Natural Language Processing

Data Science Tools 1

Fall 2021

Agenda

- Common Tasks
- WordNet Corpus
- Word Semantics
- Homework 6 Review

Is it a trivial task to divide a text into sentences?

text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We are lucky to have him.'"""

Punkt Sentence Tokenizer

Unsupervised machine learning model that understands sentence boundaries given a text

Handles Collocations, Abbreviations and many more

```
import nltk.data
text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We are
lucky to have him.'"""
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
print('\n-----\n'.join(sent_detector.tokenize(text.strip())))
```

Common Task - Sentence2Words

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>>
```

Common Task - Sentence2Words

```
>>> import nltk
>>> tweet = """@EmpireStateBldg Central Park Tower is reaaaally hiiiigh”””
>>> words = nltk.tokenize.casual.casual_tokenize(tweet,
                                                  preserve_case=True,
                                                  reduce_len=True,
                                                  strip_handles=True)
>>> words
['Central', 'Park', 'Tower', 'is', 'reaaally', 'hiiigh']
```

Common Task - Word2POSTag

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>> words_pos = [nltk.pos_tag(word_list) for word_list in words]
>>> words_pos
[[('Mr.', 'NNP'), ('Smith', 'NNP'), ('knows', 'VBZ'), ('how', 'WRB'), ('to', 'TO'),
('do', 'VB'), ('his', 'PRP$'), ('job', 'NN'), ('.', '.')], [('He', 'PRP'), ('has',
'VBZ'), ('a', 'DT'), ('Ph.D', 'NNP'), ('in', 'IN'), ('Computer', 'NNP'), ('Science',
'NNP'), ('.', '.')], [("'We", 'NNS'), ('are', 'VBP'), ('lucky', 'JJ'), ('to', 'TO'),
('have', 'VB'), ('him', 'PRP'), ('.', '.'), ("'", "''")]]
```

Common Task - Word2StemmedWord

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>> >>> from nltk.stem.snowball import SnowballStemmer
>>> stemmer = SnowballStemmer('english')
>>> stemmed_words = [[stemmer.stem(word) for word in word_list] for word_list in words]
>>> stemmed_words
[['mr.', 'smith', 'know', 'how', 'to', 'do', 'his', 'job', '.'], ['he', 'has', 'a',
'ph.d', 'in', 'comput', 'scienc', '.'], ['we', 'are', 'lucki', 'to', 'have', 'him', '.',
"'"]]
>>>
```

A similar technique to stemming is **lemmatization**. The difference is that lemmatization provides us with a real word, that is, its canonical form. For example, the lemma of the word *cats* is *cat*, and the lemma for the word *ran* is *run*.

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>> from nltk import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> words.append(['He', 'ran', 'for', 'mayor', 'in', '1990'])
>>> lemma_words = [[lemmatizer.lemmatize(word, pos='v') for word in word_list] for
word_list in words]
>>> lemma_words
[['Mr.', 'Smith', 'know', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'have', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'be', 'lucky', 'to', 'have', 'him',
'.', "'"], ['He', 'run', 'for', 'mayor', 'in', '1990']]
>>> lemmatizer.lemmatize('worse', 'a')
'bad'
>>> lemmatizer.lemmatize('worse', 'n')
'worse'
>>> lemmatizer.lemmatize('worse', 'v')
'worse'
>>>
```

Common Task - Stopwords Removal

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>> stopwords = nltk.corpus.stopwords.words('english')
>>> type(stopwords)
>>> words_filtered = [word for word_list in words for word in word_list if word not in
stopwords ]
>>> words_filtered
['Mr.', 'Smith', 'knows', 'job', '.', 'He', 'Ph.D', 'Computer', 'Science', '.', "'We",
'lucky', '.', "'", 'He', 'ran', 'mayor', '1990']
```

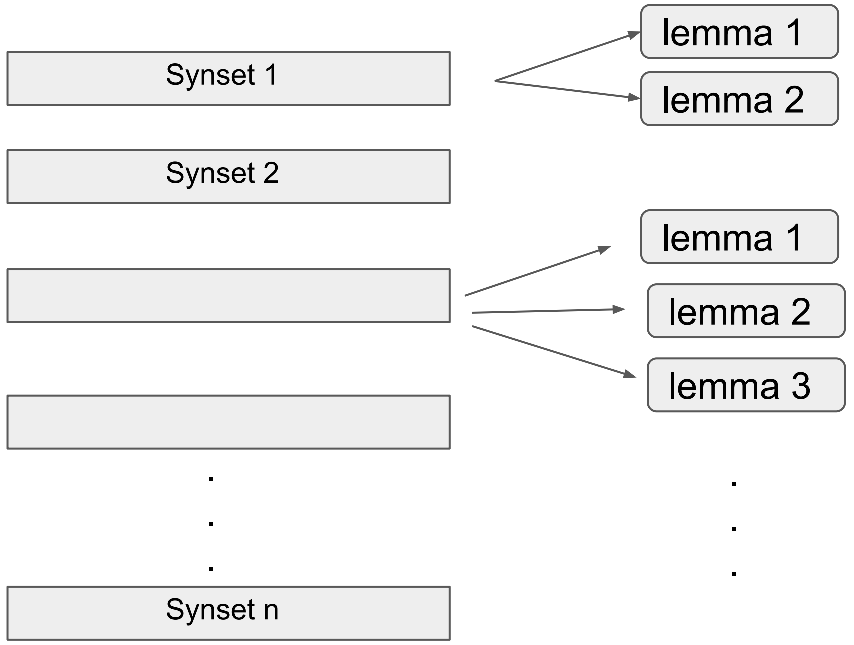
Common Task - Stopwords Removal

```
>>> import nltk.data
>>> text = """Mr. Smith knows how to do his job. He has a Ph.D in Computer Science. 'We
are lucky to have him.'"""
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> sentences = sent_detector.tokenize(text.strip())
>>> type(sentences)
<class 'list'>
>>> sentences
['Mr. Smith knows how to do his job.', 'He has a Ph.D in Computer Science.', "'We are
lucky to have him.'"]
>>> words = [ nltk.tokenize.word_tokenize(s) for s in sentences]
>>> words
[['Mr.', 'Smith', 'knows', 'how', 'to', 'do', 'his', 'job', '.'], ['He', 'has', 'a',
'Ph.D', 'in', 'Computer', 'Science', '.'], ["'We", 'are', 'lucky', 'to', 'have', 'him',
'.', "'"]]
>>> from nltk.probability import FreqDist
>>> freq = FreqDist(word.lower() for word_list in words for word in word_list)
>>> freq
FreqDist({'.': 3, 'to': 2, 'he': 2, 'in': 2, 'mr.': 1, 'smith': 1, 'knows': 1, 'how': 1,
'do': 1, 'his': 1, ...})
>>> words_with_frequencies = [(w, f) for w, f in freq.items()]
>>> words_with_frequencies
[('mr.', 1), ('smith', 1), ('knows', 1), ('how', 1), ('to', 2), ('do', 1), ('his', 1),
('job', 1), ('.', 3), ('he', 2), ('has', 1), ('a', 1), ('ph.d', 1), ('in', 2),
('computer', 1), ('science', 1), ("'we", 1), ('are', 1), ('lucky', 1), ('have', 1),
('him', 1), ("'", 1), ('ran', 1), ('for', 1), ('mayor', 1), ('1990', 1)]
>>> stopwords = [t[0] for t in words_with_frequencies if t[1] > 2]
>>> words_filtered_freq = [word for word_list in words for word in word_list if word not
in stopwords ]
```

What is WordNet Corpus?

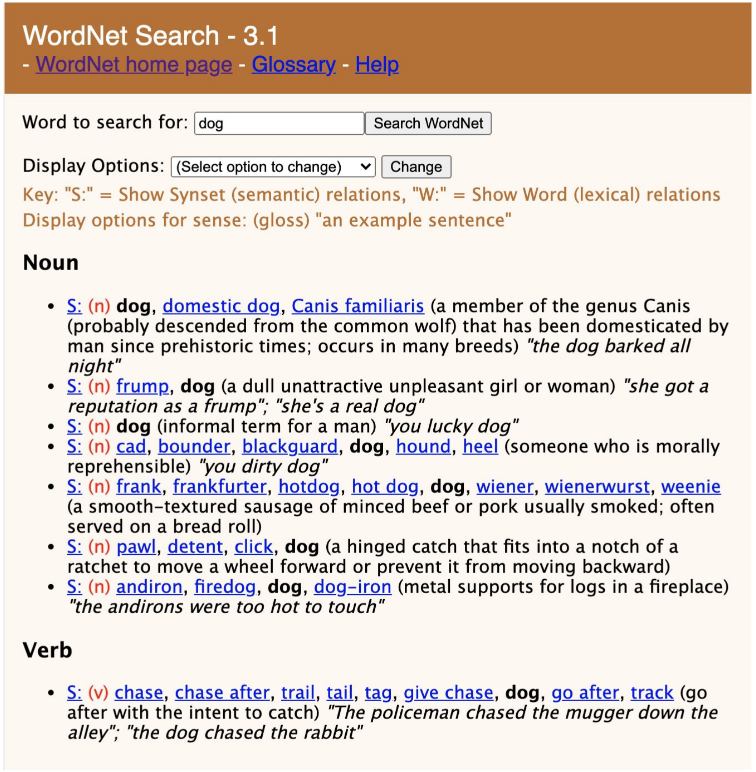
Digital lexical reference of nouns, adjectives, verbs and adverbs

More than a traditional dictionary or thesaurus



Synonym Sets (synset) - A synset is a unit that has a unique lexical sense. WordNet has a total of 117,000 synsets

WordNet Corpus - Design



WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: dog [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

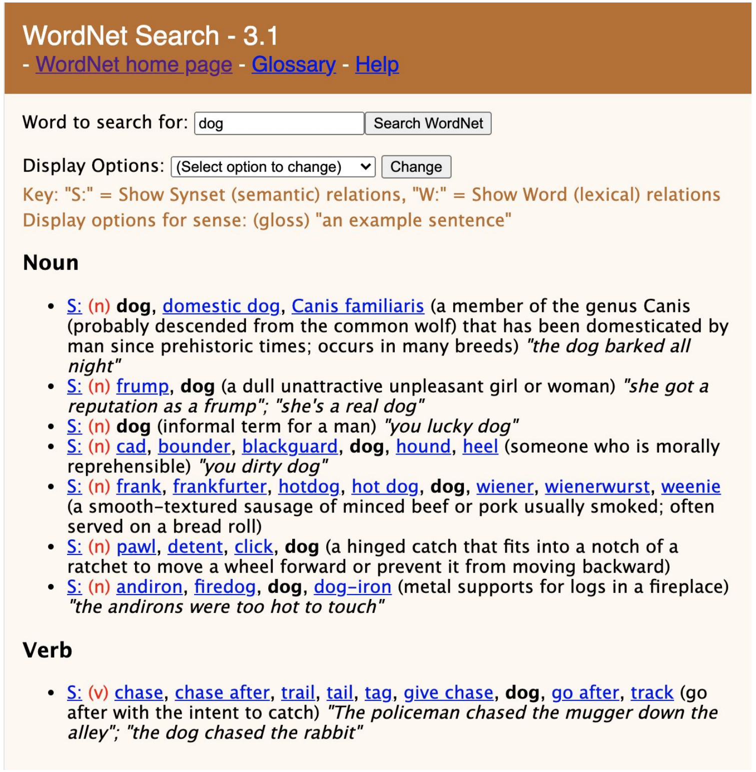
Noun

- S: (n) **dog**, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
- S: (n) frump, **dog** (a dull unattractive unpleasant girl or woman) *"she got a reputation as a frump"; "she's a real dog"*
- S: (n) **dog** (informal term for a man) *"you lucky dog"*
- S: (n) cad, bounder, blackguard, **dog**, hound, heel (someone who is morally reprehensible) *"you dirty dog"*
- S: (n) frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie (a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll)
- S: (n) pawl, detent, click, **dog** (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)
- S: (n) andiron, firedog, **dog**, dog-iron (metal supports for logs in a fireplace) *"the andirons were too hot to touch"*

Verb

- S: (v) chase, chase after, trail, tail, tag, give chase, **dog**, go after, track (go after with the intent to catch) *"The policeman chased the mugger down the alley"; "the dog chased the rabbit"*

WordNet Corpus - Design



WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: dog Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **dog**, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
- S: (n) frump, **dog** (a dull unattractive unpleasant girl or woman) *"she got a reputation as a frump"; "she's a real dog"*
- S: (n) **dog** (informal term for a man) *"you lucky dog"*
- S: (n) cad, bounder, blackguard, **dog**, hound, heel (someone who is morally reprehensible) *"you dirty dog"*
- S: (n) frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie (a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll)
- S: (n) pawl, detent, click, **dog** (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)
- S: (n) andiron, firedog, **dog**, dog-iron (metal supports for logs in a fireplace) *"the andirons were too hot to touch"*

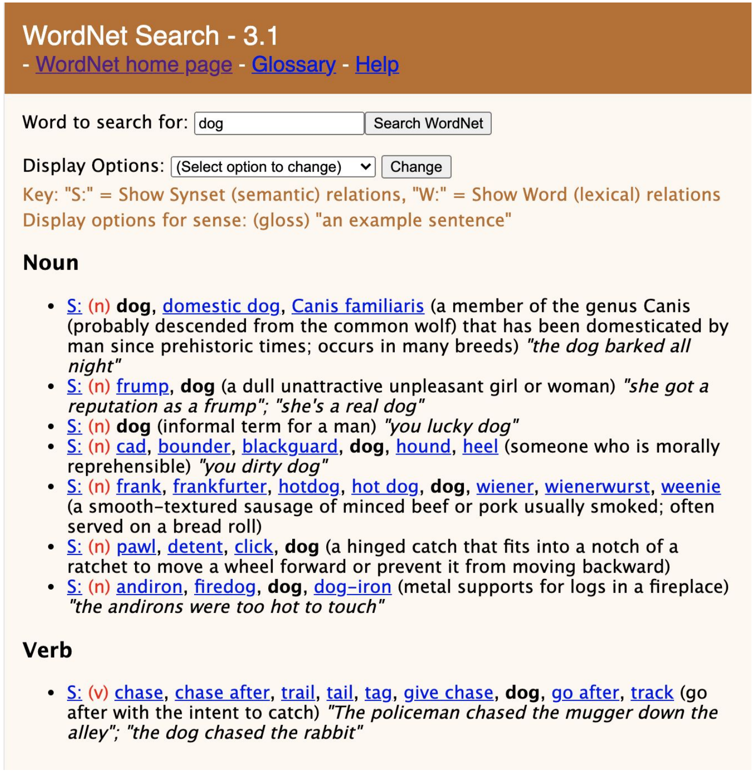
Verb

- S: (v) chase, chase after, trail, tail, tag, give chase, **dog**, go after, track (go after with the intent to catch) *"The policeman chased the mugger down the alley"; "the dog chased the rabbit"*

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('dog')
[Synset('dog.n.01'), Synset('frump.n.01'),
Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'),
Synset('andiron.n.01'), Synset('chase.v.01')]
>>> s1 = wn.synsets('dog')[0]
>>> s1
Synset('dog.n.01')
>>> help(s1)

>>> s1.definition
<bound method Synset.definition of
Synset('dog.n.01')>
>>> s1.definition()
'a member of the genus Canis (probably descended from
the common wolf) that has been domesticated by man
since prehistoric times; occurs in many breeds'
>>> s1.examples()
['the dog barked all night']
>>> s1.lemmas()
[Lemma('dog.n.01.dog'),
Lemma('dog.n.01.domestic_dog'),
Lemma('dog.n.01.Canis_familiaris')]
```

WordNet Corpus - Design



WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: dog [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **dog**, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
- S: (n) frump, **dog** (a dull unattractive unpleasant girl or woman) *"she got a reputation as a frump"; "she's a real dog"*
- S: (n) **dog** (informal term for a man) *"you lucky dog"*
- S: (n) cad, bounder, blackguard, **dog**, hound, heel (someone who is morally reprehensible) *"you dirty dog"*
- S: (n) frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie (a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll)
- S: (n) pawl, detent, click, **dog** (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)
- S: (n) andiron, firedog, **dog**, dog-iron (metal supports for logs in a fireplace) *"the andirons were too hot to touch"*

Verb

- S: (v) chase, chase after, trail, tail, tag, give chase, **dog**, go after, track (go after with the intent to catch) *"The policeman chased the mugger down the alley"; "the dog chased the rabbit"*

```
>>> s4 = wn.synsets('dog')[4]
>>> s4.definition()
'a smooth-textured sausage of minced beef or pork
usually smoked; often served on a bread roll'
>>> s4.lemmas()
[Lemma('frank.n.02.frank'),
Lemma('frank.n.02.frankfurter'),
Lemma('frank.n.02.hotdog'),
Lemma('frank.n.02.hot_dog'), Lemma('frank.n.02.dog'),
Lemma('frank.n.02.wiener'),
Lemma('frank.n.02.wienerwurst'),
Lemma('frank.n.02.weenie')]
```

Lemmas are synonyms of each other in a particular synset. A thesaurus don't have any concept of groupings of words to a synset or lexical sense

Synsets are linked to other synsets - hypernyms

```
>>> from pprint import pprint
>>> hyp = lambda s:s.hypernyms()
>>> pprint(s1.tree(hyp))
[Synset('dog.n.01'),
 [Synset('canine.n.02'),
  [Synset('carnivore.n.01'),
   [Synset('placental.n.01'),
    [Synset('mammal.n.01'),
     [Synset('vertebrate.n.01'),
      [Synset('chordate.n.01'),
       [Synset('animal.n.01'),
        [Synset('organism.n.01'),
         [Synset('living_thing.n.01'),
          [Synset('whole.n.02'),
           [Synset('object.n.01'),
            [Synset('physical_entity.n.01'),
             [Synset('entity.n.01')]]]]]]]]]]]]],
 [Synset('domestic_animal.n.01'),
  [Synset('animal.n.01'),
   [Synset('organism.n.01'),
    [Synset('living_thing.n.01'),
     [Synset('whole.n.02'),
      [Synset('object.n.01'),
       [Synset('physical_entity.n.01'), [Synset('entity.n.01')]]]]]]]]]
```

Synsets are linked to other synsets - hypernyms

```
>>> from pprint import pprint
>>> hyp = lambda s:s.hypernyms()
>>> >>> pprint(s4.tree(hyp))
[Synset('frank.n.02'),
 [Synset('sausage.n.01'),
  [Synset('meat.n.01'),
   [Synset('food.n.02'),
    [Synset('solid.n.01'),
     [Synset('matter.n.03'),
      [Synset('physical_entity.n.01'), [Synset('entity.n.01')]]]]]]]]
```

Sunsets are linked to other synsets - hypernyms

```
>>> from pprint import pprint
>>> hyp = lambda s:s.hypernyms()
>>> >>> pprint(s4.tree(hyp))
[Synset('frank.n.02'),
 [Synset('sausage.n.01'),
  [Synset('meat.n.01'),
   [Synset('food.n.02'),
    [Synset('solid.n.01'),
     [Synset('matter.n.03'),
      [Synset('physical_entity.n.01'), [Synset('entity.n.01')]]]]]]]]

>>> also possible to go the other direction
>>> hyp = lambda s:s.hyponyms()
>>> pprint(s4.tree(hyp))
[Synset('frank.n.02'), [Synset('vienna_sausage.n.01')]]
```

Lemmas are also related - derivationally_related, pertainyms, antonyms

Terms in different syntactic categories that have the same root form and are semantically related.

A relational adjective. Adjectives that are pertainyms are usually defined by such phrases as "of or pertaining to" and do not have antonyms. A pertainym can point to a noun or another pertainym.

```
Lemmas can also have relations between them:

>>> vocal = wn.lemma('vocal.a.01.vocal')
>>> vocal.derivationally_related_forms()
[Lemma('vocalize.v.02.vocalize')]
>>> vocal.pertainyms()
[Lemma('voice.n.02.voice')]
>>> vocal.antonyms()
         [Lemma('instrumental.a.01.instrumental')]
```

Note: these three relations are only present in Lemmas

Semantic Similarity between two words

```
synset1.path_similarity(synset2): Return a score denoting how similar two
word senses are, based on the shortest path that connects the senses in
the is-a (hypernym/hypnoym) taxonomy.

>>> dog = wn.synset('dog.n.01')
>>> cat = wn.synset('cat.n.01')

>>> hit = wn.synset('hit.v.01')
>>> slap = wn.synset('slap.v.01')

>>> dog.path_similarity(cat)
0.2
```

Other similarities functions available from WordNet Interface - Wu-Palmer Similarity, Leacock-Chodorow Similarity, Resnik Similarity, Jiang-Conrath Similarity, Lin Similarity

5 mins Break

Word Semantics - Bag of words

```
docs = [
 'the cat sat',
 'the cat sat in the hat',
 'the cat with the hat',
]
```

vocabulary is:

the

cat

sat

in

hat

with

Score words of new sentences based on its occurrence in the current document:

hat in the hat

[1, 0, 0, 1, 2, 0]

Word Semantics - Bag of words

docs = [

'the cat sat',

'the cat sat in the hat',

'the cat with the hat',

]

vocabulary is:

the

cat

sat

in

hat

with

Score words of new sentences based on its occurrence in the current document:

hat in the hat

[1, 0, 0, 1, 2, 0]

```
>>> docs = [
...  'the cat sat',
...  'the cat sat in the hat',
...  'the cat with the hat',
... ]
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit(docs)
>>> type(X)
<class 'sklearn.feature_extraction.text.CountVectorizer'>
>>> new_sentence = "hat in the hat"
>>> new_sentence_vector = X.transform([new_sentence])
>>> new_sentence_vector
<1x6 sparse matrix of type '<class 'numpy.int64'>'
        with 3 stored elements in Compressed Sparse Row format>
>>> new_sentence_vector.todense()
matrix([[0, 2, 1, 0, 1, 0]])
```

Word Semantics - Bag of words

```
docs = [
 'the cat sat',
 'the cat sat in the hat',
 'the cat with the hat',
]
```

```
Term frequency (tf) = count of t in d / number of words in d

Inverse Document Frequency (idf) = log (N / df ) + 1

TF-IDF = tf * idf
```

vocabulary is:

the

cat

sat

in

hat

with

Score words of new sentences based on its term frequency times inverse document frequency:

hat in the hat

[1, 0, 0, 1, 2, 0]

Word Semantics - Bag of words

docs = [

 'the cat sat',

 'the cat sat in the hat',

 'the cat with the hat',

]

vocabulary is:

the

cat

sat

in

hat

with

```
>>> docs = [
...  'the cat sat',
...  'the cat sat in the hat',
...  'the cat with the hat',
... ]
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> X = vectorizer.fit(docs)
>>> new_sentence_vector = X.transform([new_sentence])
>>> new_sentence_vector.todense()
matrix([[0. , 0.7948031 , 0.52253528, 0. , 0.30861775, 0. ]])
```

Score words of new sentences based on its occurrence in the current document and how rare the word is across documents:

hat in the hat

[1, 0, 0, 1, 2, 0]

Word Semantics - Neural Network

docs = [

'the cat sat',

'the cat sat in the hat',

'the cat with the hat',

]

vocabulary is:

the

cat

sat

in

hat

with

```
>>> from gensim.models import Word2Vec
>>> sentences = [["the", "cat", "sat"], ["the", "cat", "sat", "in", "the", "hat"],
["the", "cat", "sat", "with", "the", "hat"]]
>>> model = Word2Vec(sentences, vector_size=6, min_count=1)
>>> model.wv['hat']
array([-0.13808692, -0.15748031,  0.12186277,  0.08450437,  0.11262822,
        0.01271443], dtype=float32)
>>> model.wv['in']
array([-0.12519304, -0.0155007 ,  0.15896864, -0.12198611, -0.03889616,
       -0.0322957 ], dtype=float32)
>>> model.wv['the']
array([-0.00893712,  0.0039405 ,  0.08505583,  0.15015455, -0.15504916,
       -0.11861348], dtype=float32)
```

Score words of new sentences based on its occurrence in the current document and how rare the word is across documents:

hat in the hat

[[...], 0, 0, [...], [...], 0]

docs = [

'the cat sat',

'the cat sat in the hat',

'the cat with the hat',

]

vocabulary is:

the

cat

sat

in

hat

with

```
>>> from gensim.models import Word2Vec
>>> sentences = [["the", "cat", "sat"], ["the", "cat", "sat", "in", "the", "hat"],
["the", "cat", "sat", "with", "the", "hat"]]
>>> model = Word2Vec(sentences, vector_size=6, min_count=1)
>>> model.wv['hat']
array([-0.13808692, -0.15748031,  0.12186277,  0.08450437,  0.11262822,
        0.01271443], dtype=float32)
>>> model.wv['in']
array([-0.12519304, -0.0155007 ,  0.15896864, -0.12198611, -0.03889616,
       -0.0322957 ], dtype=float32)
>>> model.wv['the']
array([-0.00893712,  0.0039405 ,  0.08505583,  0.15015455, -0.15504916,
       -0.11861348], dtype=float32)

>>> import gensim.downloader
>>> print(list(gensim.downloader.info()['models'].keys()))
['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300',
'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50',
'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300',
'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200',
'__testing_word2vec-matrix-synopsis']
```

Score words of new sentences based on its occurrence in the current document and how rare the word is across documents:

hat in the hat

[1, 0, 0, 1, 2, 0]

Homework 6 Review