CS 4395 Homework 3: WordNet

Aditya Rathod (AGR190000)

February 24, 2023

1 WordNet

1.1 Introduction

WordNet is a legical database of English that provides a hierarchy of the English language. WordNet allows us to understand the relationships between words, with relationships ranging from the basics (synonyms, antonyms), to those unique to WordNet's hierarchy of lamguage (hypernyms, hyponyms, meronyms, holonyms).

[1]:
```
from nltk.corpus import wordnet as wn
```

1.2 Usage on nouns

Use the word "guitar" as the noun in question. Print all synsets:

[2]:
```
wn.synsets('guitar')
```

[2]:
```
[Synset('guitar.n.01')]
```

Select `guitar.n.01` as the synset to use. Print definition, examples, lemmas.

[3]:
```
ss = wn.synset('guitar.n.01')

print(f'Synset: {ss}')
print(f'Definition: {ss.definition()}\n')
print(f'Examples: {ss.examples()}\n')
print(f'Lemmas: {ss.lemmas()}')
```

```
Synset: Synset('guitar.n.01')
Definition: a stringed instrument usually having six strings; played by
strumming or plucking

Examples: []

Lemmas: [Lemma('guitar.n.01.guitar')]
```

Traverse up the WordNet hierarchy:

[4]:
```
def traverse_hierarchy(start):
    print(f'Starting from {start} and going up')
```

```
    hyp = start.hypernyms()[0]
    while hyp:
        print(f'\tSynset: {hyp}')
        if hyp.hypernyms():
            hyp = hyp.hypernyms()[0]
        else:
            break
    print('---')
```

[5]:
```
traverse_hierarchy(ss)
```

```
Starting from Synset('guitar.n.01') and going up
        Synset: Synset('stringed_instrument.n.01')
        Synset: Synset('musical_instrument.n.01')
        Synset: Synset('device.n.01')
        Synset: Synset('instrumentality.n.03')
        Synset: Synset('artifact.n.01')
        Synset: Synset('whole.n.02')
        Synset: Synset('object.n.01')
        Synset: Synset('physical_entity.n.01')
        Synset: Synset('entity.n.01')
---
```

[6]:
```
# try examples that aren't in the instructions
traverse_hierarchy(wn.synset('ghost.n.01'))
traverse_hierarchy(wn.synset('computer.n.01'))
traverse_hierarchy(wn.synset('idea.n.01'))
traverse_hierarchy(wn.synset('sandwich.n.01'))
traverse_hierarchy(wn.synset('jersey.n.03'))
```

```
Starting from Synset('ghost.n.01') and going up
        Synset: Synset('apparition.n.03')
        Synset: Synset('illusion.n.01')
        Synset: Synset('appearance.n.04')
        Synset: Synset('representation.n.01')
        Synset: Synset('content.n.05')
        Synset: Synset('cognition.n.01')
        Synset: Synset('psychological_feature.n.01')
        Synset: Synset('abstraction.n.06')
        Synset: Synset('entity.n.01')
---
Starting from Synset('computer.n.01') and going up
        Synset: Synset('machine.n.01')
        Synset: Synset('device.n.01')
        Synset: Synset('instrumentality.n.03')
        Synset: Synset('artifact.n.01')
        Synset: Synset('whole.n.02')
        Synset: Synset('object.n.01')
```

```
        Synset: Synset('physical_entity.n.01')
        Synset: Synset('entity.n.01')
---
Starting from Synset('idea.n.01') and going up
        Synset: Synset('content.n.05')
        Synset: Synset('cognition.n.01')
        Synset: Synset('psychological_feature.n.01')
        Synset: Synset('abstraction.n.06')
        Synset: Synset('entity.n.01')
---
Starting from Synset('sandwich.n.01') and going up
        Synset: Synset('snack_food.n.01')
        Synset: Synset('dish.n.02')
        Synset: Synset('nutriment.n.01')
        Synset: Synset('food.n.01')
        Synset: Synset('substance.n.07')
        Synset: Synset('matter.n.03')
        Synset: Synset('physical_entity.n.01')
        Synset: Synset('entity.n.01')
---
Starting from Synset('jersey.n.03') and going up
        Synset: Synset('shirt.n.01')
        Synset: Synset('garment.n.01')
        Synset: Synset('clothing.n.01')
        Synset: Synset('consumer_goods.n.01')
        Synset: Synset('commodity.n.01')
        Synset: Synset('artifact.n.01')
        Synset: Synset('whole.n.02')
        Synset: Synset('object.n.01')
        Synset: Synset('physical_entity.n.01')
        Synset: Synset('entity.n.01')
---
```

WordNet is organized via synonyms, called *synsets*. Antonyms are defined, if they exist. In terms of hierarchy, we have hypernyms and hyponyms, where holonyms are a lower version of the word and hypernyms is a "higher" version of the word.

Based on these five traversals, we can see that all nouns "inherit" from `entity.n.01`. (By this, I mean that all nouns have a hypernym of "entity"). Depending on what the noun is, it either is a physical entity or an abstraction, and then splits off into the proper subcategory. Physical things seem to be classified as physical entities, and then are broken down as matter or objects, and then are drilled down into their respective categories. Abstractions are usually broken down into psychological features and focus on other abstract concepts to break these abstractions down.

Print hypernyms, hyponyms, meronyms, holonyms, antonyms:

[7]:
```
print(f'Synset: {ss}')

# call method if exists on object else return empty list
```

```
# cfe = call function if exists!
cfe = lambda obj, attr: getattr(obj, attr)() if hasattr(obj, attr) else []

print(f'\tHypernyms: {cfe(ss, "hypernyms")}')
print(f'\tHyponyms: {cfe(ss, "hyponyms")}')
print(f'\tMeronyms: {cfe(ss, "part_meronyms")}')
print(f'\tHolonyms: {cfe(ss, "member_holonyms")}')
print(f'\tAntonyms: {cfe(ss, "antonyms")}')
```

```
Synset: Synset('guitar.n.01')
        Hypernyms: [Synset('stringed_instrument.n.01')]
        Hyponyms: [Synset('acoustic_guitar.n.01'), Synset('bass_guitar.n.01'),
Synset('cittern.n.01'), Synset('electric_guitar.n.01'),
Synset('hawaiian_guitar.n.01'), Synset('uke.n.01')]
        Meronyms: [Synset('fingerboard.n.03')]
        Holonyms: []
        Antonyms: []
```

1.3 Usage on verbs

Use the word "gagged" as the verb in question. Print all synsets:

[8]:
```
verb = 'gagged'
wn.synsets(verb)
```

[8]:
```
[Synset('gag.v.01'),
 Synset('choke.v.02'),
 Synset('gag.v.03'),
 Synset('gag.v.04'),
 Synset('gag.v.05'),
 Synset('gag.v.06'),
 Synset('gag.v.07')]
```

Select `gag.v.03` as the synset to use. Print definition, examples, lemmas.

[9]:
```
st = wn.synset('gag.v.03')

print(f'Synset: {st}\n-----\n')
print(f'Definition: {st.definition()}\n')
print(f'Examples: {st.examples()}\n')
print(f'Lemmas: {st.lemmas()}')
```

```
Synset: Synset('gag.v.03')
-----

Definition: tie a gag around someone's mouth in order to silence them

Examples: ['The burglars gagged the home owner and tied him to a chair']
```

```
Lemmas: [Lemma('gag.v.03.gag'), Lemma('gag.v.03.muzzle')]
```

Traverse up the WordNet hierarchy:

[10]:
```
traverse_hierarchy(st)
```

```
Starting from Synset('gag.v.03') and going up
        Synset: Synset('tie.v.01')
        Synset: Synset('fasten.v.01')
        Synset: Synset('attach.v.01')
        Synset: Synset('connect.v.01')
---
```

[11]:
```
# try examples that aren't in the instructions
traverse_hierarchy(wn.synset('run.v.01'))
traverse_hierarchy(wn.synset('question.v.01'))
traverse_hierarchy(wn.synset('ideate.v.01'))
traverse_hierarchy(wn.synset('punch.v.01'))
traverse_hierarchy(wn.synset('think.v.01'))
```

```
Starting from Synset('run.v.01') and going up
        Synset: Synset('travel_rapidly.v.01')
        Synset: Synset('travel.v.01')
---
Starting from Synset('question.v.01') and going up
        Synset: Synset('challenge.v.02')
        Synset: Synset('invite.v.04')
        Synset: Synset('request.v.02')
        Synset: Synset('ask.v.02')
        Synset: Synset('request.v.01')
        Synset: Synset('communicate.v.01')
        Synset: Synset('convey.v.03')
        Synset: Synset('transfer.v.02')
        Synset: Synset('move.v.02')
---
Starting from Synset('imagine.v.01') and going up
        Synset: Synset('create_by_mental_act.v.01')
        Synset: Synset('make.v.03')
---
Starting from Synset('punch.v.01') and going up
        Synset: Synset('hit.v.03')
        Synset: Synset('touch.v.01')
---
Starting from Synset('think.v.01') and going up
        Synset: Synset('evaluate.v.02')
        Synset: Synset('think.v.03')
---
```

Verbs use a similar organizations as nouns. But, based on these five traversals, we can see that not all verbs inherit from a common ancestor (hypernym), like nouns do. As we move up the hierarchy,

it gets reduced down into multiple different generalized nouns. We can also see that some synsets don't correlate with a specific word, but a phrase. For instance, `create_by_mental_act.v.01` isn't a word, but describes a specific category that can't be expressed via a singular word.

Find all forms of the word:

[12]:
```
all_forms = set([])
# ml = morphize lemma
ml = lambda lemma: wn.morphy(lemma.name())

for synset in wn.synsets(verb):
    all_forms.update([ml(l) for l in synset.lemmas()])

print(f'All forms of word "{verb}": {all_forms}')
```

```
All forms of word "gagged": {'gag', 'muzzle', 'heave', 'fret', 'choke', 'quip',
'strangle', 'suffocate', 'retch'}
```

Select similar words and run Wu-Palmer and Lesk on them:

[13]:
```
from nltk.wsd import lesk

wordA = wn.synsets("sucked")
wordB = wn.synsets("drank")

print(f'Word A synsets: {wordA}')
print(f'Word B synsets: {wordB}\n\n')

# pick closest synset
sA = wn.synset('suck.v.01')
sB = wn.synset('drink.v.01')

# lesk stuff
sentA = 'She sucked on the straw but the milkshake was frozen'.split(' ')
leskA = lesk(sentA, 'sucked', 'v')
sentB = 'Sarah drank her smoothie'.split(' ')
leskB = lesk(sentB, 'drank', 'v')

print(f'Wu-Palmer similarity: {sA.wup_similarity(sB)}')
print(f'Lesk for sentence {sentA}: {leskA}')
print(f'Lesk for sentence {sentB}: {leskB}')
```

```
Word A synsets: [Synset('suck.v.01'), Synset('suck.v.02'), Synset('suck.v.03'),
Synset('suck.v.04'), Synset('fellate.v.01'), Synset('absorb.v.04'),
Synset('breastfeed.v.01')]
Word B synsets: [Synset('drink.v.01'), Synset('drink.v.02'),
Synset('toast.v.02'), Synset('drink_in.v.01'), Synset('drink.v.05')]


Wu-Palmer similarity: 0.8
```

```
Lesk for sentence ['She', 'sucked', 'on', 'the', 'straw', 'but', 'the',
'milkshake', 'was', 'frozen']: Synset('suck.v.01')
Lesk for sentence ['Sarah', 'drank', 'her', 'smoothie']: Synset('toast.v.02')
```

The Wu-Palmer similarity algorithm uses synsets to determine how similar two words are. In this case, we used the words "sucked" and "drank", both of which discuss intake of a liquid. Because they share a high degree of hypernyms, this metric tells us that these words are 80% similar.

On the other hand, the Lesk algorithm focuses on word sense disambiguation based on the dictionary glosses to see how many words overlap. This algorithm worked well for the word "sucked," with it picking up that `suck.v.01` is the sense of the word. However, this algorithm didn't work as well for the word "drank," with it picking up `toast.v.02` (which refers to making a toast in a drink setting) as the sense of the word. This demonstrates the imperfect nature of the algorithm.

2 SentiWordNet

SentiWordNet builds on top of WordNet to allow for sentiment to be gleaned from text. Each synset in the database is assigned a positive, negative, and objectivity score. The sum of these three add up into 1.

You could use SentiWordNet to calculate the polarity of a sentence or a larger text, or to track the shift in polarity of a text as it progresses.

[14]:
```
from nltk.corpus import sentiwordnet as swn

def print_scores(senti_synset):
    pos_s, neg_s, obj_s = senti_synset.pos_score(), senti_synset.neg_score(),␣
 ↪senti_synset.obj_score()
    print(f'Scores for word {senti_synset.synset.name()} -', end=" ")
    print(f'Positive: {pos_s}', end=" ")
    print(f'Negative: {neg_s}', end=" ")
    print(f'Objective: {obj_s}')

# emotionally charged word
em_word = 'fucker'
em_synsets = wn.synsets(em_word)
# selected synset
em_sel_synset = em_synsets[1]
print(f'The word is "{em_word}"')
# scoring
scores = swn.senti_synsets('fucker', 'n')
for score in scores:
    print_scores(score)
```

```
The word is "fucker"
Scores for word fucker.n.01 - Positive: 0.125 Negative: 0.0 Objective: 0.875
Scores for word fucker.n.02 - Positive: 0.0 Negative: 0.25 Objective: 0.75
```

[15]:
```
def print_polarity(word):
    print(f'Scoring word "{word}"\n')
    scores = list(swn.senti_synsets(word))
    if scores:
        for score in scores:
            print_scores(score)
    if not scores or len(scores) == 0:
        print('No score available')
    print('---\n')


sentence = 'League of Legends is a terrible and incomplete game'.split(' ')
for word in sentence:
    print_polarity(word)
```

```
Scoring word "League"

Scores for word league.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word league.n.02 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word league.n.03 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word league.v.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
---

Scoring word "of"

No score available
---

Scoring word "Legends"

Scores for word legend.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word caption.n.03 - Positive: 0.0 Negative: 0.0 Objective: 1.0
---

Scoring word "is"

Scores for word be.v.01 - Positive: 0.25 Negative: 0.125 Objective: 0.625
Scores for word be.v.02 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.03 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word exist.v.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.05 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word equal.v.01 - Positive: 0.125 Negative: 0.125 Objective: 0.75
Scores for word constitute.v.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.08 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word embody.v.02 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.10 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.11 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word be.v.12 - Positive: 0.0 Negative: 0.0 Objective: 1.0
```

```
Scores for word cost.v.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
---

Scoring word "a"

Scores for word angstrom.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word vitamin_a.n.01 - Positive: 0.125 Negative: 0.25 Objective: 0.625
Scores for word deoxyadenosine_monophosphate.n.01 - Positive: 0.0 Negative: 0.0
Objective: 1.0
Scores for word adenine.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word ampere.n.02 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word a.n.06 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word a.n.07 - Positive: 0.0 Negative: 0.0 Objective: 1.0
---

Scoring word "terrible"

Scores for word awful.s.02 - Positive: 0.0 Negative: 0.625 Objective: 0.375
Scores for word atrocious.s.02 - Positive: 0.0 Negative: 0.875 Objective: 0.125
Scores for word severe.s.01 - Positive: 0.0 Negative: 0.875 Objective: 0.125
Scores for word frightful.s.02 - Positive: 0.125 Negative: 0.25 Objective: 0.625
---

Scoring word "and"

No score available
---

Scoring word "incomplete"

Scores for word incomplete.a.01 - Positive: 0.5 Negative: 0.25 Objective: 0.25
Scores for word incomplete.s.02 - Positive: 0.0 Negative: 0.125 Objective: 0.875
---

Scoring word "game"

Scores for word game.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.02 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.03 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.04 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.05 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.06 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.07 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word plot.n.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.09 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.10 - Positive: 0.0 Negative: 0.0 Objective: 1.0
Scores for word game.n.11 - Positive: 0.25 Negative: 0.5 Objective: 0.25
Scores for word bet_on.v.01 - Positive: 0.0 Negative: 0.0 Objective: 1.0
```

```
Scores for word crippled.s.01 - Positive: 0.0 Negative: 0.5 Objective: 0.5
Scores for word game.s.02 - Positive: 0.125 Negative: 0.375 Objective: 0.5
---
```

In the sentence example, we can see how words that are related but aren't even the exact word appear in the senti-synsets, likely because they were identified as synsets by WordNet. We can also see that even in emotionally charged words such as "awful," the algorithm balances the negative sentiment with the potential objectivity of the term (something can be objectively terrible). Interestingly, in the word example, it classified the word "fucker" as mostly objective rather than negative, even though that word is usually not used in an objective context.

SentiWordNet can be used in NLP applications to classify any sort of text, which in turn could be used to automate tasks like sentiment analysis or opinion mining, which are commonly used in social media monitoring, market research, and customer feedback analysis.

Even beyond that, the sentiment of a text could potentially be used to crudely identify the topic of the text. For instance, a news article that contains mostly negative sentiment words might be classified as a story about a natural disaster or a crime.

3 Collocations

Collocations refer to a group of words that take on a new meaning, and cannot be broken into their constituent parts when analyzing a piece of text. For instance, the colloquial college term "jungle juice" does not refer to juice from the jungle. Collocations can be found by calculating the chances of two words being together versus their probability appearing in general. More specifically, this calculation is the base-2 log of the probability of finding word A and B together, divided by the quantity of probability of finding A times probability of finding B. This is called a PMI, or point-wise mutual information.

Print collocations for the inaugural address corpus:

[16]:
```
from nltk.book import *
text4.collocations()
```

```
*** 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
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
```

Calculate mutual information for `Indian tribes`:

[17]:
```
import math

def pmi_calc(words, text):
    vocab = len(set(all_text))
    split = words.split(' ')
    x, y = split[0], split[1]
    xy = words
    if len(split) != 2:
        raise ValueError('Provided too many words for a collocation')
    p_xy = text.count(xy) / vocab
    p_x = text.count(x) / vocab
    p_y = text.count(y) / vocab
    pmi = math.log2(p_xy / (p_x * p_y))
    print(f'x = "{x}", y = "{y}", p(x) = {p_x}, p(y) = {p_y}, p(xy) = {p_xy}')
    return pmi



all_text = ' '.join(text4.tokens)
coll = 'Indian tribes'
print(f'pmi calculation for "{coll}" = {pmi_calc(coll, all_text)}\n')
```

```
x = "Indian", y = "tribes", p(x) = 0.13095238095238096, p(y) =
0.07142857142857142, p(xy) = 0.07142857142857142
pmi calculation for "Indian tribes" = 2.932885804141463
```

Looking at the individual values for probability: - “Indian” is about 13% likely to appear in the text - “tribes” is about 7% likely to appear in the text - Together, they are about 7% likely to appear in the text.

This means that the word “tribes” is most likely accompanied by “Indian,” while there still existing cases where “Indian” will appear alone.

Computing PMI for this phrase gave us a positive value. This indicates that the co-occurrence of these two words is more frequent than would be expected by chance, providing strong evidence that they form a collocation within this corpus.

We can then conclude that these words have a strong chance of appearing together within this corpus. As such, this gives us a method for narrowing down and analyzing potential strong collocations that we can explore further, which can only get better with larger corpus sizes and text variety.