Summary

Marina Sedinkina

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

January 15, 2019

Outline

NLP tasks

In most NLP tasks, we are searching for a specific answer to given questions:

- **Sentiment Analysis:** Is this context positive or rather negative?
- **Text Classification:** is the task of assigning predefined categories to the text documents.
- **Language Identification:** is the task of automatically detecting the language present in a document.
- **Word Sense Disambiguation (WSD):** What is the meaning of the word in this context?
- **POS tagging:** What is the POS tag of the current word?

The two camps: Rule-based and Machine Learning

Rule-based:

- **Sentiment Analysis:** if the context contains words like *great*, *perfect*, *sunny*, then it is positive

The two camps: Rule-based and Machine Learning

Rule-based:

- **Sentiment Analysis:** if the context contains words like *great*, *perfect*, *sunny*, then it is positive
- **Text Classification:** if the document contains words like *elections*, *vote*, *president*, then its category is *politics*

The two camps: Rule-based and Machine Learning

Rule-based:

- **Sentiment Analysis:** if the context contains words like *great*, *perfect*, *sunny*, then it is positive
- **Text Classification:** if the document contains words like *elections*, *vote*, *president*, then its category is *politics*
- **Language Identification:** if the document contains umlaut, then the language present in a document is German

The two camps: Rule-based and Machine Learning

Rule-based:

- **Sentiment Analysis:** if the context contains words like *great*, *perfect*, *sunny*, then it is positive
- **Text Classification:** if the document contains words like *elections*, *vote*, *president*, then its category is *politics*
- **Language Identification:** if the document contains umlaut, then the language present in a document is German
- **WSD:** compare the tokens of all possible definitions of the word with its context tokens and pick the meaning with highest overlap (**Lesk algorithm**)

The two camps: Rule-based and Machine Learning

Rule-based:

- **Sentiment Analysis:** if the context contains words like *great*, *perfect*, *sunny*, then it is positive
- **Text Classification:** if the document contains words like *elections*, *vote*, *president*, then its category is *politics*
- **Language Identification:** if the document contains umlaut, then the language present in a document is German
- **WSD:** compare the tokens of all possible definitions of the word with its context tokens and pick the meaning with highest overlap (**Lesk algorithm**)
- **POS tagging:** if the word ends in *ed*, label it as a past tense verb

The two camps: Rule-based and ML

However, NLP tasks can be solved without having to apply a predefined set of rules. We used a **machine learning approach**.

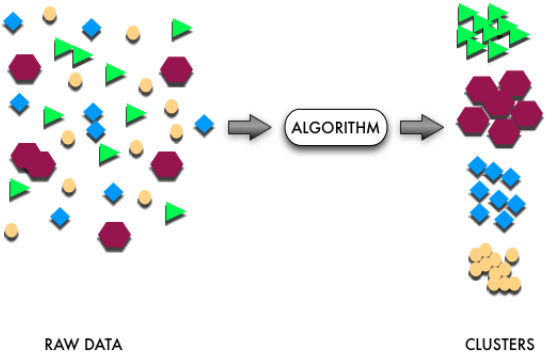
Machine Learning

Machine learning is tightly connected to artificial intelligence:

- to understand, design and improve the algorithms that can be used to build a system that is capable of learning from big amounts of data $\rightarrow$ to develop models
- making autonomous decisions about new/unseen data using these models

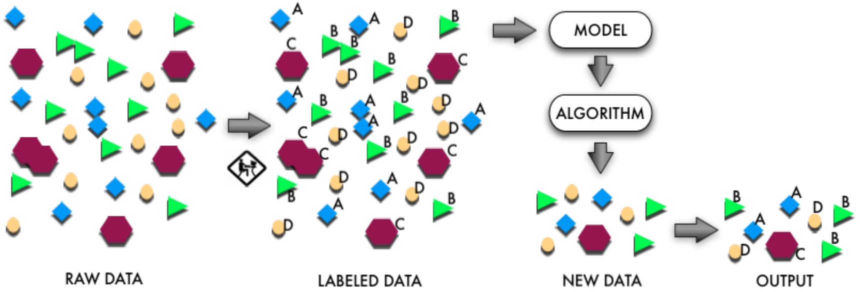
Unsupervised ML: Clustering

no label given, purely based on the given raw data → find common structure in data



Supervised ML: Classification

data labeled with the correct answers to learn from



Classification

Classification:

- choose the correct label (class)
- select the class from a predefined set
- base the decision on specific information collected for each example (so called features)

Classification. Example

Text Classification:

- choose the correct category of the document
- the category is selected from a given set of categories
- base the decision on the features for this document
- features are numerical statistics (TF-IDF) from document

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky

#ignore stopwords and create vocabulary
```

$$\mathrm{E}(t) = \begin{cases} \text{"blue"} \\ \text{"sun"} \\ \text{"bright"} \\ \text{"sky"} \end{cases}$$

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky

#ignore stopwords and create vocabulary E(t)
```

$$\mathrm{E}(t) = \begin{cases} ''\mathit{blue}'' \\ ''\mathit{sun}'' \\ ''\mathit{bright}'' \\ ''\mathit{sky}'' \end{cases}$$

$$\mathrm{tf}(t,d) = \frac{\sum\limits_{x \in d} \mathrm{fr}(x,t)}{max_{t' \in d} tf(t',d)}, \qquad \mathrm{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky.
```

Vocabulary $\mathrm{E}(t)$ contains {blue,sun,bright,sky}

$$\mathrm{tf}(t,d) = \frac{\sum_{x\in d} \mathrm{fr}(x,t)}{max_{t' \in d} tf(t',d)}, \qquad \mathrm{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

$$\vec{v_{d_n}} = (\mathrm{tf}(t_1,d_n), \mathrm{tf}(t_2,d_n), \mathrm{tf}(t_3,d_n), \ldots, \mathrm{tf}(t_n,d_n))$$
$$\vec{v_{d_2}} = (\mathrm{tf}(t_1,d_2), \mathrm{tf}(t_2,d_2), \mathrm{tf}(t_3,d_2), \ldots, \mathrm{tf}(t_n,d_2))$$

???

$$\vec{v_{d_2}} = (???)$$

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky.
```

Vocabulary $\mathrm{E}(t)$ contains {blue,sun,bright,sky}

$$\mathrm{tf}(t,d) = \frac{\sum_{x \in d} \mathrm{fr}(x,t)}{max_{t' \in d} tf(t',d)}, \qquad \mathrm{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

$$\vec{v_{d_n}} = (\mathrm{tf}(t_1, d_n), \mathrm{tf}(t_2, d_n), \mathrm{tf}(t_3, d_n), \ldots, \mathrm{tf}(t_n, d_n))$$
$$\vec{v_{d_2}} = (\mathrm{tf}(t_1, d_2), \mathrm{tf}(t_2, d_2), \mathrm{tf}(t_3, d_2), \ldots, \mathrm{tf}(t_n, d_2))$$

???

$\vec{v_{d_2}} = (0\ 0.5\ 1\ 0.5)$

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky
```

Vocabulary $\mathrm{E}(t)$ contains {blue,sun,bright,sky}

$$\mathrm{tf}(t,d) = \frac{\sum\limits_{x \in d} \mathrm{fr}(x,t)}{max_{t' \in d} tf(t',d)}, \qquad \mathrm{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

$$\vec{v_{d_n}} = (\mathrm{tf}(t_1,d_n), \mathrm{tf}(t_2,d_n), \mathrm{tf}(t_3,d_n), \ldots, \mathrm{tf}(t_n,d_n))$$
$$\vec{v_{d_1}} = (\mathrm{tf}(t_1,d_1), \mathrm{tf}(t_2,d_1), \mathrm{tf}(t_3,d_1), \ldots, \mathrm{tf}(t_n,d_1))$$

tf

$\vec{v_{d_1}} = (???)$

TF-IDF statistics

```
Document Set:
d1: The sky is blue.
d2: The sun is bright, the bright sky.
```

Vocabulary $\mathrm{E}(t)$ contains {blue,sun,bright,sky}

$$\mathrm{idf}(t) = \log_{10}\frac{|D|}{df_t}, \mathrm{tf\text{-}idf}(t) = \mathrm{tf}(t,d) \times \mathrm{idf}(t)$$

$$\mathrm{idf}(t = blue) = \log_{10}\frac{|D|}{df_t} = \log_{10}\frac{2}{1} \sim 0.3$$

$$\mathrm{idf}(t = sun) = \log_{10}\frac{|D|}{df_t} = \log_{10}\frac{2}{1} \sim 0.3$$

$$\mathrm{idf}(t = bright) = \log_{10}\frac{|D|}{df_t} = \log_{10}\frac{2}{1} \sim 0.3$$

$$\mathrm{idf}(t = sky) = \log_{10}\frac{|D|}{df_t} = \log_{10}\frac{2}{2} = 0$$

tf-idf

$\vec{v_{d_2}} =$ (0*0.3 0.5*0.3 1*0.3 0.5*0) = (0 0.15 0.3 0)

TF-IDF statistics

- **tf** $\rightarrow$ the weight how import the term in the document
- **idf** $\rightarrow$ diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely
- **tf-idf** $\rightarrow$ the product of two statistics

K Nearest Neighbors Classification

- **Classification rule**

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar
  - assign the category of this nearest neighbor

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar
  - assign the category of this nearest neighbor
- **Generalization**: take k closest neighbors instead of one

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar
  - assign the category of this nearest neighbor
- **Generalization**: take k closest neighbors instead of one
- **Cosine similarity** can be used to measure similarity between objects

$$cos(\vec{q},\vec{d}) = \frac{\vec{q}*\vec{d}}{|\vec{q}|*|\vec{d}|} = \frac{\sum_{i=1}^{V} q_i*d_i}{|\vec{q}|*|\vec{d}|}$$

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar
  - assign the category of this nearest neighbor
- **Generalization**: take k closest neighbors instead of one
- **Cosine similarity** can be used to measure similarity between objects
  $cos(\vec{q}, \vec{d}) = \frac{\vec{q} * \vec{d}}{|\vec{q}| * |\vec{d}|} = \frac{\sum_{i=1}^{V} q_i * d_i}{|\vec{q}| * |\vec{d}|}$
- Objects are represented by vectors (feature vectors)

K Nearest Neighbors Classification

- **Classification rule**
  - classify a new object
  - find the object in the training set that is most similar
  - assign the category of this nearest neighbor
- **Generalization**: take k closest neighbors instead of one
- **Cosine similarity** can be used to measure similarity between objects
  $cos(\vec{q}, \vec{d}) = \frac{\vec{q} * \vec{d}}{|\vec{q}| * |\vec{d}|} = \frac{\sum_{i=1}^{V} q_i * d_i}{|\vec{q}| * |\vec{d}|}$
- Objects are represented by vectors (feature vectors)
- Feature vectors of documents are TF-IDF statistics and cosine similarity is an indicator how close the documents are in the semantics of their content

Cosine similarity

Cosine similarity can be used to measure similarity between objects

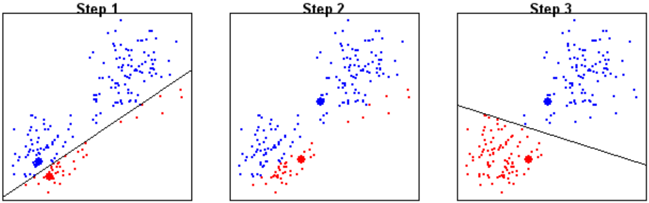
$$cos(\vec{q}, \vec{d}) = \frac{\vec{q} * \vec{d}}{|\vec{q}| * |\vec{d}|} = \frac{\sum_{i=1}^{V} q_i * d_i}{|\vec{q}| * |\vec{d}|}$$

```
import math
def dot_product(v1, v2):
    return sum([value1*value2 for value1, value2
                in zip(v1,v2)])

def cosin_sim(v1, v2):
    prod = dot_product(v1, v2)
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)

cosin_sim([1,2],[3,4])
>>> 0.9838699100999074
```

K-Means Clustering

Goal: find similarities in the data points and group similar data points together

- randomly initialize cluster centroids
- assign each point to the centroid to which it is closest
- recompute cluster centroids
- go back to 2 until nothing changes (or it takes too long)



K-nearest neighbors vs. K-Means

- K-means is a **clustering** algorithm → partitions points into K clusters: points in each cluster tend to be near each other
- K-means is a **unsupervised** algorithm → points have no external classification
- K-nearest neighbors is a **classification** algorithm → determines the classification of a new point
- K-nearest neighbors is a **supervised** algorithm → classifies a point based on the known classification of other points.

Basic Text Statistics

- `len(text)` – extract the number of **tokens** (the technical name for a sequence of characters. It can be a word but also punctuation symbol or smiles from chat corpus) in text
- `len(set(text))` – extract the number of unique tokens (**types**) in text (**vocabulary of text**). You can also use `nltk.text.Text.vocab()`.
- `sorted(set(text))` – extract the number of item types in text in sorted order
- `len(text) / len(set(text))` – lexical diversity of the text
- `sum([len(w) for w in text])/len(text)` – average word length

Brown Corpus Statsistics

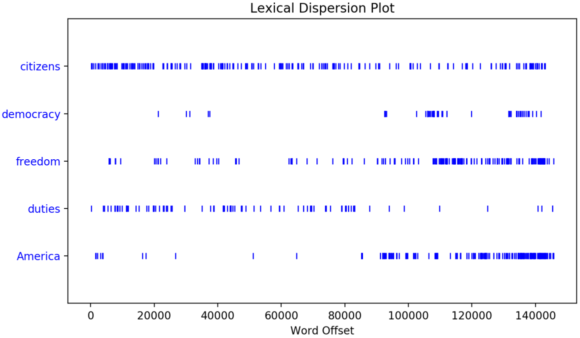
- **hapaxes**: words that only occur once in the text
- hapaxes in the Brown Corpus: `...'Ashley', 'Ashikaga', 'Asher', ...`

Lexical Dispersion Plots

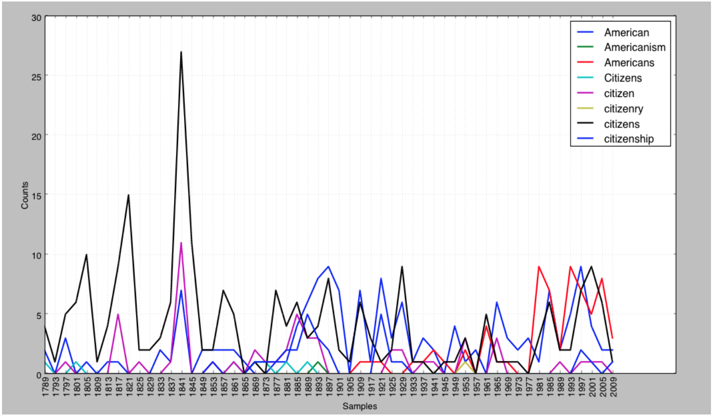
- Location of a word in the text can be displayed using a **dispersion plot**
- Dispersion plots are good for **diachronic language studies** (the exploration of natural language when time is considered as a factor)

Diachronic Language Studies

```
>>> from nltk.book import *
text4.dispersion_plot(["citizens", "democracy", "freedom",
    "duties","America"])
```



Diachronic Language Studies

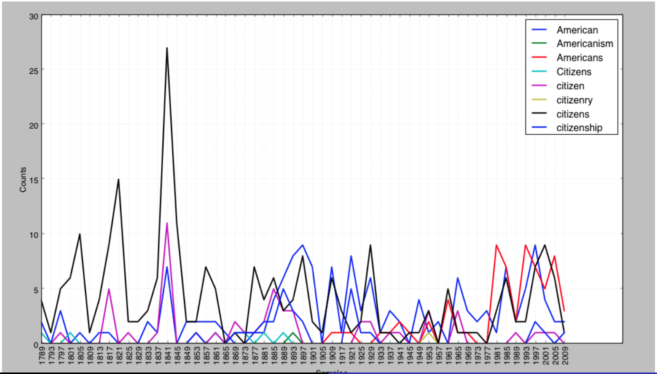


Diachronic Language Studies. Conditional Frequency Distributions (CFD)

```
import nltk
from nltk.corpus import inaugural

cfd = nltk.ConditionalFreqDist((w, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ["american", "citizen"]
    if w.lower().startswith(target))
print(cfd.plot())
```

Diachronic Language Studies

- 8 conditions: "American","Americanism","Americans",...
- for each condition we create a frequency distribution over the years



Diachronic Language Studies

How many conditions will be generated here?

```
import nltk
from nltk.corpus import inaugural
print(inaugural.fileids())
>>>['1789-Washington.txt', '1793-Washington.txt','1797-
    Adams.txt', ...
cfd = nltk.ConditionalFreqDist((w, fileid[:4])
                for fileid in inaugural.fileids()
                for w in inaugural.words(fileid)
                for target in ["american" , "free", "power"]
                if w == target)
print(cfd.conditions())
>>> ???
```

CFD: Generating Random Text

```
import nltk

text = nltk.corpus.genesis.words("english-kjv.txt")
bigrams = nltk.bigrams(text)
cfd = nltk.ConditionalFreqDist(bigrams)

print(cfd.conditions())
>>> ['In', 'the', 'beginning', 'God', 'created', ... ]
```

We treat each word as a condition, and for each one we create a frequency distribution over the following words

CFD: Generating Random Text

```
import nltk

text = nltk.corpus.genesis.words("english-kjv.txt")
bigrams = nltk.bigrams(text)
cfd = nltk.ConditionalFreqDist(bigrams)

print(list(cfd["living"]))
>>>['creature', 'thing', 'soul', '.', 'substance', ',']

print(list(cfd["living"].values()))
>>> [7, 4, 1, 1, 2, 1]

print(cfd["living"].max())
>>> creature
```

Most likely token in that context is "creature"

CFD: Language Identification

```
import nltk
from nltk.corpus import udhr

def build_language_models(list_param, dict_param):
    return nltk.ConditionalFreqDist((language, char_bigram)
                for language in list_param
                for word in dict_param[language]
                for char_bigram in nltk.bigrams(word.lower()))

languages = ['English', 'German_Deutsch']
language_base = dict((list_item, udhr.words(list_item + '-Latin1'))
    for list_item in languages)
language_model_cfd = build_language_models(languages,
    language_base)
text1 = "Peter had been to the office before they arrived."
text2 = "Das ist ein schon recht langes deutsches Beispiel."
print(guess_lang(language_model_cfd, text1))
print(guess_lang(language_model_cfd, text2))
```

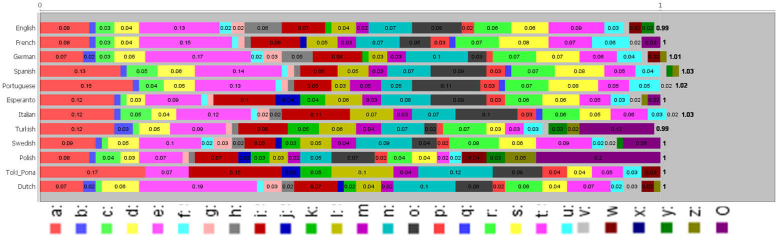
CFD: Language Identification

```
def guess_lang(cfd_param, string_param):
    max_score = 0
    for condition in cfd_param.conditions():
        counter = 0
        for word in string_param.split():
            word = word.lower()
            for char_bigram in nltk.bigrams(word):
                counter = counter + cfd_param[condition].freq(
                    char_bigram)
        if counter > max_score:
            max_language = condition
            max_score = counter
    return max_language
```

Language Guesser Task

- The distribution of characters in a languages of the same language family is usually not very different.
- Thus, it is difficult to differentiate between those languages using a unigram character model $\rightarrow$ use bigram models.

Letters Frequencies in Different Latin Languages



Collocations and Bigrams

- Bigrams are a list of word pairs extracted from a text

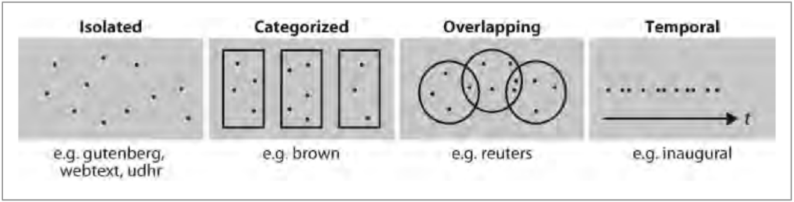
```
>>> from nltk import bigrams
>>>list(bigrams(["more","is","said","than","done"]))
>>>[('more', 'is'), ('is', 'said'), ('said', 'than'),
    ('than', 'done')]
```

- A **collocation** is a sequence of words that occur together unusually often: essentially just frequent bigrams (*red wine*, *United States*)

Corpora Structure

Corpora are designed to achieve specific goal in NLP:

- **Brown Corpus:** resource for studying systematic differences between genres (*stylistics*) → `type of categorized structure`
- **Inaugural Adress Corpus**: used for *diachronic language studies* → `type of temporal structure`



Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text
- *con_freq = ConditionalFreqDist(list_of_tuples)* – calculates conditional frequencies

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text
- *con_freq = ConditionalFreqDist(list_of_tuples)* – calculates conditional frequencies
- *Word lists*

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text
- *con_freq = ConditionalFreqDist(list_of_tuples)* – calculates conditional frequencies
- *Word lists*
  - *nltk.corpus.stopwords* → to filter out words with little lexical content such as **the**, **to**, **a**

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text
- *con_freq = ConditionalFreqDist(list_of_tuples)* – calculates conditional frequencies
- *Word lists*
  - *nltk.corpus.stopwords* → to filter out words with little lexical content such as **the**, **to**, **a**
  - *nltk.corpus.names* → **Anaphora Resolution**

Lexical Resources (Lexicon)

Lexical resource, or lexicon, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions):

- *vocab = sorted(set(my_text))* – builds the vocabulary of *my_text*
- *word_freq = FreqDist(my_text)* – counts the frequency of each word in the text
- *con_freq = ConditionalFreqDist(list_of_tuples)* – calculates conditional frequencies
- *Word lists*
    - *nltk.corpus.stopwords* → to filter out words with little lexical content such as **the**, **to**, **a**
    - *nltk.corpus.names* → **Anaphora Resolution**
    - *nltk.corpus.words* → to find unusual or misspelt words in a text

WordNet

WordNet is semantically-oriented lexical database of English where words (nouns, verbs, adjectives, etc.) are grouped into sets of synonyms (synsets), each expressing a distinct concept.

WordNet Relations

- **synonymy**

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

- **super-subordinate relation** (hyperonymy/hyponymy or is-a relation) → links general synsets like `car` to specific ones like `ambulance` or `bus`

```
>>> wn.synset("car.n.01").hyponyms()
[Synset('ambulance.n.01'), Synset('beach_wagon.n.
    01'), Synset('bus.n.04'), ...
```

WordNet Relations

- **synonymy**

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

- **super-subordinate relation** (hyperonymy/hyponymy or is-a relation) $\rightarrow$ links general synsets like `car` to specific ones like `ambulance` or `bus`
- **meronymy** $\rightarrow$ the part-whole relation holds between synsets like `tree` and `trunk, crown, limb`
- relationships between verbs $\rightarrow$ `walk` **entails** `step`
- **antonymy** $\rightarrow$ `supply` vs `demand`

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

```
>>> wn.synset("car.n.01").definition()
"a motor vehicle with four wheels; usually
    propelled by an internal combustion engine"
```

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

Semantic Similarity

You can use similarity measures defined over the collection of WordNet

- `path_similarity()` assigns a score in the range 0-1 based on the shortest path that connects the concepts in the hypernym hierarchy

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

WordNet

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

Preprocessing Steps

- **Tokeniziation** → breaking raw text into its building parts: words, phrases, symbols, or other meaningful elements called tokens
- **Punctuation removal**
- **Lowecasing**
- **Stemming** → removing morphological affixes from words, leaving only the word stem (may not be a real word)
- **Lemmatization** → removing morphological affixes from words, leaving only lemmas (**lemma** is a canonical form of the word)

```
import nltk
print(nltk.LancasterStemmer().stem("colors"))
# prints col
print(nltk.WordNetLemmatizer().lemmatize("colors"))
# prints color
```

- **Stopword removal**

Web Crawling

- **Urllib** $\rightarrow$ a high-level interface for fetching data across the World Wide Web
- **Beautiful Soup** $\rightarrow$ Python library for pulling data out of HTML and XML files

```
import nltk
import urllib
import bs4

def get_text(url):
    html = urllib.request.urlopen(url).read().decode("utf-8")
    return bs4.BeautifulSoup(html).get_text()

raw=get_raw("http://www.bbc.com/news/world-middle-east-42412729")
```

POS Tagging Overview

- **parts-of-speech (POS)** $\rightarrow$ word class, lexical categorys e.g. verb, noun, adjective, etc.
- **part-of-speech tagger** $\rightarrow$ labels words according to their POS
- **tagset** – the collection of tags used for a particular task

POS Tagging

POS Tagging allows

1. find likely words for a given tag
2. extract most ambiguous words across the word classes

POS Tagging

You want to count how many sentences in a corpus contain a form of the verb `have`. Which steps are necessary to obtain a reliable count?

POS Tagging

You want to count how many sentences in a corpus contain a form of the verb `have`. Which steps are necessary to obtain a reliable count?

1. Get tagged sentences (e.g. from a pre-tagged corpus or tag words using NLTK tagger `nltk.pos_tag`)

POS Tagging

You want to count how many sentences in a corpus contain a form of the verb `have`. Which steps are necessary to obtain a reliable count?

1. Get tagged sentences (e.g. from a pre-tagged corpus or tag words using NLTK tagger `nltk.pos_tag`)
2. Use NLTK lemmatizer to get lemma of each token in each sentence

POS Tagging

You want to count how many sentences in a corpus contain a form of the verb `have`. Which steps are necessary to obtain a reliable count?

1. Get tagged sentences (e.g. from a pre-tagged corpus or tag words using NLTK tagger `nltk.pos_tag`)
2. Use NLTK lemmatizer to get lemma of each token in each sentence
3. Iterate through the sentences

POS Tagging

You want to count how many sentences in a corpus contain a form of the verb `have`. Which steps are necessary to obtain a reliable count?

1. Get tagged sentences (e.g. from a pre-tagged corpus or tag words using NLTK tagger `nltk.pos_tag`)
2. Use NLTK lemmatizer to get lemma of each token in each sentence
3. Iterate through the sentences
4. Count those sentences, which contain at least one word with lemma "have” and pos-tag "verb”.

ML Application Development

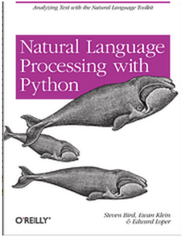
1. Implement a base version (baseline)
2. Train using training data (80% of all data)
3. Evaluate using development data (10% of all data)
4. Analyze errors (e.g. using confusion matrix)
5. Implement improvements – optimize
6. Go back to step 2
7. ...
8. Evaluate optimized version using test data (10% of all data)
9. Store the model

spaCy

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

- 

  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