook'
print(wn.synsets('cook'))

# selecting one synset of 'cook'
cook_synset = wn.synset('cook.v.01')

# extract synset definition, usage examples, and lemmas
print('\ncook.v.01:')
print(f'\ndefinition: {cook_synset.definition()}')
print(f'examples: {cook_synset.examples()}')
print(f'lemmas: {cook_synset.lemmas()}')

# traverse the hierarchy
print('\nTraversing the hierarchy:')
hyp = cook_synset.hypernyms()[0]
top = wn.synset('make.v.03') # had to make top = make, otherwise infinite loop
while hyp:
  print(hyp)
  if hyp == top:
    break;
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
```

```
[Synset('cook.n.01'), Synset('cook.n.02'), Synset('cook.v.01'), Synset('cook.v.0

cook.v.01:

definition: prepare a hot meal
examples: ["My husband doesn't cook"]
lemmas: [Lemma('cook.v.01.cook')]

Traversing the hierarchy:
Synset('create_from_raw_material.v.01')
Synset('make.v.03')
```

Let's try the same with a different verb, 'swim':

```
# output all synsets of verb 'swim'
print(wn.synsets('swim'))

# selecting one synset of 'swim'
swim_synset = wn.synset('swim.v.01')

# extract synset definition, usage examples, and lemmas
print('\nswim.v.01:')
print(f'\ndefinition: {swim_synset.definition()}')
print(f'examples: {swim_synset.examples()}')
print(f'lemmas: {swim_synset.lemmas()}')

# traverse the hierarchy
```

```
print('\nTraversing the hierarchy:')
hyp = swim_synset.hypernyms()[0]
top = wn.synset('travel.v.01') # had to make top = travel, otherwise infinite loop
while hyp:
  print(hyp)
  if hyp == top:
    break;
  if hyp.hypernyms():
    hyp = hyp.hypernyms()[0]
```

```
[Synset('swimming.n.01'), Synset('swim.v.01'), Synset('float.v.02'), Synset('swi

swim.v.01:

definition: travel through water
examples: ['We had to swim for 20 minutes to reach the shore', 'a big fish was s
lemmas: [Lemma('swim.v.01.swim')]

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

Morphy

Morphy finds and removes suffixes. For example, for nouns, morphy() removes 's', 'ses', etc. and for verbs, morphy() removes 's', 'ies', 'ed', 'ing', etc.

```
print('Use Morphy to find different forms of \'cook\':')
print('cook -> ' + wn.morphy('cook', wn.VERB))
print('cooking -> ' + wn.morphy('cooking', wn.VERB))
print('cooked -> ' + wn.morphy('cooked', wn.VERB))
print('cooks -> ' + wn.morphy('cooks', wn.VERB))

print('\nUse Morphy to find different forms of \'swim\':')
print('swim -> ' + wn.morphy('swim', wn.VERB))
print('swimming -> ' +wn.morphy('swimming', wn.VERB))
print('swam -> ' + wn.morphy('swam', wn.VERB))
print('swum -> ' + wn.morphy('swum', wn.VERB))
```

```
Use Morphy to find different forms of 'cook':
cook -> cook
cooking -> cook
cooked -> cook
cooks -> cook

Use Morphy to find different forms of 'swim':
swim -> swim
swimming -> swim
swam -> swim
swum -> swim
```

Wu-Palmer Similarity Metric & Lesk Algorithm

Wu-Palmer

The Wu-Palmer Similarity metric calculates the similarity from 0 (little) to 1 (identity) of two synsets.

As you can see in the code below, the synsets for computer and laptop have a Wu-Palmer similarity of about 0.82, meaning they are quite similar.

Lesk Algorithm

The Lesk Algorithm looks at the context of a word and compares it to dictionary glosses for word overlaps.

As you can see in the code below, the word 'break' could be related to various synsets depending on the context.

```
from nltk.wsd import lesk

# similar words
print(wn.synsets('computer'))
print(wn.synsets('laptop'))

comp_synset = wn.synset('computer.n.01')
laptop_synset = wn.synset('laptop.n.01')

# wu-palmer
wp_sim = wn.wup_similarity(comp_synset, laptop_synset)
print(f'\nWu-Palmer Similarity Metric for computer vs. laptop: {wp_sim}')

# lesk
sent1 = "He finally got his big break."
sent2 = "There was a break in the action."
sent3 = "There was a break in her voice."
sent4 = "She made a break for the door."
print('\nLesk Algorithm for \'break\':')
print(f'\n{sent1}')
print(lesk(sent1.split(" "), 'break', 'n'))

print(f'\n{sent2}')
print(lesk(sent2.split(" "), 'break', 'n'))

print(f'\n{sent3}')
print(lesk(sent3.split(" "), 'break', 'n'))

print(f'\n{sent4}')
print(lesk(sent4.split(" "), 'break', 'n'))
```

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
[Synset('laptop.n.01')]

Wu-Palmer Similarity Metric for computer vs. laptop: 0.8181818181818182

Lesk Algorithm for 'break':

He finally got his big break.
Synset('rupture.n.02')

There was a break in the action.
Synset('fault.n.04')

There was a break in her voice.
Synset('open_frame.n.01')

She made a break for the door.
Synset('fault.n.04')
```

SentiWordNet

SentiWordNet assigns 3 scores to a synset based on some analysis: positivity, negativity, and objectivity. It can be used for sentiment analysis of any text, and the scores it gives can be especially useful for analyzing customer feedback, marketing campaigns, product acceptance, etc.

The code below demonstrates using SentiWordNet for the word 'passion', which has multiple different synsets of varying scores. The example sentence using the word 'passion' is overwhelmingly positive until the phrase "despite her mother's discouragement". SentiWordNet recognizes that 'despite' and 'discouragement' have high negative scores.

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

# select emotionally charged word
print('Synsets for \'passion\':\n')
passion_senti = swn.senti_synsets('passion')
passion_syn = wn.synsets('passion')
for senti in passion_senti:
  print(f'{senti}')

# output polarity for each word in a sentence
sent = "She cultivated her passion for dancing from a young age, despite her mother's
neg = 0
pos = 0
for word in sent.split():
  syn_list = list(swn.senti_synsets(word))
  if syn_list:
    syn = syn_list[0]
```

```
    neg += syn.neg_score()
    pos += syn.pos_score()
  print(f'word: {word}\tneg: {neg}\tpos: {pos} ')
```

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
Synsets for 'passion':

<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>
word: She       neg: 0  pos: 0
word: cultivated        neg: 0.0        pos: 0.375
word: her       neg: 0.0        pos: 0.375
word: passion   neg: 0.0        pos: 0.875
word: for       neg: 0.0        pos: 0.875
word: dancing   neg: 0.0        pos: 0.875
word: from      neg: 0.0        pos: 0.875
word: a neg: 0.0        pos: 0.875
word: young     neg: 0.125      pos: 0.875
word: age,      neg: 0.125      pos: 0.875
word: despite   neg: 0.75       pos: 0.875
word: her       neg: 0.75       pos: 0.875
word: mother's  neg: 0.75       pos: 0.875
word: discouragement.   neg: 0.75       pos: 0.875
```

Collocations

Collocations are when words are seen together more often than usually expected by chance. You cannot substitute a word with another word and retain the same meaning. For example, "wild rice" is not chaotic rice, and "strong tea" is not tea that frequents the gym.

The code below shows the collocations found in text4, as well as calculations for the mutual information of the words "God bless".

"God bless" has a point-wise mutual information (PMI) = 4.1, which indicates it is likely to be a collocation. A PMI of 0 means x and y are independent, and a negative PMI indicates the words are probably not a collocation.

```
nltk.download('book')
from nltk.book import *
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
```

```
[nltk_data]     |   Package abc is already up-to-date!
[nltk_data]     | Downloading package brown to /root/nltk_data...
[nltk_data]     |   Package brown is already up-to-date!
[nltk_data]     | Downloading package chat80 to /root/nltk_data...
[nltk_data]     |   Package chat80 is already up-to-date!
[nltk_data]     | Downloading package cmudict to /root/nltk_data...
[nltk_data]     |   Package cmudict is already up-to-date!
[nltk_data]     | Downloading package conll2000 to /root/nltk_data...
[nltk_data]     |   Package conll2000 is already up-to-date!
[nltk_data]     | Downloading package conll2002 to /root/nltk_data...
[nltk_data]     |   Package conll2002 is already up-to-date!
[nltk_data]     | Downloading package dependency_treebank to
[nltk_data]     |     /root/nltk_data...
[nltk_data]     |   Package dependency_treebank is already up-to-date!
[nltk_data]     | Downloading package genesis to /root/nltk_data...
[nltk_data]     |   Package genesis is already up-to-date!
[nltk_data]     | Downloading package gutenberg to /root/nltk_data...
[nltk_data]     |   Package gutenberg is already up-to-date!
[nltk_data]     | Downloading package ieer to /root/nltk_data...
[nltk_data]     |   Package ieer is already up-to-date!
[nltk_data]     | Downloading package inaugural to /root/nltk_data...
[nltk_data]     |   Package inaugural is already up-to-date!
[nltk_data]     | Downloading package movie_reviews to
[nltk_data]     |     /root/nltk_data...
[nltk_data]     |   Package movie_reviews is already up-to-date!
[nltk_data]     | Downloading package nps_chat to /root/nltk_data...
[nltk_data]     |   Package nps_chat is already up-to-date!
[nltk_data]     | Downloading package names to /root/nltk_data...
[nltk_data]     |   Package names is already up-to-date!
[nltk_data]     | Downloading package ppattach to /root/nltk_data...
[nltk_data]     |   Package ppattach is already up-to-date!
[nltk_data]     | Downloading package reuters to /root/nltk_data...
[nltk_data]     |   Package reuters is already up-to-date!
[nltk_data]     | Downloading package senseval to /root/nltk_data...
[nltk_data]     |   Package senseval is already up-to-date!
[nltk_data]     | Downloading package state_union to /root/nltk_data...
[nltk_data]     |   Package state_union is already up-to-date!
[nltk_data]     | Downloading package stopwords to /root/nltk_data...
[nltk_data]     |   Package stopwords is already up-to-date!
[nltk_data]     | Downloading package swadesh to /root/nltk_data...
[nltk_data]     |   Package swadesh is already up-to-date!
[nltk_data]     | Downloading package timit to /root/nltk_data...
[nltk_data]     |   Package timit is already up-to-date!
[nltk_data]     | Downloading package treebank to /root/nltk_data...
[nltk_data]     |   Package treebank is already up-to-date!
[nltk_data]     | Downloading package toolbox to /root/nltk_data...
[nltk_data]     |   Package toolbox is already up-to-date!
[nltk_data]     | Downloading package udhr to /root/nltk_data...
[nltk_data]     |   Package udhr is already up-to-date!
[nltk_data]     | Downloading package udhr2 to /root/nltk_data...
[nltk_data]     |   Package udhr2 is already up-to-date!
[nltk_data]     | Downloading package unicode_samples to
[nltk_data]     |     /root/nltk_data...
[nltk_data]     |   Package unicode_samples is already up-to-date!
[nltk_data]     | Downloading package webtext to /root/nltk_data...
```

```
# collocations for text 4: Inaugural Corpus
print(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
None
```

```
# calculate mutual information for 'God bless'
import math

text4_tokens = ' '.join(text4.tokens)
words = len(set(text4))

god_bless = text4_tokens.count('God bless')/words
print(f'p(God bless) = {god_bless}')

god = text4_tokens.count('God')/words
print(f'p(God) = {god}')

bless = text4_tokens.count('bless')/words
print(f'p(bless) = {bless}')

pmi = math.log2(god_bless / (god * bless))
print(f'pmi = {pmi}')
```

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

Colab paid products - Cancel contracts here

✓ 0s completed at 4:43 PM