Web Data Mining

Lecture 4: Text Mining

Jaroslav Kuchař & Milan Dojčinovski
jaroslav.kuchar@fit.cvut.cz, milan.dojchinovski@fit.cvut.cz

Czech Technical University in Prague - Faculty of Information Technologies - Software and Web Engineering

Summer semester 2019/2020
Humla v0.3

Overview

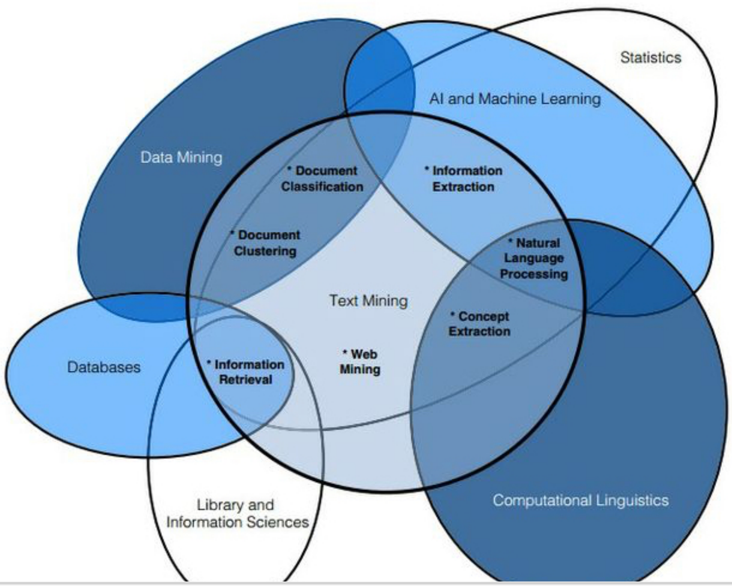
- Introduction
- Text Processing
- Named Entity Recognition
- Relation Extraction

Introduction

- What is Text Mining
  - *Extracting information and knowledge from text*
  - *Information and knowledge previously unknown to the user*
- Unknown information
  - *Information not known even for the writer*
  - *Rediscover (extract) information encoded in the text*
- Text Mining Process
  - *assembling large corpora of documents*
  - *performing preprocessing of documents*
  - *text transformation and feature generation*
  - *dimensions reduction - feature selection*
  - *pattern discovery (data mining)*
  - *results interpretation*

Text Mining Overview

- S. M. Weiss, N. Indurkhya, T. Zhang, and F. Damerau, Text mining: predictive methods for analyzing unstructured information. Springer Science and Business Media, 2010.



Main Challenges

- Text Mining is not easy!
- Main issues:
  - *High dimensionality*
  - *Different terms of same concepts*
    → e.g. car, vehicle, auto, automobile, ...
  - *Ambiguity - same term can identify one or more concepts*
    → e.g. player (sportsman, musician, performer, reproducing device, ...)
  - *Lack of structure*
    → The data (text) is not structured
      → Compared to: spreadsheets, database tables, etc.
      → Rows, columns, headings identify the meaning (semantics) of the content
- But - The data is highly redundant!

Text Mining Applications

- General applications
  - *Information retrieval (recommendation systems)*
  - *Information extraction*
- Social and Business
  - *Customer profile analysis*
  - *Social media data analysis*
  - *Trend analysis & Event tracking*
- Data Mining
  - *Document classification*
  - *Document clustering*
- Security and Crime
  - *Spam filtering*
  - *Fraud detection*

Overview

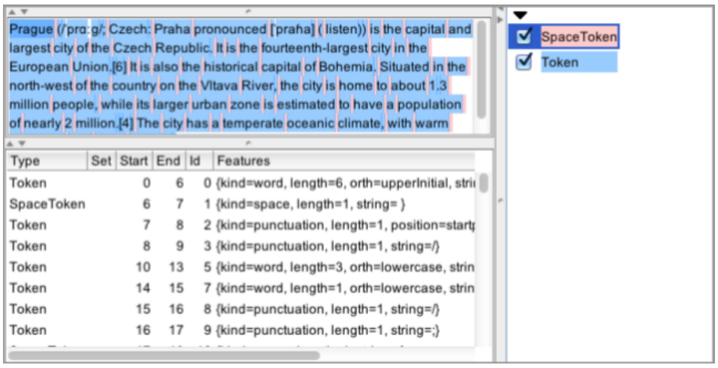
- Introduction
- Text Processing
- Named Entity Recognition
- Relation Extraction

Preprocessing

- Text needs to be preprocessed to be machine understandable
- Converting a raw text file into a well-defined sequence of linguistically-meaningful units
  – *Identifying words (tokens)*
  – *Special symbols - dots, commas, whitespaces*
  – *Identifying sentences*
- Needed for further text processing stages
  – *Part-Of-Speech tagging*
  – *Noun phrase chunking*
  – *Morphological analyzers*
  – *Named entity recognition*

Tokenization

- Grouping sequences of characters into logical elements called tokens (symbols, words, numbers)
  - *tokens are usually separated by whitespace characters or punctuation*
- Tokenizer
  - *system component performing the process of tokenization*



Tokenization (cont.)

- Trivial for a person familiar with the language structure
- More complicated for a computer program
  - *characters are sometimes token delimiters and sometimes not*
- Main groups
  - white spaces
    - → *space, tabs, new lines*
    - → *acting as delimiters and not as tokens*
  - ()<>!?"
    - → *delimiters and sometimes tokens*
  - .,;-’
    - → *delimiters only in specific situations*

Tokenization (cont.)

- Main situations
  - .,:
    - → *not delimiter in numbers*
    - → *e.g.* 100,000.5 , 12:45:37, ...
  - .
    - → *part of abbreviations and other constructs*
    - → *e.g.* P.S., D.I.Y, Dr., ...
  - '
    - → *part of the token*
      - → *e.g.* isn't
    - → *terminator*
      - → *e.g.* Tess'
    - → *quoting*
  - -
    - → *part of a phone number, accounts, ...*
    - → *e.g.* 123-456-789
- Generally language-dependent

Tokenization Example

- NLTK code:

```
import nltk
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th annivers
of our founding -- 250 years since the day we declared
our Independence.
...
In Chicago, more than 4,000 people were ...
"""
tokens = nltk.word_tokenize(text)
print(tokens)
```

- Output:

```
['Mr.', 'Speaker', ',', 'Mr.', 'Vice', 'President', ',', 'Members',
 'of', 'Congress', ',', 'the', 'First', 'Lady', 'of', 'the', 'United',
 'States', ',', 'and', 'Citizens', 'of', 'America', ':', '...', 'In',
 '9', 'years', ',', 'the', 'United', 'States', 'will', 'celebrate',
 'the', '250th', 'anniversary', 'of', 'our', 'founding', '--',
 '250', 'years', 'since', 'the', 'day', 'we', 'declared', 'our', 'Independen
 '4,000', 'people', 'were', '...']
```

Frequency Analysis

- Basic analysis of textual data
- Term frequency
  - *compute frequency distributions as ranked lists of terms*
- Lexical diversity
  - *diversity of an individual's or group's vocabulary*

```python
import nltk
from collections import Counter
text = None
with open('speech.txt', 'r') as f:
    text = f.read()

tokens = nltk.word_tokenize(text)
c = Counter(tokens)
print(c.most_common()[:10])

print(1.0*len(set(tokens))/len(tokens))
```

```
[(',', 261), ('.', 253), ('the', 215), ('and', 176), ('to', 142), ('of', 142
0.28324761204996324
```


Stemming and Lematization

- The goal
  - *Reduce the dimensionality!*
  - *Convert each of the tokens to a standard form.*
  - *Decreases the number of tokens and increases its frequency.*
- Lemmatization
  - *Finding lemma (canonical form) for a given word*
  - *words:* are, is, was, were*; lemma:* be
- Stemming
  - *Finding stem for a given word*
  - *Stem: root form of a word to which suffixes can be attached*
  - *word:* waiting*; stem:* wait

Stemming

- Part rule-based and part dictionary-based
  - *If token in dictionary return from dictionary*
  - *It token ends with* s *strip* s
  - *If token ends with* ies *replace by* y
  - ...

```
word = "printing"
regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
stem, suffix = re.findall(regexp, word)[0]
stem # print
```

```
import nltk
from nltk.stem.porter import PorterStemmer
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
"""
stemmer = PorterStemmer()
tokens = nltk.word_tokenize(text)
stems = {token:stemmer.stem(token) for token in tokens}
print(stems)
```

```
{'the': 'the', 'Citizens': 'Citizen', 'President': 'Presid',
'Congress': 'Congress', 'of': 'of', 'Members': 'Member', 'and': 'and',
'Mr.': 'Mr.', 'First': 'First', ',': ',', 'United': 'Unit', 'Lady': 'Ladi',
'America': 'America', ':': ':', 'Speaker': 'Speaker', 'Vice': 'Vice',
'States': 'State'}
```


Lemmatization

```
import nltk
from nltk.corpus.reader.wordnet import NOUN,VERB
from nltk.stem import WordNetLemmatizer
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th anniversary of our fo
-- 250 years since the day we declared our Independence.
...
In Chicago, more than 4,000 people were ...
"""
lemmatizer = WordNetLemmatizer()
tokens = nltk.word_tokenize(text)
lemmas = {token:lemmatizer.lemmatize(token, pos=VERB) for token in tokens}
print(lemmas)
```

```
{'celebrate': 'celebrate', ':': ':', 'Congress': 'Congress', 'anniversary':
'of': 'of', 'we': 'we', '250': '250', 'Independence': 'Independence',
'First': 'First', ',': ',', 'since': 'since', 'United': 'United',
'Chicago': 'Chicago', 'America': 'America', 'Speaker': 'Speaker', 'In': 'In'
'day': 'day', 'people': 'people', '250th': '250th', 'the': 'the', '9': '9',
'President': 'President', 'our': 'our', 'Members': 'Members', 'and': 'and',
'declared': 'declare', 'Mr.': 'Mr.', '.': '.', 'Citizens': 'Citizens',
'will': 'will', 'were': 'be', 'Lady': 'Lady', '--': '--', '4,000': '4,000',
'...': '...', 'years': 'years', 'Vice': 'Vice', 'States': 'States',
'more': 'more', 'founding': 'found', 'than': 'than'}
```

Other Dimensionality Reductions

- Stop words
  - *almost never provide any interesting information*
    - → *articles* a *and* the
    - → *pronouns such as* it *and* they

```
from nltk.corpus import stopwords
stops = stopwords.words('english')
text = """In Chicago, more than 4,000 people were ..."""
tokens = nltk.word_tokenize(text)
filtered_tokens = [token for token in tokens if token not in stops]
filtered_tokens
```

```
['In', 'Chicago', ',', '4,000', 'people', '...']
```

- Frequency information
  - *most frequent words are often stopwords and can be deleted*
  - *rare words are often typos and can also be dismissed*
  - *tf-idf*

Other Dimensionality Reductions (cont.)

- Synonyms vs Antonyms

```
from nltk.corpus import wordnet
token = "lady"
synonyms = []
antonyms = []
syns = wordnet.synsets(token)
for syn in syns:
    print("%s - %s " % (syn.lemmas()[0].name(), syn.definition()))
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
print(set(synonyms))
print(set(antonyms))
```

```
lady - a polite name for any woman
dame - a woman of refinement
Lady - a woman of the peerage in Britain
{'peeress', "ma'am", 'madam', 'dame', 'gentlewoman', 'Lady', 'lady',
{'Lord', 'nobleman'}
```

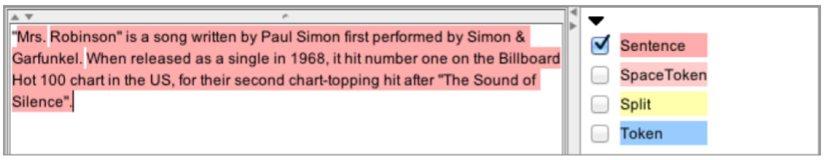
Frequency Analysis 2

```python
import nltk
from nltk.corpus import stopwords
from string import punctuation
stops = stopwords.words('english')
from collections import Counter

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

tokens = nltk.word_tokenize(text)
filtered_tokens = [token for token in tokens if token not in stops]
nopunc_tokens = [token for token in filtered_tokens if token not in punctuat

c = Counter(nopunc_tokens)
print(c.most_common()[:10])
print(1.0*len(set(nopunc_tokens))/len(nopunc_tokens))
```

```
[('--', 67), ('I', 38), ('We', 35), ('America', 30), ('American', 30),
('must', 20), ("'s", 20), ('new', 19), ('us', 18), ('country', 18)]
0.5049261083743842
```


Sentence Splitting

- Process of identifying sentences in text
- Not a trivial process
  - *Dot can have also other uses then sentence splitting symbol, for example, Mrs. Smith is ...*
- Heuristics can be used to improve the sentence splitting
- Maintain a list of patterns containing dot or write regex
  - *Mrs. | Mr. | Dr. | Ph.D. | Ing. | Bc.*



Sentence Splitting (cont.)

```python
import nltk
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th anniversary of
-- 250 years since the day we declared our Independence.
...
In Chicago, more than 4,000 people were ...
"""
sentences = nltk.sent_tokenize(text)
print(sentences)
```

```
[
'Mr. Speaker, Mr. Vice President, Members of Congress, \nthe First La
'In 9 years, the United States will celebrate the 250th anniversary o
'...',
'In Chicago, more than 4,000 people were ...\n'
]
```


N-Grams

- Set of co-occurring words within a given window
  - *Bigrams for N=2*
  - *Trigrams for N=3*
- Number of N-grams
  - $\boxed{\#NG = X-(N-1)}$ *where X is a number of tokens in the text.*

```python
import nltk
text = """
In 9 years, the United States will celebrate the 250th anniversary of
-- 250 years since the day we declared our Independence.
"""
from nltk.util import ngrams
for ng in ngrams(nltk.word_tokenize(text),3):
    print(ng)
```

```
('In', '9', 'years')
('9', 'years', ',')
('years', ',', 'the')
(',', 'the', 'United')
('the', 'United', 'States')
...
```

N-Grams application

- Collocations
  - *Expressions of multiple words which commonly co-occur*

```
import nltk
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
text = None
with open('speech.txt', 'r') as f:
    text = f.read()
bigram_finder = BigramCollocationFinder.from_words(
    nltk.word_tokenize(text))
bigram_finder.apply_freq_filter(2)
bigrams = bigram_finder.nbest(BigramAssocMeasures.chi_sq, 10)
bigrams
```

```
[('Homeland', 'Security'), ('Middle', 'East'), ('United', 'States'),
('middle', 'class'), ('43', 'million'), ('Rare', 'Disease'),
 ('Republican', 'President'), ('civil', 'rights'),
 ('illegal', 'immigrant'), ('inner', 'cities')]
```


N-Grams application (cont.)

- Generating text using N-Grams

```
import nltk
import random
text = None
with open('speech.txt', 'r') as f:
    text = f.read()
ng = ngrams(nltk.word_tokenize(text),2)
fd = nltk.ConditionalFreqDist(ng)

word = "the"
output = []
for i in range(15):
    output.append(word)
    word = random.choice(list(fd[word].keys()))
print(" ".join(output))
"""
```

```
the courage to make childcare accessible and reaffirmed our uniform.
the Bible teaches us harm . So I 've saved
the day we focus on becoming lobbyists for drugs from
...
```

Part-of-Speech Tagging

- Linguistic analysis
- Assigning linguistic categories to words in a text
  - *Number of categories is not fixed (from 6 or 7 to tens)*
  - *Difficulties:*
    - → bore *could be a noun, a present tense verb, or a past tense verb*
  - *can also take into consideration the context the words appear*

Title	Abbreviation	Example
Noun	NN	computer
Proper Noun	PNP	Africa
Adjective	JJ	nice
Pronoun	PRP	he
Verb	VB	work
Adverb	RB	word "fast" in "He runs fast."
Proposition	IN	in
Conjunction	CC	and

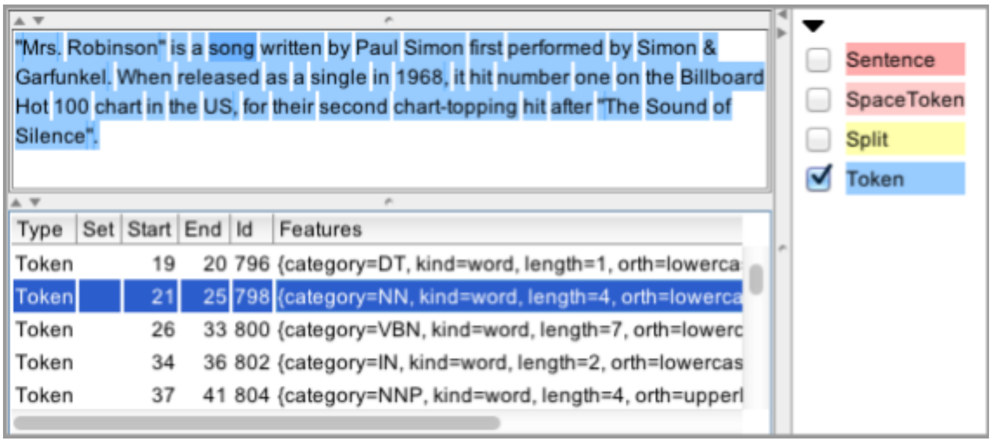
Part-of-Speech Tagging

```
nltk.help.upenn_tagset()
```

```
...
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
...
IN: preposition or conjunction, subordinating
    astride among uppon whether out inside pro despite on by throughout
    below within for towards near behind atop around if like until below
    next into if beside ...
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
...
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 ...
...
PRP: pronoun, personal
    hers herself him himself hisself it itself me myself one oneself ours
    ourselves ownself self she thee theirs them themselves they thou thy us
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly
    man baby diddle hush sonuvabitch ...
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
```

Part-Of-Speech Tagging Example

- Performing POS using the GATE framework



Simple POS Tagger

- Using large already tagged corpora and perform a mapping/classification task (with default to 'NN')
  – *e.g. ('the', 'DT'), ('day', 'NN'), ('celebrate', 'VB'), ...*
- Regular expressions

```
patterns = [
        (r'.*ing$', 'VBG'),                # gerunds
        (r'.*ed$', 'VBD'),                 # simple past
        (r'.*es$', 'VBZ'),                 # 3rd singular present
        (r'.*ould$', 'MD'),                # modals
        (r'.*\'s$', 'NN$'),                # possessive nouns
        (r'.*s$', 'NNS'),                  # plural nouns
        (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
        (r'.*', 'NN')                      # nouns (default)
]
```

- Rules based approaches
  – *e.g. VB if the tag of the preceding word is 'TO'*
- N-grams
- …
- → Combinations of all existing approaches

Part-Of-Speech Tagging Example

```python
import nltk
text = """
In 9 years, the United States will celebrate the 250th annivers
-- 250 years since the day we declared our Independence.
"""
print(nltk.pos_tag(nltk.word_tokenize(text)))
```

```
[('In', 'IN'), ('9', 'CD'), ('years', 'NNS'), (',', ','),
('the', 'DT'), ('United', 'NNP'), ('States', 'NNPS'),
('will', 'MD'), ('celebrate', 'VB'), ('the', 'DT'),
('250th', 'JJ'), ('anniversary', 'NN'), ('of', 'IN'),
('our', 'PRP$'), ('founding', 'NN'), ('--', ':'),
('250', 'CD'), ('years', 'NNS'), ('since', 'IN'),
('the', 'DT'), ('day', 'NN'), ('we', 'PRP'),
('declared', 'VBD'), ('our', 'PRP$'),
('Independence', 'NN'), ('.', '.')]
```

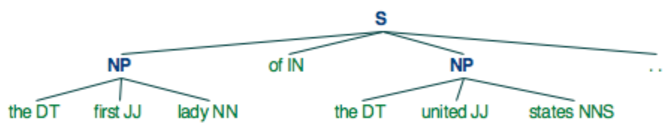

Phrase Detection

- Grouping tokens into units
  - *Also known as chunking*
  - *Utilizes grammar categories identified within the POS tagging task*
  - *Can be a prerequisite for Named Entity Recognition tasks*
- Using pattern-based classifications
- Noun Phrase Chunking (NP-chunking)
  - *Based on a chunk grammar*
    - → *set of rules that indicate how sentences should be chunked*
    - → *e.g. determiner followed by a number/adjective and terminated by a noun*

Phrase Detection (cont.)

```
import nltk
text = """
the first lady of the united states.
"""
text_pos = nltk.pos_tag(nltk.word_tokenize(text))
grammar = "NP: {<DT>?<JJ>*<NN|NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(text_pos)
print(result)
result.draw()
```

```
(S
  (NP the/DT first/JJ lady/NN)
  of/IN
  (NP the/DT united/JJ states/NNS)
  ./.)
```



Overview

- Introduction
- Text Processing
- Named Entity Recognition
- Relation Extraction

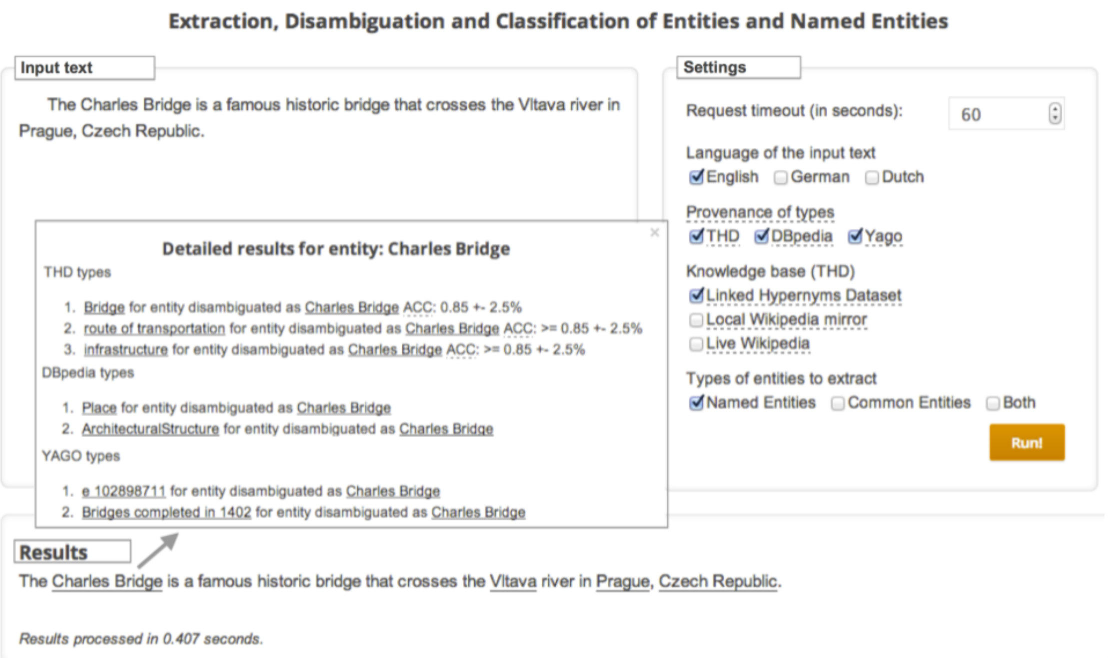
Named Entity Recognition

- Entity identification in free texts, and their classification
  - *turning verbose text data into a more compact structural form*
  - *special kind on phrase detection*
    - *→ proper noun phrases*
- NER sub-tasks
  - *recognition*
    - *→ spotting text fragments with entity mentions*
  - *classification*
    - *→ assigning class to an entity mention*
  - *disambiguation via linking*
    - *→ assigning URI (e.g., Wikipedia URI) describing the entity*
    - *→ also known as Entity Linking/Disambiguation task*
- NER systems are usually trained on a large corpus
  - *using methods such as Conditional Random Fields*
- NER based on previously defined rules (grammars) or gazetteers
  - *Gazetteers - list of countries, persons, geo locations*

Named Entity Recognition

- Results utilization
  - *dimension reduction*
  - *content enrichment*
- Various NER tools/APIs available
  - *Entityclassifier.eu*
  - *AlchemyAPI*
  - *OpenCalais*
  - *Wikimeta*
  - *DBpedia Spotlight*
  - *NERD*
  - *and many others*

Entity Recognition using Entityclassifier.eu



Entities Recognition Task

- First and main NER task
- Find entity mentions in text
- Detect start and end offset for each entity mention in a text
- Example:
  - *"The* Charles Bridge *is a famous historic bridge that crosses the* Vltava *river in* Prague*,* Czech Republic*."*
- Mentions:
  - *substring* "Charles Bridge"*, start: 4, end: 18*
  - *substring* "Vltava"*, start: 64, end: 70*
  - *substring* "Prague"*, start: 80, end: 86*
  - *substring* "Czech Republic"*, start: 88, end: 102*

Entity Recognition using NLTK

```
import nltk
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th anniversary of our fo
-- 250 years since the day we declared our Independence.
...
In Chicago, more than 4,000 people were ...
"""
tokens = nltk.word_tokenize(text)
tagged = nltk.pos_tag(tokens)

ne_chunked = nltk.ne_chunk(tagged, binary=True)
def extractEntities(ne_chunked):
    data = {}
    for entity in ne_chunked:
        if isinstance(entity, nltk.tree.Tree):
            text = " ".join([word for word, tag in entity.leaves()])
            ent = entity.label()
            data[text] = ent
        else:
            continue
    return data
extractEntities(ne_chunked)
```

```
{'America': 'NE', 'Chicago': 'NE', 'Citizens': 'NE', 'Congress': 'NE',
'Members': 'NE', 'Mr. Speaker': 'NE', 'Mr. Vice': 'NE', 'United States': 'NE
```


Custom NER Implementation

```
import nltk
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th anniversary of our founding
-- 250 years since the day we declared our Independence.
...
In Chicago, more than 4,000 people were ...
"""
tokens = nltk.word_tokenize(text)
tagged = nltk.pos_tag(tokens)

entity = []
for tagged_entry in tagged:
    if(tagged_entry[1].startswith("NN") or (entity and tagged_entry[1].startswith("IN"))):
        entity.append(tagged_entry)
    else:
        if(entity) and entity[-1][1].startswith("IN"):
            entity.pop()
        if(entity and " ".join(e[0] for e in entity)[0].isupper()):
            print(" ".join(e[0] for e in entity))
        entity = []
```

```
Mr. Speaker
Mr. Vice President
Members of Congress
First Lady
United States
Citizens of America
United States
Independence
Chicago
```

Entites Classification Task

- Task:
  – *for each mention assign a class (type)*
- The set of classes can be pre-defined (fixed) or be dynamically created
- Example:
  – *"The Charles Bridge is a famous historic bridge that crosses the Vltava river in Prague, Czech Republic."*
- Mentions:
  – *Mention: "Charles Bridge", class:* LOC
  – *Mention: "Vltava", class:* LOC
  – *Mention: "Prague", class:* LOC
  – *Mention: "Czech Republic", class:* GPE

Classification Set of Classes

- Possible fixed set of classes:
  – *LOC (location), GPE (Geo-political entity), ORG (organization), PER (person), MISC (miscellaneous entity, anything else)*
- The set of classes can be also defined using an ontology
  – *Advantages: each class is identified with an unique URI*
    → *Example: DBpedia Ontology*
      → *see DBpedia Ontology 3.9*
    → *covers 529 classes*
    → *used to describe articles in Wikipedia (DBpedia)*
    → *e.g., http://dbpedia.org/resource/Capital_city*

Entity Classification using NLTK

```python
import nltk
text = """Mr. Speaker, Mr. Vice President, Members of Congress,
the First Lady of the United States, and Citizens of America:
...
In 9 years, the United States will celebrate the 250th anniversary of our fo
-- 250 years since the day we declared our Independence.
...
In Chicago, more than 4,000 people were ...
"""
tokens = nltk.word_tokenize(text)
tagged = nltk.pos_tag(tokens)

ne_chunked = nltk.ne_chunk(tagged, binary=False)
def extractEntities(ne_chunked):
    data = {}
    for entity in ne_chunked:
        if isinstance(entity, nltk.tree.Tree):
            text = " ".join([word for word, tag in entity.leaves()])
            ent = entity.label()
            data[text] = ent
        else:
            continue
    return data
extractEntities(ne_chunked)
```

```
{'America': 'GPE', 'Chicago': 'GPE', 'Citizens': 'ORGANIZATION', 'Congress':
'First Lady': 'ORGANIZATION', 'Members': 'ORGANIZATION', 'Mr. Speaker': 'PER
```


Entities Disambiguation Task

- Human language is not exact
  - *Same text can refer to totally different entities*
- Example: the entity "Maradona"
  - *can refer to the football player* "Diego Maradona"
  - *or, to the football coach and former player* "Hugo Maradona"
  - *or, to the movie about the football player* "Diego Maradona"
- Classification does not help with the ambiguity
  - *there can exist two diferent entities of same type*
  - "Diego Maradona" *and* "Hugo Maradona", *type:* PER
- Solution: use unique URIs to solve the ambiguity
  - *perform Entity Linking*

Entity Linking Task

- Task:
  - *identifying entities using URIs (unique)*
- We can use knowledge base URIs to uniquely identify each entity
  - *e.g., Wikipedia or DBpedia or YAGO knowledge base URIs*
- Solution for the "Maradona" ambiguity:
  - *the **football player** "Diego Maradona"*
    → URI: http://dbpedia.org/resource/Diego_Maradona
  - *the **football coach and former player** "Hugo Maradona"*
    → URI: http://dbpedia.org/resource/Hugo_Maradona
  - *the **movie** about the football player "Diego Maradona"*
    → URI:
    http://dbpedia.org/resource/Maradona_by_Kusturica

Coreference Resolution

- In text, different text fragments can refer to same entity
- Main objective
  - *mentioned subjects, pronouns and other referring expressions must be connected to the right individuals*
  - *match proper names and their variants in a document*
- Example:
  - Mary Smith *and* Mrs. Smith
    → should be matched as same person
  - International Business Machines Ltd. *and* IBM
    → should be matched as same company
- State of the art algorithms have accuracy of around 75%

Overview

- Introduction
- Text Processing
- Named Entity Recognition
- Relation Extraction

Relation Extraction

- Extracting semantic relation between entities
- Examples
  - *PERSON works for ORGANIZATION*
  - *PERSON attends EVENT*
  - *PERSON lives in LOCATION*
- Using domain-specific patterns
- Naive approach:
  - *fixed patterns*
    $\rightarrow$ *X(Subject) works for Y(Object)*
  - *does not scale*
- Approaches based on linguistic analysis
  - *based on linguistic features (e.g. POS tagging)*

Pattern based Relation Extraction

```
import nltk
import re
from nltk.sem import extract_rels,rtuple

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

sentences = nltk.sent_tokenize(text)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in sentences]
tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_sentences]

OF = re.compile(r'.*\bof\b.*')

for i, sent in enumerate(tagged_sentences):
    sent = nltk.ne_chunk(sent)
    rels = extract_rels('PERSON', 'GPE', sent, corpus='ace', pattern=OF, window=5)
    for rel in rels:
        print('{0:<5}{1}'.format(i, rtuple(rel)))
```

```
1     [PER: 'Jewish/NNP Community/NNP Centers/NNPS']
                                        'and/CC vandalism/NN of/IN' [GPE: 'Jewish/JJ']
123   [PER: 'Matt/NNP Bevin/NNP'] 'of/IN' [GPE: 'Kentucky/NNP']
198   [PER: 'Carryn/NNP Owens/NNP'] ',/, the/DT widow/NN of/IN a/DT' [GPE: 'U.S./NNP']
```


Pattern based Relation Extraction 2

- https://en.wikipedia.org/wiki/Donald_Trump

```
import nltk
import re
from nltk.sem import extract_rels,rtuple
import wikipedia

p = wikipedia.page("Donald Trump")
text = p.content

sentences = nltk.sent_tokenize(text)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in sentences]
tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_sentences]

BORN = re.compile(r'.*\bborn\b.*')

for i, sent in enumerate(tagged_sentences):
    sent = nltk.ne_chunk(sent)
    rels = extract_rels('PERSON', 'GPE', sent, corpus='ace', pattern=BORN, window=5)
    for rel in rels:
        print('{0:<5}{1}'.format(i, rtuple(rel)))
```

```
28    [PER: 'Fred/NNP'] 'was/VBD born/VBN in/IN the/DT' [GPE: 'Bronx/NNP']
53    [PER: 'Donald/NNP Jr./NNP'] '(/( born/JJ 1977/CD )/) ,/,' [GPE: 'Ivanka/NNP']
310   [PER: 'Obama/NNP'] 'was/VBD born/VBN in/IN the/DT' [GPE: 'United/NNP States/NNPS']
311   [PER: 'Obama/NNP'] 'was/VBD born/VBN in/IN the/DT' [GPE: 'U.S./NNP']
```

Relation Extraction based on Linguistic Analysis

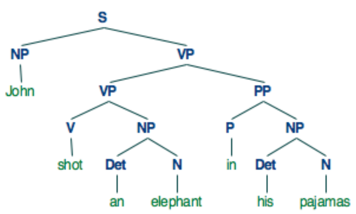
- Identify the subject, predicate and object of the sentence
  - *the relation is the predicate*
  - *the subject is the source entity*
  - *the object is the target entity*
- Perform dependency parsing of the sentence

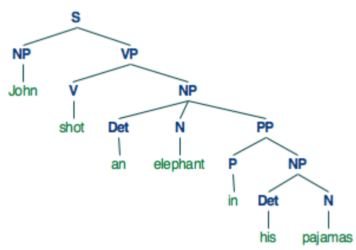
```
                S
       _________|__________
      |                    VP
      |          __________|______
      |         |                 NP
      |         |     ____________|___
      |         |    |      |         PP
      |         |    |      |      ___|___
      |         |    |      |     |       NP
      |         |    |      |     |    ___|_____
      NP        V   Det     N     P  Det        N
      |         |    |      |     |   |         |
     John      shot  an  elephant in his     pajamas
```


Parsing

- Allows to analyze a sentence structure
  - *Dealing with the ambiguity of the natural language*

```
import nltk
grammar = nltk.CFG.fromstring("""
 S -> NP VP
 PP -> P NP
 NP -> Det N | Det N PP | 'John'
 VP -> V NP | VP PP
 Det -> 'an' | 'his'
 N -> 'elephant' | 'pajamas'
 V -> 'shot'
 P -> 'in'
""")
sentence = ['John', 'shot', 'an', 'elephant', 'in', 'his', 'pajamas']
parser = nltk.ChartParser(grammar)
for tree in parser.parse(sentence):
    print(tree)
    tree.draw()
```





Identification of a subject in parsed English tree

```
            S
        ____|__________
       |              VP
       |     _________|_____
       |    |              NP
       |    |     _________|___
       |    |    |     |      PP
       |    |    |     |   ___|___
       |    |    |     |  |      NP
       |    |    |     |  |   ___|_____
       NP   V   Det    N  P  Det       N
       |    |    |     |  |   |        |
      John shot  an elephant in his   pajamas
```

- Identification of a subject for English
  - S *indicates the sentence*
  - NP *is noun phrase*
  - VP *is verb phrase*
  - ***subject is NP that is the child of S and the sibling of VP***

Relation extraction from parsed trees

```
            S
        ____|__________
       |              VP
       |     _________|_____
       |    |              NP
       |    |     _________|___
       |    |    |     |      PP
       |    |    |     |   ___|___
       |    |    |     |  |      NP
       |    |    |     |  |   ___|_____
       NP   V   Det    N  P  Det       N
       |    |    |     |  |   |        |
      John shot  an elephant in his   pajamas
```

- The subject is the NP
  - John
- The relation is the verb (V) in the verb phrase (VP)
  - shot
- The object is the noun (N) in the noun phrase (NP) in the verb phrase (VP)
  - elephant