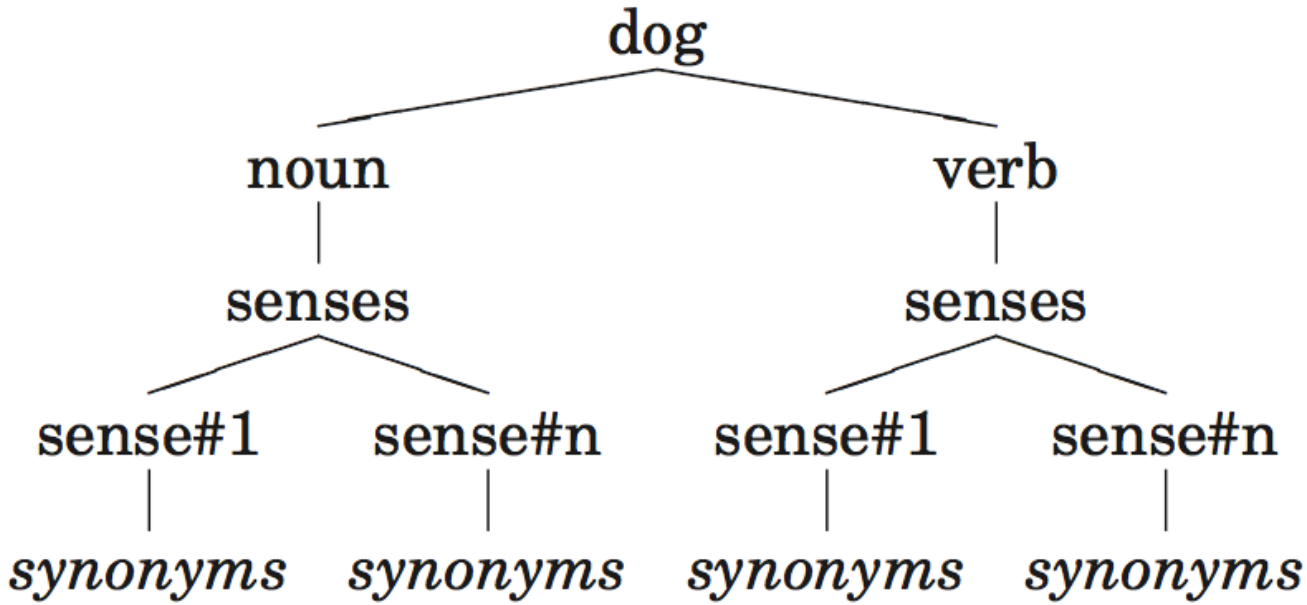
WordNet

Florian Fink
- Folien von Desislava Zhekova -

January 19, 2021

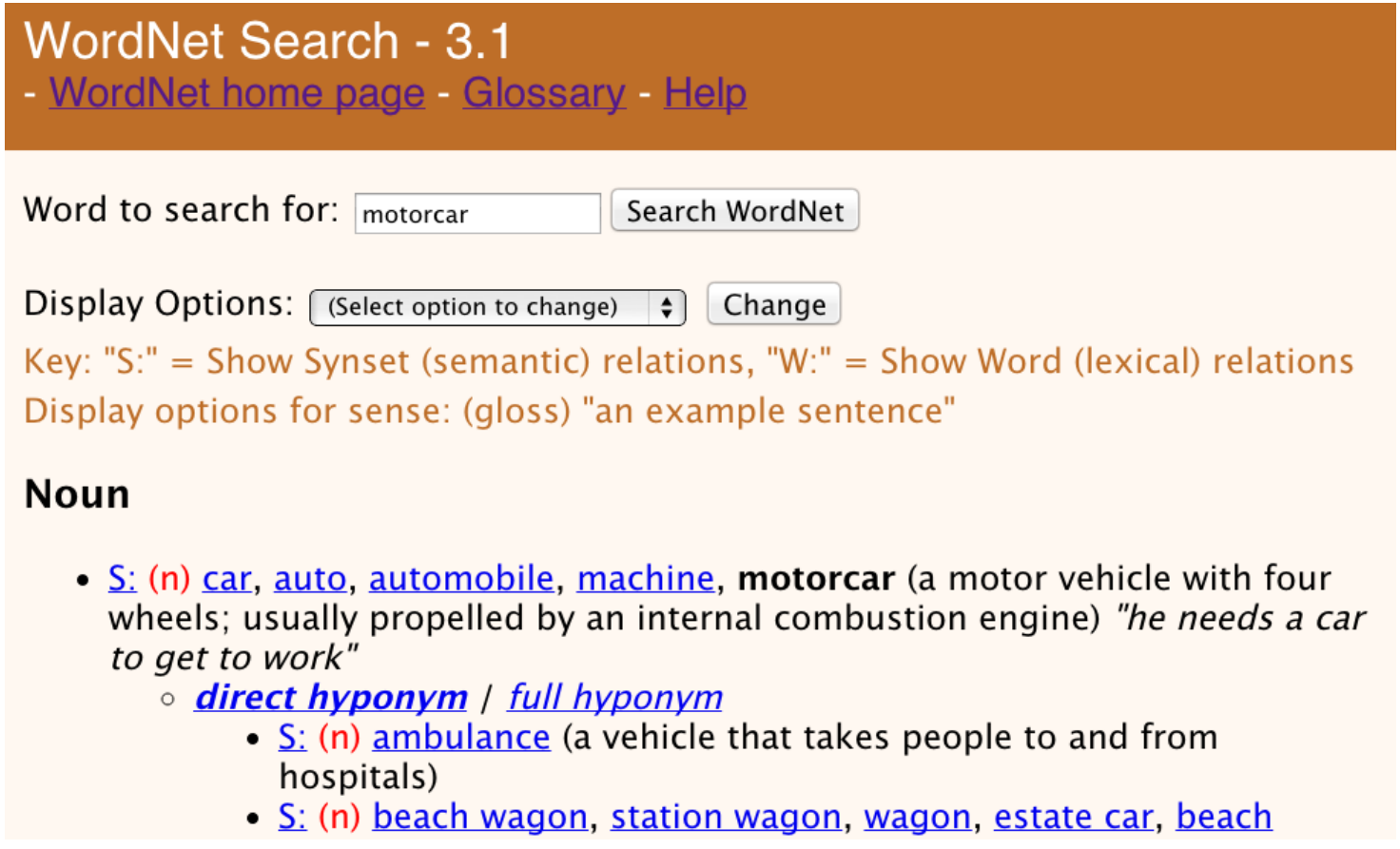
WordNet

- 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

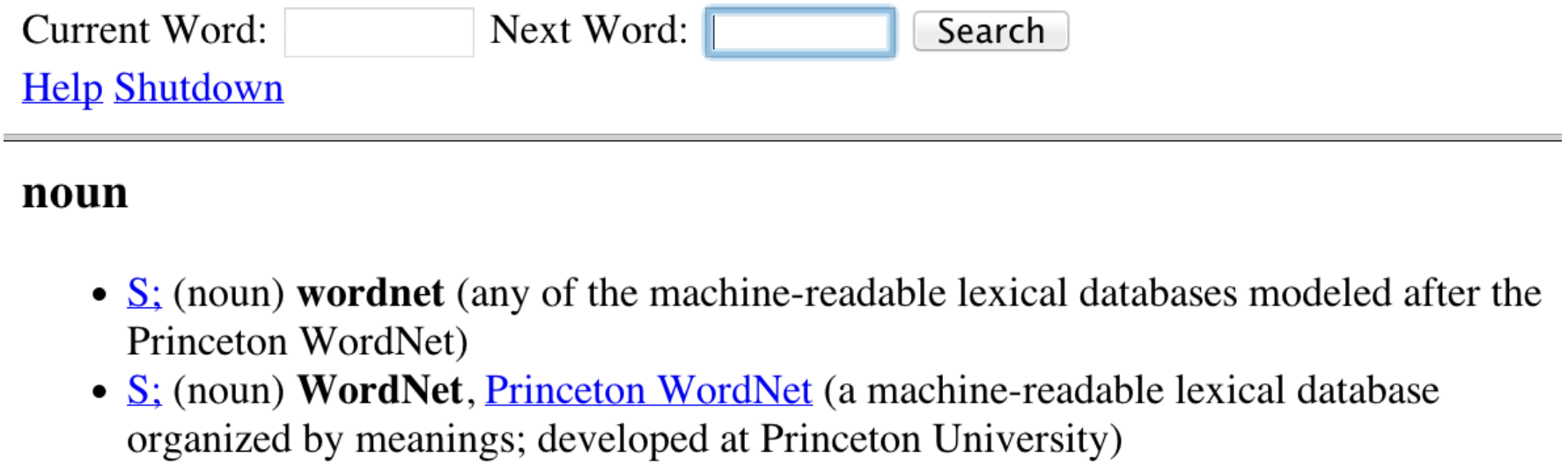
http://globalwordnet.org/

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

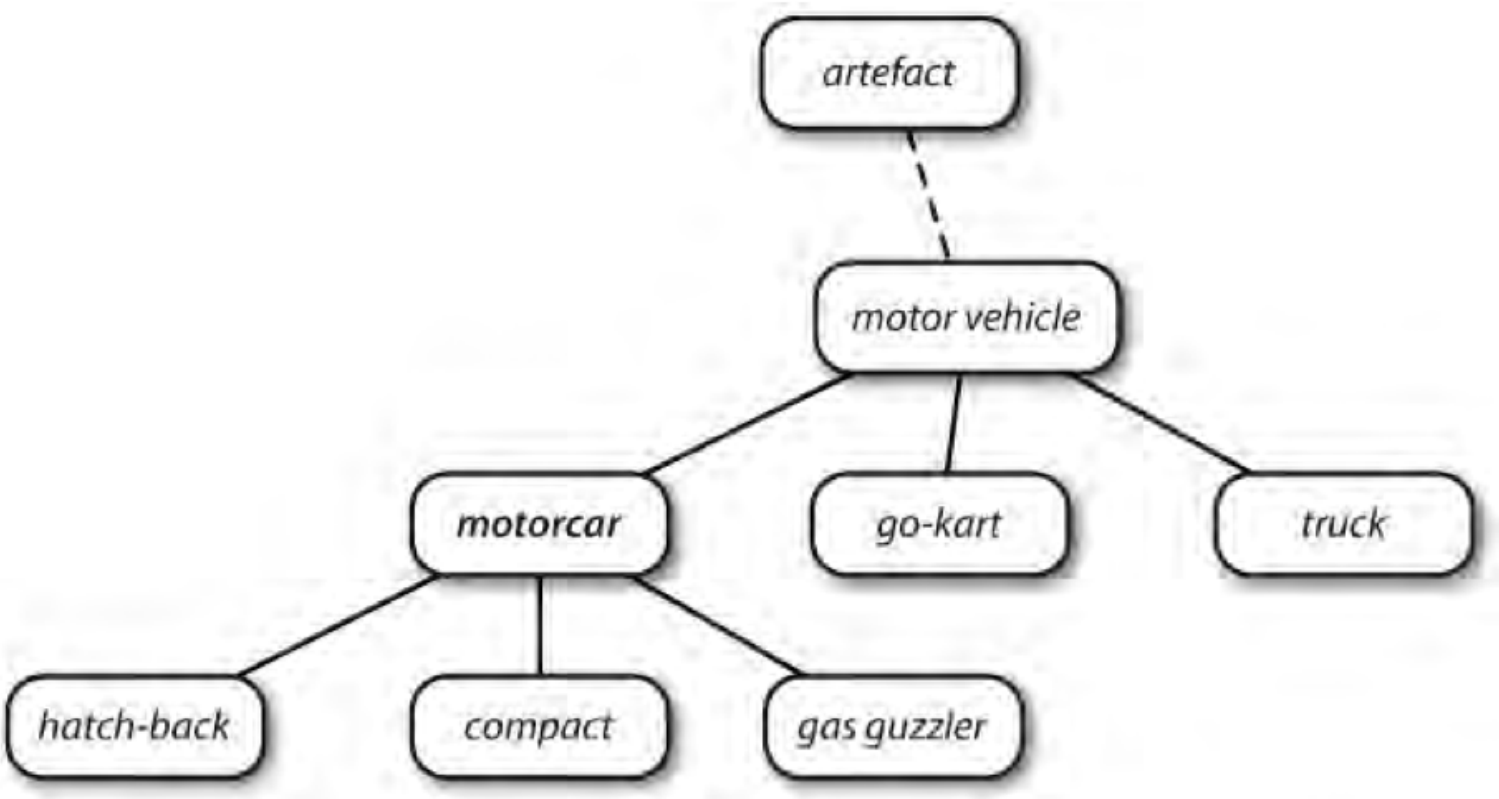
Unlike the words automobile and motorcar, which are unambiguous and have one synset, the word car is ambiguous, having five synsets:

```
>>> 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

```
1 >>> motorcar = wn.synset("car.n.01")
2 >>> types_of_motorcar = motorcar.hyponyms()
3 >>> types_of_motorcar[26]
4 Synset("ambulance.n.01")
5 >>> sorted([lemma.name() for synset in types_of_motorcar
      for lemma in synset.lemmas()])
6 ["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

```
1 >>> wn.synset("tree.n.01").part_meronyms()
2 [Synset("burl.n.02"), Synset("crown.n.07"), Synset("
      stump.n.01"), Synset("trunk.n.01"), Synset("limb.
      n.02")]
3 >>> wn.synset("tree.n.01").substance_meronyms()
4 [Synset("heartwood.n.01"), Synset("sapwood.n.01")]
5 >>> wn.synset("tree.n.01").member_holonyms()
6 [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 can express 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:

```
1  import nltk
2  from nltk.corpus import wordnet as wn
3
4  print(wn.synsets("motorcar"))
5
6
7  print(dir(wn.synsets("motorcar")[0]))
```

Semantic Similarity

Two synsets linked to the same root may have several hypernyms in common. 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)
0.07692307692307692
>>> 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 $\rightarrow$ 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**. This **whale** was huge.

Polysemy

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

```
1 from nltk.corpus import wordnet as wn
2 num_senses=len(wn.synsets("dog","n"))
3
4 print(num_senses)
5 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)`

```
1 from nltk.corpus import wordnet as wn
2 all_lemmas=set(wn.all_lemma_names("n"))
3 print(len(all_lemmas))
4 prints 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"

```
1 from nltk.corpus import wordnet as wn
2 meanings=wn.synsets("car","n")
3 print(meanings)
```

- Sum up the number of meanings of each lemma (restricted to nouns) and devide this by the total number of lemmas

Lesk Similarity

???

Compute the average polysemy of nouns ’car’, ’automobile’, ’motorcar’

```
1
2 senses_motorcar = [Synset("car.n.01")]
```

average polysemy

average_polysemy = ???

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

Normalization

Once the text has been segmented into its tokens (paragraphs, sentences, words), most NLP pipelines do a number of other basic procedures for text normalization, e.g.:

- lowercasing
- stemming
- lemmatization
- stopword removal

Lowercasing

Lowercasing:

```
1 import nltk
2
3 string = "The boy,s cars are different colors."
4 tokens = nltk.word_tokenize(string)
5 lower = [x.lower() for x in tokens]
6 print(" ".join(lower))
7
8 # prints
9 # the boy ,s cars are different colors .
```

- Often, however, instead of working with all word forms, we would like to extract and work with their base forms (e.g. lemmas or stems)
- Thus with **stemming** and **lemmatization** we aim to reduce inflectional (and sometimes derivational) forms to their base forms.

Stemming

Stemming: removing morphological affixes from words, leaving only the word stem.

```
1  import nltk
2
3  string = "The boy,s cars are different colors."
4  tokens = nltk.word_tokenize(string)
5  lower = [x.lower() for x in tokens]
6  stemmed = [stem(x) for x in lower]
7  print(" ".join(stemmed))
8
9  def stem(word):
10     for suffix in ["ing", "ly", "ed", "ious", "ies", "ive", "es",
           "s", "ment"]:
11         if word.endswith(suffix):
12             return word[:-len(suffix)]
13     return word
14 # prints
15 # the boy ,s car are different color .
```

Stemming

Stemming:

```
1  import nltk
2  import re
3
4  string = "The boy,s cars are different colors."
5  tokens = nltk.word_tokenize(string)
6  lower = [x.lower() for x in tokens]
7  stemmed = [stem(x) for x in lower]
8  print(" ".join(stemmed))
9
10 def stem(word):
11     regexp = r"^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$"
12     stem, suffix = re.findall(regexp, word)[0]
13     return stem
14
15 # prints
16 # the boy ,s car are different color .
```

Stemming

NLTK's stemmers:

- **Porter Stemmer** is the oldest stemming algorithm supported in NLTK, originally published in 1979.
  `http://www.tartarus.org/~martin/PorterStemmer/`
- **Lancaster Stemmer** is much newer, published in 1990, and is more aggressive than the Porter stemming algorithm.
- **Snowball stemmer** currently supports several languages: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Porter, Portuguese, Romanian, Russian, Spanish, Swedish.
- **Snowball stemmer**: slightly faster computation time than porter.

Stemming

NLTK's stemmers:

```
1  import nltk
2
3  string = "The boy,s cars are different colors."
4  tokens = nltk.word_tokenize(string)
5  lower = [x.lower() for x in tokens]
6
7  porter = nltk.PorterStemmer()
8  stemmed = [porter.stem(t) for t in lower]
9  print(" ".join(stemmed))
10 # prints
11 # the boy ,s car are differ color .
12
13 lancaster = nltk.LancasterStemmer()
14 stemmed = [lancaster.stem(t) for t in lower]
15 print(" ".join(stemmed))
16 # prints
17 # the boy ,s car ar diff col .
```

Stemming

NLTK's stemmers:

```
1  import nltk
2
3  string = "The boy,s cars are different colors."
4  tokens = nltk.word_tokenize(string)
5  lower = [x.lower() for x in tokens]
6
7  snowball = nltk.SnowballStemmer("english")
8  stemmed = [snowball.stem(t) for t in lower]
9  print(" ".join(stemmed))
10 # prints
11 # the boy ,s car are differ color .
```

Lemmatization

- stemming can often create non-existent words, whereas lemmas are actual words
- **NLTK WordNet Lemmatizer** uses the WordNet Database to lookup lemmas

```
1  import nltk
2  string = "The boy,s cars are different colors."
3  tokens = nltk.word_tokenize(string)
4  lower = [x.lower() for x in tokens]
5  porter = nltk.PorterStemmer()
6  stemmed = [porter.stem(t) for t in lower]
7  print(" ".join(lemmatized))
8  # prints the boy ,s car are differ color .
9  wnl = nltk.WordNetLemmatizer()
10 lemmatized = [wnl.lemmatize(t) for t in lower]
11 print(" ".join(lemmatized))
12 # prints the boy ,s car are different color .
```

Stopword removal:

Stopword removal:

```
1  import nltk
2
3  string = "The boy,s cars are different colors."
4  tokens = nltk.word_tokenize(string)
5  lower = [x.lower() for x in tokens]
6  wnl = nltk.WordNetLemmatizer()
7  lemmatized = [wnl.lemmatize(t) for t in lower]
8
9  content = [x for x in lemmatized if x not in nltk.
       corpus.stopwords.words("english")]
10 print(" ".join(content))
11 # prints
12 # boy ,s car different color .
```

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)
...
```

Lesk Algorithm

Check implementation via

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

```
1  def lesk(context_sentence, ambiguous_word, pos=None,
       synsets=None):
2      context = set(context_sentence)
3      if synsets is None:
4          synsets = wordnet.synsets(ambiguous_word)
5      if pos:
6          synsets = [ss for ss in synsets if str(ss.pos()) ==
               pos]
7      if not synsets:
8          return None
9
10     _, sense = max(
11         (len(context.intersection(ss.definition().split())
               , ss) for ss in synsets
12     )
13     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 Babbages early mechanica
general-purpose computer, the Analytical Engine. ... "
```

Step 2: Sentence Splitting

Split the text into sentences:

```
>>> sentences = nltk.sent_tokenize(text)
["Augusta Ada King, Countess of Lovelace ... "],
["Her notes on the engine ... "],
["Because of this , she is often ... "],
```

Step 3: Tokenize

Split the sentences in to tokens.

```
>>> print tokens
[["Augusta", "Ada", "King", ",", "Countess", ... ],
["Her", "notes", "on", "the", "engine", ... ],
["Because", "of", "this", ",", "she", "is", ... ]
 ... ]
```

Step 4: Part-of-Speech tagging

Find the POS-tag of each token using NLTK's recommended POS tagger `nltk.pos_tag`.

```
>>> print tags
[[("Augusta", "NNP"), ("Ada", "NNP"), ... ],
[("Her", "PRP$"), ("notes", "NNS"), ("on", "IN"), ... ],
 ... ]
```

POS-tags

NLTK provides documentation for each tag, which can be queried using **nltk.help.upenn_tagset**.

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

Step 5: Lemmatize

Put the lemma of each noun from the text into a list.

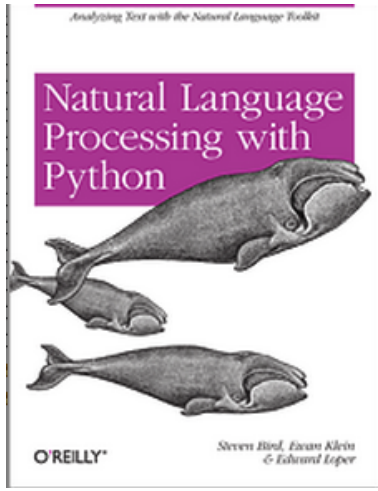
```
>>> from nltk.stem.wordnet import WordNetLemmatizer
>>> from nltk.corpus import wordnet
>>> lemmata = ...
>>> print lemmata
["Augusta", "Ada", ... , "notes", ... ]
```

Step 6: Find Hyperonyms

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

```
>>> relative = wordnet.sysnsets("relative", pos="n")[0]
>>> relative = wordnet.sysnsets("scienece", pos="n")[0]
>>> relative = wordnet.sysnsets("illness", pos="n")[0]
```

References

- 

  http://www.nltk.org/book/
- https://github.com/nltk/nltk
- Christopher D. Manning, Hinrich Schütze 2000. Foundations of Statistical Natural Language Processing. *The MIT Press Cambridge, Massachusetts London, England*. http://ics.upjs.sk/~pero/web/documents/pillar/Manning_Schuetze_StatisticalNLP.pdf