WordNet

Marina Sedinkina
- Folien von Desislava Zhekova -

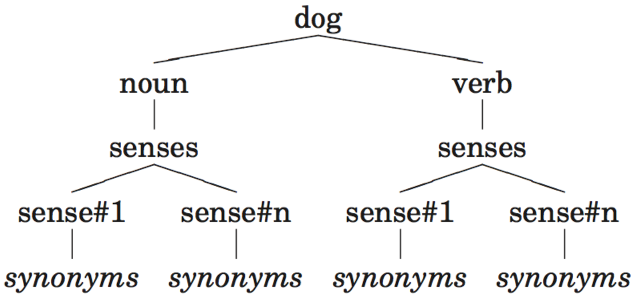
CIS, LMU
marina.sedinkina@campus.lmu.de

December 18, 2018

Outline

WordNet

- 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 [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) 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

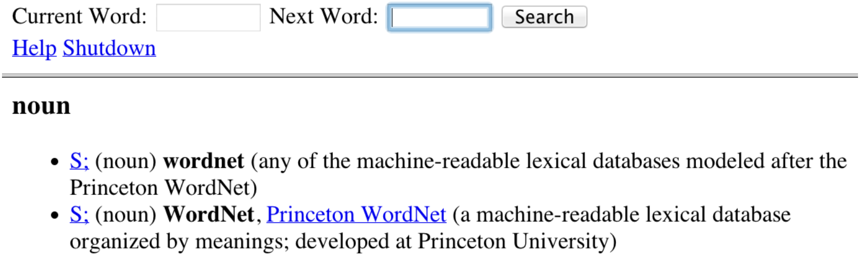
http://globalwordnet.org/

Wordnets in the World

Language	Resource name	Developer(s)	Contact	Online Browsing	License	Other Resources
Afrikaans	Afrikaans WordNet	North-West University, South Africa	Gerhard van Huyssteen Ané Bekker	NO	OPEN FOR ACADEMIC USE	
Albanian	AlbaNet	Vlora University, Vlora, Albania	Ervin Ruci	YES	OPEN (GPL)	
Arabic	Arabic WordNet	Arabic WordNet	Horacio Rodriguez	NO	OPEN	
Multilingual (Arabic/ English/ Malaysian/ Indonesian/ Finnish/ Hebrew/ Japanese/ Persian/ Thai/ French)	Open Multilingual Wordnet	Linguistics and Multilingual Studies, NTU	Francis Bond	NO	OPEN	

WordNet

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



Senses and Synonyms

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:

1. Benz is credited with the invention of the motorcar.
2. Benz is credited with the invention of the automobile.

$\Rightarrow$ Motorcar and automobile are synonyms.

Let's explore these words with the help of WordNet

Senses and Synonyms

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets("motorcar")
    [Synset("car.n.01")]
```

- 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"):

```
>>> wn.synset("car.n.01").lemma_names()
["car", "auto", "automobile", "machine", "
    motorcar"]
```

Senses and Synonyms

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

```
>>> wn.synset("car.n.01").definition()
"a motor vehicle with four wheels; usually propelled
    by an internal combustion engine"
>>> wn.synset("car.n.01").examples()
["he needs a car to get to work"]
```

Senses and Synonyms

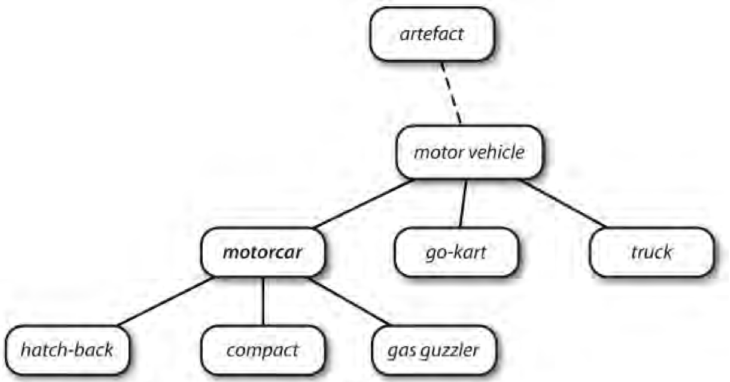
Unlike the words automobile and motorcar, which are unambiguous and have one synset, the word car is ambiguous, having five synsets:

```
>>> 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()
...
["car", "auto", "automobile", "machine", "motorcar"]
["car", "railcar", "railway_car", "railroad_car"]
["car", "gondola"]
["car", "elevator_car"]
["cable_car", "car"]
```

The WordNet Hierarchy

Hypernyms and hyponyms ("is-a relation")

- *motor vehicle* is a hypernym of *motorcar*
- *ambulance* is a hyponym of *motorcar*



The WordNet Hierarchy

```
>>> 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

```
>>> 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"]
```

More Lexical Relations

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

```
>>> 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

```
>>> 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")]
```

More Lexical Relations

Some lexical relationships hold between lemmas, e.g., **antonymy**:

```
>>> 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")]
```

More Lexical Relations

You can see the lexical relations, and the other methods defined on a synset, using `dir()`. For example:

```
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', 'lch_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']
```

Semantic Similarity

If two synsets share a very specific hypernym (low down in the hypernym hierarchy), they must be closely related.

```
>>> 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:

```
>>> 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

```
>>> right.path_similarity(minke)
0.25
>>> right.path_similarity(orca)
0.16666666666666666
>>> right.path_similarity(tortoise)
0.07692307692307693
>>> right.path_similarity(novel)
0.043478260869565216
```

Similarity between nouns

- `("car", "automobile")`
- synsets1("car") = [$synset_{11}$, $synset_{12}$, $synset_{13}$]
  `nltk.corpus.wordnet.synsets("car")`
- synsets2("automobile") = [$synset_{21}$, $synset_{22}$, $synset_{23}$]
  `nltk.corpus.wordnet.synsets("automobile")`
- consider all combinations of synsets formed by the synsets of the words in the word pair `("car”, "automobile”)`
  $[(synset_{11}, synset_{21}), (synset_{11}, synset_{22}), (synset_{11}, synset_{23}), \ldots]$
- determine score of each combination e.g.:
  $synset_{11}$`.path_similarity(`$synset_{21}$`)`
- determine the maximum score → indicator of similarity

Semantic Similarity

???

Can you think of an NLP application for which semantic similarity will be helpful?

Semantic Similarity

???

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.

Polysemy

- The **polysemy** of a word is the number of senses it has.
- The noun **dog** has 7 senses in WordNet:

```
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:
  `nltk.corpus.wordnet.all_lemma_names(POS)`

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

```
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')]
```

Polysemy of nouns

```python
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

Sense	**Definition**
s1: tree	a tree of the olive family
s2: burned stuff	the solid residue left when combustible material is burned

Table: Two senses of **ash**

Lesk Algorithm: Example

Sense	Definition
s1: tree	a tree of the olive family
s2: burned stuff	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

Scores	Context
s1 s2	This cigar burns slowly and creates a stiff ash

Table: Disambiguation of ash with Lesk's algorithm

Lesk Algorithm: Example

Sense	Definition
s1: tree	a tree of the olive family
s2: burned stuff	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

Scores	Context
s1 s2	This cigar burns slowly and creates a stiff ash

Table: Disambiguation of ash with Lesk's algorithm

Lesk Algorithm: Example

Sense	Definition
s1: tree	a tree of the olive family
s2: burned stuff	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

Scores	Context
s1 s2	This cigar burns slowly and
0 1	creates a stiff ash

Table: Disambiguation of ash with Lesk's algorithm

Lesk Algorithm: Example

Sense	Definition
s1: tree	a tree of the olive family
s2: burned stuff	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

Scores	Context
s1 s2	The ash is one of the last trees
???	to come into leaf

Table: Disambiguation of ash with Lesk's algorithm

Lesk Algorithm: Example

Sense	Definition
s1: tree	a tree of the olive family
s2: burned stuff	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

Scores	Context
s1 s2	The ash is one of the last trees
1 0	to come into leaf

Table: Disambiguation of ash with Lesk's algorithm

Lesk Algorithm

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

>>> print(lesk(sent, "bank","n"))
Synset("savings_bank.n.02")
```

Lesk Algorithm

The definitions for "bank" are:

```
>>> from nltk.corpus import wsordnet 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

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?

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.

```
>>> 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)**

```
>>> 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).

```
>>> 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.

```
pos_tags = nltk.pos_tag(tokens)
print pos_tags

[('Augusta', 'NNP'), ('Ada', 'NNP'), ('King', 'NNP'),
(',', ','), ('Countess', 'NNP'), ('of', 'IN'),
('Lovelace', 'NNP'), ('(', 'NNP'), ('10', 'CD'),
('December', 'NNP'), ('1815', 'CD'), ('27', 'CD'),
('November', 'NNP'), ('1852', 'CD'), (')', 'CD'),
(',', ','), ('born', 'NN'), ('Augusta', 'NNP'),
('Ada', 'NNP'), ('Byron', 'NNP'), ('and', 'CC'),
('now', 'RB'), ('commonly', 'RB'), ('known',
'VBN'), ('as', 'IN'), ('Ada', 'NNP'), ... ]
```

Print out all the nouns occurring in the text.

Step 4: Part-of-Speech tagging

```
[ ...  (',', ','), ('born', 'NN'), ('Augusta', 'NNP'),
('Ada', 'NNP'), ('Byron', 'NNP'), ('and', 'CC'),
('now', 'RB'), ('commonly', 'RB'), ('known',
'VBN'), ('as', 'IN'), ('Ada', 'NNP'),  ... ]
```

- CC – coordinating conjunction
- RB – adverb
- IN – preposition
- NN – noun
- JJ – adjective
- VB – verb

Step 4: Part-of-Speech tagging

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

```
>>> 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.

```
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', ... ]
```

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`)

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

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

```
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
```

Ada Lovelace Task: Finding Hypernyms

Reminder:

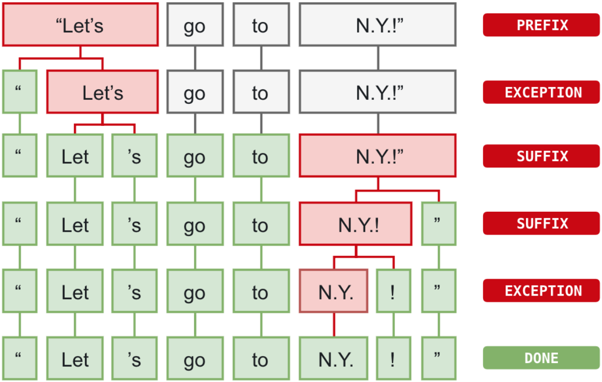
- We have a list of the lemmas of all nouns, `noun_lemmas`.
- Retrieve the synsets for each lemma.
- Check whether it's a hyponym of one of the three synsets of interest.
- Counts the relevant nouns, and collect them.

Tokenization with spaCy

```
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**

LANGUAGE	CODE	LANGUAGE DATA	MODELS
English	`en`	`lang/en`	4 models
German	`de`	`lang/de`	1 model
Spanish	`es`	`lang/es`	2 models
Portuguese	`pt`	`lang/pt`	1 model
French	`fr`	`lang/fr`	2 models
Italian	`it`	`lang/it`	1 model

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}

```
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

```
doc = nlp(u'I like New York in Autumn.')
span = doc[2:4]
span.merge()
assert len(doc) == 6
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?

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

- Step 1: use spaCy's **named entity recognizer** to extract money and currency values (entities labelled as MONEY)
- Step2: use spaCy's **dependency parser** to find the noun phrase they are referring to.

Relation extraction with spaCy

Step 1: use spaCy's **named entity recognizer** to extract money and currency values (entities labelled as MONEY)

```
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

Step2: use spaCy's **dependency parser** to find the noun phrases

```
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
```

Relation extraction with spaCy

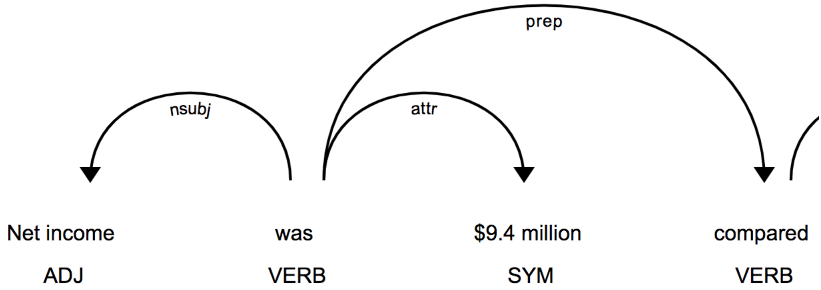
Step 3: convert MONEY phrases and noun phrases to one token

```
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
...
```

Relation extraction with spaCy

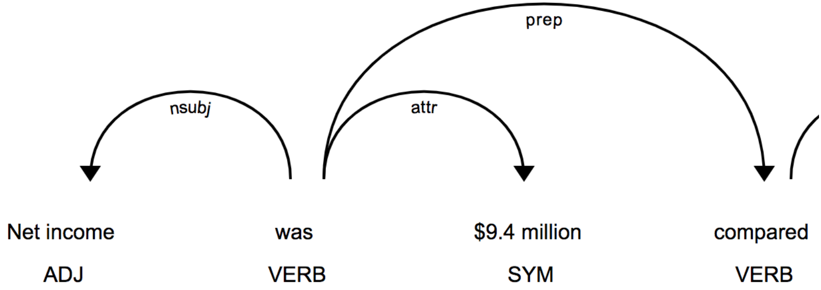
Step4: link named entities (MONEY) to the noun phrases they are referring to: use **dependency labels**

```python
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.



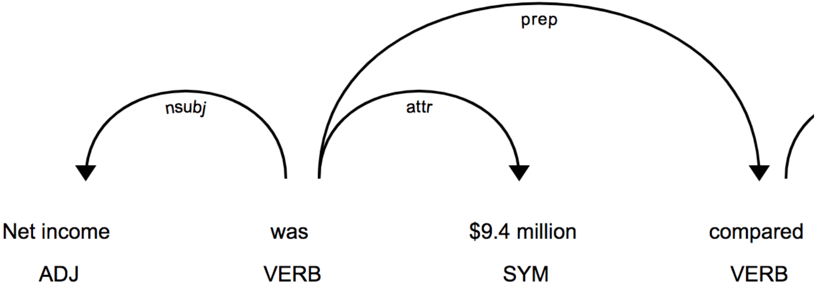
Relation extraction with spaCy

Step4: link named entities (MONEY) to the noun phrases they are referring to: use **dependency labels**

```
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]
 ...
```

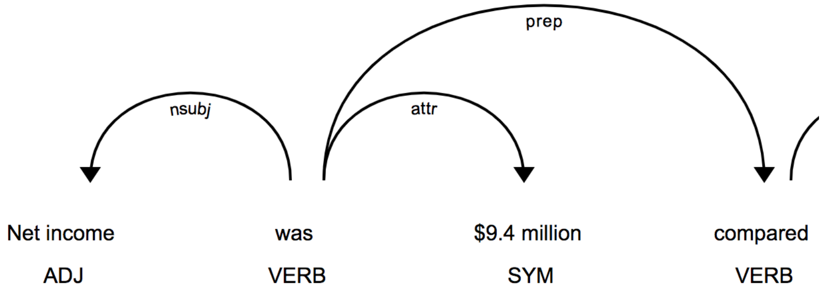
Relation extraction with spaCy

token	token.dep_	token.head	token.head.lefts
Net income	nsubj	was	[Net income]
was	ROOT	was	[Net income]
$9.4 million	**attr**	was	[Net income]



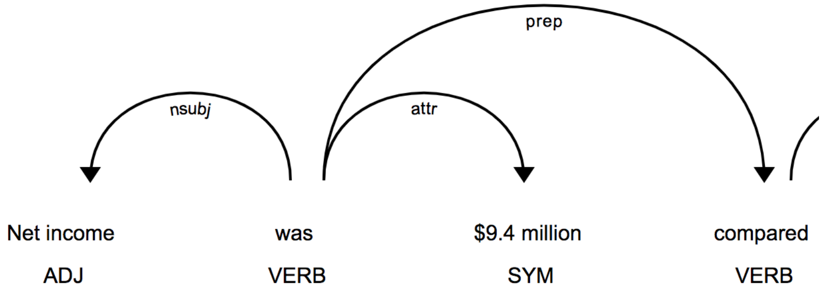
Relation extraction with spaCy

token	token.dep_	token.head	**token.head.lefts**
Net income	nsubj	was	[Net income]
was	ROOT	was	[Net income]
$9.4 million	**attr**	was	**[Net income]**



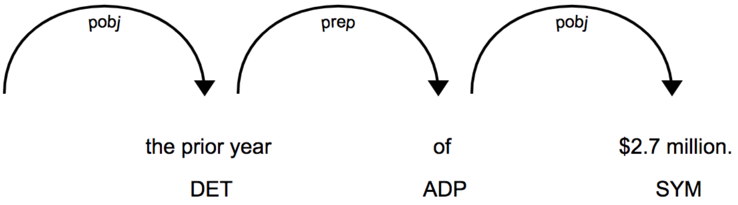
Relation extraction with spaCy

token	token.dep_	token.head	**token.head.lefts**
Net income	**nsubj**	was	[Net income]
was	ROOT	was	[Net income]
$9.4 million	**attr**	was	**[Net income]**



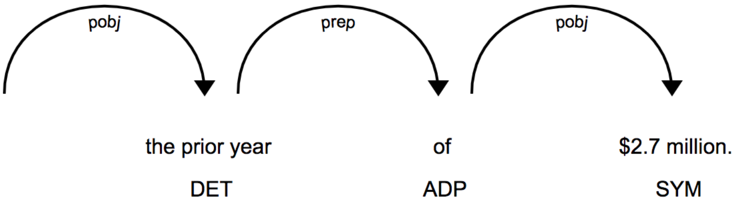
Relation extraction with spaCy

- 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.



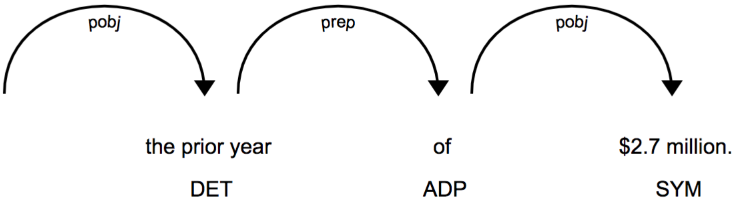
Relation extraction with spaCy

token	token.dep_	token.head	**token.head.lefts**
the prior year	pobj	to	[]
of	prep	the prior year	[]
$2.7 million	**pobj**	of	[]



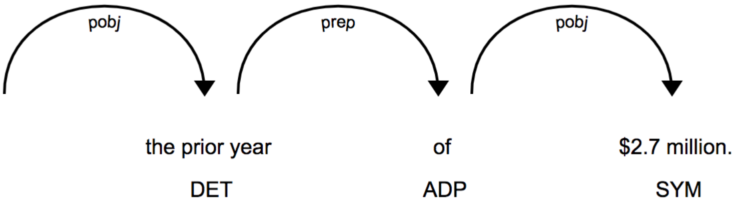
Relation extraction with spaCy

token	token.dep_	token.head	**token.head.lefts**
the prior year	pobj	to	[]
of	**prep**	the prior year	[]
$2.7 million	**pobj**	**of**	[]



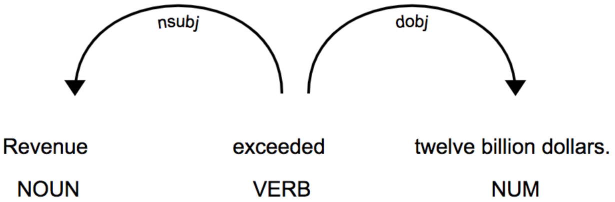
Relation extraction with spaCy

token	token.dep_	token.head	**token.head.lefts**
the prior year	pobj	to	[]
of	**prep**	**the prior year**	[]
$2.7 million	**pobj**	**of**	[]



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.



Conclusion

- **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)


Merry Christmas

References

- http://www.nltk.org/book/
- https://github.com/nltk/nltk
- https://spacy.io/