NLP PROGRAM-4

A program to get synonyms from WordNet

Soundarya G_ 2048057

Downloading and Importing Wordnet

```
# Downloading wordnet
import nltk
nltk.download('wordnet')
nltk.download('wordnet_ic')
nltk.download('genesis')
```

```
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data]   Package wordnet is already up-to-date!
    [nltk_data] Downloading package wordnet_ic to /root/nltk_data...
    [nltk_data]   Package wordnet_ic is already up-to-date!
    [nltk_data] Downloading package genesis to /root/nltk_data...
    [nltk_data]   Package genesis is already up-to-date!
    True
```

```
# Importing required libraries
import re
import pandas as pd
from nltk.corpus import wordnet
from nltk.corpus import wordnet_ic
from nltk.corpus import genesis
```

Wordnet

A really useful lexical resource is WordNet. Its unique semantic network helps us find word relations, synonyms, grammars, etc. WordNet is just another NLTK corpus reader. The WordNet corpus reader gives access to the Open Multilingual WordNet, using ISO-639 language codes.

> **Applications:**

> This helps support NLP tasks such as sentiment analysis, automatic language translation, text similarity, and more.

Word

Discipline

```
# Finding syset of the word ‘discipline’

syn = wordnet.synsets("discipline")
print(syn[0].name())
```

```
discipline.n.01
```

```
type(syn[0].name())
```

```
str
```

Care

```
syn = wordnet.synsets("care")
syn
```

```
[Synset('care.n.01'),
 Synset('caution.n.03'),
 Synset('concern.n.02'),
 Synset('care.n.04'),
 Synset('care.n.05'),
 Synset('care.n.06'),
 Synset('care.v.01'),
 Synset('care.v.02'),
 Synset('wish.v.02'),
 Synset('manage.v.02'),
 Synset('worry.v.02')]
```

```
type(syn)
```

```
list
```

```
syn[0]
```

```
Synset('care.n.01')
```

Lemmas

Lemmas of code.v.02 (as in "convert ordinary language into code") are code.v.02.encipher, code.v.02.cipher, code.v.02.cypher, code.v.02.encrypt, code.v.02.inscribe, code.v.02.write_in_code

```
print(syn[0].lemmas()[0].name())
```

```
care
```

Definition

```
print(syn[0].definition())
```

```
the work of providing treatment for or attending to someone or something
```

```
print(syn[3].definition())
```

```
a cause for feeling concern
```

Examples

```
print(syn[0].examples())
```

```
['no medical care was required', 'the old car needs constant attention']
```

```
print(syn[1].examples())
```

```
['he exercised caution in opening the door', 'he handled the vase with care']
```

```
print(syn[3].examples())
```

```
['his major care was the illness of his wife']
```

Antonyms

> Synonyms of the word active are searched in the module synsets and are appended in the list synonyms. The same process is repeated for the Antonym also.

```
synonyms = []
antonyms = []
for syn_set in wordnet.synsets("light"):
  for l in syn_set.lemmas():
    synonyms.append(l.name())
    if l.antonyms():
      antonyms.append(l.antonyms()[0].name())
print("\nSet of synonyms of the word:", set(synonyms))
print("\nSet of antonyms of the word:", set(antonyms))
```

```

Set of synonyms of the word: {'unaccented', 'illumine', 'unhorse', 'scant', 'light-colo

Set of antonyms of the word: {'heavy', 'extinguish', 'dark'}
```

```
synonyms = []
antonyms = []
for syn_set in wordnet.synsets("light"):
  #for l in syn_set.lemmas():
  synonyms.append(syn_set.name())
    #if l.antonyms():
       #antonyms.append(l.antonyms()[0].name())
print("\nSet of synonyms of the word:", set(synonyms))
print("\nSet of antonyms of the word:", set(antonyms))
```

```
Set of synonyms of the word: {'light.v.01', 'light.n.10', 'sparkle.n.01', 'light.s.11',

Set of antonyms of the word: set()
```

Main Code

> To find a word's synonym, part of speech, rank and definition.The words returned in range Minimum 10 Words-Maximum N Words.

```
def Program_4():
  user_word = input("Enter the word: ")
  syn = wordnet.synsets(user_word)

  word_list =[]
  pos_list = []
  rank_list = []
  defn_list = []

  pos_dict = {'n':'noun','v':'verb','a':'adjective','r':'adverb','s':'singular'}

  syn=list(syn)
  list1=[]
  for k in syn:
    list1.append(k)

  for i in range(len(list1)):
    value = str(list1[i])
    chunks = re.split("['.]",value)
    word_list.append(chunks[1])
    pos_list.append(pos_dict[chunks[2]])
    rank_list.append(chunks[3])
    defn_list.append(list1[i].definition())

  df = pd.DataFrame(list(zip(word_list, pos_list,rank_list,defn_list)),
                    columns =['Synonym Word', 'POS','Rank','Definition'])
  return df
```

```
Program_4()
```

```
Enter the word: care
```

	Synonym Word	POS	Rank	Definition
0	care	noun	01	the work of providing treatment for or attendi...
1	caution	noun	03	judiciousness in avoiding harm or danger
2	concern	noun	02	an anxious feeling
3	care	noun	04	a cause for feeling concern
4	care	noun	05	attention and management implying responsibili...
5	care	noun	06	activity involved in maintaining something in ...
6	care	verb	01	feel concern or interest
7	care	verb	02	provide care for
8	wish	verb	02	prefer or wish to do something
9	manage	verb	02	be in charge of, act on, or dispose of
10	worry	verb	02	be concerned with

Similarity

1.Thesaurus-based

```
from nltk.corpus import wordnet as wn

dog = wn.synset('dog.n.01')
cat = wn.synset('cat.n.01')
hit = wn.synset('hit.v.01')
slap = wn.synset('slap.v.01')
discovery = wn.synset('discovery.n.01')
find = wn.synset('find.v.01')
care_n = wn.synset('care.n.01')
care_v = wn.synset('care.v.02')
```

- Path Similarity:

> It is a similarity measure that finds the distance that is the length of the shortest path between two synsets. The score is in the range 0 to 1.

```
print(dog.path_similarity(cat))
```

```
0.2
```

```
print(wn.path_similarity(hit, slap))
```

```
0.14285714285714285
```

```
print(find.path_similarity(discovery))
print(find.path_similarity(discovery))
```

```
0.125
```

```
print(find.path_similarity(find))
```

```
1.0
```

```
print(care_n.path_similarity(care_v))
```

```
None
```

```
syn1 = wordnet.synsets('football')
syn2 = wordnet.synsets('soccer')

# A word may have multiple synsets, so need to compare each synset of word1 with
for s1 in syn1:
    for s2 in syn2:
        print("Path similarity of: ")
        print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')
        print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')
        print("   is", s1.path_similarity(s2))
        print()
```

```
Path similarity of:
Synset('football.n.01') ( n ) [ any of various games played with a ball (round or oval)
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to k
   is 0.5

Path similarity of:
Synset('football.n.02') ( n ) [ the inflated oblong ball used in playing American footb
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to k
   is 0.05
```

Interpretation: The highest path similarity score of the words is 0.5, indicating they are closely related.

- Leacock-Chodorow (LCH) Similarity:

> It is a similarity measure which is an extended version of Path-based similarity as it incorporates the depth of the taxonomy. Therefore, it is the negative log of the shortest path (spath) between two concepts (synset_1 and synset_2) divided by twice the total depth of the taxonomy (D). The LCH similarity scores are between 0 and 3.689

```
print(dog.lch_similarity(cat))
```

```
2.0281482472922856
```

```
print(wn.lch_similarity(hit, slap))
print(wn.lch_similarity(hit, slap))
```

```
1.3121863889661687
```

- Wu-Palmer (WUP) Similarity:

> Return a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node). The score can be 0 < score <= 1.

```
print(dog.wup_similarity(cat))
```

```
0.8571428571428571
```

```
print(wn.wup_similarity(hit, slap))
```

```
0.25
```

2. Information Content metrics(Thesaurus and Corpus)

- Resnik (RES) Similarity:

> Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node). It ranges from 0 for terms without similarity to infinity.

```
# wordnet_ic Information Content: Load an information content file from the wordr
brown_ic = wordnet_ic.ic('ic-brown.dat')
semcor_ic = wordnet_ic.ic('ic-semcor.dat')
# Or you can create an information content dictionary from a corpus
genesis_ic = wn.ic(genesis, False, 0.0)
```

```
print(dog.res_similarity(cat, brown_ic))
```

```
7.911666509036577
```

```
print(dog.res_similarity(cat, genesis_ic))
```

```
7.204023991374837
```

- Jiang-Conrath (JCN) Similarity:

> Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets.

Equation: 1 / (IC(s1) + IC(s2) - 2 * IC(lcs)).

```
print(dog.jcn_similarity(cat, brown_ic))
```

```
0.4497755285516739
```

```
print(dog.jcn_similarity(cat, genesis_ic))
```

```
0.28539390848096946
```

- Lin Similarity:

> Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets.

Equation: 2 * IC(lcs) / (IC(s1) + IC(s2)).

```
print(dog.lin_similarity(cat, brown_ic))
```

```
0.8768009843733973
```

```
print(dog.lin_similarity(cat, genesis_ic))
```

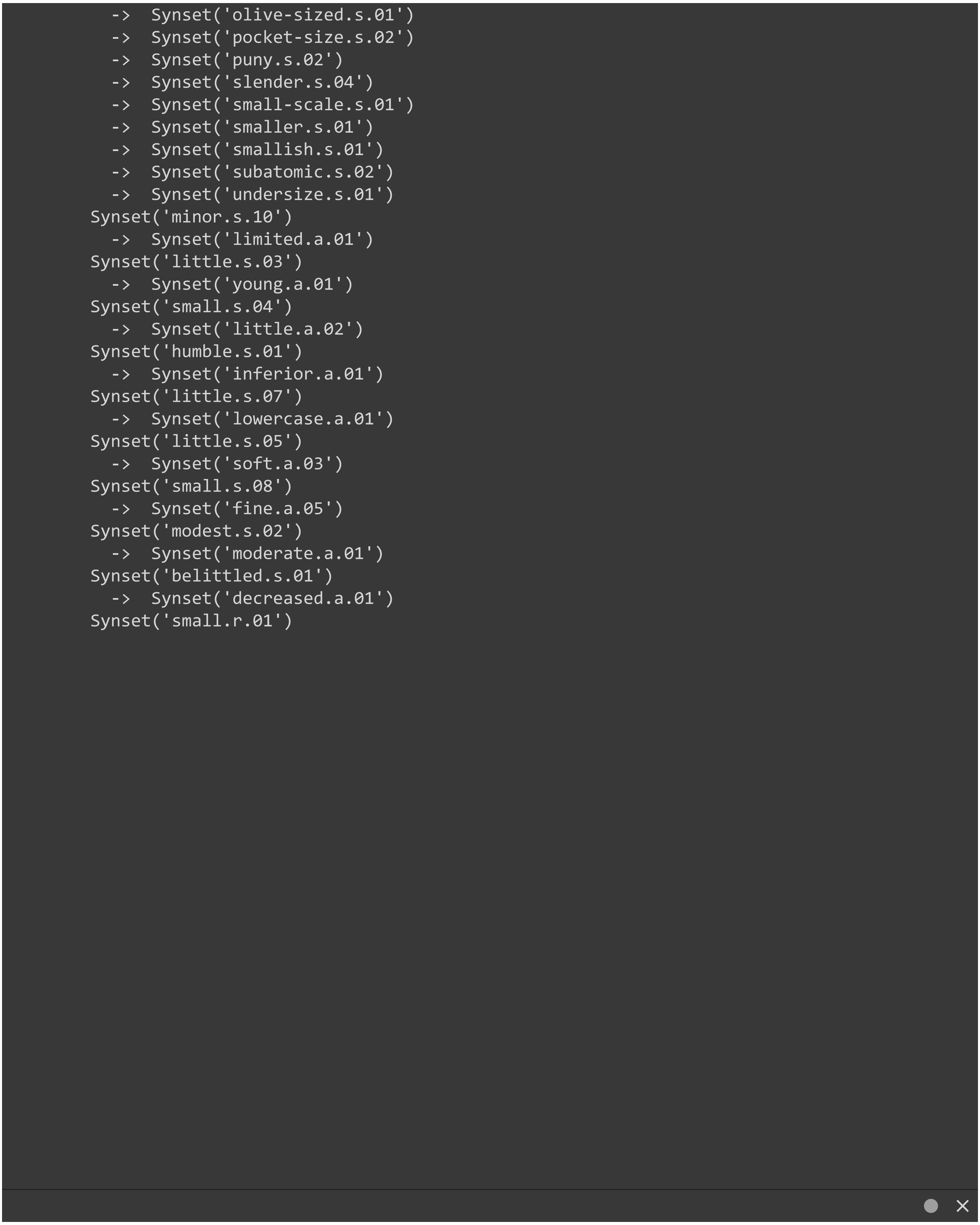
```
0.8043806652422293
```

Additional to Similarity

If you also want the "similar to" list, that's not the same thing as the synonyms. For that, you call similar_tos() on each Synset.

```
for s in wn.synsets('small'):
    print(s)
    for sim in s.similar_tos():
        print('  ->  {}'.format(sim))
```

```
Synset('small.n.01')
Synset('small.n.02')
Synset('small.a.01')
  ->  Synset('atomic.s.03')
  ->  Synset('bantam.s.01')
  ->  Synset('bitty.s.01')
  ->  Synset('dinky.s.01')
  ->  Synset('dwarfish.s.01')
  ->  Synset('elfin.s.02')
  ->  Synset('gnomish.s.01')
  ->  Synset('half-size.s.01')
  ->  Synset('infinitesimal.s.01')
  ->  Synset('lesser.s.02')
  ->  Synset('micro.s.01')
  ->  Synset('microscopic.s.04')
  ->  Synset('miniature.s.01')
  ->  Synset('minuscule.s.03')
```



```
      ->  Synset('olive-sized.s.01')
      ->  Synset('pocket-size.s.02')
      ->  Synset('puny.s.02')
      ->  Synset('slender.s.04')
      ->  Synset('small-scale.s.01')
      ->  Synset('smaller.s.01')
      ->  Synset('smallish.s.01')
      ->  Synset('subatomic.s.02')
      ->  Synset('undersize.s.01')
    Synset('minor.s.10')
      ->  Synset('limited.a.01')
    Synset('little.s.03')
      ->  Synset('young.a.01')
    Synset('small.s.04')
      ->  Synset('little.a.02')
    Synset('humble.s.01')
      ->  Synset('inferior.a.01')
    Synset('little.s.07')
      ->  Synset('lowercase.a.01')
    Synset('little.s.05')
      ->  Synset('soft.a.03')
    Synset('small.s.08')
      ->  Synset('fine.a.05')
    Synset('modest.s.02')
      ->  Synset('moderate.a.01')
    Synset('belittled.s.01')
      ->  Synset('decreased.a.01')
    Synset('small.r.01')
```