WordNet Protfolio Assignment

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

```
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True
```

1. Wordnet Summary:

WordNet is a hierarchial organization of nouns, verbs, adjectives and adverbs listing glosses (short definitions), sysnets (synonym sets), use examples and relations to other words. It is a project started by the Priceton psychologist George Miller who wanted to investigate peoples' cocepts of hierarchial organization.

2. Select a noun. Output all synsets

```
# get all synsets of 'anime'

wn.synsets('anime')
```

```
[Synset('zanzibar_copal.n.01'), Synset('anime.n.02')]
```

3. Select one synset from the list of synsets.

3.a Extract its definition, usage examples, and lemmas.

```
print('Definition: ', wn.synset('anime.n.02').definition())
print('Usage Examples: ', wn.synset('anime.n.02').examples())
print('Lemmas: ', wn.synset('anime.n.02').lemmas())
```

```
Definition:  any of various resins or oleoresins
Usage Examples:  []
```

```
Lemmas:  [Lemma('anime.n.02.anime'), Lemma('anime.n.02.gum_anime')]
```

3.b From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go

```
anime_synsets = wn.synsets('anime', pos = wn.NOUN)
for sense in anime_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + st
```

```
Synset: zanzibar_copal.n.01(a hard copal derived from an African tree)
         Lemmas:['Zanzibar_copal', 'anime']
Synset: anime.n.02(any of various resins or oleoresins)
         Lemmas:['anime', 'gum_anime']
```

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

Nouns are the most highly-connected synets. Nouns are organized with a hierarchy, where the entity is at the top.

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

```
anime = wn.synset('anime.n.02')
print('Hypernymns: ', anime.hypernyms())
print('Hyponyms: ', anime.hyponyms())
print('Meronyms: ', anime.part_meronyms())
print('Holonyms: ', anime.part_holonyms())
print('Antonym: ', anime.lemmas()[0].antonyms(), anime.lemmas()[1].antonyms())
```

```
Hypernymns:  [Synset('natural_resin.n.01')]
Hyponyms:  []
Meronyms:  []
Holonyms:  []
Antonym:  [] []
```

5. Select a verb. Output all synsets.

```
# get all synsets of 'sleep'
```

```
wn.synsets('sleep')
```

```
[Synset('sleep.n.01'),
 Synset('sleep.n.02'),
 Synset('sleep.n.03'),
 Synset('rest.n.05'),
 Synset('sleep.v.01'),
 Synset('sleep.v.02')]
```

6. Select one synset from the list of synsets.

6.a Extract its definition, usage examples, and lemmas.

```
print('Definition: ', wn.synset('rest.n.05').definition())
print('Usage Examples: ', wn.synset('rest.n.05').examples())
print('Lemmas: ', wn.synset('rest.n.05').lemmas())
```

```
Definition:  euphemisms for death (based on an analogy between lying in a bed an
Usage Examples:  ['she was laid to rest beside her husband', 'they had to put th
Lemmas:  [Lemma('rest.n.05.rest'), Lemma('rest.n.05.eternal_rest'), Lemma('rest.
```

6.b From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting the synsets as you go

```
sleep_synsets = wn.synsets('sleep', pos = wn.VERB)
for sense in sleep_synsets:
    lemmas = [l.name() for l in sense.lemmas()]
    print("Synset: " + sense.name() + "(" +sense.definition() + ")  \n\t Lemmas:" + st
```

```
Synset: sleep.v.01(be asleep)
         Lemmas:['sleep', 'kip', 'slumber', "log_Z's", "catch_some_Z's"]
Synset: sleep.v.02(be able to accommodate for sleeping)
         Lemmas:['sleep']
```

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

Verns can be in hypernym/hyponum relations. Unlike nouns, there is no top level for all verbs.

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

```
wn.morphy('sleep', wn.VERB)
```

```
'sleep'
```

```
wn.morphy('sleep', wn.NOUN)
```

```
'sleep'
```

```
wn.morphy('sleepy', wn.ADJ)
```

```
'sleepy'
```

```
wn.morphy('sleepier', wn.ADJ)
```

```
'sleepy'
```

```
wn.morphy('slept', wn.VERB)
```

```
'sleep'
```

8. Select two words that you think might be similar.

- make
- create

8.a Find the specific synsets you are interested in.

- make.v.01
- create.v.03

```
print('Make synstets: ', wn.synsets('make'))
print('Craete synstets: ', wn.synsets('create'))
```

```
Make synstets:  [Synset('brand.n.02'), Synset('shuffle.n.01'), Synset('make.v.01
Craete synstets:  [Synset('make.v.03'), Synset('create.v.02'), Synset('create.v.
```

8.a Run the Wu-Palmer similarity metric and the Lesk algorithm. Write a couple of sentences with your observations.

From the code below, we can see that the chosen synsets from above have a Wu palmer similarity score of 0.4. This means that they are not too similar. Additionally, looking at the lesk algorthim ran on 'make' and 'crate; using the context senteces below, we see that the synsets most relevant to ake and create are make.v.02 and create.v.04 respectively

```
make = wn.synset('make.v.01')
create = wn.synset('create.v.03')
```

```
wn.wup_similarity(make, create)
```

```
0.4
```

```
from nltk.wsd import lesk
```

```
make_sent = 'I like to make sandwiches when I am hungry'.split(' ')
create_sent = 'I like to create new characters in NBA2k when I get bored with the defa

print(lesk(make_sent, 'make', pos = 'v'))
print(lesk(create_sent, 'create', pos = 'v'))
```

```
Synset('make.v.42')
Synset('create.v.04')
```

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

SentiWordNet is built on top of WordNet and assigns 3 scores: positive, negative, objectivity. SentiWordNet requires that a corpus already be downloaded and performe sentiment analysis to see if text is postive/neative ot objective/subjective. Use cases include seeing which product reviews are postive/negative in order to find which aspects of a product customers are happy/unhappy about for further improvements/innovations

9.a Select an emotionally charged word. Find its senti-synsets and output the polarity scores for each word.

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

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

```
wn.synsets('sad')
```

```
[Synset('sad.a.01'), Synset('sad.s.02'), Synset('deplorable.s.01')]
```

```
wn.synset('sad.a.01').definition()
```

```
'experiencing or showing sorrow or unhappiness; ; - Christina Rossetti'
```

```
sad = swn.senti_synset('sad.a.01')
print("Positive score = ", sad.pos_score())
print("Negative score = ", sad.neg_score())
print("Objective score = ", sad.obj_score())
```

```
Positive score =  0.125
Negative score =  0.75
Objective score =  0.125
```

9.b Make up a sentence. Output the polarity for each word in the sentence.

```
sent = 'i am very excited to see the new anime releasing during the upcoming fall seas
```

```
for word in sent:
    if len(wn.synsets(word)) > 0:
        syn = wn.synsets(word)[0].name()
        print(syn)
        senti_syn = swn.senti_synset(syn)
        print(senti_syn)
        print("Positive score = ", senti_syn.pos_score())
        print("Negative score = ", senti_syn.neg_score())
        print("Objective score = ", senti_syn.obj_score())
        print()
```

```
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

very.s.01
<very.s.01: PosScore=0.5 NegScore=0.0>
Positive score =  0.5
Negative score =  0.0
Objective score =  0.5

excite.v.01
<excite.v.01: PosScore=0.25 NegScore=0.375>
Positive score =  0.25
Negative score =  0.375
Objective score =  0.375

see.n.01
<see.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

new.a.01
```

```
new.a.01
<new.a.01: PosScore=0.375 NegScore=0.0>
Positive score =  0.375
Negative score =  0.0
Objective score =  0.625

zanzibar_copal.n.01
<zanzibar_copal.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

let_go_of.v.01
<let_go_of.v.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

approaching.s.01
<approaching.s.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

fall.n.01
<fall.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

season.n.01
<season.n.01: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
```

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

Looking at the polarity scores for my sentence, I notice that most of the words are objective. This makes sence as most of the words were nouns that do not carry too much meaning. Surprisingly, the word 'excite' only had a positivity score of 0.25, with a higher negativity score of 0.35. This is surpsing because, given the context of the sentence, I ould expect the positivity score to be much higher than the negativity score. This is likely due to the way I selected the synsets to pass into the senti_synsets function. Selecting the first synset of a word might not always garuntee that the correct meaning of the word is bein conveyed. For example, words like 'i' and 'anime' have bizzare first synsets. Because of this, I think using the lesk algorithm to first find the appropriate synsets and then feeding those into the senti_synets function would be better. In an NLP application that first finds the correct synsets, the polarity scores could be very insightfull in the feild of product

analytics or any space with customer reviews. These polarity scores could very easily help an organization gage how their target population is responding to products or new changes.

10. Write a couple of sentences about what a collocation is.

A collocation is when two or more words combine more often than expected by chance, and you cannot sunstitute a word and get the same meaning. These can be found with point-wise mutual information (PMI):

$$PMI = log_2 \frac{P(x,y)}{P(x)P(y)}$$

- PMI = 0: x and y are independent
- PMI > 0: likely to be a collocation
- PMI < 0: not likely to be a collocation

10.b Output collocations for text4, the Inaugural corpus

```
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('stopwords')

from nltk.book import text4
```

```
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural 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 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!
```

```
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
```

10.c Select one of the collocations identified by NLTK. Calculate mutual information. Write commentary on the results of the mutual information formula and your interpretation.

Looking at the code below, we see that PMI for United States is around 4.815. Looking specifically at $P(UnitedStates)$, $P(United)$ and $P(States)$, we see that the values are around 0.016, 0.017 and 0.033 respectively. This means that the word 'States' occurs more often than 'United'. Also since $P(States) > P(UnitedStates)$, 'States' appears more times than the combination of 'United' and 'States'. However, since the PMI is larger than 0, we can say that it is likely that 'United States' is a collocation.

```
import math


text = ' '.join(text4.tokens)
text[:50]
```

```
'Fellow - Citizens of the Senate and of the House o'
```

```
vocab = len(set(text4))

p_US = text.count('United States') / vocab
print('p(United States) = ', p_US)

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

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

pmi = math.log2(p_US / (p_United * p_States))
print('PMI = ', pmi)
```

```
p(United States) =  0.015860349127182045
p(United) =  0.0170573566084788
p(States) =  0.03301745635910224
PMI =  4.815657649820885
```

Colab paid products - Cancel contracts here

✓ 0s completed at 4:44 PM