CS 4395 - Assignment 4

Hamna Mustafa

hbm170002

WordNet

Wordnet is a lexical database that provides short definitions of words as well as examples. It consists of verbs, nouns, adverbs and adjectives. It groups words into synonym sets calles 'synsets'. It can help to identify meanings of words as well as their relationships to other words.

```
import nltk
nltk.download("omw-1.4")
nltk.download('wordnet')
from nltk.corpus import wordnet as wn
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

+ Code + Text

Nouns

The code below gets all the synsets of 'promise'.

```
#get all synsets of promise
wn.synsets('promise')
```

```
[Synset('promise.n.01'),
 Synset('promise.n.02'),
 Synset('promise.v.01'),
 Synset('promise.v.02'),
 Synset('predict.v.01'),
 Synset('promise.v.04')]
```

Selecting the second synset and extracting its definition, usage examples and lemmas:

```
print("Definition:")
print(wn.synset('promise.n.02').definition())

print("\nExample:")
print(wn.synset('promise.n.02').examples())
```

```
print("\nLemmas")
print(wn.synset('promise.n.01').lemmas())
```

```
Definition:
grounds for feeling hopeful about the future

Example:
['there is little or no promise that he will recover']

Lemmas
[Lemma('promise.n.01.promise')]
```

Traversing up the hierarchy

We can see that in the case of nouns, WordNet has a top-most noun level of ‘entity’. This means that any word that is classified by wordnet as noun will eventually traverse up to 'entity'.

```
promise = wn.synset('promise.n.02')
hyper = lambda s: s.hypernyms()
list(promise.closure(hyper))
```

```
[Synset('expectation.n.01'),
 Synset('belief.n.01'),
 Synset('content.n.05'),
 Synset('cognition.n.01'),
 Synset('psychological_feature.n.01'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

Outputting the following (or an empty list if none exist): hypernyms, hyponyms, meronyms, holonyms, antonym.

```
current = wn.synset('promise.n.02')
print("Hypernyms:")
print(current.hypernyms())

print("Hyponyms:")
print(current.hyponyms())

print("Meronyms:")
print(current.member_meronyms())
print(current.part_meronyms())

print("Holonyms:")
print(current.member_holonyms())
print(current.part_holonyms())

print("Antonyms:")
print(current.lemmas()[0].antonyms())
```

```
Hypernyms:
[Synset('expectation.n.01')]
Hyponyms:
[Synset('rainbow.n.02')]
Meronyms:
[]
[]
Holonyms:
[]
[]
Antonyms:
[]
```

Verb

The code below gets all synsets of 'dance'

```
#get all synsets of dance
wn.synsets('dance')
```

```
[Synset('dance.n.01'),
 Synset('dance.n.02'),
 Synset('dancing.n.01'),
 Synset('dance.n.04'),
 Synset('dance.v.01'),
 Synset('dance.v.02'),
 Synset('dance.v.03')]
```

Selecting the second synset and extracting its definition, usage examples and lemmas:

```
print("Definition:")
print(wn.synset('dance.v.01').definition())

print("\nExample:")
print(wn.synset('dance.v.01').examples())

print("\nLemmas")
print(wn.synset('dance.v.01').lemmas())
```

```
Definition:
move in a graceful and rhythmical way

Example:
['The young girl danced into the room']

Lemmas
[Lemma('dance.v.01.dance')]
```

Traversing up the hierarchy

Unlike with nouns, verbs don't have a top-level synset that it uniform for all verbs. Thus, traversing a verb can become confusing because it won't necessarily traverse upwards towards the same direction. This can cause a semantic drift.

```
dance = wn.synset('dance.v.01')
hyper = lambda s: s.hypernyms()
list(dance.closure(hyper))
```

```
[Synset('move.v.03')]
```

Using morphy:

```
print(wn.morphy('dance',wn.VERB))
print(wn.morphy('danced',wn.VERB))
print(wn.morphy('dances',wn.VERB))
print(wn.morphy('dancing',wn.VERB))
```

```
dance
dance
dance
dance
```

Similarity

```
friends = wn.synsets('friend')
print(friends)
companions = wn.synsets('companion')
print(companions)
```

```
[Synset('friend.n.01'), Synset('ally.n.02'), Synset('acquaintance.n.03'), Synset
[Synset('companion.n.01'), Synset('companion.n.02'), Synset('companion.n.03'), S
```

```
friend = wn.synset('friend.n.01')
companion = wn.synset('companion.n.01')

print("Wu Palmer similarity:")
wn.wup_similarity(friend, companion)
```

```
Wu Palmer similarity:
0.7058823529411765
```

We can see that the Wu Palmer metric did a good job of figuring out that friend and companion are similar words. Wu Palmer looks at common ancestor words to decide if the words are similar. In this case, it worked pretty well.

```
from nltk.wsd import lesk
# look at the definitions for 'friend'
print("Definitions of 'friend':")
for ss in wn.synsets('friend'):
    print(ss, ss.definition())

sentence1 = ["I","love","you",",","my","friend","."]
print("\nLesk for first sentence")
print(lesk(sentence1, 'friend'))
print("\nDefinitions of 'companion':")
for ss in wn.synsets('companion'):
    print(ss, ss.definition())

print("\nLesk for second sentence")
sentence2 = ["I","love","you",",","my","companion","."]
print(lesk(sentence2, 'companion'))
```

```
Definitions of 'friend':
Synset('friend.n.01') a person you know well and regard with affection and trust
Synset('ally.n.02') an associate who provides cooperation or assistance
Synset('acquaintance.n.03') a person with whom you are acquainted
Synset('supporter.n.01') a person who backs a politician or a team etc.
Synset('friend.n.05') a member of the Religious Society of Friends founded by Ge

Lesk for first sentence
Synset('friend.n.01')

Definitions of 'companion':
Synset('companion.n.01') a friend who is frequently in the company of another
Synset('companion.n.02') a traveler who accompanies you
Synset('companion.n.03') one paid to accompany or assist or live with another
Synset('company.v.01') be a companion to somebody

Lesk for second sentence
Synset('company.v.01')
```

The Lesk algorithm looks at context words and compares them to the words in dictionary glosses in order to figure out what context the word is being used in. As we can see, it was able to figure out what context the word 'friend' was being used in the first sentence. It was also somewhat able to figure out the context of 'companion' in the second sentence.

SentiWordNet

SentiWordNet assigns sentiment scores to synsets. The three sentiments are: positivity, negativity and objectivity. It is built on top of WordNet, hence the name. Each synset gets a score between 0 and 1 for each sentiment category. All the 3 sentiment category scores add up to 1.

```
from nltk.corpus import sentiwordnet as swn
nltk.download('sentiwordnet')
hates = wn.synsets('hate')
print(hates)
```

```
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
[Synset('hate.n.01'), Synset('hate.v.01')]
```

```
hate = swn.senti_synset('hate.n.01')
print(hate)
print("Positive score = ", hate.pos_score())
print("Negative score = ", hate.neg_score())
print("Objective score = ", hate.obj_score())
```

```
<hate.n.01: PosScore=0.125 NegScore=0.375>
Positive score =  0.125
Negative score =  0.375
Objective score =  0.5
```

```
print("Word: hate\nSentiSynsets:")
senti_list = list(swn.senti_synsets('hate'))
for item in senti_list:
    print(item)

print('\nSentence:')
sent = 'I hate my life so much'
print(sent)
print("Polarity:")
neg = 0
pos = 0
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        neg += syn.neg_score()
        pos += syn.pos_score()
```

```
print("neg\tpos counts")
print(neg, '\t', pos)
```

```
Word: hate
SentiSynsets:
<hate.n.01: PosScore=0.125 NegScore=0.375>
<hate.v.01: PosScore=0.0 NegScore=0.75>

Sentence:
I hate my life so much
Polarity:
neg     pos counts
0.5      0.25
```

We can see from the scores that in the case of the word 'hate', SentiWordNet was able to deciper that it would be a negative word and gave it a higher negative score than positive (both as a verb and as a noun). Similary, the sentence was also a negative sentence and SentiWordNet was able to distinguish that. This can be very helpful when coming up for responses for human language to computer situations like the use of Alexa.

Collocation

A collocation is when two or more words join together with a frequency that's higher than expected. In a collocation, synonyms of those words cannot be substituted into the collocation. For example, 'pay attention' is does not have the same meaning as 'pay observation'.

```
import nltk
nltk.download("book")
from nltk.book import *
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
```

```
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ieer.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package nps_chat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/nps_chat.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package ppattach to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ppattach.zip.
[nltk_data]    | Downloading package reuters to /root/nltk_data...
[nltk_data]    | Downloading package senseval to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/senseval.zip.
[nltk_data]    | Downloading package state_union to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/state_union.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/stopwords.zip.
[nltk_data]    | Downloading package swadesh to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/swadesh.zip.
[nltk_data]    | Downloading package timit to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/timit.zip.
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package toolbox to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/toolbox.zip.
[nltk_data]    | Downloading package udhr to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr.zip.
[nltk_data]    | Downloading package udhr2 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr2.zip.
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/unicode_samples.zip.
[nltk_data]    | Downloading package webtext to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/webtext.zip.
```

```
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
```

```
import math
text = ' '.join(text4.tokens)
vocab = len(set(text4))
```

```
pd = text.count('public debt')/vocab
print("Mutual Information:")
print("p(public debt) = ",pd )
p = text.count('public')/vocab
print("p(public) = ", p)
d = text.count('debt')/vocab
print('p(debt) = ', d)
pmi = math.log2(pd / (p * d))
pmi = math.log2(pd / (p * d))
print('pmi = ', pmi)
```

```
Mutual Information:
p(public debt) =  0.001396508728179551
p(public) =  0.03341645885286783
p(debt) =  0.004189526184538653
pmi =  3.318334830163354
```

We can see that the mutual information was calculated by the dividing the probability of the collocation with the probability of the words in it. This shows how much those words appear separately rather than apart. Thus, the higher the mutual information, the more the likelihood that we are dealing with a collocation.