WordNet

Marina Sedinkina
- Folien von Desislava Zhekova -

CIS, LMU
marina.sedinkina@campus.lmu.de

December 18, 2018
Outline

1. WordNet
2. Lesk Algorithm
3. Finding Hypernyms with WordNet
4. Relation Extraction with spaCy
5. References
WordNet is a large lexical database of English (semantically-oriented).

Nouns, verbs, adjectives and adverbs are grouped into sets of synonyms (synsets).

Basis for grouping the words is their meanings.
WordNet

English WordNet online: http://wordnet.princeton.edu

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

Word to search for: motorcar

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) car, auto, automobile, machine, motorcar (a motor vehicle with four wheels; usually propelled by an internal combustion engine) "he needs a car to get to work"
  - direct hyponym / full hyponym
    - S: (n) ambulance (a vehicle that takes people to and from hospitals)
    - S: (n) beach wagon, station wagon, wagon, estate car, beach
# WordNet

## Wordnets in the World

<table>
<thead>
<tr>
<th>Language</th>
<th>Resource name</th>
<th>Developer(s)</th>
<th>Contact</th>
<th>Online Browsing</th>
<th>License</th>
<th>Other Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>Afrikaans WordNet</td>
<td>North-West University, South Africa</td>
<td>Gerhard van Huyssteen, Ané Bekker</td>
<td>NO</td>
<td>OPEN FOR ACADEMIC USE</td>
<td></td>
</tr>
<tr>
<td>Albanian</td>
<td>AlbaNet</td>
<td>Vlora University, Vlora, Albania</td>
<td>Ervin Ruci</td>
<td>YES</td>
<td>OPEN (GPL)</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>Arabic WordNet</td>
<td>Arabic WordNet</td>
<td>Horacio Rodriguez</td>
<td>NO</td>
<td>OPEN</td>
<td></td>
</tr>
<tr>
<td>Multilingual</td>
<td>Open Multilingual Wordnet</td>
<td>Linguistics and Multilingual Studies, NTU</td>
<td>Francis Bond</td>
<td>NO</td>
<td>OPEN</td>
<td></td>
</tr>
</tbody>
</table>

WordNet

- NLTK includes the English WordNet (155,287 words and 117,659 synonym sets)
- NLTK graphical WordNet browser: `nltk.app.wordnet()`

Current Word:  Next Word:  

Help Shutdown

noun

- **S:** (noun) **wordnet** (any of the machine-readable lexical databases modeled after the Princeton WordNet)
- **S:** (noun) **WordNet**, **Princeton WordNet** (a machine-readable lexical database organized by meanings; developed at Princeton University)
Consider the sentence in (1). If we replace the word motorcar in (1) with automobile, to get (2), the meaning of the sentence stays pretty much the same:

- Benz is credited with the invention of the motorcar.
- Benz is credited with the invention of the automobile.

⇒ Motorcar and automobile are synonyms.

Let’s explore these words with the help of WordNet
Motorcar has one meaning **car.n.01** (=the first noun sense of car).

The entity **car.n.01** is called a **synset**, or "synonym set", a collection of synonymous words (or "lemmas"):
Senses and Synonyms

Synsets are described with a **gloss** (= definition) and some example sentences.

```python
1 >>> wn.synset("car.n.01").definition()
2 "a motor vehicle with four wheels; usually propelled by an internal combustion engine"
3 >>> wn.synset("car.n.01").examples()
4 ["he needs a car to get to work"]
```
Unlike the words automobile and motorcar, which are unambiguous and have one synset, the word car is ambiguous, having five synsets:

```python
>>> wn.synsets("car")
[Synset("car.n.01"), Synset("car.n.02"), Synset("car.n.03"), Synset("car.n.04"), Synset("cable_car.n.01"))]

>>> for synset in wn.synsets("car"):
    ... print synset.lemma_names()
```

...
The WordNet Hierarchy

Hypernyms and hyponyms ("is-a relation")

- *motor vehicle* is a hypernym of *motorcar*
- *ambulance* is a hyponym of *motorcar*
```python
>>> motorcar = wn.synset("car.n.01")
>>> types_of_motorcar = motorcar.hyponyms()
>>> types_of_motorcar[26]
Synset("ambulance.n.01")
>>> sorted([lemma.name() for synset in types_of_motorcar
          for lemma in synset.lemmas()])
["Model_T", "S.U.V.", "SUV", "Stanley_Steamer", "ambulance", 
  "beach_waggon", "beach_wagon", "bus", "cab", "compact", 
  "compact_car", "convertible", "coupe", "cruiser", "electric", 
  "electric_automobile", "electric_car", "estate_car", 
  "gas_guzzler", "hack", "hardtop", "hatchback", "heap", 
  "horseless_carriage", "hot-rod", "hot_rod", "jalopy", 
  "jeep", "landrover", "limo", "limousine", "loaner", 
  "minicar", "minivan", "pace_car", "patrol_car", "phaeton", 
  "police_car", "police_cruiser", "prowl_car", "race_car", 
  "racer", "racing_car" ... ]
```
The WordNet Hierarchy

```python
>>> motorcar.hypernyms()
[Synset("motor_vehicle.n.01")]

>>> paths = motorcar.hypernym_paths()

>>> len(paths)
2

>>> [synset.name() for synset in paths[0]]
["entity.n.01", "physical_entity.n.01", "object.n.01", "whole.n.02", "artifact.n.01", "instrumentality.n.03", "container.n.01", "wheeled_vehicle.n.01", "self−propelled_vehicle.n.01", "motor_vehicle.n.01", "car.n.01"]

>>> [synset.name() for synset in paths[1]]
["entity.n.01", "physical_entity.n.01", "object.n.01", "whole.n.02", "artifact.n.01", "instrumentality.n.03", "conveyance.n.03", "vehicle.n.01", "wheeled_vehicle.n.01", "self−propelled_vehicle.n.01", "motor_vehicle.n.01", "car.n.01"]
```
Meronyms and holonyms

- *branch* is a meronym (*part meronym*) of *tree*
- *heartwood* is a meronym (*substance meronym*) of *tree*
- *forest* is a holonym (*member holonym*) of *tree*
More Lexical Relations

```python
>>> wn.synset("tree.n.01").part_meronyms()
[Synset("burl.n.02"), Synset("crown.n.07"), Synset("stump.n.01"), Synset("trunk.n.01"), Synset("limb.n.02")]

>>> wn.synset("tree.n.01").substance_meronyms()
[Synset("heartwood.n.01"), Synset("sapwood.n.01")]

>>> wn.synset("tree.n.01").member_holonyms()
[Synset("forest.n.01")]
```
More Lexical Relations

- Relationships between verbs:
  - The act of walking involves the act of stepping, so walking entails stepping.
  - Some verbs have multiple entailments.

```python
>>> wn.synset("walk.v.01").entailments()
[Synset("step.v.01")]

>>> wn.synset("eat.v.01").entailments()
[Synset("swallow.v.01"), Synset("chew.v.01")]

>>> wn.synset("tease.v.03").entailments()
[Synset("arouse.v.07"), Synset("disappoint.v.01")]
```
Some lexical relationships hold between lemmas, e.g., **antonymy**:

```python
>>> wn.lemma("supply.n.02.supply").antonyms()
[Lemma("demand.n.02.demand")]

>>> wn.lemma("rush.v.01.rush").antonyms()
[Lemma("linger.v.04.linger")]

>>> wn.lemma("horizontal.a.01.horizontal").antonyms()
[Lemma("vertical.a.01.vertical"), Lemma("inclined.a.02.inclined")]

>>> wn.lemma("staccato.r.01.staccato").antonyms()
[Lemma("legato.r.01.legato")]
```
You can see the lexical relations, and the other methods defined on a synset, using `dir()`.

```python
import nltk
from nltk.corpus import wordnet as wn

print(wn.synsets("motorcar"))
>>> [Synset('car.n.01')]

print(dir(wn.synsets("motorcar")[0]))
>>> [ ... , 'hyponyms', 'instance_hypernyms', 'instance_hyponyms', 'jcn_similarity', 'Ich_similarity', 'lemma_names', 'lemmas', 'lexname', 'lin_similarity', 'lowest_common_hypernyms', 'max_depth', 'member_holonyms', 'member_meronyms', 'min_depth', 'name', 'offset', 'part_holonyms', 'part_meronyms', 'path_similarity', 'pos', 'region_domains', 'res_similarity', 'root_hypernyms', 'shortest_path_distance', 'similar_tos', 'substance_holonyms', 'substance_meronyms', 'topic_domains', 'tree', 'unicode_repr', 'usage_domains', 'verb_groups', 'wup_similarity']
```
If two synsets share a very specific hypernym (low down in the hypernym hierarchy), they must be closely related.

```python
>>> right = wn.synset("right_whale.n.01")
>>> orca = wn.synset("orca.n.01")
>>> minke = wn.synset("minke_whale.n.01")
>>> tortoise = wn.synset("tortoise.n.01")
>>> novel = wn.synset("novel.n.01")

>>> right.lowest_common_hypernyms(minke)
[Synset("baleen_whale.n.01")]

>>> right.lowest_common_hypernyms(orca)
[Synset("whale.n.02")]

>>> right.lowest_common_hypernyms(tortoise)
[Synset("vertebrate.n.01")]

>>> right.lowest_common_hypernyms(novel)
[Synset("entity.n.01")]
```
Semantic Similarity

We can quantify this concept of generality by looking up the depth of each synset:

```python
>>> wn.synset("baleen_whale.n.01").min_depth()
14

>>> wn.synset("whale.n.02").min_depth()
13

>>> wn.synset("vertebrate.n.01").min_depth()
8

>>> wn.synset("entity.n.01").min_depth()
0
```
Semantic Similarity

Similarity measures have been defined over the collection of WordNet synsets that incorporate this insight:

- `path_similarity()` assigns a score in the range 0-1 based on the shortest path that connects the concepts in the hypernym hierarchy.
- -1 is returned in those cases where a path cannot be found.
- Comparing a synset with itself will return 1.
Semantic Similarity

```python
def right.path_similarity(minke):
    return 0.25

>>> right.path_similarity(orca)
0.16666666666666666

>>> right.path_similarity(tortoise)
0.076923076923076927

>>> right.path_similarity(novel)
0.043478260869565216
```
Similarity between nouns

- ("car", "automobile")
- $\text{synsets1("car") = [synset}_{11}, \text{synset}_{12}, \text{synset}_{13}]$
  \[\text{nltk.corpus.wordnet.synsets("car")}\]
- $\text{synsets2("automobile") = [synset}_{21}, \text{synset}_{22}, \text{synset}_{23}]$
  \[\text{nltk.corpus.wordnet.synsets("automobile")}\]
- consider all combinations of synsets formed by the synsets of the words in the word pair ("car", "automobile")
  \[(\text{synset}_{11}, \text{synset}_{21}), (\text{synset}_{11}, \text{synset}_{22}), (\text{synset}_{11}, \text{synset}_{23}), ...\]
- determine score of each combination e.g.:
  \[\text{synset}_{11}.\text{path\_similarity}(\text{synset}_{21})\]
- determine the maximum score $\rightarrow$ indicator of similarity
Can you think of an NLP application for which semantic similarity will be helpful?
Can you think of an NLP application for which semantic similarity will be helpful?

**Suggestion**

**Coreference Resolution:**
I saw an orca. The whale was huge.
The polysemy of a word is the number of senses it has.

The noun `dog` has 7 senses in WordNet:

```python
from nltk.corpus import wordnet as wn
num_senses=len(wn.synsets('dog','n'))
print(num_senses)
```

```
prints 7
```

We can also compute the average polysemy of nouns, verbs, adjectives and adverbs according to WordNet.
Polysemy of nouns

We can also compute the average polysemy of nouns.

- Fetch all lemmas in WordNet that have a given POS:
  
  ```python
  nltk.corpus.wordnet.all_lemma_names(POS)
  ```

  ```python
  from nltk.corpus import wordnet as wn
  all_lemmas=set(wn.all_lemma_names("n"))
  print(len(all_lemmas))
  >>>117798
  ```

- Determine meanings of each lemma:
  
  ```python
  nltk.corpus.wordnet.synsets(lemma,pos) returns list of senses to a given lemma and POS, e.g. for "car"
  ```

  ```python
  from nltk.corpus import wordnet as wn
  meanings=wn.synsets("car","n")
  print(meanings)
  >>>
  [Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'),
   Synset('car.n.04'), Synset('cable_car.n.01')]```
def average_polysemy(part_of_speech):
    lemmas = set(nltk.corpus.wordnet.all_lemma_names(part_of_speech))

    nr_of_synsets = 0
    for lemma in lemmas:
        nr_of_synsets += len(nltk.corpus.wordnet.synsets(lemma, pos=part_of_speech))

    return nr_of_synsets / len(lemmas)
Lesk Algorithm

- classical algorithm for Word Sense Disambiguation (WSD) introduced by Michael E. Lesk in 1986
- idea: word’s dictionary definitions are likely to be good indicators for the senses they define
Lesk Algorithm: Example

<table>
<thead>
<tr>
<th>Sense</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1: tree</td>
<td>a tree of the olive family</td>
</tr>
<tr>
<td>s2: burned stuff</td>
<td>the solid residue left when combustible material is burned</td>
</tr>
</tbody>
</table>

Table: Two senses of **ash**
Lesk Algorithm: Example

<table>
<thead>
<tr>
<th>Sense</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1: tree</td>
<td>a tree of the olive family</td>
</tr>
<tr>
<td>s2: burned stuff</td>
<td>the solid residue left when combustible material is burned</td>
</tr>
</tbody>
</table>

Table: Two senses of ash

Score = number of (stemmed) words that are shared by sense definition and context

<table>
<thead>
<tr>
<th>Scores</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 s2</td>
<td>This cigar burns slowly and creates a stiff ash</td>
</tr>
</tbody>
</table>

Table: Disambiguation of ash with Lesk’s algorithm
Lesk Algorithm: Example

<table>
<thead>
<tr>
<th>Sense</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1: tree</td>
<td>a tree of the olive family</td>
</tr>
<tr>
<td>s2: burned stuff</td>
<td>the solid residue left when combustible material is burned</td>
</tr>
</tbody>
</table>

Table: Two senses of *ash*

Score = number of (stemmed) words that are shared by sense definition and context

<table>
<thead>
<tr>
<th>Scores</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 s2</td>
<td>This cigar <em>burns</em> slowly and creates a stiff ash</td>
</tr>
</tbody>
</table>

Table: Disambiguation of ash with Lesk’s algorithm
Lesk Algorithm: Example

**Sense**

s1: tree

s2: burned stuff

**Definition**

a tree of the olive family

the solid residue left when combustible material is **burned**

*Table: Two senses of ash*

Score = number of (stemmed) words that are shared by sense definition and context

<table>
<thead>
<tr>
<th>Scores</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 s2</td>
<td>This cigar <strong>burns</strong> slowly and creates a stiff ash</td>
</tr>
</tbody>
</table>

*Table: Disambiguation of ash with Lesk’s algorithm*
**Lesk Algorithm: Example**

### Sense

- s1: tree
- s2: burned stuff

### Definition

- a tree of the olive family
- the solid residue left when combustible material is burned

**Table: Two senses of ash**

<table>
<thead>
<tr>
<th>Scores</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>The ash is one of the last trees to come into leaf</td>
</tr>
</tbody>
</table>
| s2     | ???

**Table: Disambiguation of ash with Lesk’s algorithm**
Lesk Algorithm: Example

<table>
<thead>
<tr>
<th>Sense</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1: tree</td>
<td>a tree of the olive family</td>
</tr>
<tr>
<td>s2: burned stuff</td>
<td>the solid residue left when combustible material is burned</td>
</tr>
</tbody>
</table>

*Table: Two senses of ash*

Score = number of (stemmed) words that are shared by sense definition and context

<table>
<thead>
<tr>
<th>Scores</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 s2</td>
<td>The ash is one of the last trees to come into leaf</td>
</tr>
</tbody>
</table>

*Table: Disambiguation of ash with Lesk’s algorithm*
Lesk Algorithm

```python
>>> from nltk.wsd import leks
>>> sent = ["I", "went", "to", "the", "bank", "to", "deposit", "money", "."]

>>> print(lexk(sent, "bank","n"))
Synset("savings_bank.n.02")
```
The definitions for "bank" are:

```python
>>> from nltk.corpus import wordnet as wn
>>> for ss in wn.synsets("bank"):
    ...     print(ss, ss.definition())
```

- Synset('bank.n.01') sloping land (especially the slope beside a body of water)
- Synset('depository_financial_institution.n.01') a financial institution that accepts deposits and channels the money into lending activities
- Synset('bank.n.03') a long ridge or pile
- Synset('bank.n.04') an arrangement of similar objects in a row or in tiers
- Synset('bank.n.05') a supply or stock held in reserve for future use (especially in emergencies)
- Synset('bank.n.06') the funds held by a gambling house or the dealer in some gambling games
- Synset('bank.n.07') a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force
- Synset('savings_bank.n.02') a container (usually with a slot in the top) for keeping money at home
- Synset('bank.n.09') a building in which the business of banking transacted
- Synset('bank.n.10') a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
- Synset('bank.v.01') tip laterally
- Synset('bank.v.02') enclose with a bank
Lesk Algorithm

Check implementation via

http://www.nltk.org/_modules/nltk/wsd.html

```python
def lesk(context_sentence, ambiguous_word, pos=None, synsets=None):
    context = set(context_sentence)
    if synsets is None:
        synsets = wordnet.synsets(ambiguous_word)
    if pos:
        synsets = [ss for ss in synsets if str(ss.pos()) == pos]
    if not synsets:
        return None
    _, sense = max(
        (len(context.intersection(ss.definition().split())), ss) for ss in synsets)
    return sense
```
Lesk Algorithm

- Information derived from a dictionary is insufficient for high quality **Word Sense Disambiguation (WSD)**.
- Lesk reports accuracies between 50% and 70%.
- Optimizations: to expand each word in the context with a list of synonyms.
In the Wikipedia article on Ada Lovelace,

- how many words refer to a **relative**? (excluding names)
- how many words refer to an **illness**?
- how many words refer to a **science**?

In each case: which words?
Task

TASK TO SOLVE

In the Wikipedia article on Ada Lovelace,

- how many words refer to a relative? (excluding names)
- how many words refer to an illness?
- how many words refer to a science?

In each case: which words?

Let’s solve this using WordNet...
Step 1: Read in file

Read `ada_lovelace.txt` as one text string.

```python
>>> print text
"Augusta Ada King, Countess of Lovelace (10 December 1815
27 November 1852), born Augusta Ada Byron and
now commonly known as Ada Lovelace, was an
English mathematician and writer chiefly known
for her work on Charles Babbage's early mechanical
general-purpose computer, the Analytical Engine. ... "
```
Step 2: Sentence Splitting

Split the text into sentences: `nltk.sent_tokenize(text)`

```python
>>> print sentences[:3]
['Augusta Ada King, Countess of Lovelace (10 December 1815 – 27 November 1852), born Augusta Ada Byron and now commonly known as Ada Lovelace, was an English mathematician and writer chiefly known for her work on Charles Babbage’s early mechanical general-purpose computer, the Analytical Engine.‘, 'Her notes on the engine include what is recognised as the first algorithm intended to be carried out by a machine.’, 'Because of this, she is often described as the world’s first computer programmer.’, ... ]
```
### Step 3: Tokenize

Split the sentences into tokens: `nltk.word_tokenize(text)`

Create one list of tokens (containing all tokens of the text).

```python
>>> print tokens
[' Augusta ', ' Ada ', ' King ', ',', ',', ' Countess ', ', of ',
' Lovelace ', ',', ' ( ', ' 10 ', ', December ', ', 1815 ', ', 27 ',
' November ', ', 1852 ', ' ) ', ',', ',', ' born ', ' Augusta ',
' Ada ', ' Byron ', ' and ', ' now ', ' commonly ', ' known ',
' as ', ' Ada ', ' Lovelace ', ',', ',', ' was ', ' an ',
' English ', ' mathematician ', ' and ', ' writer ',
' chiefly ', ' known ', ' for ', ' her ', ' work ', ' on ',
' Charles ', ' Babbage ', ' ', ' s ', ' early ', ' mechanical ',
' general-purpose ', ' computer ', ',', ',', ' the ',
' Analytical ', ' Engine ', ' ', ' Her ', ' notes ', ' on ',
' the ', ' engine ', ' include ', ' what ', ' is ',
' recognised ', ' as ', ' the ', ' first ', ' algorithm ',
' intended ', ' to ', ' be ', ' carried ', ' out ',
```
Step 4: Part-of-Speech tagging

Find the POS-tag of each token using NLTK’s recommended POS tagger.

```python
pos_tags = nltk.pos_tag(tokens)
present pos_tags
```

Print out all the nouns occurring in the text.
Step 4: Part-of-Speech tagging

1. CC – coordinating conjunction
2. RB – adverb
3. IN – preposition
4. NN – noun
5. JJ – adjective
6. VB – verb
Step 4: Part-of-Speech tagging

NLTK provides documentation for each tag, which can be queried using the tag, e.g:

```python
>>> nltk.help.upenn_tagset('NN')
NN: noun, common, singular or mass
  common–carrier cabbage knuckle–duster Casino
  afghan shed thermostat investment slide
  humour falloff slick wind hyena override
  subhumanity machinist ...

>>> nltk.help.upenn_tagset('CC')
CC: conjunction, coordinating
  & and both but either et for less minus neither
  nor or plus so therefore times v. versus vs.
  whether yet
```
Step 4: Part-of-Speech tagging

Note!

Some POS tags denote variation of the same word type, e.g. NN, NNS, NNP, NNPS, such can be looked up via regular expressions.

```
>>> nltk.help.upenn_tagset('NN*')
NN: noun, common, singular or mass
    common−carrier cabbage knuckle−duster Casino ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita ...
NNPS: noun, proper, plural
    Americans Americas Amharas Amityvilles ...
NNS: noun, common, plural
    undergraduates scotches bric−a−brac ...
```
Step 4: Lemmatize

Now, put the lemma of each noun from the text into one list.

```python
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

# your code ...

lemmatizer.lemmatize(lemma, wordnet.NOUN)

# your code ...

>>> print noun_lemmas
>>> ['Augusta', 'Ada', 'King', 'Countess', 'Lovelace',
  '(', 'December', 'November', 'born', 'Augusta',
  'Ada', 'Byron', 'Ada', 'Lovelace',
  'mathematician', 'writer', 'work', 'Charles',
  'Babbage', 'computer', ... ]
```
These are the three hypernyms of interest:
(as there are multiple synsets for a lemma, we pick the first one in each list returned by `nltk.wordnet`)

1. `relative = wordnet.synsets("relative", pos='n')[0]`
2. `science = wordnet.synsets("science", pos='n')[0]`
3. `illness = wordnet.synsets("illness", pos='n')[0]`
Ada Lovelace Task: Hypernyms

These are the three hypernyms of interest:
(as there are multiple synsets for a lemma, we pick the first one in each list returned by `nltk.wordnet`)

1. `relative = wordnet.synsets("relative", pos='n')[0]`
2. `science = wordnet.synsets("science", pos='n')[0]`
3. `illness = wordnet.synsets("illness", pos='n')[0]`

How can we find out whether one synset is a hyponym of another?
Ada Lovelace Task: Hypernym Code

```python
def hypernymOf(synset1, synset2):
    """ Returns True if synset2 is a hypernym of synset1, or if they are the same synset. Returns False otherwise. """
    if synset1 == synset2:
        return True
    for hypernym in synset1.hypernyms():
        if synset2 == hypernym:
            return True
        if hypernymOf(hypernym, synset2):
            return True
    return False
```
Reminder:

- We have a list of the lemmas of all nouns, `noun_lemmas`.
- Retrieve the synsets for each lemma.
- Check whether it’s a hyponym of one of the three synsets of interest.
- Counts the relevant nouns, and collect them.
Tokenization with spaCy

```python
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1 billion')
for token in doc:
    print(token.text)

>>> Apple
    is
    looking
    at
    buying
    U.K.
    ...
```
Tokenization with spaCy

- Does the substring match a tokenizer exception rule? (U.K.)
- Can a prefix, suffix or infix be split off? (e.g. punctuation)
Tokenization with spaCy

- Tokenizer exceptions strongly depend on the specifics of the individual language
- Global and language-specific tokenizer data is supplied via the language data in `spacy/lang`

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>CODE</th>
<th>LANGUAGE DATA</th>
<th>MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>en</td>
<td>lang/en</td>
<td>4 models</td>
</tr>
<tr>
<td>German</td>
<td>de</td>
<td>lang/de</td>
<td>1 model</td>
</tr>
<tr>
<td>Spanish</td>
<td>es</td>
<td>lang/es</td>
<td>2 models</td>
</tr>
<tr>
<td>Portuguese</td>
<td>pt</td>
<td>lang/pt</td>
<td>1 model</td>
</tr>
<tr>
<td>French</td>
<td>fr</td>
<td>lang/fr</td>
<td>2 models</td>
</tr>
<tr>
<td>Italian</td>
<td>it</td>
<td>lang/it</td>
<td>1 model</td>
</tr>
</tbody>
</table>
Adding special case tokenization rules

The tokenizer exceptions define special cases like "don’t" in English, which needs to be split into two tokens: {ORTH: do} and {ORTH: n’t, LEMMA: not}

```python
import spacy
from spacy.symbols import ORTH, LEMMA, POS, TAG

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'gimme that')  # phrase to tokenize
print([w.text for w in doc])  # ['gimme', 'that']

# add special case rule
special_case = [{ORTH: u'gim', LEMMA: u'give', POS: u'VERB'},
                {ORTH: u'me'}]
nlp.tokenizer.add_special_case(u'gimme', special_case)

# check new tokenization
print([w.text for w in nlp(u'gimme that')])  # ['gim', 'me', 'that']
```
Adding special case tokenization rules

```python
1  doc = nlp(u'I like New York in Autumn. ')
2  span = doc[2:4]
3  span.merge()
4  assert len(doc) == 6
5  assert doc[2].text == 'New York'
```
Relation extraction with spaCy

**TASK TO SOLVE**

Extract money and currency values (entities labelled as MONEY) and find the noun phrase they are referring to - for example:

“Net income was $9.4 million compared to the prior year of $2.7 million.”

$9.4 million → Net income.

$2.7 million → the prior year

???

How can we solve this task?
TASK TO SOLVE
Extract money and currency values (entities labelled as MONEY) and find the noun phrase they are referring to - for example:
“Net income was $9.4 million compared to the prior year of $2.7 million.”
$9.4 million $\rightarrow$ Net income.
$2.7$ million $\rightarrow$ the prior year

- Step 1: use spaCy’s **named entity recognizer** to extract money and currency values (entities labelled as MONEY)
- Step 2: use spaCy’s **dependency parser** to find the noun phrase they are referring to.
Step 1: use spaCy's **named entity recognizer** to extract money and currency values (entities labelled as MONEY)

```python
import spacy

model = spacy.load('en_core_web_sm')

doc = nlp(u'Net income was $9.4 million compared to the prior year of $2.7 million.')

print(doc.ents)

>>> $9.4 million, the prior year, $2.7 million

print([token.ent_type_ for token in doc])

['', '', '', 'MONEY', 'MONEY', 'MONEY', '', '', 'DATE', 'DATE', 'DATE', '', 'MONEY', 'MONEY', 'MONEY', '', '']
```
Relation extraction with spaCy

Step 2: use spaCy’s **dependency parser** to find the noun phrases

```python
import spacy
model = spacy.load('en_core_web_sm')
doc = nlp(u'Net income was $9.4 million compared to the prior year of $2.7 million.')
for noun_phrase in doc.noun_chunks:
    print(noun_phrase)
Net income
the prior year
```
Step 3: convert MONEY phrases and noun phrases to one token

```python
import spacy
model = spacy.load('en_core_web_sm')
doc = nlp(u'Net income was $9.4 million compared to the prior year of $2.7 million.')
#your code
for token in doc:
    print(token.text)
Net income
was
$9.4 million
...
Step 4: link named entities (MONEY) to the noun phrases they are referring to: use **dependency labels**

```python
1 from spacy import displacy
displacy.serve(doc, style='dep')
```
Relation extraction with spaCy

- An attribute (**attr**) is a noun phrase that is a non-VP (verbal phrase) predicate usually following a copula verb such as “to be”
- A nominal subject (**nsubj**) is a noun phrase which is the syntactic subject of a clause.
Step 4: link named entities (MONEY) to the noun phrases they are referring to: use **dependency labels**

```python
import spacy
model = spacy.load('en_core_web_sm')
doc = nlp(u'Net income was $9.4 million compared to the prior year of $2.7 million. ')
for token in doc:
    print(token.text, token.dep_, token.head.text,
          [el for el in token.head.lefts])
```

- Net income nsubj was [Net income]
- was ROOT was [Net income]
- $9.4 million attr was [Net income]
- ...
### Relation extraction with spaCy

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net income</td>
<td>nsubj</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>was</td>
<td>ROOT</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>$9.4 million</td>
<td>attr</td>
<td>was</td>
<td>[Net income]</td>
</tr>
</tbody>
</table>

**Diagram:**

```
Net income (ADJ) → was (VERB) → $9.4 million (SYM) → compared (VERB)
```
Relation extraction with spaCy

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.heads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net income</td>
<td>nsubj</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>was</td>
<td>ROOT</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>$9.4 million</td>
<td>attr</td>
<td>was</td>
<td>[Net income]</td>
</tr>
</tbody>
</table>

Net income was $9.4 million compared.
Relation extraction with spaCy

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net income</td>
<td>nsubj</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>was</td>
<td>ROOT</td>
<td>was</td>
<td>[Net income]</td>
</tr>
<tr>
<td>$9.4 million</td>
<td>attr</td>
<td>was</td>
<td>[Net income]</td>
</tr>
</tbody>
</table>
Relation extraction with spaCy

- An object of a preposition (**pobj**) is a noun phrase that modifies the head of a prepositional phrase, which is usually a preposition.
- A prepositional modifier (**prep**) is any prepositional phrase that modifies the meaning of its head.

```
the prior year

DET

of

ADP

$2.7 million.

SYM
```
Relation extraction with spaCy

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td>the prior year of $2.7 million</td>
<td>pobj prep pobj</td>
<td>to the prior year of</td>
<td>[] [] []</td>
</tr>
</tbody>
</table>

The prior year of $2.7 million.
**Relation extraction with spaCy**

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td>the prior year</td>
<td>pobj</td>
<td>to</td>
<td>[]</td>
</tr>
<tr>
<td>of</td>
<td>prep</td>
<td>the prior year</td>
<td>[]</td>
</tr>
<tr>
<td>$2.7 million</td>
<td>pobj</td>
<td>of</td>
<td>[]</td>
</tr>
</tbody>
</table>

```

the prior year
DET

of
ADP

$2.7 million.
SYM
```
Relation extraction with spaCy

<table>
<thead>
<tr>
<th>token</th>
<th>token.dep_</th>
<th>token.head</th>
<th>token.head.lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td>the prior year</td>
<td>pobj</td>
<td>to</td>
<td>[]</td>
</tr>
<tr>
<td>of</td>
<td>prep</td>
<td>the prior year</td>
<td>[]</td>
</tr>
<tr>
<td>$2.7 million</td>
<td>pobj</td>
<td>of</td>
<td>[]</td>
</tr>
</tbody>
</table>
Relation extraction with spaCy

From which sentences the information will be extracted?

- Research and product development expenses were $6 million.
- Net loss for the year ended December 31, 2017 was $11 million.
- an increase of $0.4 million
- greater by $2.9 million

What about a direct object (dobj)? It is a noun phrase that is the accusative object of the verb.

- Revenue exceeded twelve billion dollars.

![Diagram showing the relationship between nsubj and dobj](image)

Revenue exceeded twelve billion dollars.

NOUN VERB NUM
**WordNet** is a large lexical database where nouns, verbs, adjectives and adverbs are grouped into sets of synonyms:
- word sense disambiguation - **Lesk Algorithm** (also implemented in **NLTK**)
- find hypernyms and hyponyms

**spaCy** is open-source library for advanced Natural Language Processing (NLP) in Python
- use pre-trained models (e.g. **en_core_web_sm**)
- use the models to preprocess the text: e.g. tokenization, pos-tagging and lemmatization
- customize tokenizer
- use the models for information extraction: named entities, dependency labels (use both for relation extraction)
References

- https://github.com/nltk/nltk
- https://spacy.io/