WordNet

February 25, 2023

Justin Hardy | JEH180008 | Dr. Mazidi | CS 4395.001

The purpose of this assignment is to explore the NLTK Corpus WordNet library, and demonstrate both our understanding of recently discussed natural language processing concepts, as well as our basic skills with WordNet and SentiWordNet.

1 What is WordNet?

WordNet is a project started by Princeton University that was designed to hierarchically organize grammar in sentences. It provides short definitions, synonym sets, usage examples, and word relations to all sorts of nouns, verbs, adjectives, and adverbs in the dictionary. As you can imagine, this can be very helpful to those wishing to explore natural language in their projects.

1.1 Exploring Synsets (Noun)

In WordNet, Synsets are Synonym Sets, which can be used to view specific definitions about a particular word. You'll notice that with most words, there are multiple synsets for the word, largely because most words have multiple definitions. These synsets are connected to other synsets semantically, giving a sort-of hierarchy to the word structure.

We'll go ahead and explore a word, its first synset, and uncover various info about its definition, uses, lemmas, and place in the WordNet hierarchy.

[15]:
```
# Import WordNet
from nltk.corpus import wordnet

# Choose a noun to use
noun = "computer"

# Print out all of its synsets
noun_synsets = wordnet.synsets(noun)
print("Chosen Word:", noun)
print("Synsets:", noun_synsets, "\n")

# Select one synset, and output its information.
noun_synset = noun_synsets[0]
for synset in noun_synsets:
    if not str(synset.name()).find(".n.") == -1:
        noun_synset = synset
        break

print("Synset Selected:", noun_synset.name())
print("Definition:", noun_synset.definition())
print("Usage Example:", noun_synset.examples() if not noun_synset.examples() ==␣
  ↪[] else "None available.")
print("Lemmas:", noun_synset.lemmas(), "\n")

# Traverse up the WordNet hierarchy for the selected noun
print("Traversing Up The WordNet Hierarchy:")
current_noun_synset = noun_synset
while not current_noun_synset.hypernyms() == []:
    # Get hypernym synset, and output
    current_noun_synset = current_noun_synset.hypernyms()[0]
    print(current_noun_synset)
```

```
Chosen Word: computer
Synsets: [Synset('computer.n.01'), Synset('calculator.n.01')]

Synset Selected: computer.n.01
Definition: a machine for performing calculations automatically
Usage Example: None available.
Lemmas: [Lemma('computer.n.01.computer'),
Lemma('computer.n.01.computing_machine'),
Lemma('computer.n.01.computing_device'), Lemma('computer.n.01.data_processor'),
Lemma('computer.n.01.electronic_computer'),
Lemma('computer.n.01.information_processing_system')]

Traversing Up The WordNet Hierarchy:
Synset('machine.n.01')
Synset('device.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')
```

From what we can observe, WordNet seems to organize nouns to be derivative of the noun “entity”, as all nouns are going to describe some type of thing, whether it be physical or an abstraction.

1.2 Finding ‘-nyms’ of Synsets

In the previous section (1a), we ended by traversing up the WordNet hierarchy and printing each synset we traverse. How we did this was by grabbing a word’s first hypernym, and doing this continuously until there are no more hypernyms to traverse.

There are various other ‘-nyms’ besides hypernyms.

Hypernyms, as described, give a more generalized definition of what the synset is (what a computer is to a laptop). The opposite of that is a hyponym, which gives a more specific definition of that synset (what a puppy is to a dog). Think like, subsets and supersets in set theory.

Other '-nyms' include Meronyms and Holonyms; the former representing something that is "part of" the synset (what a screen is to a tv), and the latter representing something that is the "whole" of the synset (what pants are to a zipper). And of course, there are Synonyms and Antonyms, which are likely known by you already, assuming you've completed 3rd grade in the past.

Specifically, we'll output all the '-nyms' for our chosen noun synset.

[16]:
```python
# Output Hypernyms
print("Hypernyms:", noun_synset.hypernyms(), "\n")

# Output Hyponyms
print("Hyponyms:", noun_synset.hyponyms(), "\n")

# Output Meronyms
print("Meronyms:", noun_synset.part_meronyms(), "\n")

# Output Holonyms
print("Holonyms:", noun_synset.part_holonyms(), "\n")

# Output Antonyms
print("Antonyms:", noun_synset.lemmas()[0].antonyms())
```

```
Hypernyms: [Synset('machine.n.01')]

Hyponyms: [Synset('analog_computer.n.01'), Synset('digital_computer.n.01'),
Synset('home_computer.n.01'), Synset('node.n.08'),
Synset('number_cruncher.n.02'), Synset('pari-mutuel_machine.n.01'),
Synset('predictor.n.03'), Synset('server.n.03'), Synset('turing_machine.n.01'),
Synset('web_site.n.01')]

Meronyms: [Synset('busbar.n.01'), Synset('cathode-ray_tube.n.01'),
Synset('central_processing_unit.n.01'), Synset('chip.n.07'),
Synset('computer_accessory.n.01'), Synset('computer_circuit.n.01'),
Synset('data_converter.n.01'), Synset('disk_cache.n.01'),
Synset('diskette.n.01'), Synset('hardware.n.03'), Synset('keyboard.n.01'),
Synset('memory.n.04'), Synset('monitor.n.04'), Synset('peripheral.n.01')]

Holonyms: [Synset('platform.n.03')]

Antonyms: []
```

1.3 Exploring Synsets (Verb)

Now, let's explore synsets with a verb of choosing.

[17]:
```python
# Choose a verb to use
verb = "procrastinate"

# Print out all of its synsets
verb_synsets = wordnet.synsets(verb)
print("Chosen Word:", verb)
print("Synsets:", verb_synsets, "\n")

# Select one synset, and output its information.
verb_synset = verb_synsets[0]
for synset in verb_synsets:
    if not str(synset.name()).find(".v.") == -1:
        verb_synset = synset
        break

print("Synset Selected:", verb_synset.name())
print("Definition:", verb_synset.definition())
print("Usage Example:", verb_synset.examples() if not verb_synset.examples() ==␣
 ↪[] else "None available.")
print("Lemmas:", verb_synset.lemmas(), "\n")

# Traverse up the WordNet hierarchy for the selected verb
print("Traversing Up The WordNet Hierarchy:")
current_verb_synset = verb_synset
while not current_verb_synset.hypernyms() == []:
    # Get hypernym synset, and output
    current_verb_synset = current_verb_synset.hypernyms()[0]
    print(current_verb_synset)
```

```
Chosen Word: procrastinate
Synsets: [Synset('procrastinate.v.01'), Synset('procrastinate.v.02')]

Synset Selected: procrastinate.v.01
Definition: postpone doing what one should be doing
Usage Example: ['He did not want to write the letter and procrastinated for
days']
Lemmas: [Lemma('procrastinate.v.01.procrastinate'),
Lemma('procrastinate.v.01.stall'), Lemma('procrastinate.v.01.drag_one's_feet'),
Lemma('procrastinate.v.01.drag_one's_heels'),
Lemma('procrastinate.v.01.shillyshally'), Lemma('procrastinate.v.01.dilly-
dally'), Lemma('procrastinate.v.01.dillydally')]

Traversing Up The WordNet Hierarchy:
Synset('delay.v.02')
Synset('wait.v.02')
Synset('act.v.01')
```

From what we can observe, it's likely that WordNet structures verbs to all be derivative of the verb "act", "change", and other action roots, since verbs simply describe actions.

1.4 Determining Various Forms of the Word

EXPLANATION OF WORD FORMS GOES HERE.

[18]:
```python
# Utilize morphy to find word forms of chosen verb
## Base Form
verb_forms_base = wordnet.morphy(verb)
print("Base Form:", verb_forms_base)

## Verb
verb_forms_noun = wordnet.morphy(verb, wordnet.VERB)
print("Verb Form:", verb_forms_noun)

## Noun
verb_forms_noun = wordnet.morphy(verb, wordnet.NOUN)
print("Noun Form:", verb_forms_noun)

## Adjective
verb_forms_noun = wordnet.morphy(verb, wordnet.ADJ)
print("Adjective Form:", verb_forms_noun)
```

```
Base Form: procrastinate
Verb Form: procrastinate
Noun Form: None
Adjective Form: None
```

1.5 Determining Word Similarity

There exists algorithms that allow us to determine how similar two words are. This is done by calculating a metric that represents their similarity. We'll specifically try the Wu-Palmer algorithm's implementation in WordNet. We'll also check out the Lesk algorithm, and see how it can be used to determine what word synset is being used.

[19]:
```python
# Import lesk algorithm
from nltk import word_tokenize
from nltk.wsd import lesk

# Choose words to compare, and write example sentence
word1 = "forgive"
word1_example = "Please forgive me for my behavior."

word2 = "excuse"
word2_example = "Please excuse me for my behavior."

# Tokenize example sentences
word1_example = word_tokenize(word1_example)
word2_example = word_tokenize(word2_example)
```

```
# Print inputted words
print("Example Sentence 1:", word1_example, "\nTarget Word:", word1, "\n")
print("Example Sentence 2:", word2_example, "\nTarget Word:", word2, "\n")

# Run Lesk algorithm to determine word synset being used
word1_synset = lesk(word1_example, word1)
word2_synset = lesk(word2_example, word2)
print("Word Synsets Selected (Lesk):")
print(word1_synset.name() + ",", word1_synset.definition())
print(word2_synset.name() + ",", word2_synset.definition())
print()

# Print Wu-Palmer similarity metric
print("Wu-Palmer similarity:", wordnet.wup_similarity(word1_synset,␣
 ↪word2_synset))
```

```
Example Sentence 1: ['Please', 'forgive', 'me', 'for', 'my', 'behavior', '.']
Target Word: forgive

Example Sentence 2: ['Please', 'excuse', 'me', 'for', 'my', 'behavior', '.']
Target Word: excuse

Word Synsets Selected (Lesk):
forgive.v.02, absolve from payment
excuse.v.01, accept an excuse for

Wu-Palmer similarity: 0.25
```

Wu-Palmer seems to do decently with determining similarity. While the words I'd chosen; forgive and excuse, are similar, they have different meanings in context, and Wu-Palmer seems to have picked up on that. Lesk seems to be alright at picking which synset to use, but I can imagine it wouldn't do so well at picking up on context in complex sentences.

2 What is SentiWordNet?

SentiWordNet can help us determine the sentiments of a word or sentence. This is basically a measure of how positive, negative, or objective a sentence is. SentiWordNet allows you to create senti-synsets that can be used to output these scores, known as polarity scores, to gauge how emotional a sentence or word is.

[20]:
```
# Import SentiWordNet
from nltk.corpus import sentiwordnet

# Choose a word to evaluate
emotional_word = "overzealous"
print("Word:", emotional_word)
```

```python
# Determine the word's senti-synsets, and output their polarity scores
ew_senti_synsets = list(sentiwordnet.senti_synsets(emotional_word))
for ss in ew_senti_synsets:
    print(ss)
    print("+ Score:", ss.pos_score())
    print("- Score:", ss.neg_score())
    print("o Score:", ss.obj_score())
    print()

# Choose a sentence to evaluate
emotional_sentence = "Where is the hug you usually give me after work?"
print("Sentence:", emotional_sentence)

# Tokenize sentence
emotional_sentence = word_tokenize(emotional_sentence)

# Output polarity for each word in the sentence
## Determine positivity/negativity score
for token in emotional_sentence:
    print("Word:", token)
    ss = list(sentiwordnet.senti_synsets(token))
    print("+ Score:", ss[0].pos_score() if not ss == [] else "None")
    print("- Score:", ss[0].neg_score() if not ss == [] else "None")
    print("o Score:", ss[0].obj_score() if not ss == [] else "None")
```

```
Word: overzealous
<fanatic.s.01: PosScore=0.375 NegScore=0.5>
+ Score: 0.375
- Score: 0.5
o Score: 0.125

Sentence: Where is the hug you usually give me after work?
Word: Where
+ Score: None
- Score: None
o Score: None
Word: is
+ Score: 0.25
- Score: 0.125
o Score: 0.625
Word: the
+ Score: None
- Score: None
o Score: None
Word: hug
+ Score: 0.125
- Score: 0.0
o Score: 0.875
```

```
Word: you
+ Score: None
- Score: None
o Score: None
Word: usually
+ Score: 0.0
- Score: 0.0
o Score: 1.0
Word: give
+ Score: 0.0
- Score: 0.0
o Score: 1.0
Word: me
+ Score: 0.0
- Score: 0.0
o Score: 1.0
Word: after
+ Score: 0.0
- Score: 0.0
o Score: 1.0
Word: work
+ Score: 0.0
- Score: 0.0
o Score: 1.0
Word: ?
+ Score: None
- Score: None
o Score: None
```

3 What is a Collocation?

A collocation describes a phrase that occurs more often than expected by chance, and cannot be substituted with different words to achieve the same meaning. For example, you wouldn't call heavy rain "strong rain".

3.1 Exploring Collocations

NLTK includes a number of large texts in its 'book' library, which we will use to explore collocations. Collocations themselves can be identified via the point-wise mutual information (PMI) metric. It's essentially an all-real measure of how likely something is to be a collocation. NLTK does not include support for this metric, so we will have to calculate it ourselves.

[21]:
```
# Import NLTK Collocation utilities
from nltk.book import text4
import math, random

# Output text4 collocations
```

```
print("Inaugural Corpus Collocations (text4):")
text4.collocations()
print()

# Select one of the collocations
rand_index = random.randint(0, len(text4.collocation_list())-1)
collocation = text4.collocation_list()[rand_index]
print("Collocation:", collocation)

# Calculate Point-Wise Mutual Information (PMI) metric for selected collocation
## As per the formula log( P(x,y) / [P(x) * P(y)] )
text = ' '.join(text4.tokens)
total = len(set(text4))
Pxy = text.count(collocation[0] + " " + collocation[1]) / total
Px = text.count(collocation[0]) / total
Py = text.count(collocation[1]) / total
PMI = math.log2( Pxy / (Px * Py) )
## Print PMI Score
print("Probability of \"" + collocation[0] + " " + collocation[1] + "\":", Pxy)
print("Probability of \"" + collocation[0] + "\":", Px)
print("Probability of \"" + collocation[1] + "\":", Py)
print("PMI =", PMI)
print()
print("PMI NOTE:\n 0 = Independent\n + = likely a collocation\n - = unlikely a␣
↪collocation")
```

```
Inaugural Corpus Collocations (text4):
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

Collocation: ('Vice', 'President')
Probability of "Vice President": 0.0017955112219451373
Probability of "Vice": 0.0018952618453865336
Probability of "President": 0.010773067331670824
PMI = 6.458424602064904

PMI NOTE:
 0 = Independent
 + = likely a collocation
 - = unlikely a collocation
```

Each unique run of this notebook will choose a different collocation to calculate PMI score for. Since we are selecting from a list of collocations, we can expect the PMI score for each of the collocations to be positive. However, plugging in a different phrase that exists in text4 may give different results. Needless to say, the probabilistic nature of the mutual information formula is definitely an interesting application here.