NLP_2022_exercise4_Wordnet

May 15, 2022

1 Practical exercise 4 - experiments with Wordnet

WordNet is a large digital lexicon made by hand. The kernel of WordNet are the so called synsets that can be understood as meanings. Each word belongs to one or more synsets and each synset is made up of one or more words. Semantic relations like hypernymy and hyponymy exist between synsets, not between words! Consequently, there is no such thing like synonymy in Wordnet. If two words are synonymous the will share one or several synsets. It is possible to access Wordnet is via the web interface: http://wordnetweb.princeton.edu/perl/webwn. There we can see e.g. the synsets of a word.

2 1. WordNet in Python

The NLTK package offers some easy methods to access WordNet. Before you use WordNet you have to run once the following code:

[1]:
```
import nltk

nltk.download("wordnet")
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\dell\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[1]: True

How to access synsets:

[2]:
```
from nltk.corpus import wordnet as wn

# get synsets of a word
synsets = wn.synsets("rock")
for s in synsets:
    print(s)
print()

# use synset identifier directly
dog = wn.synset("dog.n.01")
print(dog.hypernyms())
print(dog.hyponyms())
```

```python
print(dog.lemmas())  # ??
```

```
Synset('rock.n.01')
Synset('rock.n.02')
Synset('rock.n.03')
Synset('rock.n.04')
Synset('rock_candy.n.01')
Synset('rock_'n'_roll.n.01')
Synset('rock.n.07')
Synset('rock.v.01')
Synset('rock.v.02')

[Synset('canine.n.02'), Synset('domestic_animal.n.01')]
[Synset('basenji.n.01'), Synset('corgi.n.01'), Synset('cur.n.01'),
Synset('dalmatian.n.02'), Synset('great_pyrenees.n.01'), Synset('griffon.n.02'),
Synset('hunting_dog.n.01'), Synset('lapdog.n.01'), Synset('leonberg.n.01'),
Synset('mexican_hairless.n.01'), Synset('newfoundland.n.01'),
Synset('pooch.n.01'), Synset('poodle.n.01'), Synset('pug.n.01'),
Synset('puppy.n.01'), Synset('spitz.n.01'), Synset('toy_dog.n.01'),
Synset('working_dog.n.01')]
[Lemma('dog.n.01.dog'), Lemma('dog.n.01.domestic_dog'),
Lemma('dog.n.01.Canis_familiaris')]
```

An easy way to compute the similarity between two synsets is to measure the length of the path between the synsets in the WordNet hierarchy made up by the hypernym relations. The method path_simiarity returns 1/p where p is the length of the path between two synsets.

[3]:
```python
ape = wn.synset("ape.n.01")
monkey = wn.synset("monkey.n.01")
zoo = wn.synset("zoo.n.01")

print( "Similarity between ape and monkey: ", ape.path_similarity(monkey))
print( "Similarity between ape and zoo: ", ape.path_similarity(zoo))
```

```
Similarity between ape and monkey:  0.3333333333333333
Similarity between ape and zoo:  0.07692307692307693
```

Wordnet is not completely connected. The path similarity method therefore assumes a fake root node that connect all parts. The path similarity has the problem that words are less similar if they are part of the hierarchy that is worked out in more detail. In general we would assume that the first divisions at the top of the hierarchy imply large semantic differences, while a division at a very deep position in the hierarchy makes only small semantic distinctions. Therefore some alternative measures have been defined, e.g. the Wu-Palmer similarity and the Leacock-Chodorow similarity (feel free to read up on those measures).

[4]:
```python
print("Wu-Palmer similarity between ape and monkey: ", ape.
 ↪wup_similarity(monkey))
print("Wu-Palmer similarity between ape and zoo: ", ape.wup_similarity(zoo))
```

```
print("Leacock Chodorow similarity between ape and monkey: ", ape.
↪lch_similarity(monkey))
print("Leacock Chodorow similarity between ape and zoo: ", ape.
↪lch_similarity(zoo))
```

```
Wu-Palmer similarity between ape and monkey:  0.9230769230769231
Wu-Palmer similarity between ape and zoo:  0.4
Leacock Chodorow similarity between ape and monkey:  2.538973871058276
Leacock Chodorow similarity between ape and zoo:  1.072636802264849
```

Both measures give higher weight to distances between nodes that are closer to the root. However, the distance to the root is also a design decision and a number of measures try to include other information sources as well. E.g. the similarity measures of Resnik and Lin include the frequency of words in a corpus as well.

3 2. Exercise:

0. Read in the email dataset (see exercise 3). You may copy some of the code from that notebook.
1. Let us investigate the coverage of this data in Wordnet:
    - Count the unique words (types) in the data and store them in a list.
    - How many of those items have synsets in Wordnet? (calculate a percentage value)
    - What is the average number of synsets per type?
2. Not all words have lexical meaning. We can filter certain word classes. Apply POS-tagging (for example https://www.nltk.org/book/ch05.html) to extract only nouns (just NN - not proper nouns NNP). Check the coverage in Wordnet for these nouns. How many have synsets in Wordnet? (calculate a percentage value)
3. Experiments with the similarity of words:
    - Choose 10 out of the 50 most frequent nouns from the data set (they all should have at least one synset in Wordnet).
    - Now compute for each of the 10 words the Wu-Palmer or Leacock-Chodorow similarity to each of the other 9 words (you may use the first synset for each word for this calculation when words have multiple synsets). You might want to display the resulting numbers in a table. Which words are most similar to each other?
    - Check for all sentences which contain the word 'Obama': How often does each of the 10 words you selected occur in these sentences? Have words with similar meaning also similar co-occurrence counts with 'Obama'?

0. Reading the Email Dataset

[5]:
```
import zipfile
with zipfile.ZipFile("emails-body.txt.zip", 'r') as zip_f:
    zip_f.extractall('.')
```

[6]:
```
texts = open('emails-body.txt', encoding='utf8').read().split('<cmail>\n')
```

1. Investigating the coverage of this data in Wordnet

Count the unique words (types) in the data and store them in a list.

[7]:
```
from somajo import SoMaJo

somajo_tokenizer = SoMaJo(language="en_PTB",
                          split_camel_case=True)
```

[8]:
```
data_tok = []
for sentence in somajo_tokenizer.tokenize_text(texts):
    data_tok.extend([token.text for token in sentence])
unique_data_tok = list(set(data_tok))
```

[9]:
```
print('Total Unique words (types) in the Data: {}'.format(len(unique_data_tok)))
```

```
Total Unique words (types) in the Data: 37340
```

How many of those items have synsets in Wordnet? (calculate a percentage value)

[10]:
```
def calculate_synsets_percentage(word_tokens):
    count = 0
    for word in word_tokens:
        if wn.synsets(word):
            count += 1
    return (count/len(word_tokens))*100
```

[11]:
```
synsets_per = calculate_synsets_percentage(unique_data_tok)
print('Total Percentage of items having Synsets is: {:.2f}%'.
 ↪format(synsets_per))
```

```
Total Percentage of items having Synsets is: 66.25%
```

What is the average number of synsets per type?

[12]:
```
import pandas as pd
synsets_per_type = {word: len(wn.synsets(word)) for word in unique_data_tok if␣
 ↪len(wn.synsets(word)) > 0}
synsets_per_type = dict(sorted(synsets_per_type.items(), key=lambda item:␣
 ↪item[1], reverse=True))
pd.Series(synsets_per_type, name='Average Number of Synsets per Type')
```

[12]:
```
Break           75
break           75
breaks          75
broken          72
cut             70
                ..
conveners        1
HIKERS           1
bankrolling      1
incoherently     1
reignite         1
```

```
Name: Average Number of Synsets per Type, Length: 24737, dtype: int64
```

[13]:
```
average_synsets_type = sum(synsets_per_type.values())/len(unique_data_tok)
print('Average Number of Synsets Per Type: {:.2f}'.format(average_synsets_type))
```

```
Average Number of Synsets Per Type: 3.10
```

2. Not all words have lexical meaning. We can filter certain word classes. Apply POS-tagging to extract only nouns (just NN - not proper nouns NNP). Check the coverage in Wordnet for these nouns. How many have synsets in Wordnet? (calculate a percentage value)

[14]:
```
NN_tokens = [word for (word, tag) in nltk.pos_tag(unique_data_tok) if tag ==␣
↪'NN']
```

[15]:
```
NN_synsets_per = calculate_synsets_percentage(NN_tokens)
print('Total Percentage of Nouns (NN) having Synsets is: {:.2f}%'.
↪format(NN_synsets_per))
```

```
Total Percentage of Nouns (NN) having Synsets is: 78.35%
```

3. Experiments with the similarity of words

Choose 10 out of the 50 most frequent nouns from the data set (they all should have at least one synset in Wordnet).

[16]:
```
from collections import Counter
data_counter = Counter(data_tok)
NN_token_counts = {noun: data_counter[noun] for noun in NN_tokens if len(wn.
↪synsets(noun)) > 0}
NN_token_counts = dict(sorted(NN_token_counts.items(), key=lambda item:␣
↪item[1], reverse=True))
top_10_nouns = list(NN_token_counts.keys())[:10]
print('Top 10 out of the 50 most frequent nouns are: \n{}'.format(top_10_nouns))
```

```
Top 10 out of the 50 most frequent nouns are:
['call', 'time', 'w', 'get', 'work', 'government', 'today', 'see', 'support',
'right']
```

Now compute for each of the 10 words the Wu-Palmer or Leacock-Chodorow similarity to each of the other 9 words (you may use the first synset for each word for this calculation when words have multiple synsets). You might want to display the resulting numbers in a table. Which words are most similar to each other?

[17]:
```
import numpy as np
WP_distances = np.zeros(shape=(10,10))
LC_distances = np.zeros(shape=(10,10))
```

[18]:
```
for index1, noun1 in enumerate(top_10_nouns):
    n1 = wn.synsets(noun1)[0]
    for index2, noun2 in enumerate(top_10_nouns):
```

```
        WP_distances[index1, index2] = n1.wup_similarity(wn.synsets(noun2)[0])
        LC_distances[index1, index2] = n1.lch_similarity(wn.synsets(noun2)[0])
WP_distances = pd.DataFrame(WP_distances, columns=top_10_nouns,␣
 ↪index=top_10_nouns)
LC_distances = pd.DataFrame(LC_distances, columns=top_10_nouns,␣
 ↪index=top_10_nouns)
```

[19]:
```
def find_similar_words(df, similarity_metric):
    print('Printing Words Similar to each other using metric: {}\n'.
 ↪format(similarity_metric))
    for i in range(df.shape[0]):
        similar_word = df.iloc[i].nlargest(2).index[-1]
        print('{} is Similar to {}'.format(df.index[i],similar_word))
```

Wu-Palmer Similarity

[20]:
```
WP_distances
```

[20]:

	call	time	w	get	work	government
call	1.000000	0.117647	0.235294	0.086957	0.117647	0.117647
time	0.117647	1.000000	0.266667	0.400000	0.571429	0.285714
w	0.235294	0.266667	1.000000	0.190476	0.266667	0.266667
get	0.086957	0.400000	0.190476	1.000000	0.500000	0.200000
work	0.117647	0.571429	0.266667	0.500000	1.000000	0.285714
government	0.117647	0.285714	0.266667	0.200000	0.285714	1.000000
today	0.125000	0.307692	0.285714	0.210526	0.307692	0.307692
see	0.315789	0.125000	0.250000	0.090909	0.125000	0.125000
support	0.117647	0.571429	0.266667	0.500000	0.857143	0.285714
right	0.105263	0.375000	0.235294	0.272727	0.375000	0.250000

\

	today	see	support	right
call	0.125000	0.315789	0.117647	0.105263
time	0.307692	0.125000	0.571429	0.375000
w	0.285714	0.250000	0.266667	0.235294
get	0.210526	0.090909	0.500000	0.272727
work	0.307692	0.125000	0.857143	0.375000
government	0.307692	0.125000	0.285714	0.250000
today	1.000000	0.133333	0.307692	0.266667
see	0.133333	1.000000	0.125000	0.111111
support	0.307692	0.125000	1.000000	0.375000
right	0.266667	0.111111	0.375000	1.000000

[21]:
```
find_similar_words(WP_distances, 'Wu-Palmer')
```

```
Printing Words Similar to each other using metric: Wu-Palmer

call is Similar to see
time is Similar to work
```

```
w is Similar to today
get is Similar to work
work is Similar to support
government is Similar to today
today is Similar to time
see is Similar to call
support is Similar to work
right is Similar to time
```

Leacock-Chodorow Similarity

[22]:
```python
LC_distances
```

[22]:

	call	time	w	get	work	government
call	3.637586	0.864997	0.998529	0.546544	0.864997	0.864997
time	0.864997	3.637586	1.152680	1.072637	1.691676	1.239691
w	0.998529	1.152680	3.637586	0.747214	1.152680	1.152680
get	0.546544	1.072637	0.747214	3.637586	1.239691	0.804373
work	0.864997	1.691676	1.152680	1.239691	3.637586	1.239691
government	0.864997	1.239691	1.152680	0.804373	1.239691	3.637586
today	0.929536	1.335001	1.239691	0.864997	1.335001	1.335001
see	0.998529	0.929536	1.072637	0.593064	0.929536	0.929536
support	0.864997	1.691676	1.152680	1.239691	2.538974	1.239691
right	0.747214	1.239691	0.998529	0.804373	1.239691	1.072637

\

	today	see	support	right
call	0.929536	0.998529	0.864997	0.747214
time	1.335001	0.929536	1.691676	1.239691
w	1.239691	1.072637	1.152680	0.998529
get	0.864997	0.593064	1.239691	0.804373
work	1.335001	0.929536	2.538974	1.239691
government	1.335001	0.929536	1.239691	1.072637
today	3.637586	0.998529	1.335001	1.152680
see	0.998529	3.637586	0.929536	0.804373
support	1.335001	0.929536	3.637586	1.239691
right	1.152680	0.804373	1.239691	3.637586

[23]:
```python
find_similar_words(LC_distances, 'Leacock-Chodorow')
```

```
Printing Words Similar to each other using metric: Leacock-Chodorow

call is Similar to w
time is Similar to work
w is Similar to today
get is Similar to work
work is Similar to support
government is Similar to today
today is Similar to time
see is Similar to w
```

```
support is Similar to work
right is Similar to time
```

Check for all sentences which contain the word ‘Obama’: How often does each of the 10 words you selected occur in these sentences? Have words with similar meaning also similar co-occurrence counts with ‘Obama’?

[24]:

```
obama_sentences = []
for sentence in texts:
    if 'obama' in sentence.lower():
        obama_sentences.append(sentence)
```

[25]:

```
word_obama_count = np.zeros(shape=(len(obama_sentences), 10))
for index1, word in enumerate(top_10_nouns):
    for index2, ob_sent in enumerate(obama_sentences):
        if word in ob_sent.lower():
            word_obama_count[index2, index1] += 1
word_obama_count = pd.DataFrame(word_obama_count, columns=top_10_nouns, index=['Sentence ' + str(i) for i in range(1, len(obama_sentences) + 1)])
```

[26]:

```
word_obama_count
```

[26]:

```
              call  time    w  get  work  government  today  see  support  \
Sentence 1     1.0   1.0  1.0  1.0   1.0         1.0    0.0  0.0      1.0
Sentence 2     1.0   1.0  1.0  1.0   0.0         0.0    1.0  1.0      0.0
Sentence 3     0.0   0.0  1.0  0.0   0.0         0.0    1.0  1.0      0.0
Sentence 4     0.0   0.0  1.0  0.0   0.0         0.0    1.0  1.0      0.0
Sentence 5     0.0   1.0  1.0  1.0   1.0         1.0    0.0  1.0      1.0
…              …    …    …   …    …           …      …    …        …
Sentence 206   0.0   1.0  1.0  1.0   1.0         0.0    1.0  1.0      1.0
Sentence 207   1.0   1.0  1.0  1.0   1.0         0.0    0.0  0.0      0.0
Sentence 208   0.0   1.0  1.0  1.0   1.0         1.0    1.0  1.0      1.0
Sentence 209   0.0   0.0  1.0  1.0   0.0         1.0    0.0  0.0      1.0
Sentence 210   0.0   1.0  1.0  1.0   1.0         0.0    0.0  1.0      1.0

              right
Sentence 1      1.0
Sentence 2      0.0
Sentence 3      0.0
Sentence 4      0.0
Sentence 5      1.0
…               …
Sentence 206    0.0
Sentence 207    1.0
Sentence 208    1.0
Sentence 209    1.0
Sentence 210    1.0
```

```
[210 rows x 10 columns]
```