WordNet

WordNet

WordNet is a hierarchal collection that contains the following: nouns, verbs, adjectives, and adverbs. It al contains glosses, which are short definitions, synsets, which are synonym sets, use examples, and relations to other words.

This project aims to demonstrate the use of WordNet and SentiWordNet.

Author: Meinhard Benedict Capucao

```
from nltk.corpus import wordnet as wn
from nltk.stem import WordNetLemmatizer
from nltk.corpus import sentiwordnet as swn
import nltk
nltk.download('sentiwordnet')
nltk.download('wordnet')
nltk.download('omw-1.4')

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

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[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!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg 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 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]   Unzipping corpora/nps_chat.zip.
True
```

What is a **Sysnet**?

A sysnet is a hierarchical grouping of words with similar meaning. They are divided into:

- Hypernyms: The higher word, where similar words fall under.
- Hyponyms: The lower word, which falls under the category of a broader word.
- Meronyms: A word that is part of another; elements that combine to make a whole.
- Holoynms: A word that is the whole of other parts, opposite of meronyms.
- Tropoynms: A word that is a more specific action of another word.

Noun Sysnet Exploration

First, we will find the sysnets for a noun. Let's use 'drink'.

```
# Find the sysnets for 'drink', and output them.
wn.synsets('drink')
```

```
[Synset('drink.n.01'),
 Synset('drink.n.02'),
 Synset('beverage.n.01'),
 Synset('drink.n.04'),
 Synset('swallow.n.02'),
 Synset('drink.v.01'),
 Synset('drink.v.02'),
 Synset('toast.v.02'),
 Synset('drink_in.v.01'),
 Synset('drink.v.05')]
```

Let's select one of the sysnets. In this demonstration, we'll select '**beverage.n.01**', since it is also a noun. We will get the definition, usage example, and lemmas for beverage.n.01:

- ** Definition: **any liquid suitable for drinking
- ** Usage examples:** ['may I take your beverage order?']
- **Examples:** [Lemma('beverage.n.01.beverage'), Lemma('beverage.n.01.drink'), Lemma('beverage.n.01.drinkable'), Lemma('beverage.n.01.potable')]

```
print(wn.synset('beverage.n.01').definition() + '\n' + str(wn.synset('beverage.n.01').examples()) + '\n' + str(wn.synset('beverage.n.01').lem
```

```
any liquid suitable for drinking
['may I take your beverage order?']
[Lemma('beverage.n.01.beverage'), Lemma('beverage.n.01.drink'), Lemma('beverage.n.01.drinkable'), Lemma('beverage.n.01.potable')]
```

Then, let's walk up the hierarchy for the noun synset beverage.

WordNet is hierarchically organized for nouns with entity at the top. Then, depending on the sysnet, it becomes more specific. The top of the beverage sysnet is entity, as all knowns. However, above beverage is food, substance, matter, physical entity, then entity.

```
beverage = wn.synset('beverage.n.01')
bev = beverage.hypernyms()[0]
top = wn.synset('entity.n.01')
while bev:
    print(bev)
    if bev == top:
        break
    if bev.hypernyms():
        bev = bev.hypernyms()[0]
```

```
Synset('food.n.01')
Synset('substance.n.07')
Synset('matter.n.03')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Next we will output the **hypernyms**, **hyponyms**, **meronyms**, **holonyms**, and **antonyms**. Looks like beverage has a few hypernyms, and a lot of hyponyms, which makes sense since there are many types of beverages.

```
print('hypernyms: ', beverage.hypernyms())
print('hyponyms: ', beverage.hyponyms())
print('meronyms: ', beverage.part_meronyms())
print('holonyms: ', beverage.part_holonyms())
print('antonyms: ', beverage.lemmas()[0].antonyms())
```

```
hypernyms:  [Synset('food.n.01'), Synset('liquid.n.01')]
hyponyms:  [Synset('alcohol.n.01'), Synset('cider.n.01'), Synset('cocoa.n.01'), Synset('coffee.n.01'), Synset('cooler.n.02'), Synset('dr
meronyms:  []
holonyms:  []
antonyms:  []
```

Verb Sysnet Exploration

First, we will find the sysnets for a verb. Let's use 'run' and the first verb that appears (**'run.v.01').**

```
# Find the sysnets for 'drink', and output them.
wn.synsets('run')
```

```
[Synset('run.n.01'),
 Synset('test.n.05'),
 Synset('footrace.n.01'),
 Synset('streak.n.01'),
 Synset('run.n.05'),
 Synset('run.n.06'),
 Synset('run.n.07'),
 Synset('run.n.08'),
 Synset('run.n.09'),
 Synset('run.n.10'),
 Synset('rivulet.n.01'),
```

```
 Synset('political_campaign.n.01'),
 Synset('run.n.13'),
 Synset('discharge.n.06'),
 Synset('run.n.15'),
 Synset('run.n.16'),
 Synset('run.v.01'),
 Synset('scat.v.01'),
 Synset('run.v.03'),
 Synset('operate.v.01'),
 Synset('run.v.05'),
 Synset('run.v.06'),
 Synset('function.v.01'),
 Synset('range.v.01'),
 Synset('campaign.v.01'),
 Synset('play.v.18'),
 Synset('run.v.11'),
 Synset('tend.v.01'),
 Synset('run.v.13'),
 Synset('run.v.14'),
 Synset('run.v.15'),
 Synset('run.v.16'),
 Synset('prevail.v.03'),
 Synset('run.v.18'),
 Synset('run.v.19'),
 Synset('carry.v.15'),
 Synset('run.v.21'),
 Synset('guide.v.05'),
 Synset('run.v.23'),
 Synset('run.v.24'),
 Synset('run.v.25'),
 Synset('run.v.26'),
 Synset('run.v.27'),
 Synset('run.v.28'),
 Synset('run.v.29'),
 Synset('run.v.30'),
 Synset('run.v.31'),
 Synset('run.v.32'),
 Synset('run.v.33'),
 Synset('run.v.34'),
 Synset('ply.v.03'),
 Synset('hunt.v.01'),
 Synset('race.v.02'),
 Synset('move.v.13'),
 Synset('melt.v.01'),
 Synset('ladder.v.01'),
 Synset('run.v.41')]
```

Let's select one of the sysnets. In this demonstration, we'll select **"run.v.01"**, since it is the first verb. We will get the definition, usage example, and lemmas for run.v.01:

- **Definition:** move fast by using one's feet, with one foot off the ground at any given time
- **Usage examples:** ["Don't run--you'll be out of breath", 'The children ran to the store']
- **Examples:** [Lemma('run.v.01.run')]

```
print(wn.synset('run.v.01').definition() + '\n' + str(wn.synset('run.v.01').examples()) + '\n' + str(wn.synset('run.v.01').lemmas()))
```

```
 move fast by using one's feet, with one foot off the ground at any given time
 ["Don't run--you'll be out of breath", 'The children ran to the store']
 [Lemma('run.v.01.run')]
```

Then, let's walk up the hierarchy for the verb synset run.

WordNet is hierarchically organized for verbs differently, with the hypernym for run being travel. This is more specific than noun's top hypernym, which is entity. This means there is more distinction in actions and that they cannot be generalized as easy.

```
runverb = wn.synset('run.v.01')
run = runverb.hypernyms()[0]
top = wn.synset('travel.v.01')
while run:
    print(run)
    if run == top:
        break
    if run.hypernyms():
        run = run.hypernyms()[0]
```

```
Synset('travel_rapidly.v.01')
Synset('travel.v.01')
```

Next we will output the **hypernyms**, **hyponyms**, **meronyms**, **holonyms**, and **antonyms**. Looks like run's hypernym is travel rapidy, and has many hyponyms. This makes sense since there are many other words that are similar to the verb run.

```
print('hypernyms: ', runverb.hypernyms())
print('hyponyms: ', runverb.hyponyms())
print('meronyms: ', runverb.part_meronyms())
print('holonyms: ', runverb.part_holonyms())
print('antonyms: ', runverb.lemmas()[0].antonyms())
```

```
hypernyms:  [Synset('travel_rapidly.v.01')]
hyponyms:  [Synset('hare.v.01'), Synset('jog.v.03'), Synset('lope.v.01'), Synset('outrun.v.01'), Synset('romp.v.02'), Synset('run.v.33')]
meronyms:  []
holonyms:  []
antonyms:  []
```

Next, we will find as many forms of the verb 'run' as we can. To do this, we use the morphy function.

```
print(wn.morphy('running', pos = 'v'))
print(wn.morphy('ran', pos = 'v'))
print(wn.morphy('run', pos = 'v'))
```

```
run
run
run
```

Next, we will select two words that are maybe similar: for this task we will select "boat" and "yacht".

We will also run the Wu-Palmer similarity metric and the Lesk algorithm.

The Wu-Palmer similarity metric is based on the common specific ancenstor node.

```
pond = wn.synset('pond.n.01')
lake = wn.synset('lake.n.01')
print('Path Similarity: ', pond.path_similarity(lake))
print('Wu-Palmer similarity: ', wn.wup_similarity(pond, lake))
print()
```

```
Path Similarity:  0.5
Wu-Palmer similarity:  0.9090909090909091
```

We see that pond and lake are 50% similar, according to path_similarity.

The Lesk algorithm returns the synset with the highest number of overlapping words between the context sentence and each synsets definition in the target word. In this case, the lesk algorithm correctly tags lake since water and body is there. This isn't always accurate though, especially for smaller bodies of text. It also makes sense that lake and pond are similar, as their hypernyms will both lead to water. That's why the Wu-Palmer similairty index scores it really high.

```
from nltk.wsd import lesk
# look at the definitions for 'bank'
for ss in wn.synsets('lake'):
    print(ss, ss.definition())

sent = ['Lets', 'jump', 'in', 'the', 'body', 'of', 'water', ',', 'the', 'lake', '!']
print('n', lesk(sent, 'lake'))
```

```
Synset('lake.n.01') a body of (usually fresh) water surrounded by land
Synset('lake.n.02') a purplish red pigment prepared from lac or cochineal
Synset('lake.n.03') any of numerous bright translucent organic pigments
n Synset('lake.n.01')
```

SentiWordNet

Next, we will explore SentiWordNet. This assigns three semantic scores for a given sysnet: positivity, negativity, and objectivity. These are assigned from a score of 0.0 to 1.0.

We will use an emotionally charged word to explore the applications of SentiWordNet... Let's use 'shame'. As expected, shame is used mostly in a negative connotation. The last verb is slightly positive, which is interesting.

```
print(wn.synsets('shame'))
shamenoun1 = swn.senti_synset('shame.n.01')
print(shamenoun1)

print("Positive score = ", shamenoun1.pos_score())
print("Negative score = ", shamenoun1.neg_score())
print( "Objective score = ", shamenoun1.obj_score(), '\n')

senti_list = list(swn.senti_synsets('shame'))
for item in senti_list:
    print(item)
```

```
[Synset('shame.n.01'), Synset('shame.n.02'), Synset('pity.n.02'), Synset('dishonor.v.01'), Synset('shame.v.02'), Synset('shame.v.03'), S
<shame.n.01: PosScore=0.0 NegScore=0.75>
Positive score =  0.0
Negative score =  0.75
Objective score =  0.25

<shame.n.01: PosScore=0.0 NegScore=0.75>
<shame.n.02: PosScore=0.0 NegScore=0.5>
<pity.n.02: PosScore=0.0 NegScore=0.625>
<dishonor.v.01: PosScore=0.0 NegScore=0.625>
<shame.v.02: PosScore=0.0 NegScore=0.5>
<shame.v.03: PosScore=0.0 NegScore=0.375>
<shame.v.04: PosScore=0.125 NegScore=0.125>
```

Let's try to make up a sentence and see it's polarity, also for each word...

'Keep going, you did a good job!'

We see that the positive count is 0.5... Where does this come from?

We can see that it all comes from job, which makes sense. If we switch out good, the semantics of the sentence changes completely. This is a good tool to use to determine if sentences lean positive or negative. If a sentence has overwhelmingly positive/negative polarity, then it can be used in various ways; for example, a chatbot can respond to negative or positive words differently.

```
sent = 'Keep going, you did a good job!'
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()

print("neg\tpos counts")
print(neg, '\t', pos)

print('Score for keep: ')
p = list(swn.senti_synsets('keep'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')

print('Score for going: ')
p = list(swn.senti_synsets('going'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')

print('Score for did: ')
p = list(swn.senti_synsets('did'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')
```

```
print('Score for a: ')
p = list(swn.senti_synsets('a'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')


print('Score for good: ')
p = list(swn.senti_synsets('good'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')


print('Score for job: ')
p = list(swn.senti_synsets('job'))[0]
print("negative: ", p.neg_score())
print("positive: ", p.pos_score())
print("objective: ", p.obj_score(), '\n')
```

```
neg     pos counts
0.0      0.5
Score for keep:
negative:  0.0
positive:  0.0
objective:  1.0

Score for going:
negative:  0.0
positive:  0.0
objective:  1.0

Score for did:
negative:  0.0
positive:  0.0
objective:  1.0

Score for a:
negative:  0.0
positive:  0.0
objective:  1.0

Score for good:
negative:  0.0
positive:  0.5
objective:  0.5

Score for job:
negative:  0.0
positive:  0.0
objective:  1.0
```

Lastly, let's explore collocations. Collocations are words that have greater means when put together. They are also more likely to be combined because of association, for example, 'fast food' combines to have a whole new meaning.

We will output collocations for text4, the Inaugural corpus.

```
from nltk.book import *
text4

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
```

Let's select 'foreign nations' and calculate the mutual information.

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

import math
```

```
vocab = len(set(text4))
hg = text.count('foreign nations')/vocab
print("p(foreign nations) = ",hg )
h = text.count('foreign')/vocab
print("p(foreign) = ", h)
g = text.count('nations')/vocab
print('p(nations) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
vocab = len(set(text4))
hg = text.count('and I')/vocab
print("p(and I) = ",hg )
h = text.count('and')/vocab
print("p(and) = ", h)
g = text.count('I')/vocab
print('p(I) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
p(foreign nations) =  0.0014962593516209476
p(foreign) =  0.01027431421446384
p(nations) =  0.020448877805486283
pmi =  2.8322245851494996
p(and I) =  0.004788029925187032
p(and) =  0.5833416458852868
p(I) =  0.16987531172069825
pmi =  -4.371313198680179
```

The 2.8 pmi indicates this is more likely to be a collocation, as it is positive and is more than the 'as I', which is not a collocation. The higher the pmi that is calculate through correlation and the log of probability from the vocab, the greater chance the two words form a meaning that is beyond its parts.

✓ 0s completed at 8:36 PM