Charles Wallis

2/24/2023

WordNet is an English list of semantic vocabulary. It classifies words into a synonym group called 'synset', providing a brief and general definition, and records various semantic relationships betweens these vocabulary lists. There are two purposes to wordnet, and one: it is to create a combination of dictionaries (word collections) and sisorus (symmetrical and antonymic dictionaries), which can be used more intuitively, and second: to support automated text analysis and AI applications.

```
import nltk
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('stopwords')
from nltk.corpus import sentiwordnet as swn
from nltk.corpus import wordnet as wn
from nltk.book import text4
from nltk.wsd import lesk
import math
```

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

```
wn.synsets('tea') #all synsets of 'charles'
```

```
[Synset('tea.n.01'),
 Synset('tea.n.02'),
 Synset('tea.n.03'),
 Synset('tea.n.04'),
 Synset('tea.n.05')]
```

```
print(wn.synset('tea.n.01').definition()) #definition
print(wn.synset('tea.n.02').definition()) #definition
print(wn.synset('tea.n.02').examples()) #usage examples (I like 02 example)
print(wn.synset('tea.n.02').lemmas()) #lemmas
```

```
a beverage made by steeping tea leaves in water
a light midafternoon meal of tea and sandwiches or cakes
['an Englishman would interrupt a war to have his afternoon tea']
[Lemma('tea.n.02.tea'), Lemma('tea.n.02.afternoon_tea'), Lemma('tea.n.02.teatime')]
```

```
#Traversing the Hierarchy
tea = wn.synset('tea.n.01')
hyper = lambda s: s.hypernyms()
list(tea.closure(hyper))
```

```
[Synset('beverage.n.01'),
 Synset('food.n.01'),
 Synset('liquid.n.01'),
 Synset('substance.n.07'),
 Synset('fluid.n.01'),
 Synset('matter.n.03'),
 Synset('substance.n.01'),
 Synset('physical_entity.n.01'),
 Synset('part.n.01'),
 Synset('entity.n.01'),
 Synset('relation.n.01'),
 Synset('abstraction.n.06')]
```

For the word 'tea', wordnet organized it into a beverage, which is a food, which is a category of liquid, which is a substance, and so on until it reaches abstraction which I thought was interesting, since most other nouns stopped at being an entity.

```
print(wn.synset('tea.n.01').hypernyms())
print(wn.synset('tea.n.01').hyponyms())
print(wn.synset('tea.n.03').part_meronyms())
print(wn.synset('tea.n.05').part_holonyms())
print(wn.synset('tea.n.01').lemmas()[0].antonyms())
```

```
[Synset('beverage.n.01')]
[Synset('cambric_tea.n.01'), Synset('cuppa.n.01'), Synset('herb_tea.n.01'), Synset('ice_tea.n.01'), S
[Synset('tea.n.05')]
[Synset('tea.n.03')]
[]
```

meronyms, holonyms, and antonyms did not exist in the WordNet for 'tea', except for tea.n.03 and tea.n.05 were meronyms and holonyms for each other. There are no antonyms for the word tea, since it's just a beverage.

```
#Displaying synsets of the verb drink
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')]
```

```
print(wn.synset('drink.n.05').definition())
print(wn.synset('drink.n.05').examples())
print(wn.synset('drink.n.05').lemmas())
```

```
the act of swallowing
['one swallow of the liquid was enough', 'he took a drink of his beer and smacked his lips']
[Lemma('swallow.n.02.swallow'), Lemma('swallow.n.02.drink'), Lemma('swallow.n.02.deglutition')]
```

Interestingly, going through drink.n.01 up to 05 reminded me that the word 'drink' is also used as a noun, and 02 and 05 were the only "verb" definitions, and the 02 definition was specific to drinking alcoholic beverages. (even the 05 gave a beer example)

```
#Traversing the Hierarchy
drinking = wn.synset('drink.n.05')
hyper = lambda s: s.hypernyms()
list(drinking.closure(hyper))
```

```
[Synset('consumption.n.01'),
 Synset('bodily_process.n.01'),
 Synset('organic_process.n.01'),
 Synset('process.n.06'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

Just like nouns, it was processing up one broader idea at a time on what the "drinking" act is actually doing. A living thing will consume a liquid as a bodily process, which is an organic one, done by a physical entity. Perhaps I used a verb that is also used as a noun, but it seems like there are not significant difference in traversing through a noun hierarchy and a verb hierarchy.

```
print(wn.morphy('drink', wn.NOUN))
print(wn.morphy('drinks', wn.NOUN))
print(wn.morphy('drink', wn.VERB))
print(wn.morphy('drank', wn.VERB))
print(wn.morphy('drunk', wn.VERB))
print(wn.morphy('drinks', wn.VERB))
print(wn.morphy('drinked', wn.VERB))
print(wn.morphy('drinking', wn.VERB))
```

```
drink
drink
drink
drink
drink
drink
```

```
    drink
    drink


#Selecting two words that I think are similar and finding synsets
print(wn.synsets('tea'))
print(wn.synsets('wine'))

    [Synset('tea.n.01'), Synset('tea.n.02'), Synset('tea.n.03'), Synset('tea.n.04'), Synset('tea.n.05')]
    [Synset('wine.n.01'), Synset('wine.n.02'), Synset('wine.v.01'), Synset('wine.v.02')]


for n in wn.synsets(str('wine')):
  print(n, n.definition())
#I will be using wine.n.01, and tea.n.01 (from above)

    Synset('wine.n.01') fermented juice (of grapes especially)
    Synset('wine.n.02') a red as dark as red wine
    Synset('wine.v.01') drink wine
    Synset('wine.v.02') treat to wine


print("Wu-Palmer Similarity: ", wn.wup_similarity(wn.synset('wine.n.01'), wn.synset('tea.n.01')))

    Wu-Palmer Similarity:  0.8421052631578947
```

Trying out the Lesk algorithm

```
#display all of the term definitions
for n in wn.synsets(str('tea')):
  print(n, n.definition())

print()

for n in wn.synsets(str('wine')):
  print(n, n.definition())

    Synset('tea.n.01') a beverage made by steeping tea leaves in water
    Synset('tea.n.02') a light midafternoon meal of tea and sandwiches or cakes
    Synset('tea.n.03') a tropical evergreen shrub or small tree extensively cultivated in e.g. China and
    Synset('tea.n.04') a reception or party at which tea is served
    Synset('tea.n.05') dried leaves of the tea shrub; used to make tea

    Synset('wine.n.01') fermented juice (of grapes especially)
    Synset('wine.n.02') a red as dark as red wine
    Synset('wine.v.01') drink wine
    Synset('wine.v.02') treat to wine
    ['Our relatives in Italy wined and dined us for a week']


#writing sample sentences for lesk
sentence1 = ['The', 'tea', 'is', 'too', 'hot', 'to', 'drink']
sentence2 = ['black', 'tea', 'with', 'some', 'sugar', 'goes', 'well', 'with', 'a', 'strawberry', 'shortcak
sentence3 = ['I', 'planted', 'a', 'tea', 'tree', 'to', 'harvest', 'some', 'oil', 'from', 'it']
sentence4 = ['I', 'am', 'going', 'to', 'a', 'tea', 'party', 'with', 'my', 'friends']
sentence5 = ['I', 'packaged', 'the', 'dried', 'tea', 'leaves', 'so', 'I', 'can', 'brew', 'it', 'later']

sentence6 = ['Would', 'you', 'like', 'a', 'glass', 'of', 'wine?']
```

```
sentence7 = ['my', 'favorite', 'color', 'is', 'wine']
sentence8 = ['I', 'would', 'like', 'to', 'get', 'some', 'wine', 'with', 'my', 'meal']
sentence9 = ['Would', 'you', 'like', 'to', 'wine', 'and', 'dine', 'with' ,'me?']

#running the lesk algorithm to check if it detects with correct noun for each different uses of noun
print(lesk(sentence1, 'tea')) #I'll count that as a correct
print(lesk(sentence2, 'tea')) #wrong
print(lesk(sentence3, 'tea')) #correct
print(lesk(sentence4, 'tea')) #correct
print(lesk(sentence5, 'tea')) #correct

print(lesk(sentence6, 'wine')) #wrong
print(lesk(sentence7, 'wine')) #wrong
print(lesk(sentence8, 'wine')) #wrong
print(lesk(sentence9, 'wine')) #correct
```

```
Synset('tea.n.05')
Synset('tea.n.04')
Synset('tea.n.03')
Synset('tea.n.04')
Synset('tea.n.05')
Synset('wine.n.02')
Synset('wine.v.02')
Synset('wine.v.02')
Synset('wine.v.02')
```

So the Wu-Palmer algorithm seems to capture the similarity between words very well, since it gave a pretty high number for something I did expect to have a high similarity.

The lesk algorithm worked surprisingly well for the tea example, even when the definitions confused me when I wrote the sentences. surprisingly the wine words didn't work as intended, even when I thought I got those down (like sentence 7, where I used it as a color).

SentiWordNet

SentiWordNet uses WordNet's resources to attach sentience attributes to words and sentences. Each synset has 3 sentiment scores (positive, negative, and objectivity), where the sum of all three is 1. This is applicable in sentiment analysis, trying to read the tone of a sentence, identify tone of a product review, or detect toxic behaviors in an online game

```
print('Scores for the word \'Like\'')
print('Positive: ', swn.senti_synset('like.v.02').pos_score())
print('Negative: ', swn.senti_synset('like.v.02').neg_score())
print('Objective: ', swn.senti_synset('like.v.02').obj_score())
```

```
Scores for the word 'Like'
positive:  1.0
negative:  0.0
objective:  0.0
```

```
print('Scores for the word \'Fear\'')
print('Positive: ', swn.senti_synset('fear.v.01').pos_score())
```

```
print('Negative: ', swn.senti_synset('fear.v.01').neg_score())
print('Objective: ', swn.senti_synset('fear.v.01').obj_score())
```

```
Scores for the word "Fear"
Positive:  0.125
Negative:  0.625
Objective:  0.25
```

SentiWordNet in Sentences

```
#writing sample sentences for SentiWordNet
sentiSentence1 = 'I enjoy drinking warm tea to calm myself down'
sentiSentence2 = 'My teammate is being so unhelpful'


def wordsInSentencePolarity(phrase):
  word = phrase.split()
  print(phrase)
  for w in word:
      synsets = list(swn.senti_synsets(w))
      if synsets: #if synsets exist for the word, choose the first one
          print(w)
          print("Positive : ", synsets[0].pos_score())
          print("Negative : ", synsets[0].neg_score())
          print("Objective: ", synsets[0].obj_score(), '\n')


#Output the polarity for each word in the first sentence
wordsInSentencePolarity(sentiSentence1)
```

```
I enjoy drinking warm tea to calm myself down
I
Positive :  0.0
Negative :  0.0
Objective:  1.0

enjoy
Positive :  0.375
Negative :  0.0
Objective:  0.625

drinking
Positive :  0.0
Negative :  0.0
Objective:  1.0

warm
Positive :  0.25
Negative :  0.0
Objective:  0.75

tea
Positive :  0.0
Negative :  0.0
Objective:  1.0

calm
Positive :  0.375
Negative :  0.0
Objective:  0.625
```

```
down
Positive :  0.125
Negative :  0.0
Objective:  0.875
```

The first sentence, "I enjoy drinking warm tea to calm myself down" gave mostly objective, but positive polarity scores and no negative polarity scores in the words. An application I can think of would be that I would get an average score for each to determine the polarity of a sentence.

```
#Output the polarity for each word in the second sentence
wordsInSentencePolarity(sentiSentence2)
```

```
My teammate is being so unhelpful
teammate
Positive :  0.0
Negative :  0.0
Objective:  1.0

is
Positive :  0.25
Negative :  0.125
Objective:  0.625

being
Positive :  0.0
Negative :  0.0
Objective:  1.0

so
Positive :  0.0
Negative :  0.0
Objective:  1.0

unhelpful
Positive :  0.125
Negative :  0.25
Objective:  0.625
```

The second sentence, "My teammate is being so unhelpful " was intended to be a negative sentence, but the positive polarity sums up to 0.375 and negative polarity also sums up to 0.375, which makes this sentence not particularly positive or negative, but rather objective.

Which makes me think, maybe this sentence isn't necessarily negative. SentiWordNet is pretty accurate in word polarity, but I think there is a gap between how computers understand these sentences from humans because we are also able to read context and infer that the speaker of this sentence would be upset about the situation.

In NLP, this techniques can definitely be used in detecting and capturing different sentiences online when a mass-processing is needed. I would trust this to label sentences correctly so I don't have to read through everything and label them myself.

Collocation

Collocations are words that are put together to mean something else. 'pay attention' is a common word, where both words mean something completely different from when they are put together.

```
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
```

```
#I will choose 'United States' as a collocation to calculate mutual information
text4words = ' '.join(text4.tokens)
d = len(set(text4))

#set the P info / how often is this word shown in text4
US = text4words.count('United States')/d
U = text4words.count('United')/d
S = text4words.count('States')/d

#mutual information = log( P(x,y) / (P(x)*P(y)) )
MI = math.log2(US/(U*S))

#display results
print("P(United States): ", US)
print("P(United): ", U)
print("P(States): ", S)
print("Mutual Information: ", MI)
```

```
P(United States):  0.015860349127182045
P(United):  0.0170573566084788
P(States):  0.03301745635910224
Mutual Information:  4.815657649820885
```

The word United and States both appear pretty often, at 0.01705 and 0.03302 respectively. The probability of "United States" appearing is at 0.01586 which is barely below the chance of seeing the word "United", at only 7% difference between them. The Mutual Information score is really high due to this, indicating that the word "United States" is likely to be a collocation.

If the Mutual Information scores are closer to 0, it would mean that the words are less likely of being a collocation, and if the MI score is negative then with high significance we can assume that the word is not a collocation.