```
"""
Soham Mukherjee
Professor Mazidi
CS 4395
2/18/2023
"""
```

```
'\nSoham Mukherjee\nProfessor Mazidi\nCS 4395\n2/18/2023\n'
```

Homework 3

Soham Mukherjee

Professor Mazidi

CS 4395

2/18/2023

What is WordNet?

WordNet is a lexical database that catalogues the relations between words as well as their alternatives. It can be thought of as a thesaurus, as it is like a dictionary that also offers synonyms for similar-context words. Semantically equivalent words are often bundled into the same groups known as synsets, which are considered referentially similar for all intents and purposes. An example would be "home" and "house" as they can be used interchangeably, and thus fall under the same synset in theory.

```
# Select a noun. Output all synsets.
import nltk
from nltk.corpus import wordnet as wn
noun = 'car'
wn.synsets(noun)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[Synset('car.n.01'),
 Synset('car.n.02'),
 Synset('car.n.03'),
 Synset('car.n.04'),
 Synset('cable_car.n.01')]
```

+ Code + Text

```
"""
Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas.
From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the
synsets as you go. Write a couple of sentences observing the way that WordNet is organized for
nouns.
"""
selected_synset = wn.synset('car.n.01')

definition = selected_synset.definition()
examples = selected_synset.examples()
lemmas = selected_synset.lemma_names()

print("Definition: ", definition)
print("Examples:", examples)
print("Lemmas: ", lemmas)

print("Traversing up the WordNet Hierarchy...")
while True:
    print(selected_synset.name())
    hypernyms = selected_synset.hypernyms()
    if hypernyms:
        selected_synset = hypernyms[0]
    else:
        break
```

```
Definition:  a motor vehicle with four wheels; usually propelled by an internal combustion engine
Examples: ['he needs a car to get to work']
Lemmas:  ['car', 'auto', 'automobile', 'machine', 'motorcar']
Traversing up the WordNet Hierarchy...
```

```
car.n.01
motor_vehicle.n.01
self-propelled_vehicle.n.01
wheeled_vehicle.n.01
container.n.01
instrumentality.n.03
artifact.n.01
whole.n.02
object.n.01
physical_entity.n.01
entity.n.01
```

Write a couple of sentences observing the way that WordNet is organized for nouns.

At the top of the hierarchy, WordNet stores the most general categories. The lower the hierarchy, the more specific the given noun. The noun hierarchy is organized according to the "ISA" relationship, where from bottom up, the lower noun ISA instance of the upper noun. For instance, above we can see that *motor_vehicle* is a hypernym of *car*, indicating that *car* is a type of *motor_vehicle*.

```
# Output the following (or an empty list if none exist): hypernyms, hyponyms, meronyms,
# holonyms, antonym.

selected_synset = wn.synset('car.n.01')

# hypernyms
print("Hypernyms:")
print(selected_synset.hypernyms())

# hyponyms
print("Hyponyms:")
print(selected_synset.hyponyms())

# meronyms
print("Meronyms:")
print(selected_synset.part_meronyms() + selected_synset.member_meronyms())

# holonyms
print("Holonyms:")
print(selected_synset.part_holonyms() + selected_synset.member_holonyms())

# antonyms
print("Antonyms:")
antonyms = []
for lemma in selected_synset.lemmas():
    for antonym in lemma.antonyms():
        antonyms.append(antonym)
print(antonyms)
```

```
Hypernyms:
[Synset('motor_vehicle.n.01')]
Hyponyms:
[Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bus.n.04'), Synset('cab.n.03'), Synset('compact.n.03'), Synset('convertib
Meronyms:
[Synset('accelerator.n.01'), Synset('air_bag.n.01'), Synset('auto_accessory.n.01'), Synset('automobile_engine.n.01'), Synset('automobile
Holonyms:
[]
Antonyms:
[]
```

```
# Select a verb. Output all synsets.
verb = "drive"
wn.synsets(verb)
```

```
[Synset('drive.n.01'),
 Synset('drive.n.02'),
 Synset('campaign.n.02'),
 Synset('driveway.n.01'),
 Synset('drive.n.05'),
 Synset('drive.n.06'),
 Synset('drive.n.07'),
 Synset('drive.n.08'),
 Synset('drive.n.09'),
 Synset('drive.n.10'),
 Synset('drive.n.11'),
 Synset('drive.n.12'),
 Synset('drive.v.01'),
```

```
 Synset('drive.v.02'),
 Synset('drive.v.03'),
 Synset('force.v.06'),
 Synset('drive.v.05'),
 Synset('repel.v.01'),
 Synset('drive.v.07'),
 Synset('drive.v.08'),
 Synset('drive.v.09'),
 Synset('tug.v.02'),
 Synset('drive.v.11'),
 Synset('drive.v.12'),
 Synset('drive.v.13'),
 Synset('drive.v.14'),
 Synset('drive.v.15'),
 Synset('drive.v.16'),
 Synset('drive.v.17'),
 Synset('drive.v.18'),
 Synset('drive.v.19'),
 Synset('drive.v.20'),
 Synset('drive.v.21'),
 Synset('drive.v.22')]
```

```
"""
Select one synset from the list of synsets. Extract its definition, usage examples, and lemmas.
From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the
synsets as you go. Write a couple of sentences observing the way that WordNet is organized for
verbs.
"""
selected_synset = wn.synset('drive.v.02')

definition = selected_synset.definition()
examples = selected_synset.examples()
lemmas = selected_synset.lemma_names()

print("Definition: ", definition)
print("Examples:", examples)
print("Lemmas: ", lemmas)

print("Traversing up the WordNet Hierarchy...")
while True:
    print(selected_synset.name())
    hypernyms = selected_synset.hypernyms()
    if hypernyms:
        selected_synset = hypernyms[0]
    else:
        break
```

```
Definition:  travel or be transported in a vehicle
Examples: ['We drove to the university every morning', 'They motored to London for the theater']
Lemmas:  ['drive', 'motor']
Traversing up the WordNet Hierarchy...
drive.v.02
travel.v.01
```

Write a couple of sentences observing the way that WordNet is organized for verbs.

At the top of the hierarchy, WordNet stores the most general categories. The lower the hierarchy, the more specific the given verb. The verb hierarchy is also organized according to the "ISA" relationship, where from bottom up, the lower verb ISA instance of the upper verb. For instance, above we can see that *travel* is a hypernym of *drive*, indicating that *drive* is a type of *travel*.

```
# Use morphy to find as many different forms of the word as you can.
word = 'drive'
synsets = wn.synsets(word)

# get all the inflected forms for each synset.
inflected_forms = set()
for synset in synsets:
    for lemma in synset.lemmas():
        inflected_forms.add(lemma.name())

# print the set
print(inflected_forms)
```

```
{'driving_force', 'force', 'private_road', 'crusade', 'effort', 'tug', 'movement', 'parkway', 'campaign', 'thrust', 'ram', 'force_back',
```

```
# Select two words that you think might be similar. Find the specific synsets you are interested in.
# Run the Wu-Palmer similarity metric and the Lesk algorithm. Write a couple of sentences with
# your observations

# Wu-Palmer Algorithm
word_1 = "freezer"
word_2 = "fridge"

w1_synsets = wn.synsets(word_1)
w2_synsets = wn.synsets(word_2)

similarity = w1_synsets[0].wup_similarity(w2_synsets[0])
print(f"Similarity of {word_1} and {word_2}: {similarity}")

# Lesk algorithm
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize

sentence_1 = ["I", "like", "to", "put", "icecream", "in", "the", "freezer"]
sentence_2 = ["Why", "does", "the", "fridge", "have", "my", "keys?"]
sense_1 = lesk(sentence_1, word_1, 'n')
sense_2 = lesk(sentence_2, word_2, 'n')

print(sense_1)
print(sense_2)

def_1 = sense_1.definition()
def_2 = sense_2.definition()

print(def_1)
print(def_2)

overlap = set(def_1.split()).intersection(set(def_2.split()))
print(f"Overlap between {word_1} and {word_2}: {overlap}")
```

```
Similarity of freezer and fridge: 0.9629629629629629
Synset('deep-freeze.n.01')
Synset('electric_refrigerator.n.01')
electric refrigerator (trade name Deepfreeze) in which food is frozen and stored for long periods of time
a refrigerator in which the coolant is pumped around by an electric motor
Overlap between freezer and fridge: {'refrigerator', 'in', 'is', 'electric', 'which'}
```

Write a couple of sentences with your observations.

There is a significant amount of overlap between freezer and fridge, as Wu-Palmer yields a high similarity of ~0.9629.

Using the Lesk Algorithm returns most notably 'refrigerator' and 'electric', both terms that refer to the fridge and freezer which are both electric cooling machines.

With the above results, we can see that both a fridge and a freezer are very similar words - not only in spelling but also in word origin.

Write a couple of sentences about SentiWordNet, describing its functionality and possible use cases.

SentiWordNet judges words based on their sentiment. It checks for strong opinions, primarily if a given data is positive, negative, or neutral when it comes to discerning the author's point of view. While WordNet gives purely binary scores (+ or -) for every word, SentiWordNet is continuous, giving positive and negative values in the range [0, 1]. SentiWordNet would probably be best for dissecting opinion articles.

```
# Select an emotionally charged word. Find its senti-synsets and output the polarity scores
# for each word. Make up a sentence. Output the polarity for each word in the sentence. Write a
# couple of sentences about your observations of the scores and the utility of knowing these
# scores in an NLP application.
from nltk.corpus import sentiwordnet as swn

strong_word = "awful"

print("Senti-synsets: ")
strong_synsets=list(swn.senti_synsets(strong_word))
print(strong_synsets)
print()
```

```
13
14 print("Polarity Scores: ")
15 for syn in strong_synsets:
16   print(syn)
17   print("Positive score = ", syn.pos_score())
18   print("Negative score = ", syn.neg_score())
19   print("Objective score = ", syn.obj_score())
20 print()
21
22 print("Polarity of words in a sentence: ")
23 sent = "I never want to see you here, there, or anywhere near my house again."
24 print(sent)
25 neg = 0
26 pos = 0
27 tokens = sent.split()
28 for token in tokens:
29     syn_list = list(swn.senti_synsets(token))
30     if syn_list:
31         syn = syn_list[0]
32         neg += syn.neg_score()
33         pos += syn.pos_score()
34
35 print("neg\tpos counts")
36 print(neg, '\t', pos)
```

```
Senti-synsets:
[SentiSynset('atrocious.s.02'), SentiSynset('awful.s.02'), SentiSynset('nasty.a.01'), SentiSynset('awed.s.01'), SentiSynset('frightful.s

Polarity Scores:
<atrocious.s.02: PosScore=0.0 NegScore=0.875>
Positive score =  0.0
Negative score =  0.875
Objective score =  0.125
<awful.s.02: PosScore=0.0 NegScore=0.625>
Positive score =  0.0
Negative score =  0.625
Objective score =  0.375
<nasty.a.01: PosScore=0.0 NegScore=0.875>
Positive score =  0.0
Negative score =  0.875
Objective score =  0.125
<awed.s.01: PosScore=0.5 NegScore=0.5>
Positive score =  0.5
Negative score =  0.5
Objective score =  0.0
<frightful.s.02: PosScore=0.125 NegScore=0.25>
Positive score =  0.125
Negative score =  0.25
Objective score =  0.625
<amazing.s.02: PosScore=0.875 NegScore=0.125>
Positive score =  0.875
Negative score =  0.125
Objective score =  0.0
<terribly.r.01: PosScore=0.25 NegScore=0.0>
Positive score =  0.25
Negative score =  0.0
Objective score =  0.75

Polarity of words in a sentence:
I never want to see you here, there, or anywhere near my house again.
neg     pos counts
0.875    0.0
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
```

Write a couple of sentences about your observations of the scores and the utility of knowing these scores in an NLP application

I notice that these scores have three types of scores - positive, negative, and objective scores. The lower the objective score, the more negative the word. Objective scores are determined by subtracting the negative score from 1.

These scores would be useful for an NLP application because it could help researchers and people alike classify sentiment from a given text to help readers understand what type of article they're reading. Additionally, it could check for strong positive or negative bias in news media, and see if there's a way to maintain neutrality of the given information.

Write a couple of sentences about what a collocation is.

Collocations are two or more words that tend to appear together frequently; an example would be "social media". When combined, these words form a meaning that is normally different from the separate words. For instance, "social media" refers to the public posting of a creator's content on the internet.

```
# Write a couple of sentences about what a collocation is. Output collocations for text4, the
# Inaugural corpus. Select one of the collocations identified by NLTK. Calculate mutual
# information. Write commentary on the results of the mutual information formula and your
# interpretation.
from nltk.book import text4

# output collocations for text4
print(text4.collocations())

# Select one of the collocations identified by NLTK. Calculate mutual information.
text = ' '.join(text4.tokens)
print(text[:50])

import math
vocab = len(set(text4))
ow = text.count('Old World')/vocab
print("p(Old World) = ",ow )
o = text.count('Old')/vocab
print("p(Old) = ", o)
w = text.count('World')/vocab
print('p(World) = ', w)
pmi = math.log2(ow / (o * w))
print('pmi = ', pmi)
```

```
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
p(Old World) =  0.000997506234413965
p(Old) =  0.0010972568578553616
p(World) =  0.0017955112219451373
pmi =  8.983886091037398
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
[nltk_data] Downloading package webtext to /root/nltk_data...
[nltk_data]   Package webtext 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 stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Write commentary on the results of the mutual information formula and your interpretation.

The higher the Pointwise Mutual Information (PMI), the more likely the word is to be a collocation. It makes sense that the probability of the words separately is more than the probability of the words together as they can be treated as independent objects. My interpretation is that because "old" and "world" frequently appears as "old world" and that the probability is very close to the separate probabilities of the two words, this means that it's highly likely to be a collocation, hence the high PMI.