wordnet

February 25, 2023

1 Portfolio Assignment: WordNet

Linus Fackler

1.1 WordNet

WordNet is a lexical database that organizes English words based on their meanings and relationships with other words. It groups words into sets of synonyms called synsets, and each synset is linked to other synsets through semantic relationships such as hypernymy (broader term) and hyponymy (narrower term). WordNet has many applications in natural language processing and computational linguistics.

1.1.1 Imports necessary

[ ]:
```
from nltk.corpus import wordnet as wn
```

1.1.2 Synsets of noun 'car'

[ ]:
```
wn.synsets('car')
```

[ ]:
```
[Synset('car.n.01'),
 Synset('car.n.02'),
 Synset('car.n.03'),
 Synset('car.n.04'),
 Synset('cable_car.n.01')]
```

1.1.3 Selecting a synset

Definition

[ ]:
```
car = wn.synset('car.n.01')
car.definition()
```

[ ]:
```
'a motor vehicle with four wheels; usually propelled by an internal combustion
engine'
```

Examples

[ ]:
```
car.examples()
```

[ ]:
```
['he needs a car to get to work']
```

Lemmas

[ ]:
```
car.lemmas()
```

[ ]:
```
[Lemma('car.n.01.car'),
 Lemma('car.n.01.auto'),
 Lemma('car.n.01.automobile'),
 Lemma('car.n.01.machine'),
 Lemma('car.n.01.motorcar')]
```

Traversing up the WordNet hierarchy

[ ]:
```
print("root hypernym:", car.root_hypernyms())

print("\n10 highest nouns in hierarchy:")
i = 1
for synset in list(wn.all_synsets('n')):
    print(synset)
    if i > 10:
        break
    i += 1



# car_synsets = wn.synsets('car', pos=wn.NOUN)
# for sense in car_synsets:
#     lemmas = [l.name() for l in sense.lemmas()]
#     print("Synset: " + sense.name() + "(" + sense.definition() + ") \n\t␣
 ↪Lemmas:" + str(lemmas))
```

```
root hypernym: [Synset('entity.n.01')]

10 highest nouns in hierarchy:
Synset('entity.n.01')
Synset('physical_entity.n.01')
Synset('abstraction.n.06')
Synset('thing.n.12')
Synset('object.n.01')
Synset('whole.n.02')
Synset('congener.n.03')
Synset('living_thing.n.01')
Synset('organism.n.01')
Synset('benthos.n.02')
Synset('dwarf.n.03')
```

Here we see that “entity” is at the top of the noun hierarchy. We traverse from car up to entity in the hierarchy.

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

```
Synset('motor_vehicle.n.01')
Synset('self-propelled_vehicle.n.01')
Synset('wheeled_vehicle.n.01')
Synset('container.n.01')
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')
```

WordNet organizes their nouns into hierarchies based on the hypernymy/hyponymy relation between synsets. The more we got up in the hierarchy, the more "general" the nouns get. "Activity" is a hypernym of "car", "act" a hypernym of "activity".

Hypernyms

[ ]:
```
hyper = lambda s: s.hypernyms()
list(car.closure(hyper))
```

[ ]:
```
[Synset('motor_vehicle.n.01'),
 Synset('self-propelled_vehicle.n.01'),
 Synset('wheeled_vehicle.n.01'),
 Synset('container.n.01'),
 Synset('vehicle.n.01'),
 Synset('instrumentality.n.03'),
 Synset('conveyance.n.03'),
 Synset('artifact.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

Hyponyms

[ ]:
```
hypo = lambda s: s.hyponyms()
list(car.closure(hypo))
```

[ ]:
```
[Synset('ambulance.n.01'),
 Synset('beach_wagon.n.01'),
 Synset('bus.n.04'),
 Synset('cab.n.03'),
 Synset('compact.n.03'),
 Synset('convertible.n.01'),
 Synset('coupe.n.01'),
 Synset('cruiser.n.01'),
 Synset('electric.n.01'),
 Synset('gas_guzzler.n.01'),
 Synset('hardtop.n.01'),
 Synset('hatchback.n.01'),
 Synset('horseless_carriage.n.01'),
 Synset('hot_rod.n.01'),
 Synset('jeep.n.01'),
 Synset('limousine.n.01'),
 Synset('loaner.n.02'),
 Synset('minicar.n.01'),
 Synset('minivan.n.01'),
 Synset('model_t.n.01'),
 Synset('pace_car.n.01'),
 Synset('racer.n.02'),
 Synset('roadster.n.01'),
 Synset('sedan.n.01'),
 Synset('sport_utility.n.01'),
 Synset('sports_car.n.01'),
 Synset('stanley_steamer.n.01'),
 Synset('stock_car.n.01'),
 Synset('subcompact.n.01'),
 Synset('touring_car.n.01'),
 Synset('used-car.n.01'),
 Synset('funny_wagon.n.01'),
 Synset('shooting_brake.n.01'),
 Synset('gypsy_cab.n.01'),
 Synset('minicab.n.01'),
 Synset('panda_car.n.01'),
 Synset('berlin.n.03'),
 Synset('minicab.n.01'),
 Synset('finisher.n.05'),
 Synset('stock_car.n.02'),
 Synset('brougham.n.02')]
```

Meronyms

[ ]:
```
mero = lambda s: s.part_meronyms()
list(car.closure(mero))
```

```
[ ]: [Synset('accelerator.n.01'),
      Synset('air_bag.n.01'),
      Synset('auto_accessory.n.01'),
      Synset('automobile_engine.n.01'),
      Synset('automobile_horn.n.01'),
      Synset('buffer.n.06'),
      Synset('bumper.n.02'),
      Synset('car_door.n.01'),
      Synset('car_mirror.n.01'),
      Synset('car_seat.n.01'),
      Synset('car_window.n.01'),
      Synset('fender.n.01'),
      Synset('first_gear.n.01'),
      Synset('floorboard.n.02'),
      Synset('gasoline_engine.n.01'),
      Synset('glove_compartment.n.01'),
      Synset('grille.n.02'),
      Synset('high_gear.n.01'),
      Synset('hood.n.09'),
      Synset('luggage_compartment.n.01'),
      Synset('rear_window.n.01'),
      Synset('reverse.n.02'),
      Synset('roof.n.02'),
      Synset('running_board.n.01'),
      Synset('stabilizer_bar.n.01'),
      Synset('sunroof.n.01'),
      Synset('tail_fin.n.02'),
      Synset('third_gear.n.01'),
      Synset('window.n.02'),
      Synset('exhaust.n.02'),
      Synset('horn_button.n.01'),
      Synset('bumper_guard.n.01'),
      Synset('armrest.n.01'),
      Synset('doorlock.n.01'),
      Synset('hinge.n.01'),
      Synset('back.n.08'),
      Synset('headrest.n.01'),
      Synset('seat_belt.n.01'),
      Synset('inlet_manifold.n.01'),
      Synset('hood_ornament.n.01'),
      Synset('exhaust_manifold.n.01'),
      Synset('exhaust_pipe.n.01'),
      Synset('exhaust_valve.n.01'),
      Synset('silencer.n.02'),
      Synset('tailpipe.n.01'),
      Synset('pintle.n.01')]
```

Holonyms

[ ]:
```
holo = lambda s: s.part_holonyms()
list(car.closure(holo))
```

[ ]:
```
[]
```

Antonym

[ ]:
```
antonyms = []

for l in car.lemmas():
    if l.antonyms():
        antonyms.append(l.antonyms(0).name())

print(antonyms)
```

```
[]
```

1.1.4 Synsets of verb 'drive'

[ ]:
```
wn.synsets('drive')
```

[ ]:
```
[Synset('drive.n.01'),
 Synset('drive.n.02'),
 Synset('campaign.n.02'),
 Synset('driveway.n.01'),
 Synset('drive.n.05'),
 Synset('drive.n.06'),
 Synset('drive.n.07'),
 Synset('drive.n.08'),
 Synset('drive.n.09'),
 Synset('drive.n.10'),
 Synset('drive.n.11'),
 Synset('drive.n.12'),
 Synset('drive.v.01'),
 Synset('drive.v.02'),
 Synset('drive.v.03'),
 Synset('force.v.06'),
 Synset('drive.v.05'),
 Synset('repel.v.01'),
 Synset('drive.v.07'),
 Synset('drive.v.08'),
 Synset('drive.v.09'),
 Synset('tug.v.02'),
 Synset('drive.v.11'),
 Synset('drive.v.12'),
 Synset('drive.v.13'),
```

```
 Synset('drive.v.14'),
 Synset('drive.v.15'),
 Synset('drive.v.16'),
 Synset('drive.v.17'),
 Synset('drive.v.18'),
 Synset('drive.v.19'),
 Synset('drive.v.20'),
 Synset('drive.v.21'),
 Synset('drive.v.22')]
```

1.1.5 Selecting a synset

Definition

[ ]:
```
drive = wn.synset('drive.v.01')
drive.definition()
```

[ ]:
```
'operate or control a vehicle'
```

Examples

[ ]:
```
drive.examples()
```

[ ]:
```
['drive a car or bus', 'Can you drive this four-wheel truck?']
```

Lemmas

[ ]:
```
drive.lemmas()
```

[ ]:
```
[Lemma('drive.v.01.drive')]
```

Traversing up the WordNet hierarchy

[ ]:
```
print("root hypernym:", drive.root_hypernyms())

print("\n10 highest verbs in hierarchy:")
i = 1
for synset in list(wn.all_synsets('v')):
    print(synset)
    if i > 10:
        break
    i += 1
```

```
root hypernym: [Synset('touch.v.01')]

10 highest verbs in hierarchy:
Synset('breathe.v.01')
Synset('respire.v.02')
Synset('respire.v.01')
```

```
Synset('choke.v.01')
Synset('hyperventilate.v.02')
Synset('hyperventilate.v.01')
Synset('aspirate.v.03')
Synset('burp.v.01')
Synset('force_out.v.08')
Synset('hiccup.v.01')
Synset('sigh.v.01')
```

As we can see here, verbs are different than nouns when it comes to their hierarchy. There is not just one verb that covers all other verbs, basically is the root hypernym for everything.

[ ]:
```
hyp = drive
top = wn.synset('touch.v.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Synset('drive.v.01')
Synset('operate.v.03')
Synset('manipulate.v.02')
Synset('handle.v.04')
Synset('touch.v.01')
```

Here, it doesn't go much higher in the hierarchy after "drive". It ends on a different verb than most other vers do. This is completely different than for nouns, where every noun eventually reaches "entity".

1.1.6 Morphy for "drive"

Base

[ ]:
```
print("Base form of 'drive': ", wn.morphy('drive'))
```

```
Base form of 'drive':  drive
```

Verb

[ ]:
```
print("Verb form of 'drive': ", wn.morphy('drive', wn.VERB))
```

```
Verb form of 'drive':  drive
```

Adjective

[ ]:
```
print("Adjective form of 'drive': ", wn.morphy('drive', wn.ADJ))
```

```
Adjective form of 'drive':  None
```

Noun

[ ]:
```
print("Base form of 'drive': ", wn.morphy('drive', wn.NOUN))
```

```
Base form of 'drive':  drive
```

1.2 Word similarity

Imports necessary

[ ]:
```
from nltk.wsd import lesk
from nltk import word_tokenize as tokenize
```

Selecting 2 similar words

[ ]:
```
# example words
word1 = 'drive'
word2 = 'ride'

# example sentences
sent1 = 'Do you want to drive with me?'
sent2 = 'Do you want to ride with me?'

# tokenizing example sentences
sent1_tok = tokenize(sent1)
sent2_tok = tokenize(sent2)
```

Lesk algorithm

[ ]:
```
synset1 = lesk(sent1_tok, word1)
print("Sentence 1: ", sent1_tok)
print("Lesk' selected Synset:", synset1)

synset2 = lesk(sent2_tok, word2)
print("\nSentence 2: ", sent2_tok)
print("Lesk' selected Synset:", synset2)
```

```
Sentence 1:  ['Do', 'you', 'want', 'to', 'drive', 'with', 'me', '?']
Lesk' selected Synset: Synset('drive.v.09')

Sentence 2:  ['Do', 'you', 'want', 'to', 'ride', 'with', 'me', '?']
Lesk' selected Synset: Synset('ride.n.02')
```

Wu-Palmer similarity

[ ]:
```
wn.wup_similarity(synset1, synset2)
```

[ ]:
```
0.125
```

Although I would have thought the opposite, 'drive' and 'ride' seem to be relatively dissimilar and only moderately related in the semantic hierarchy. They share some common characteristics/associations though, but they're not as closely related. This might be because 'ride' can have

different meanings and I should have selected the words based on their description, rather than just the word itself, to get a better result

1.3 SentiWordNet

Functionality SemiWordNet is an extension of WordNet that measures the strength and direction of semantic relationships between words, with information on uncertainty. It enhances WordNet by providing information about the degree of semantic relatedness between words.

Possible use cases It helps with natural language processing tasks and is useful for researchers and developers. Specifically, if accuracte measures of semantic similarity are required. These include information retrieval (improving accuracy of search engines), text classification, sentiment analysis, and machine translation.

1.3.1 Example

Imports

[ ]:
```
from nltk.corpus import sentiwordnet as swn
```

[ ]:
```
# selecting emotionally charged word
emo_word = 'excitement'

senti_syn = list(swn.senti_synsets(emo_word))
```

Polarity scores for each word

[ ]:
```
for syn in senti_syn:
    print(syn)
    print("Positive score = ", syn.pos_score())
    print("Negative score = ", syn.neg_score())
    print("Objective score = ", syn.obj_score())
```

```
<exhilaration.n.01: PosScore=0.5 NegScore=0.5>
Positive score =  0.5
Negative score =  0.5
Objective score =  0.0
<excitement.n.02: PosScore=0.375 NegScore=0.25>
Positive score =  0.375
Negative score =  0.25
Objective score =  0.375
<excitation.n.03: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
<agitation.n.04: PosScore=0.0 NegScore=0.0>
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0
```

Example sentence

```
[ ]: # example sentence
emo_sentence = "Surrounded by love and beauty, she found true happiness."

emo_sentence_tok = tokenize(emo_sentence)
emo_sentence_tok = [t for t in emo_sentence_tok if t.isalpha()]
# punctuation is unnecessary for this example
```

Polarity scores for each word

```
[ ]: for t in emo_sentence_tok:
    print(t)
    syn = list(swn.senti_synsets(t))
    if (syn != []):
        print("Positive score = ", syn[0].pos_score())
        print("Negative score = ", syn[0].neg_score())
        print("Objective score = ", syn[0].obj_score())
    else:
        print("No score available")
    print()
```

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

by
Positive score =  0.0
Negative score =  0.0
Objective score =  1.0

love
Positive score =  0.625
Negative score =  0.0
Objective score =  0.375

and
No score available

beauty
Positive score =  0.5
Negative score =  0.0
Objective score =  0.5

she
No score available

found
```

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

true
Positive score =  0.5
Negative score =  0.0
Objective score =  0.5

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

Some scores that seem like they would be 100% positive, like 'beauty', are not. This might be because the context can affect these words specifically a lot. 'Love', for example, is mostly positive. 'Surrounded' is neutral for obvious reasons. 'Happiness' is completely positive, as even context can't completely change the meaning of the word.

Knowing these scores is crucial for technologies like chatbots, for example. Sentences and words' objectiveness is different in every scenario and affects the overall meaning of what a person wants the machine to understand.

1.4 Collocation

1.4.1 What is Collocation?

Collocation refers to the frequent appearance of certain words together in a language, indicating a strong tendency for those words to be used in a particular combination. Collocation can help language learners understand how words are commonly used together and how to use them in context. It can also aid in natural language processing tasks, such as machine translation and text generation, by helping to ensure that words are used in a way that is consistent with the way they are used in natural language.

1.4.2 Example text

Imports

[ ]:
```
from nltk.book import text4
import math
```

Text 4 Collocations

[ ]:
```
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
```

Select collocation

```
[ ]: coll = text4.collocation_list()[4]
     coll
```

```
[ ]: ('Federal', 'Government')
```

Calculate mutual information

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

     vocab = len(set(text4))
     hg = text.count(coll[0] + " " + coll[1]) / vocab
     print("p(" + coll[0] + " " + coll[1] + ") = ", hg)

     h = text.count(coll[0]) / vocab
     print("p(" + coll[0] + ") = ", h)

     g = text.count(coll[1]) / vocab
     print("p(" + coll[1] + ") = ", g)

     pmi = math.log2(hg / (h * g))
     print("pmi = ", pmi)
```

```
p(Federal Government) =  0.0031920199501246885
p(Federal) =  0.006483790523690773
p(Government) =  0.03371571072319202
pmi =  3.868067366919006
```

The results show that those 2 words tend to be of relatively high collocation. Meaning they will appear together more likely than expected when randomly throwing words together.

Example run 2 - collocation Just to be able to compare this to another result, I will run this again on a different set of words.

```
[ ]: coll2 = text4.collocation_list()[7]
     coll2
```

```
[ ]: ('Vice', 'President')
```

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

     vocab = len(set(text4))
     hg = text.count(coll2[0] + " " + coll2[1]) / vocab
     print("p(" + coll2[0] + " " + coll2[1] + ") = ", hg)

     h = text.count(coll2[0]) / vocab
     print("p(" + coll2[0] + ") = ", h)
```

```
g = text.count(coll2[1]) / vocab
print("p(" + coll2[1] + ") = ", g)

pmi = math.log2(hg / (h * g))
print("pmi = ", pmi)
```

```
p(Vice President) =  0.0017955112219451373
p(Vice) =  0.0018952618453865336
p(President) =  0.010773067331670824
pmi =  6.458424602064904
```

Here, the pmi is higher, meaning, those 2 words appear even more frequent together. This seems like obvious information for humans, since 'Vice' and 'President' are heard together often. But for a machine to recognize this information makes NLP and its application even more superior.