1

WordNet

Wordnet is a hierarchical organization of nouns, verbs, adjectives and adverbs. This was developed at Princeton. We can use it to find synset, definitions, example and relation to other words (hypernym, hyponym, meronym, holonym, troponym etc.)

2

In [199…
```
import math

import sympy
# choosing a noun to work with
from nltk.corpus import wordnet
wordnet.synsets('way')
```

Out[199]:
```
[Synset('manner.n.01'),
 Synset('means.n.01'),
 Synset('direction.n.01'),
 Synset('way.n.04'),
 Synset('way.n.05'),
 Synset('way.n.06'),
 Synset('way.n.07'),
 Synset('room.n.02'),
 Synset('way.n.09'),
 Synset('way.n.10'),
 Synset('way.n.11'),
 Synset('way.n.12'),
 Synset('way.r.01')]
```

3

In [200…
```
# extracting definition from one of the synsets
wordnet.synset('direction.n.01').definition()
```

Out[200]:
```
'a line leading to a place or point'
```

In [201…
```
# getting example
wordnet.synset('direction.n.01').examples()
```

Out[201]:
```
['he looked the other direction', "didn't know the way home"]
```

In [202…
```
# getting lemma
wordnet.synset('direction.n.01').lemmas()
```

Out[202]:
```
[Lemma('direction.n.01.direction'), Lemma('direction.n.01.way')]
```

In [203…
```
# traversing the hierarchy
word_hyp = wordnet.synset('direction.n.01')
hyper = lambda s: s.hypernyms()
list(word_hyp.closure(hyper))
```

Out[203]:
```
[Synset('path.n.03'),
 Synset('line.n.11'),
 Synset('location.n.01'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

Observation

In the output we can see that the further up the hierarchy we go, the word gets more generalized For example, lions->wildcats->carnivores->mammals->animals and so on. Which makes sense as the further up the hypernyms of a word we go we get to see more generalized word for the given word

4

In [204…
```
wordnet.synset('direction.n.01').hypernyms()
```

Out[204]:
```
[Synset('path.n.03')]
```

In [205…
```
wordnet.synset('direction.n.01').hyponyms()
```

Out[205]:
```
[Synset('bearing.n.02'),
 Synset('course.n.03'),
 Synset('east-west_direction.n.01'),
 Synset('north-south_direction.n.01'),
 Synset('qibla.n.01'),
 Synset('tendency.n.04')]
```

In [206…
```
wordnet.synset('direction.n.01').part_meronyms()
```

Out[206]:
```
[]
```

In [207…
```
wordnet.synset('direction.n.01').part_holonyms()
```

Out[207]:
```
[]
```

In [208…
```
antonyms = []
lemmatized = wordnet.synset('direction.n.01').lemmas()
for x in lemmatized:
    if x.antonyms():
        antonyms.append(x.antonyms()[0].name())

print(antonyms)
```

```
[]
```

5

In [209…
```
# synset of verb
wordnet.synsets('study')
```

Out[209]:
```
[Synset('survey.n.01'),
 Synset('study.n.02'),
 Synset('report.n.01'),
 Synset('study.n.04'),
 Synset('study.n.05'),
 Synset('discipline.n.01'),
 Synset('sketch.n.01'),
 Synset('cogitation.n.02'),
 Synset('study.n.09'),
 Synset('study.n.10'),
 Synset('analyze.v.01'),
 Synset('study.v.02'),
 Synset('study.v.03'),
 Synset('learn.v.04'),
 Synset('study.v.05'),
 Synset('study.v.06')]
```

6

In [210…
```
# extracting definition from one of the synsets
wordnet.synset('analyze.v.01').definition()
```

Out[210]:
```
'consider in detail and subject to an analysis in order to discover essential features or meaning'
```

In [211…
```
# getting example
wordnet.synset('analyze.v.01').examples()
```

Out[211]:
```
['analyze a sonnet by Shakespeare',
 'analyze the evidence in a criminal trial',
 'analyze your real motives']
```

In [212…
```
# getting lemma
wordnet.synset('analyze.v.01').lemmas()
```

Out[212]:
```
[Lemma('analyze.v.01.analyze'),
 Lemma('analyze.v.01.analyse'),
 Lemma('analyze.v.01.study'),
 Lemma('analyze.v.01.examine'),
 Lemma('analyze.v.01.canvass'),
 Lemma('analyze.v.01.canvas')]
```

In [213…
```
# traversing the hierarchy
word_hyp = wordnet.synset('analyze.v.01')
hyper = lambda s: s.hypernyms()
list(word_hyp.closure(hyper))
```

Out[213]:
```
[]
```

Observation

In the output we don't have any hierarchical data to make assumptions with. Usually the hierarchy for example looks like, lions->wildcats->carnivores->mammals->animals and so on. From the observation we can see that not all verbs will have hypernyms

7

In [214…
```
# using morphy to find NOUN, VERB, ADJ and ADV
wordnet.morphy('analyze', wordnet.NOUN)
```

In [215…
```
wordnet.morphy('analyze', wordnet.VERB)
```

Out[215]:
```
'analyze'
```

In [216…
```
wordnet.morphy('analyze', wordnet.ADJ)
```

In [217…
```
wordnet.morphy('analyze', wordnet.ADV)
```

8

In [218…
```
# finding two synsets from two words to compare later
wordnet.synsets('bus')
```

Out[218]:
```
[Synset('bus.n.01'),
 Synset('bus_topology.n.01'),
 Synset('busbar.n.01'),
 Synset('bus.n.04'),
 Synset('bus.v.01'),
 Synset('bus.v.02'),
 Synset('bus.v.03')]
```

In [219…
```
wordnet.synsets('ship')
```

Out[219]:
```
[Synset('ship.n.01'),
 Synset('transport.v.04'),
 Synset('ship.v.02'),
 Synset('embark.v.01'),
 Synset('ship.v.04'),
 Synset('ship.v.05')]
```

In [220…
```
# finding similarity using Wu-Palmer

bus = wordnet.synset('bus.n.01')
ship = wordnet.synset('ship.n.01')
wordnet.wup_similarity(bus, ship)
```

Out[220]:
```
0.7
```

In [221…
```
from nltk.wsd import lesk
from nltk import word_tokenize

# applying the Lesk algorithm
bus_sentence = word_tokenize(wordnet.synset('bus.n.01').examples()[0])
bus_sentence_string =' '.join(bus_sentence)
print(f'Given sentence: {bus_sentence_string}')
print(lesk(bus_sentence, 'bus', 'n'))

ship_sentence = ['they', 'booked', 'a', 'ticket', 'on', 'the', 'cruise', 'ship']
ship_sentence_string =' '.join(ship_sentence)
print(f'Given sentence: {ship_sentence_string}')
print(lesk(ship_sentence, 'ship', 'n'))
```

```
Given sentence: he always rode the bus to work
Synset('bus_topology.n.01')
Given sentence: they booked a ticket on the cruise ship
Synset('ship.n.01')
```

Observation

In the output for 'bus' we see that it is not the correct meaning in regard to the context, here it assumes we mean bus as a topology instead of the vehicle In the output for 'ship' we see that it is the correct meaning in regard to the context, where it means the ship as a noun From the observation we can see that the Lesk algorithm is not 100% right but is usually able to get the meaning from context given it has the information in its corpus

9

SentiWordNet

This functionality is built on top of WordNet, it can be used to do sentiment analysis. It gives a rating of positivity, negativity and objectivity to the given sentence. This can be used to assess user input and what they are trying to express by it. It becomes specially helpful for voice assistants and AI in general.

In [222…
```
from nltk.corpus import sentiwordnet

# 'compassion.n.01'
# selecting an emotionally charged word
wordnet.synsets('affection')
affection = sentiwordnet.senti_synset('affection.n.01')
print(affection)
```

```
<affection.n.01: PosScore=0.625 NegScore=0.25>
```

In [223…
```
senti_sent = "I enjoy spending time with my family".split()
#senti_sent = "I enjoy spending time family".split()
for word in senti_sent:
    syn_list = list(sentiwordnet.senti_synsets(word))
    if len(syn_list) > 0 :
        print(f'Polarity of \"{word}\" : {syn_list[0]}')
    else:
        print(f'Sorry polarity of \"{word}\" can not be determined as there is no senti_synset for it')
```

```
Polarity of "I" : <iodine.n.01: PosScore=0.0 NegScore=0.0>
Polarity of "enjoy" : <enjoy.v.01: PosScore=0.375 NegScore=0.0>
Polarity of "spending" : <spending.n.01: PosScore=0.0 NegScore=0.0>
Polarity of "time" : <time.n.01: PosScore=0.0 NegScore=0.0>
Sorry polarity of "with" can not be determined as there is no senti_synset for it
Sorry polarity of "my" can not be determined as there is no senti_synset for it
Polarity of "family" : <family.n.01: PosScore=0.0 NegScore=0.0>
```

Observation

In the output we can see that polarity is not available for all words as they don't exist in the senti_synset. Also, it may give wrong answers as it makes assumptions, here "I" was to define a person/myself but the function assumed it meant Iodine But for all emotionally charged words like "enjoy" we can get the polarity fairly accurately. As we mentioned before, this can be used in AI, voice assistants and more. If I was talking to an AI and I said this sentence to it then I'd expect it to reply in a positive manner. If I said "I am feeling down" then it should reply in an encouraging manner. The AI would only be able to understand they proper way to respond if it is able to use sentiment analysis well.

10

Collocation

We have many common set of words that are highly likely to be found right next to each other. Some general example can be, "social media", "desktop computer", "artificial intelligence" Having this knowledge we are able to fill in what word might come next.

In [224…

```
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
```

In [225…

```
from collections import Counter
```

```
from nltk import bigrams
import math

# Calculating mutual information of "years ago"
count_bigram = 0
bi_grams = list(bigrams(text4))
for freq in bi_grams:
    if freq[0] == 'years' and freq[1] == 'ago':
        count_bigram = count_bigram + 1

count_one = 0
count_two = 0
for freq in text4:
    if freq == 'years':
        count_one = count_one + 1
    if freq == 'ago':
        count_two = count_two + 1


print(f'\"years ago\" appears: {count_bigram} times')
print(f'tokens in \"text4\" with no preprocessing: {len(text4)}')
print(f'\"years\" appears: {count_one} times')
print(f'\"ago\" appears: {count_two} times')

print('Formula for PMI is :')
print('\tlog2((P(x,y))/(P(x) * P(y))')
print(f'\tlog2({count_bigram}/({len(text4)}-1))/(({count_one}/{len(text4)}) * ({count_two}/{len(text4)}))')
pxy = count_bigram/(len(text4)-1)
px = count_one/len(text4)
py = count_two/len(text4)
print(f'\tlog2({pxy})/(({px}) * ({py}))')
pxpy = px * py
print(f'\tlog2({pxy})/({pxpy})')
div = pxy / pxpy
print(f'\tlog2({div})')
ans = math.log2(div)
ans = "{:.2f}".format(ans)
print(f'Answer is : {ans}')
if float(ans)>0:
    print('It is likely to be a collocation')
if float(ans)==0:
    print('It is likely both words are independent')
```

```
if float(ans)<0:
    print('It is NOT likely to be a collocation')
```

```
"years ago" appears: 27 times
tokens in "text4" with no preprocessing: 152901
"years" appears: 143 times
"ago" appears: 44 times
Formula for PMI is :
        log2((P(x,y))/(P(x) * P(y))
        log2(27/(152901-1))/((143/152901) * (44/152901))
        log2(0.00017658600392413343)/((0.0009352456818464235) * (0.00028776790210659187))
        log2(0.00017658600392413343)/(2.6913368781919437e-07)
        log2(656.1274634737103)
Answer is : 9.36
It is likely to be a collocation
```

Observation

The bi-gram chosen was happening often in the text, so it is likely to be a collocation. As we calculated the PMI to be a positive number it is very likely to be a collocation The collocation was done on text4 which is the Inaugural corpus. Just from the context we can say that this bi-gram is likely to happen, but now we have the data to prove it.