

# NLKT WordNet Exploration

**Author:** Andrew Sen

**Date:** 9/25/2022

WordNet is a hierarchical organization of nouns, verbs, adjectives, and adverbs. For each word, WordNet includes information about definitions of the word, synonym sets called synsets, usage examples, and relations to other words.

## Synsets for Nouns

I will use the noun "game" and explore its associated synsets with WordNet. First, I will output all synsets for "game."

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

wn.synsets('game')
```

```
Out[ ]: [Synset('game.n.01'),
          Synset('game.n.02'),
          Synset('game.n.03'),
          Synset('game.n.04'),
          Synset('game.n.05'),
          Synset('game.n.06'),
          Synset('game.n.07'),
          Synset('plot.n.01'),
          Synset('game.n.09'),
          Synset('game.n.10'),
          Synset('game.n.11'),
          Synset('bet_on.v.01'),
          Synset('crippled.s.01'),
          Synset('game.s.02')]
```

Using the first synset in the list, I will output its definition, usage examples, and lemmas.

```
In [ ]: game = wn.synset('game.n.01')
# extracting definitions
print(f'Definition:\n{game.definition()}\n')
# extracting usage examples
print(f'Usage Examples:\n{game.examples()}\n')
# extracting lemmas
print(f'Lemmas:\n{game.lemmas()}\n')
```

Definition:

a contest with rules to determine a winner

Usage Examples:

['you need four people to play this game']

Lemmas:

[Lemma('game.n.01.game')]

We can also use WordNet to traverse up a word's hierarchy of hypernyms.

```
In [ ]: # extracting hypernyms of 'game'
hyp = game.hypernyms()[0]
while hyp:
    print(hyp)
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
    else:
        break
```

Synset('activity.n.01')

Synset('act.n.02')

Synset('event.n.01')

Synset('psychological\_feature.n.01')

Synset('abstraction.n.06')

Synset('entity.n.01')

As we can see with this example, WordNet has a topmost hypernym for nouns in the form of the 'entity' synset. Under this system, all other nouns are a hyponym of 'entity.' As we will see, this is unlike how verbs are organized in WordNet.

## Synsets for Verbs

As we did for the noun 'game,' we will do the same to explore the synsets of the verb 'play.'

```
In [ ]: wn.synsets('play')
```

```

Out[ ]: [Synset('play.n.01'),
        Synset('play.n.02'),
        Synset('play.n.03'),
        Synset('maneuver.n.03'),
        Synset('play.n.05'),
        Synset('play.n.06'),
        Synset('bid.n.02'),
        Synset('play.n.08'),
        Synset('playing_period.n.01'),
        Synset('free_rein.n.01'),
        Synset('shimmer.n.01'),
        Synset('fun.n.02'),
        Synset('looseness.n.05'),
        Synset('play.n.14'),
        Synset('turn.n.03'),
        Synset('gambling.n.01'),
        Synset('play.n.17'),
        Synset('play.v.01'),
        Synset('play.v.02'),
        Synset('play.v.03'),
        Synset('act.v.03'),
        Synset('play.v.05'),
        Synset('play.v.06'),
        Synset('play.v.07'),
        Synset('act.v.05'),
        Synset('play.v.09'),
        Synset('play.v.10'),
        Synset('play.v.11'),
        Synset('play.v.12'),
        Synset('play.v.13'),
        Synset('play.v.14'),
        Synset('play.v.15'),
        Synset('play.v.16'),
        Synset('play.v.17'),
        Synset('play.v.18'),
        Synset('toy.v.02'),
        Synset('play.v.20'),
        Synset('dally.v.04'),
        Synset('play.v.22'),
        Synset('dally.v.01'),
        Synset('play.v.24'),
        Synset('act.v.10'),
        Synset('play.v.26'),
        Synset('bring.v.03'),
        Synset('play.v.28'),
        Synset('play.v.29'),
        Synset('bet.v.02'),
        Synset('play.v.31'),
        Synset('play.v.32'),
        Synset('play.v.33'),
        Synset('meet.v.10'),
        Synset('play.v.35')]

```

I will select the first verb synset in the list and extract its definition, usage examples, and lemmas.

```
In [ ]: play = wn.synset('play.v.01')
# extracting definitions
print(f'Definition:\n{play.definition()}\n')
# extracting usage examples
print(f'Usage Examples:\n{play.examples()}\n')
# extracting Lemmas
print(f'Lemmas:\n{play.lemmas()}\n')
```

Definition:  
participate in games or sport

Usage Examples:  
['We played hockey all afternoon', 'play cards', 'Pele played for the Brazilian teams in many important matches']

Lemmas:  
[Lemma('play.v.01.play')]

We can also traverse the heirarchy of hypernyms for a given verb.

```
In [ ]: # extracting hypernyms of 'play'
hyp = play.hypernyms()[0]
while hyp:
    print(hyp)
    if hyp.hypernyms():
        hyp2 = hyp.hypernyms()[0]
    else:
        break
```

Synset('compete.v.01')

Here we see that the topmost hypernym for 'play' is 'compete.' This cannot be a general hypernym for all verbs. This shows that, unlike with nouns, WordNet does not categorize all verbs as being hyponyms to some universal umbrella verb.

The `morphy()` function returns the base form of a word. We can use it to confirm that certain words are just different forms of the word 'play.'

```
In [ ]: forms = ['played', 'playing', 'plays', 'player']
print("word: base\n-----")
for f in forms:
    print(f'{f}: {wn.morphy(f)}')
```

word: base  
-----  
played: play  
playing: playing  
plays: play  
player: player

# Similarity and Word Sense Disambiguation

WordNet includes the Wu-Palmer algorithm for determining the similarity between two words. NLTK also has an implementation of the Lesk algorithm for determining which definition of a word is being used in a given sentence. To test both, I will use two different forms of the word 'punch.'

First, I will output the definitions of each synset of 'punch' to pick the right synsets to work with.

```
In [ ]: for ss in wn.synsets('punch'):
        print(ss.name() + ': ' + ss.definition())

# Lesk

punch.n.01: (boxing) a blow with the fist
punch.n.02: an iced mixed drink usually containing alcohol and prepared for multiple servings; normally served in a punch bowl
punch.n.03: a tool for making holes or indentations
punch.v.01: deliver a quick blow to
punch.v.02: drive forcibly as if by a punch
punch.v.03: make a hole into or between, as for ease of separation
```

I will use `punch.n.02` and `punch.v.01` for this example. Now let's use Wu-Palmer to determine their similarity.

```
In [ ]: punch_hit = wn.synset('punch.v.01')
        punch_drink = wn.synset('punch.n.02')

        wn.wup_similarity(punch_hit, punch_drink)
```

```
Out[ ]: 0.13333333333333333
```

We see that Wu-Palmer gave the two senses of 'punch' a low similarity score, which is to be expected.

Now let's use Lesk to disambiguate the use of 'punch' in a sentence.

```
In [ ]: from nltk.wsd import lesk

sentence = ['I', 'walked', 'to', 'the', 'table',
            'and', 'grabbed', 'some', 'punch']
print(lesk(sentence, 'punch'))

sentence2 = ['I', 'wanted', 'to', 'punch', 'him']
print(lesk(sentence2, 'punch'))
```

```
Synset('punch.n.02')
Synset('punch.v.02')
```

Here we see that Lesk was able to correctly disambiguate the first sentence. The disambiguation suggested for the second sentence was not what I expected, but the definitions of `punch.v.01` and `punch.v.02` are similar nonetheless.

## SentiWordNet

SentiWordNet is a tool for programmatically determining the sentiment of a piece of text. Given some text, it will assign scores in positivity, negativity, and objectivity.

For this example, I will use the word 'attack.'

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

for ss in swn.senti_synsets('attack'):
    print(ss)
```

```
<attack.n.01: PosScore=0.0 NegScore=0.0>
<attack.n.02: PosScore=0.0 NegScore=0.0>
<fire.n.09: PosScore=0.125 NegScore=0.5>
<approach.n.01: PosScore=0.0 NegScore=0.0>
<attack.n.05: PosScore=0.0 NegScore=0.0>
<attack.n.06: PosScore=0.0 NegScore=0.0>
<attack.n.07: PosScore=0.0 NegScore=0.25>
<attack.n.08: PosScore=0.0 NegScore=0.125>
<attack.n.09: PosScore=0.25 NegScore=0.125>
<attack.v.01: PosScore=0.0 NegScore=0.0>
<attack.v.02: PosScore=0.0 NegScore=0.0>
<attack.v.03: PosScore=0.0 NegScore=0.5>
<assail.v.01: PosScore=0.0 NegScore=0.375>
<attack.v.05: PosScore=0.0 NegScore=0.0>
<attack.v.06: PosScore=0.0 NegScore=0.0>
```

Now I will make up a sentence and find the polarity of each word in the sentence.

```
In [ ]: sentence = 'I really hate drinking iced coffee'
tokens = sentence.split()

for token in tokens:
    ss = lesk(tokens, token) # use Lesk to get best synset
    print(f'{token}: {swn.senti_synset(ss.name())}')
```

```
I: <one.s.01: PosScore=0.0 NegScore=0.25>
really: <very.r.01: PosScore=0.25 NegScore=0.25>
hate: <hate.v.01: PosScore=0.0 NegScore=0.75>
drinking: <drink.n.02: PosScore=0.0 NegScore=0.0>
iced: <ice.v.03: PosScore=0.0 NegScore=0.0>
coffee: <coffee_bean.n.01: PosScore=0.0 NegScore=0.0>
```

We see that the word 'I' interestingly has a slightly negative polarity. The word 'really' has equal positive and negative polarities, which makes sense because it can precede either a positive or negative word. 'Hate' is very negative as to be expected, and the remaining words have no polarity.

In a real NLP application, it would be useful to have these sentiment scores because it gives extra information about the meaning of a text. Sentiment information could be used as extra factors in a given language model.

## Collocations

A collocation is a group of words that, when put together, refer to a particular thing or action, and the same effect is not achieved if one of the words is replaced with a synonym. An example would be the term 'fast food.' 'Fast food' means something very particular, and saying 'quick food' either sounds wrong or is referring to a different concept altogether.

We will list the collocations found in one of NLTK's built-in texts.

```
In [ ]: from nltk.book import 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

I will now calculate the mutual information of the collocation 'Federal Government.'

```
In [ ]: import math

text = ' '.join(text4.tokens)
vocab = len(set(text))
fg = text.count('Federal Government') / vocab
print("p(Federal Government) = ", fg)
f = text.count('Federal') / vocab
print("p(Federal) = ", f)
g = text.count('Government') / vocab
print('p(Government) = ', g)
pmi = math.log2(fg / (f * g))
print('pmi = ', pmi)
```

```
p(Federal Government) = 0.38095238095238093
p(Federal) = 0.7738095238095238
p(Government) = 4.023809523809524
pmi = -3.0309298265318785
```

A negative pmi indicates that 'Federal Government' is not likely to be a collocation in this text. Since NLTK did consider it a collocation, we can assume that NLTK uses some other means of determining whether or not a phrase is a collocation.

