WordNet

Marina Sedinkina
- Folien von Desislava Zhekova -

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

January 9, 2018
Outline

1. WordNet
2. Lesk Algorithm
3. Preprocessing
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.

```
dog
   /\   /
  /   \
noun| verb
   |    |
  /    |
senses| senses
      |    |
sense#1| sense#n| sense#1| sense#n
          |      |       |
synonyms| synonyms| synonyms| synonyms
```
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**
### 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 (Arabic/English/Malaysian/Indonesian/Finnish/Hebrew/Japanese/Persian/Thai/French)</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:  
Search

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:

1. Benz is credited with the invention of the motorcar.
2. 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 $\text{car.n.01}$ (the first noun sense of car).

The entity $\text{car.n.01}$ 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"]
```
Synsets are described with a **gloss** (= definition) and some example sentences

```python
>>> 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"]
```
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()
  ...
["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

```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 can express antonymy:

```python
1 >>> wn.lemma("supply.n.02.supply").antonyms()
2 [Lemma("demand.n.02.demand")]
3 >>> wn.lemma("rush.v.01.rush").antonyms()
4 [Lemma("linger.v.04.linger")]
5 >>> wn.lemma("horizontal.a.01.horizontal").antonyms()
6 [Lemma("vertical.a.01.vertical"), Lemma("inclined.a.02.inclined")]
7 >>> wn.lemma("staccato.r.01.staccato").antonyms()
8 [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:

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

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

print(dir(wn.synsets("motorcar")[0]))
# [ ... common_hypernyms, definition, entailments, examples, frame_ids, hypernym_distances, hypernym_paths, hypernyms, 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 ]
```
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.

```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")]
```
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
>>> right.path_similarity(minke)
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")

\[
\text{synsets1("car")} = [\text{synset}_{11}, \text{synset}_{12}, \text{synset}_{13}]
\]
\[
\text{nltk.corpus.wordnet.synsets("car")}
\]

\[
\text{synsets2("automobile")} = [\text{synset}_{21}, \text{synset}_{22}, \text{synset}_{23}]
\]
\[
\text{nltk.corpus.wordnet.synsets("automobile")}
\]

- consider all combinations of synsets formed by the synsets of the words in the word pair ("car", "automobile")

\[
[ (\text{synset}_{11}, \text{synset}_{21}), (\text{synset}_{11}, \text{synset}_{22}), (\text{synset}_{11}, \text{synset}_{23}), \ldots ]
\]

- determine score of each combination e.g.:

\[
\text{synset}_{11}.\text{path_similarity}(\text{synset}_{21})
\]

- determine the maximum score → 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
  from nltk.corpus import wordnet as wn
  all_lemmas=set(wn.all_lemma_names("n"))
  print(len(all_lemmas))
  # prints 117798
  ```

- Determine meanings of each lemma:
  
  ```python
  from nltk.corpus import wordnet as wn
  meanings=wn.synsets("car","n")
  print(meanings)
  # [Synset( car.n.01 ), Synset( car.n.02 ), ... ]
  ```

- Sum up the number of meanings of each lemma (restricted to nouns) and divide this by the total number of lemmas
Compute the average polysemy of nouns 'car', 'automobile', 'motorcar'

```python
1 all_lemma_nouns = [ car , automobile , motorcar ]
2 senses_car = [ Synset( car.n.01 ) , Synset( car.n.02 ) ,
                   Synset( car.n.03 ) , Synset( cable_car.n.01 )]
3 senses_automobile = [ Synset( car.n.01 )]
4 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

<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 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</td>
<td>This cigar burns slowly and creates a stiff ash</td>
</tr>
<tr>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></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>The ash is one of the last trees to come into leaf</td>
</tr>
<tr>
<td>???</td>
<td>??</td>
</tr>
</tbody>
</table>

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

Sense
s1: tree

Definition
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
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 )
```
The definitions for "bank" are:

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

1. **Synset (bank.n.01)** sloping land (especially the slope beside a body of water)
2. **Synset (depository_financial_institution.n.01)** a financial institution that accepts deposits and channels the money into lending activities
3. **Synset (bank.n.03)** a long ridge or pile
4. **Synset (bank.n.04)** an arrangement of similar objects in a row or in tiers
5. **Synset (bank.n.05)** a supply or stock held in reserve for future use (especially in emergencies)
6. **Synset (bank.n.06)** the funds held by a gambling house or the dealer in some gambling games
7. **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
8. **Synset (savings_bank.n.02)** a container (usually with a slot in the top) for keeping money at home
9. **Synset (bank.n.09)** a building in which the business of banking transacted
10. **Synset (bank.n.10)** a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
11. **Synset (bank.v.01)** tip laterally
12. **Synset (bank.v.02)** enclose with a bank
13. **Synset (bank.v.03)** do business with a bank or keep an account at a bank
Lesk Algorithm

Check implementation via

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

```python
def leks(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
```
Check implementation via:
http://www.nltk.org/_modules/nltk/wsd.html

```python
def lesk(context_sentence, ambiguous_word, pos=None, synsets=None):
    ...

    if not synsets:
        return None

    inters = []
    for ss in synsets:
        defin_words = ss.definition().split()
        intersec_words = context.intersection(defin_words)
        len_iter = len(intersec_words)
        inters.append((len_iter, ss))

    _, sense = max(inters)

    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.
Lesk Similarity

- The Lesk similarity of two concepts is defined as the textual overlap between the corresponding definitions, as provided by a dictionary.

- **Punctuation in definitions should be eliminated**, because they do not have a meaning. If two definitions contain punctuation, the score increases.

- The larger a text, the higher can its score be. It should be normalized to allow a fair comparison → **divide overlap by maximum matching number**
def lesk_similarity(synset1, synset2):
    #TODO find tokens of wordnet definition of synset1, ignore punctuation
    definition_words1 = ...

    #TODO find tokens of wordnet definition of synset2, ignore punctuation
    definition_words2 = ...

    #TODO calculate maximum matching number (length of shortest definition)
    max_match = ...

    #TODO find overlap in definitions, consider words occurring twice
    overlap = ...

    return overlap / max_match

print(lesk_similarity(wn.synset('car.n.01'), wn.synset('wheel.n.01')))
Lesk Similarity

Find overlap in definitions, consider word occurring twice?

1. \( \text{def1} = [ \text{a, motor, vehicle, propelled, by, a, combustion, engine} ] \)
2. \( \text{def2} = [ \text{a, vehicle, that, takes, people, to, a, hospital} ] \)

\( \text{overlap\_number} \)

\( \text{overlap\_number} = ??? \)
<table>
<thead>
<tr>
<th>Preprocessing Step</th>
<th>Processed Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>The boy’s cars are different colors.</td>
</tr>
<tr>
<td>Tokenized</td>
<td>[&quot;The&quot;, &quot;boy’s&quot;, &quot;cars&quot;, &quot;are&quot;, &quot;different&quot;, &quot;colors.&quot;&quot;]</td>
</tr>
<tr>
<td>Punctuation removal</td>
<td>[&quot;The&quot;, &quot;boy’s&quot;, &quot;cars&quot;, &quot;are&quot;, &quot;different&quot;, &quot;colors&quot;]</td>
</tr>
<tr>
<td>Lowecased</td>
<td>[&quot;the&quot;, &quot;boy’s&quot;, &quot;cars&quot;, &quot;are&quot;, &quot;different&quot;, &quot;colors&quot;]</td>
</tr>
<tr>
<td>Stemmed</td>
<td>[&quot;the&quot;, &quot;boy’s&quot;, &quot;car&quot;, &quot;are&quot;, &quot;differ&quot;, &quot;color&quot;]</td>
</tr>
<tr>
<td>Lemmatized</td>
<td>[&quot;the&quot;, &quot;boy’s&quot;, &quot;car&quot;, &quot;are&quot;, &quot;different&quot;, &quot;color&quot;]</td>
</tr>
<tr>
<td>Stopword removal</td>
<td>[&quot;boy’s&quot;, &quot;car&quot;, &quot;different&quot;, &quot;color&quot;]</td>
</tr>
</tbody>
</table>
Tokenization is the process of breaking raw text into its building parts: words, phrases, symbols, or other meaningful elements called tokens.

A list of tokens is almost always the first step to any other NLP task, such as part-of-speech tagging and named entity recognition.
Tokenization

- **token** – is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing
- **type** – is the class of all tokens containing the same character sequence
Tokenization

- What is Token?
  
  Fairly trivial: chop on whitespace and throw away punctuation characters.

- Tricky cases: various uses of the apostrophe for possession and contractions?
Tokenization

Mrs. O’Reilly said that the girls’ bags from H&M’s shop in New York aren’t cheap.

<table>
<thead>
<tr>
<th>Mrs.</th>
<th>&quot;Mrs.”; &quot;Mrs”, &quot;.&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>O’Reilly</td>
<td>&quot;O’Reilly”; &quot;OReilly”; &quot;O”; &quot;Reilly”; &quot;O”, &quot;’”, &quot;Reilly”;</td>
</tr>
<tr>
<td>aren’t</td>
<td>&quot;aren’t”; &quot;arent”; &quot;are”; &quot;n’t”; &quot;aren”, &quot;t”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Tokenize manually the following sentence. How many tokens do you get?

Mrs. O’Reilly said that the girls’ bags from H&M’s shop in New York aren’t cheap.
Tokenize manually the following sentence:
Mrs. O’Reilly said that the girls’ bags from H&M’s shop in New York aren’t cheap.

Answer
NLTK returns the following 20 tokens:
["Mrs.", "O’Reilly", "said", "that", "the", "girls", ",", "bags", "from", "H", ",", "&", ",", "M", ",", ",", "shop", ",", "in", ",", "New", ",", "York", ",", "are", ",", ",", "n’t", "cheap", ",", "."]
Most decisions need to be met depending on the language at hand. Some problematic cases for English include:

- hyphenation – *ex-wife*, *Cooper-Hofstadter*, *the bazinga-him-again maneuver*
- apostrophe – *O’Reilly*, *aren’t*
- other cases – *H&M’s*
Tokenization can be approached at any level:

- word segmentation
- sentence segmentation
- paragraph segmentation
- other elements of the text
NLTK comes with a whole bunch of tokenization possibilities:

```python
>>> from nltk import word_tokenize, wordpunct_tokenize
>>> s = "Good muffins cost $3.88\nin New York.
Please buy me\n\ntwo of them.\n\nThanks."

>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']

>>> wordpunct_tokenize(s)
['Good', 'muffins', 'cost', '$', '3', '.', '88', 'in', 'New', 'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```
NLTK comes with a whole bunch of tokenization possibilities:

```python
>>> from nltk.tokenize import *
>>> # same as s.split():
>>> WhitespaceTokenizer().tokenize(s)
[ Good, muffins, cost, $3.88, in, New, York, Please, buy, me, two, of, them, Thanks. ]
```

```python
>>> # same as s.split(" "):  # same as s.split():
>>> SpaceTokenizer().tokenize(s)
[ Good, muffins, cost, $3.88\nin, New, York, Please, buy, me\ntwo, of, them\nThanks. ]
```
NLTK comes with a whole bunch of tokenization possibilities:

```python
>>> # same as s.split(\n):
>>> LineTokenizer(blanklines=keep).tokenize(s)
[Good muffins cost $3.88, in New York. Please buy
 me, two of them., , Thanks. ]

>>> # same as [l for l in s.split(\n) if l.strip()]:
>>> LineTokenizer(blanklines=discard).tokenize(s)
[Good muffins cost $3.88, in New York. Please buy
 me, two of them., , Thanks. ]

>>> # same as s.split(\t):
>>> TabTokenizer().tokenize(a\tb c\n\t d)
[a, b c\n, d ]
NLTK PunktSentenceTokenizer: divides a text into a list of sentences

```python
>>> import nltk.data
>>> text = "Punkt knows that the periods in Mr. Smith and Johann S. Bach do not mark sentence boundaries. And sometimes sentences can start with non–capitalized words. i is a good variable name."

>>> sent_detector = nltk.data.load( tokenizers/punkt/english.pickle )

>>> print \n−−−−−
 . join(sent_detector.tokenize(text.strip()))

# Punkt knows that the periods in Mr. Smith and Johann S. Bach do not mark sentence boundaries.

# −−−−−

# And sometimes sentences can start with non–capitalized words.

# −−−−−

# i is a good variable name.
```
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:

```python
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
print(" ".join(lower))

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

```python
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
stemmed = [stem(x) for x in lower]
print(" ".join(stemmed))

def stem(word):
    for suffix in ["ing", "ly", "ed", "ious", "ies", "ive", "es", "s", "ment"]:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word

# prints
# the boy's car are different color .
```
Stemming:

```python
import nltk
import re

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
stemmed = [stem(x) for x in lower]
print(" ".join(stemmed))

def stem(word):
    regexp = r"^(.*?)(ing|ly|ious|ies|ive|es|s|ment)?$"
    stem, suffix = re.findall(regexp, word)[0]
    return stem

# prints
# 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/](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:

```python
import nltk

string = "The boy´s cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]

porter = nltk.PorterStemmer()
stemmed = [porter.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# the boy´s car are differ color.

lancaster = nltk.LancasterStemmer()
stemmed = [lancaster.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# the boy´s car ar diff col.
```
Stemming

NLTK’s stemmers:

```python
import nltk

string = "The boy´s cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]

snowball = nltk.SnowballStemmer("english")
stemmed = [snowball.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# 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

```python
import nltk
string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
porter = nltk.PorterStemmer()
stemmed = [porter.stem(t) for t in lower]
print(" ".join(lemmatized))
# prints the boy's car are differ color.
wnl = nltk.WordNetLemmatizer()
lemmatized = [wnl.lemmatize(t) for t in lower]
print(" ".join(lemmatized))
# prints the boy's car are different color.
```
Stopword removal:

```python
import nltk

# Importing the necessary library

string = "The boy's cars are different colors."

tokens = nltk.word_tokenize(string)

# Tokenizing the string

lower = [x.lower() for x in tokens]

# Converting all tokens to lowercase

wnl = nltk.WordNetLemmatizer()

# Initializing the WordNetLemmatizer

lemmatized = [wnl.lemmatize(t) for t in lower]

# Lemmatizing the tokens

content = [x for x in lemmatized if x not in nltk.corpus.stopwords.words("english")]

# Filtering out stopwords

print(" ".join(content))

# Prints the content without stopwords

# prints

# boy's car different color.
```
References

- https://github.com/nltk/nltk
  
  http://ics.upjs.sk/~pero/web/documents/pillar/Manning_Schuetze_StatisticalNLP.pdf