# Assignment 3: WordNet

Name: Aarya Patil

Date: 02/21/23

---

## 1. What is WordNet?

WordNet is a large lexical database of words in the English language that are sorted by concept and meaning. The concept part of WordNet groups words into cognitive synonyms called "synsets". The meaning part of WordNet is similar to a thesaurus.

## Using WordNet

### 2. Get all synsets of a noun. In this example the noun is 'car'.

In [346…
```
from nltk.corpus import wordnet as wn # import wordnet
wn.synsets('car')
```

Out[346]:
```
[Synset('car.n.01'),
 Synset('car.n.02'),
 Synset('car.n.03'),
 Synset('car.n.04'),
 Synset('cable_car.n.01')]
```

### 3a. Select one synset from the list of synsets and extract it's information.

In [347…
```
synEx = wn.synset('car.n.01') # picks the first synset

print("Definition:", synEx.definition()) # extract the definition
print("Usage examples:", synEx.examples()) # extract the usage examples
print("Lemmas:", synEx.lemmas())# extract the lemmas
```

```
Definition: a motor vehicle with four wheels; usually propelled by an inter
nal combustion engine
Usage examples: ['he needs a car to get to work']
Lemmas: [Lemma('car.n.01.car'), Lemma('car.n.01.auto'), Lemma('car.n.01.aut
omobile'), Lemma('car.n.01.machine'), Lemma('car.n.01.motorcar')]
```

### 3b. Traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

In [348…

```
# Iterate over synsets
exercise_synsets = wn.synsets('car', pos = wn.NOUN)
for sense in exercise_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lem
```

```
Synset: car.n.01(a motor vehicle with four wheels; usually propelled by an
internal combustion engine)
         Lemmas:['car', 'auto', 'automobile', 'machine', 'motorcar']
Synset: car.n.02(a wheeled vehicle adapted to the rails of railroad)
         Lemmas:['car', 'railcar', 'railway_car', 'railroad_car']
Synset: car.n.03(the compartment that is suspended from an airship and that
carries personnel and the cargo and the power plant)
         Lemmas:['car', 'gondola']
Synset: car.n.04(where passengers ride up and down)
         Lemmas:['car', 'elevator_car']
Synset: cable_car.n.01(a conveyance for passengers or freight on a cable ra
ilway)
         Lemmas:['cable_car', 'car']
```

### 3c. How does WordNet organize nouns?

WordNet organizes nouns into hierarchies based on the hyponym-hypernym relations between synsets. A hypernym is essentially the broader grouping of a word. For example, feline is a hypernym of cat. A hyponym is the opposite. It is a specific subgroup of a a word. For example, cat is a hyponym of feline. There can also be other relationships between synsets such as meronym (part of), holonym (whole), and troponym (defines a more specific action).

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

In [349…
```
car = wn.synset('car.n.01') # restore first synset into separate variable

print("Hypernyms:", car.hypernyms()) # print out the hypernyms
print("\nHyponyms:", car.hyponyms()) # print out the hyponyms
print("\nMeronyms:", car.part_meronyms()) # print out the meronyms
print("\nHolonyms:", car.part_holonyms()) # print out the holonyms

# Print out the antonyms
antonyms_list = [] # variable to store the antonyms
for lemma in car.lemmas(): # find all antonyms for each lemma
    if lemma.antonyms():
        antonyms_list.append(lemma.antonyms()) # add to list of antonyms
print("\nAntonyms:", antonyms_list) # print the antonyms
```

```
Hypernyms: [Synset('motor_vehicle.n.01')]

Hyponyms: [Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bu
s.n.04'), Synset('cab.n.03'), Synset('compact.n.03'), Synset('convertible.n
.01'), Synset('coupe.n.01'), Synset('cruiser.n.01'), Synset('electric.n.01'
), Synset('gas_guzzler.n.01'), Synset('hardtop.n.01'), Synset('hatchback.n.
01'), Synset('horseless_carriage.n.01'), Synset('hot_rod.n.01'), Synset('je
ep.n.01'), Synset('limousine.n.01'), Synset('loaner.n.02'), Synset('minicar
.n.01'), Synset('minivan.n.01'), Synset('model_t.n.01'), Synset('pace_car.n
.01'), Synset('racer.n.02'), Synset('roadster.n.01'), Synset('sedan.n.01'),
Synset('sport_utility.n.01'), Synset('sports_car.n.01'), Synset('stanley_st
eamer.n.01'), Synset('stock_car.n.01'), Synset('subcompact.n.01'), Synset('
touring_car.n.01'), Synset('used-car.n.01')]

Meronyms: [Synset('accelerator.n.01'), Synset('air_bag.n.01'), Synset('auto
_accessory.n.01'), Synset('automobile_engine.n.01'), Synset('automobile_hor
n.n.01'), Synset('buffer.n.06'), Synset('bumper.n.02'), Synset('car_door.n.
01'), Synset('car_mirror.n.01'), Synset('car_seat.n.01'), Synset('car_windo
w.n.01'), Synset('fender.n.01'), Synset('first_gear.n.01'), Synset('floorbo
ard.n.02'), Synset('gasoline_engine.n.01'), Synset('glove_compartment.n.01'
), Synset('grille.n.02'), Synset('high_gear.n.01'), Synset('hood.n.09'), Sy
nset('luggage_compartment.n.01'), Synset('rear_window.n.01'), Synset('rever
se.n.02'), Synset('roof.n.02'), Synset('running_board.n.01'), Synset('stabi
lizer_bar.n.01'), Synset('sunroof.n.01'), Synset('tail_fin.n.02'), Synset('
third_gear.n.01'), Synset('window.n.02')]

Holonyms: []

Antonyms: []
```

## 5. Get all synsets of a verb. In this example the verb is 'jump'.

In [350…
```
wn.synsets('jump')
```

Out[350]:
```
[Synset('jump.n.01'),
 Synset('leap.n.02'),
 Synset('jump.n.03'),
 Synset('startle.n.01'),
 Synset('jump.n.05'),
 Synset('jump.n.06'),
 Synset('jump.v.01'),
 Synset('startle.v.02'),
 Synset('jump.v.03'),
 Synset('jump.v.04'),
 Synset('leap_out.v.01'),
 Synset('jump.v.06'),
 Synset('rise.v.11'),
 Synset('jump.v.08'),
 Synset('derail.v.02'),
 Synset('chute.v.01'),
 Synset('jump.v.11'),
 Synset('jumpstart.v.01'),
 Synset('jump.v.13'),
 Synset('leap.v.02'),
 Synset('alternate.v.01')]
```

## 6a. Select one synset from the list of synsets and extract it's information.

In [351…
```
synEx = wn.synset('jump.v.01') # picks the first verb synset

print("Definition:", synEx.definition()) # extract the definition
print("Usage examples:", synEx.examples()) # extract the usage examples
print("Lemmas:", synEx.lemmas())# extract the lemmas
```

```
Definition: move forward by leaps and bounds
Usage examples: ['The horse bounded across the meadow', 'The child leapt ac
ross the puddle', 'Can you jump over the fence?']
Lemmas: [Lemma('jump.v.01.jump'), Lemma('jump.v.01.leap'), Lemma('jump.v.01
.bound'), Lemma('jump.v.01.spring')]
```

## 6b. Traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go.

In [352…
```
# Iterate over synsets
exercise_synsets = wn.synsets('jump', pos = wn.VERB)
for sense in exercise_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lem
```

```
Synset: jump.v.01(move forward by leaps and bounds)
         Lemmas:['jump', 'leap', 'bound', 'spring']
Synset: startle.v.02(move or jump suddenly, as if in surprise or alarm)
         Lemmas:['startle', 'jump', 'start']
Synset: jump.v.03(make a sudden physical attack on)
         Lemmas:['jump']
Synset: jump.v.04(increase suddenly and significantly)
         Lemmas:['jump']
Synset: leap_out.v.01(be highly noticeable)
         Lemmas:['leap_out', 'jump_out', 'jump', 'stand_out', 'stick_out']
Synset: jump.v.06(enter eagerly into)
         Lemmas:['jump']
Synset: rise.v.11(rise in rank or status)
         Lemmas:['rise', 'jump', 'climb_up']
Synset: jump.v.08(jump down from an elevated point)
         Lemmas:['jump', 'leap', 'jump_off']
Synset: derail.v.02(run off or leave the rails)
         Lemmas:['derail', 'jump']
Synset: chute.v.01(jump from an airplane and descend with a parachute)
         Lemmas:['chute', 'parachute', 'jump']
Synset: jump.v.11(cause to jump or leap)
         Lemmas:['jump', 'leap']
Synset: jumpstart.v.01(start (a car engine whose battery is dead) by connec
ting it to another car's battery)
         Lemmas:['jumpstart', 'jump-start', 'jump']
Synset: jump.v.13(bypass)
         Lemmas:['jump', 'pass_over', 'skip', 'skip_over']
Synset: leap.v.02(pass abruptly from one state or topic to another)
         Lemmas:['leap', 'jump']
Synset: alternate.v.01(go back and forth; swing back and forth between two
states or conditions)
         Lemmas:['alternate', 'jump']
```

## 6c. How does WordNet organize verbs?

WordNet also organizes verbs into hierarchies based on the hyponym-hypernym relations between synsets. A hypernym is essentially the broader grouping of a word. For example, feline is a hypernym of cat. A hyponym is the opposite. It is a specific subgroup of a a word. For example, cat is a hyponym of feline. There can also be other relationships between synsets such as meronym (part of), holonym (whole), and troponym (defines a more specific action).

## 7. Use morphy to find as many different forms of the word as you can.

In [353…
```
wn.morphy('jump', wn.VERB)
```

Out[353]:
```
'jump'
```

## 8a. Run the Wu-Palmer similarity metric and the Lesk algorithm on two words that you think may be similar.

In [354…

```
# Word 1
sun = wn.synset('sun.n.01') # find a specific synset

# Word 2
star = wn.synset('star.n.01') # find a specific synset

# Wu-Palmer similarity metric
print("Wu-Palmer metric:", wn.wup_similarity(sun, star))

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

print("\nLesk algorithm:") # print a divider
print("\tDefinitions of sun:")
for ss in wn.synsets('sun'): # find all the definitions for sun
    print("\t\t", ss, ss.definition())

print('\tSpecific definition for "sun" in the sentence - "The Earth revolves
sent = ['The', 'Earth', 'revolves', 'around', 'the', 'Sun' '.'] # sentence t
print("\t\t", lesk(sent, 'sun', 'n')) # lesk algorithm is run
```

```
Wu-Palmer metric: 0.9333333333333333

Lesk algorithm:
        Definitions of sun:
                Synset('sun.n.01') the star that is the source of light an
d heat for the planets in the solar system
                Synset('sunlight.n.01') the rays of the sun
                Synset('sun.n.03') a person considered as a source of warm
th or energy or glory etc
                Synset('sun.n.04') any star around which a planetary syste
m revolves
                Synset('sunday.n.01') first day of the week; observed as a
day of rest and worship by most Christians
                Synset('sun.v.01') expose one's body to the sun
                Synset('sun.v.02') expose to the rays of the sun or affect
by exposure to the sun
        Specific definition for "sun" in the sentence - "The Earth revolves
around the Sun.":
                Synset('sun.n.04')
```

## 8b. Observations of the Wu-Palmer similarity metric and the Lesk algorithm:

The Wu-Palmer similarity metric worked in a way that I did not expect. Many words that I tried that were similar did not return a high metric. For example when I did the words 'cry' and 'bawl' which are synonyms, the metric returned was 0.2, implying that they are not similar. I tried many other words that did not work, until I finally landed on 'sun' and 'star' which returned a metric of 0.9333.

The Lesk algorithm worked when certain words in the sentence matched parts of the definition. I tried a simple sentence at first and got the correct definition, but when I tried the sentence "He is the sun in my life" to get the 3rd noun definition, it returned that it was the 1st noun definition. So after rewording the sentence a few times to try to get different definitions for the word 'sun', I finally got it to correctly identify that sun referred to the 4th definition in the sentence "The Earth revolves around the Sun."

# 9a. What is SentiWordNet?

SentiWordNet is a lexical resource created around WordNet that calculates sentiment scores for each synset. Specifically these are positivity, negativity, and objectivity scores. Use cases of SentiWordNet are centered around sentiment analysis, such as reading online posts to determine how the public feels about controversial events.

# Using SentiWordNet

### 9b. Get all the senti-synsets of an emotionally charged word. Output the polarity scores for each word.

In [355…

```
from nltk.corpus import sentiwordnet as swn # import sentiwordnet

# Choose an emotionally charged word
anxiety = swn.senti_synset('anxiety.n.01')
print(anxiety)

# Print out sentiment scores for the word anxiety
print("Positive score = ", anxiety.pos_score())
print("Negative score = ", anxiety.neg_score())
print("Objective score = ", anxiety.obj_score())

print("\nPolarity scores for each senti-synset:")
senti_list = list(swn.senti_synsets('anxiety')) # find polarity scores for t
for item in senti_list:
    print("\t", item)
```

```
<anxiety.n.01: PosScore=0.0 NegScore=0.125>
Positive score =  0.0
Negative score =  0.125
Objective score =  0.875

Polarity scores for each senti-synset:
         <anxiety.n.01: PosScore=0.0 NegScore=0.125>
         <anxiety.n.02: PosScore=0.125 NegScore=0.75>
```

## 9c. Output the polarity for each word in a sentence of your choice.

In [356…

```
# Pick a sentence
sent = "The boy was happy and excited to finally eat his favorite candy bar"
print("Sentence:", sent)

tokens = sent.split() # tokenizes sentence

for token in tokens:
    syn_list = list(swn.senti_synsets(token)) # get all senti_synsets for ea
    if syn_list: # continue as long as there are words in the list
        syn = syn_list[0]
        print("\nWord: ", token) # print out the word
        print("\tPositive score =", syn.pos_score()) # print out the positiv
        print("\tNegative score =", syn.neg_score()) # print out the negativ
        print("\tObjective score =", syn.obj_score()) # print out the object
```

```
Sentence: The boy was happy and excited to finally eat his favorite candy b
ar

Word:  boy
        Positive score = 0.25
        Negative score = 0.0
        Objective score = 0.75

Word:  was
        Positive score = 0.0
        Negative score = 0.0
        Objective score = 1.0

Word:  happy
        Positive score = 0.875
        Negative score = 0.0
        Objective score = 0.125

Word:  excited
        Positive score = 0.25
        Negative score = 0.375
        Objective score = 0.375

Word:  finally
        Positive score = 0.0
        Negative score = 0.0
        Objective score = 1.0

Word:  eat
        Positive score = 0.0
        Negative score = 0.0
        Objective score = 1.0

Word:  favorite
        Positive score = 0.25
        Negative score = 0.0
        Objective score = 0.75

Word:  candy
        Positive score = 0.0
        Negative score = 0.0
        Objective score = 1.0

Word:  bar
        Positive score = 0.0
        Negative score = 0.0
        Objective score = 1.0
```

## 9d. Observations of the SentiWordNet polarity scores and the utility of knowing then in an NLP application.

Since the program chooses the first senti_synset for each word, the polarity scores can sometimes be off. For example, when I tried to use the word 'frightened' in my sentence, the polarity scores for it were as follows: positive score = 0.375, negative score = 0.0, objective score = 0.625. These numbers do not make sense as 'frightened' usually does not have any positive connotation to it, only negative. When I ran the sentence "The boy was happy and excited to finally eat his favorite candy bar" the polarity scores returned were fairly accurate, with only a few exceptions. The scores can prove fairly important in an NLP application as they can help developers quickly analyze user opinion on an update on the application.

## 10a. What is a collocation?

A collocation is two or more words that are often put together to create a specific meaning. They consist of common phrases that people use regularly and can be made up any kind of words. Some examples include "fast food", "take a look", "see you later", and "pay attention".

## 10b. Output collocations for text4, the Inaugural corpus.

In [357…

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

# Get and print 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
```

## 10c. Select one of the collocations identified by NLTK and calculate mutual information.

In [358…
```
# Import math
import math

# Store text4 as 1 string and calculate the vocabulary
text = ' '.join(text4.tokens)
vocab = len(set(text4))

# Calculate and print mutual information for "Vice President"
info = text.count('Vice President')/vocab
print('p(Vice President) =', info)

v = text.count('Vice')/vocab
print("p(Vice) = ", v)

p = text.count('President')/vocab
print('p(President) = ', p)

PMI = math.log2(info / (v * p))
print('PMI = ', PMI)
```

```
p(Vice President) = 0.0017955112219451373
p(Vice) =  0.0018952618453865336
p(President) =  0.010773067331670824
PMI =  6.458424602064904
```

In [359…
```
# Calculate and print mutual information for "of the" to use as a comparator
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
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
```

## 10d. Analysis on the results of the mutual information formula.

Since the PMI, or Pointwise Mutual Information, is positive for the collocation "Vice President", we can conclude that the words 'Vice' and 'President' co-occur more frequently than they do independently. Also as an additional comparison, I calculated the PMI for "of the" and since the PMI for "Vice President" is higher than the PMI of "of the", it is more likely to be a collocation.