WordNet

Wordnet is a lexical database of English that uses synsets to group nounse, verbs, adjectives, and adverbs into sets of cognitive synonyms, each group expressing a distinct concept.

These include:

- hypernyms and hyponyms: represent higher and lower level words according to a hierarchy
- meronyms and holonyms: represent words that make up or are whole versions (respectively) of a word
- troponyms: a more specific version of a word

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
```

```
[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!
```

```
wn.synsets('table')
```

```
[Synset('table.n.01'),
 Synset('table.n.02'),
 Synset('table.n.03'),
 Synset('mesa.n.01'),
 Synset('table.n.05'),
 Synset('board.n.04'),
 Synset('postpone.v.01'),
 Synset('table.v.02')]
```

```
print(wn.synset('table.n.02').definition())
print(wn.synset('table.n.02').examples())
wn.synset('table.n.02').lemmas()
```

```
a piece of furniture having a smooth flat top that is usually supported by one or more vertical legs
['it was a sturdy table']
[Lemma('table.n.02.table')]
```

```
table_set = wn.synset('table.n.02')
while table_set:
  print(table_set)
  table_set = table_set.hypernyms()[0]
  if table_set == wn.synset('entity.n.01'):
    print(table_set)
    break
```

```
Synset('table.n.02')
Synset('furniture.n.01')
Synset('furnishing.n.02')
Synset('instrumentality.n.03')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Observations of Wordnet Hierarchy

After the above code block, it's evident that wordnet organizes its verbs into a tree, with levels for synsets and "parents" that make a path back up to the root, entity.n.01. Synsets with the same parent are seen to be more related to each other, and wordnet's path_similarity() function suggests that the similarity of any two words can be modeled by the length of the path traversal to go from one word in the hierarchy to another.

```
table_set = wn.synset('table.n.02')
print(f'Holonyms: {table_set.member_holonyms()}')
print(f'Meronyms: {table_set.member_meronyms()}')
print(f'Hypernyms: {table_set.hypernyms()}')
print(f'Hyponyms: {table_set.hyponyms()}')
print(f'Antonyms: {table_set.lemmas()[0].antonyms()}')
```

```
Holonyms: []
Meronyms: []
Hypernyms: [Synset('furniture.n.01')]
Hyponyms: [Synset('altar.n.01'), Synset('booth.n.01'), Synset('breakfast_table.n.01'), Synset('card_table.n.01')
Antonyms: []
```

```
wn.synsets('fight')
```

```
[Synset('battle.n.01'),
 Synset('fight.n.02'),
 Synset('competitiveness.n.01'),
 Synset('fight.n.04'),
 Synset('fight.n.05'),
 Synset('contend.v.06'),
 Synset('fight.v.02'),
 Synset('fight.v.03'),
 Synset('crusade.v.01')]
```

```
print(wn.synset('fight.v.02').definition())
print(wn.synset('fight.v.02').examples())
wn.synset('fight.v.02').lemmas()
```

```
fight against or resist strongly
['The senator said he would oppose the bill', "Don't fight it!"]
[Lemma('fight.v.02.fight'),
 Lemma('fight.v.02.oppose'),
 Lemma('fight.v.02.fight_back'),
 Lemma('fight.v.02.fight_down'),
 Lemma('fight.v.02.defend')]
```

```
fight_set = wn.synset('fight.v.02')
while fight_set:
  print(fight_set)
  if len(fight_set.hypernyms()) == 0:
    break
  fight_set = fight_set.hypernyms()[0]
```

```
Synset('fight.v.02')
Synset('contend.v.06')
```

Return all words that have the same root word as 'fight'

```
fight_forms = [word for word in wn.words() if wn.morphy(word) == wn.morphy('fight')]
```

```
fight_forms
```

```
['fight']
```

Comparing two words I think are similar

I chose intelligent and smart, since I use them interchangeably.

Choosing the synset that represents my words most accurately

```
#@title Choosing the synset that represents my words most accurately

print(wn.synsets("smart"))
print(wn.synsets("intelligent"))
```

```
[Synset('smart.n.01'), Synset('ache.v.03'), Synset('smart.a.01'), Synset('chic.s.01'), Synset('bright.s.03'), Sy
[Synset('intelligent.a.01'), Synset('intelligent.s.02'), Synset('healthy.s.04'), Synset('intelligent.s.04')]
```

Showing definitions for the synsets I chose

```
#@title Showing definitions for the synsets I chose
print(f'Intelligent.a.01 def: {wn.synset("intelligent.a.01").definition()}')
print(f'Smart.a.01 def: {wn.synset("smart.a.01").definition()}')
```

```
Intelligent.a.01 def: having the capacity for thought and reason especially to a high degree
Smart.a.01 def: showing mental alertness and calculation and resourcefulness
```

Wu-Palmer similarity metric

```
#@title Wu-Palmer similarity metric
wn.wup_similarity(wn.synset("intelligent.a.01"), wn.synset("smart.a.01"))
```

```
0.5
```

Lesk algorithm for smart describing Aditya

```
#@title Lesk algorithm for smart describing Aditya
lesk("Aditya is very smart.".split(), "smart")
```

```
Synset('smart.s.07')
```

Lesk algorithm for smart describing guy

```
#@title Lesk algorithm for smart describing guy
lesk("Aditya is a very smart guy".split(), "smart")
```

```
Synset('smart.n.01')
```

Showing definition for what the lesk algorithm predicted for the sentences

```
#@title Showing definition for what the lesk algorithm predicted for the sentences
display(wn.synset("smart.s.07").definition())
display(wn.synset("smart.n.01").definition())
```

```
'capable of independent and apparently intelligent action'
'a kind of pain such as that caused by a wound or a burn or a sore'
```

As seen above, even though the two sentences have basically the same meaning, and same usage of smart as an adjective, for some reason the lesk algorithm classified the second usage of smart as a noun describing pain rather than correctly identifying it as a descriptor for Aditya.

The only thing that changed was that the word "smart" in the second context was used to describe "guy" was was then attributed to Aditya, as opposed to the first sentence directly attributing the quality of "smart" to Aditya.

Unfortunately for Aditya, it seems the Lesk algorithm doesn't seem too keen on allowing him to both be smart and a person at the same time, poor guy.

SentiWordNet

Description

SentiWordNet is a free lexical resource for opinion mining. It assigns each synset of WordNet 3 sentiment scores, one for positivity, one for negativity, and one for objectivity.

This resource supports sentiment classification tasks and other uses for opinion mining, and works closely with WordNet to annotate each synset with the above 3 sentiment scores.

The resource seems to be useful for sentiment analysis only once a sentence has been correctly analyzed and assigned synsets for each word. However this already implies an assumption that individual words can be considered as a unit of opinion information, and can be evaluated independently of one another to generate sentiment scores to then somehow combine into a final sentiment score for the whole passage. Not to mention that having a model correctly classify each word into its synset is arguably a more difficult task than attributing a sentiment score to each individual usage of a word. Both of these culminate in a fact that can be seen in a number of papers: that lexicon-based sentiment analysis does not perform as well as contemporary NN/Transformer based sentiment analysis.

Sources:

https://www.researchgate.net/publication/356427417_Lexicon-based_Methods_vs_BERT_for_Text_Sentiment_Analysis

https://doi.org/10.3390/electronics11030374

However there have been recent studies arguing for a combination of lexical-based approaches and NN architectures for sentiment analysis, especially in languages other than english:

https://doi.org/10.1109/KSE53942.2021.9648599

https://doi.org/10.1088/1742-6596/1802/3/032113

I believe the motivation for this is that whereas Lexicon-based approaches perform poorly on large passages filled with context (because it evaluates one work at a time), a BERT model can evaluate entire sequences at a time. Meanwhile, the BERT model (which is built for general NLP tasks and then fine-tuned) can gain very specific sentiment information from Lexicon-based sentiment values. Therefore it can be said that these two methods can work off of each other and account for the others' weaknesses.

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

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

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

Using an emotionally charged word to view SentiWordNet evaluation

```
#@title Using an emotionally charged word to view SentiWordNet evaluation

# Choosing the word 'stupid', viewing synsets
wn.synsets("stupid")
```

```
[Synset('stupid.n.01'),
 Synset('stupid.a.01'),
 Synset('dazed.s.01'),
 Synset('unintelligent.a.01')]
```

```
# Viewing polarity scores for all synsets for 'stupid'
def view_polarity(word: str):
  print(f'-----------\nViewing polarity scores for all synsets of word: {word}')
  word_synsets = swn.senti_synsets(word)
  for w_synset in word_synsets:
    print(f'3 sentiment scores for \'{w_synset}\':')
    print(f'\tPositivity score: {w_synset.pos_score()}')
    print(f'\tNegativity score: {w_synset.neg_score()}')
    print(f'\tObjectivity score: {w_synset.obj_score()}')
  print('-----------')
view_polarity('stupid')
```

```
-----------
Viewing polarity scores for all synsets of word: stupid
3 sentiment scores for '<stupid.n.01: PosScore=0.0 NegScore=0.125>':
        Positivity score: 0.0
        Negativity score: 0.125
        Objectivity score: 0.875
3 sentiment scores for '<stupid.a.01: PosScore=0.0 NegScore=0.75>':
        Positivity score: 0.0
        Negativity score: 0.75
        Objectivity score: 0.25
3 sentiment scores for '<dazed.s.01: PosScore=0.0 NegScore=0.125>':
        Positivity score: 0.0
        Negativity score: 0.125
        Objectivity score: 0.875
3 sentiment scores for '<unintelligent.a.01: PosScore=0.0 NegScore=0.375>':
        Positivity score: 0.0
        Negativity score: 0.375
        Objectivity score: 0.625
-----------
```

Tokenizing an emotionally charged sentence and garnering polarity scores (I really didn't like my Advanced Algo class)

```
#@title Tokenizing an emotionally charged sentence and garnering polarity scores (I really didn't like my Advanced Al
sentence = "I loved taking advanced algorithms SO MUCH, especially the fact that 90% of the grade was based on 9 stup

tokens = [t.lower() for t in nltk.word_tokenize(sentence) if t.isalpha()]

for token in tokens:
  view_polarity(token)
```

```
    -----------
    Viewing polarity scores for all synsets of word: from
    -----------
    -----------
    Viewing polarity scores for all synsets of word: stupid
    3 sentiment scores for '<stupid.n.01: PosScore=0.0 NegScore=0.125>':
            Positivity score: 0.0
```

```
        Positivity score: 0.0
        Negativity score: 0.125
        Objectivity score: 0.875
3 sentiment scores for '<stupid.a.01: PosScore=0.0 NegScore=0.75>':
        Positivity score: 0.0
        Negativity score: 0.75
        Objectivity score: 0.25
3 sentiment scores for '<dazed.s.01: PosScore=0.0 NegScore=0.125>':
        Positivity score: 0.0
        Negativity score: 0.125
        Objectivity score: 0.875
3 sentiment scores for '<unintelligent.a.01: PosScore=0.0 NegScore=0.375>':
        Positivity score: 0.0
        Negativity score: 0.375
        Objectivity score: 0.625
-----------
-----------
Viewing polarity scores for all synsets of word: dumb
3 sentiment scores for '<dense.s.04: PosScore=0.0 NegScore=0.25>':
        Positivity score: 0.0
        Negativity score: 0.25
        Objectivity score: 0.75
3 sentiment scores for '<speechless.s.01: PosScore=0.0 NegScore=0.625>':
        Positivity score: 0.0
        Negativity score: 0.625
        Objectivity score: 0.375
3 sentiment scores for '<dumb.s.03: PosScore=0.0 NegScore=0.0>':
        Positivity score: 0.0
        Negativity score: 0.0
        Objectivity score: 1.0
3 sentiment scores for '<dumb.s.04: PosScore=0.0 NegScore=0.0>':
        Positivity score: 0.0
        Negativity score: 0.0
        Objectivity score: 1.0
-----------
-----------
Viewing polarity scores for all synsets of word: and
-----------
-----------
Viewing polarity scores for all synsets of word: horrible
3 sentiment scores for '<atrocious.s.03: PosScore=0.0 NegScore=0.625>':
        Positivity score: 0.0
        Negativity score: 0.625
        Objectivity score: 0.375
-----------
-----------
Viewing polarity scores for all synsets of word: exams
3 sentiment scores for '<examination.n.02: PosScore=0.0 NegScore=0.0>':
        Positivity score: 0.0
        Negativity score: 0.0
        Objectivity score: 1.0
-----------
```

As you can see, the word 'stupid' correctly got synsets that were very negatively charged for the most part.

As for the sentence, since the first part of it was sarcastic, it evaluated "love" as a positive word, but sarcasm has always been a thorn in the side of sentiment analysis to the point of being its own task due to how widely it's used.

The rest of the words were evaluated correctly, so overall the usefulness of SentiWordNet is not to be underestimated despite seeming so simplistic.

Collocations

Collocations are words that, when placed together, hold a meaning different than the sum of their parts. For example, the words "pay attention" hold a meaning that can't be garnered from the meanings of "pay" and "attention" separately.

Importing text 4: The Inaugural Address Corpus

```
#@title Importing text 4: The Inaugural Address Corpus
nltk.download('gutenberg')
nltk.download('genesis')
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 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]   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!
<Text: Inaugural Address Corpus>
```

Printing collocations for text4

```
#@title Printing collocations for text4
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
```

Calculating mutual information for 'public debt'

```
#@title Calculating mutual information for 'public debt'
import math
text = ' '.join(text4.tokens)
vocab = len(set(text4))
hg = text.count('public debt')/vocab
print(f'p(public debt) = {hg}')
h = text.count('public') / vocab
print(f'p(public) = {h}')
g = text.count('debt')/vocab
print(f'p(debt) = {g}')
pmi = math.log2(hg/(h*g))
print(f'pmi = {pmi}')
```

```
p(public debt) = 0.001396508728179551
p(public) = 0.03341645885286783
p(debt) = 0.004189526184538653
pmi = 3.318334830163354
```

Observations

Since the pmi is much higher than 0, this means that the probability of the two words appearing next to each other in that order (represented by p(public debt)) is much higher than the probability of them appearing next to each other by happenstance (represented by the p(public) * p(debt) which assumes that the respective probabilities are independent)

This mirrors the calcuation for correlation within probability, and the concept is similar. A higher score means that the two words are "correlated", and in language that means that those two words are likely to have their own meaning together rather than just happening to appear next to each other.