Empolis Machine Learning Workshop

– März 2016 –

Features für Text

Prof. Dr. Adrian Ulges
Fachbereich DCSM / Informatik
Hochschule RheinMain

21. März 2016

Some Remarks regarding Text Features

ML Applications involving Text

- Information extraction / part-of-speech tagging
- Sentiment analysis
- Spam filtering
- Information retrieval
- Recommendation (of news, videos, movies, jobs, ...)

Some Remarks regarding Text Features

ML Applications involving Text

- Information extraction / part-of-speech tagging
- Sentiment analysis
- Spam filtering
- Information retrieval
- Recommendation (of news, videos, movies, jobs, ...)

Example: Bag-of-words Features

- Features for text should satisfy our criteria: **compactness**, **discriminativity**, **invariance**
- Common baseline (**bag-of-words** features): Store the count of appearances for each term in a document
- How would you **rank** bag-of-words features regarding compactness / discriminativity / invariance ?

Some Remarks regarding Text Features

Remarks

- In **this chapter**, we will have a look at some improvements over bag-of-words features
- The focus will still be on **simple text statistics**
- A very useful reference: Python's `nltk` module!

Text Features: Segmentation

- First Question: What is a "**term**"?
- **Text segmentation** into terms is not a trivial problem *(splitting by spaces does not entirely do the job!)*

Text Features: Segmentation

- First Question: What is a "**term**"?
- **Text segmentation** into terms is not a trivial problem *(splitting by spaces does not entirely do the job!)*

Example	**Approach**
Germany's chancellor	*rule-based recognition*
3/20/91 vs. Mar 12, 1991	*rule-based recognition*
(0049) 611/9495-1215	*rule-based recognition*
San Francisco	

Text Features: Segmentation

- First Question: What is a "**term**"?
- **Text segmentation** into terms is not a trivial problem
  (splitting by spaces does not entirely do the job!)

Example	**Approach**
Germany's chancellor	*rule-based recognition*
3/20/91 vs. Mar 12, 1991	*rule-based recognition*
(0049) 611/9495-1215	*rule-based recognition*
San Francisco	*statistical methods*
Lebensversicherungsgesellschaft vs. Malerei	

Text Features: Segmentation

- First Question: What is a "**term**"?
- **Text segmentation** into terms is not a trivial problem
  (splitting by spaces does not entirely do the job!)

Example	**Approach**
Germany's chancellor	*rule-based recognition*
3/20/91 vs. Mar 12, 1991	*rule-based recognition*
(0049) 611/9495-1215	*rule-based recognition*
San Francisco	*statistical methods*
Lebensversicherungsgesellschaft vs. Malerei	*compound splitter (dictionary-based vs. statistical methods)*

Text Features: Segmentation

Code Example: Python

- This code uses **regular expressions**, which allow us to search a wide range of text patterns in strings

Operator	Behavior
`.`	Wildcard, matches any character
`^abc`	Matches some pattern *abc* at the start of a string
`abc$`	Matches some pattern *abc* at the end of a string
`[abc]`	Matches one of a set of characters
`[A-Z0-9]`	Matches one of a range of characters
`ed|ing|s`	Matches one of the specified strings (disjunction)
`*`	Zero or more of previous item, e.g., `a*`, `[a-z]*` (also known as *Kleene Closure*)
`+`	One or more of previous item, e.g., `a+`, `[a-z]+`
`?`	Zero or one of the previous item (i.e., optional), e.g., `a?`, `[a-z]?`
`{n}`	Exactly *n* repeats where *n* is a non-negative integer
`{n,}`	At least *n* repeats
`{,n}`	No more than *n* repeats
`{m,n}`	At least *m* and no more than *n* repeats
`a(b|c)+`	Parentheses that indicate the scope of the operators

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Text Features: Normalization

- We also **normalize** text to increase robustness to flexion and sentence structure
- **Step 1**: Lower-casing *(Sometimes $\rightarrow$ sometimes)*
- **Step 2**: Stemming = reducing words to their stem

Text Features: Normalization

- We also **normalize** text to increase robustness to flexion and sentence structure
- **Step 1**: Lower-casing *(Sometimes → sometimes)*
- **Step 2**: Stemming = reducing words to their stem

Stemming: Methods

- Rule-based Methods
  - Example rule: `*t → *` (geht → geh)
  - Example rule: `*en → *` (gehen → geh)
- Dictionary-based Methods
  - Example: `stem['ging'] = 'geh'`
  - popular for languages with strong flexion *(like German)*

Stemming: Code Example

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

>>> tokens = ['women', 'swords', 'is', 'lying']

>>> [naive_stem(t) for t in tokens]

    ['women', 'sword', 'i', 'ly']          // naive
```

Stemming: Code Example

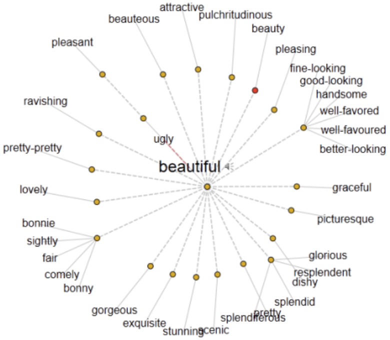
```
def naive_stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

>>> tokens = ['women', 'swords', 'is', 'lying']

>>> [naive_stem(t) for t in tokens]

    ['women', 'sword', 'i', 'ly']          // naive

>>> [nltk.WordNetLemmatizer().lemmatize(t)
      for t in tokens]

    ['woman', 'sword', 'is', 'lying']      // dict-based

>>> [nltk.PorterStemmer().stem(t)
      for t in tokens]

    ['women', 'sword', 'is', 'lie']        // rule-based
```

Text Features: Synsets

- Can we achieve invariance to **synonyms**?

  "What a beautiful day!" vs.
  "What a lovely day!"

- A frequent approach are **thesauri**: A thesaurus is a collection of terms, connected by (pre-defined) relations

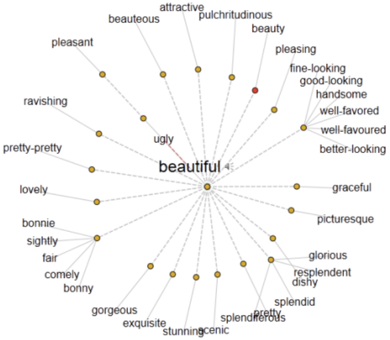


Text Features: Synsets

- Can we achieve invariance to **synonyms**?

  "What a beautiful day!" vs.
  "What a lovely day!"

- A frequent approach are **thesauri**: A thesaurus is a collection of terms, connected by (pre-defined) relations
- Typical relations
  - synonyms *(beautiful vs. lovely)*
  - antonym *(beautiful vs. ugly)*
  - generalization/specialization *(a boat* **is a** *vehicle)*
- Synonyms form so-called **synsets**



Synsets: Python Example

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets("dog")

  [Synset('dog.n.01'),
   Synset('frump.n.01'),
   Synset('dog.n.03'),
   Synset('cad.n.01'),
   Synset('frank.n.02'),
   Synset('pawl.n.01'),
   Synset('andiron.n.01'),
   Synset('chase.v.01')]
```

Synsets: Python Example

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets("dog")

  [Synset('dog.n.01'),
   Synset('frump.n.01'),
   Synset('dog.n.03'),
   Synset('cad.n.01'),
   Synset('frank.n.02'),
   Synset('pawl.n.01'),
   Synset('andiron.n.01'),
   Synset('chase.v.01')]

>>> for synset in wn.synsets("dog"):
        print "dog =", synset.definition

   dog = a member of the genus Canis ...
   dog = a dull unattractive unpleasant woman
   dog = informal term for a man
   dog = a smooth-textured sausage ...
   dog = metal supports for logs in a fireplace
   dog = go after with the intent to catch
   ...
```

From Thesauri to Ontologies

- We can extend the concept of a thesaurus to *ontologies*
- An ontology can be thought of as a generalized **knowledge base** containing objects and relations between them
- Ontologies can be **combined** by linking their objects

```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa =  InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```

Innsbruck

Country	Austria
State	Tyrol
Administrative region	Statutory city
Population	117,342 (2006)
Area	104.91 km²
Population density	1,119 /km²
Elevation	574 m
Coordinates	47°16′ N 11°23′ E
Postal code	6010-6080
Area code	0512
Licence plate code	I
Mayor	Hilde Zach
Website	www.innsbruck.at

From Thesauri to Ontologies

- We can extend the concept of a thesaurus to *ontologies*
- An ontology can be thought of as a generalized **knowledge base** containing objects and relations between them
- Ontologies can be **combined** by linking their objects

Example: The DBPedia Project

- 20.8 mio. "things", crawled from Wikipedia infoboxes
- $>$ 500 mio. "facts"
- representation by RDF *(Resource Description Framework)*

```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa = InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```

From Thesauri to Ontologies

- We can extend the concept of a thesaurus to *ontologies*
- An ontology can be thought of as a generalized **knowledge base** containing objects and relations between them
- Ontologies can be **combined** by linking their objects

Example: The DBPedia Project

- 20.8 mio. "things", crawled from Wikipedia infoboxes
- $>$ 500 mio. "facts"
- representation by RDF *(Resource Description Framework)*
- allows **smarter search** ("give me all cities in New Jersey with more than 10,000 inhabitants")

```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa = InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```

Text Features: N-Grams

- So far, we have neglected the **order** of words in the document

  "I can **not** *believe it –* **What** *a* **cool** *video!" vs.*
  "This video is **not cool** *–* **What** *a..."*

Text Features: N-Grams

- So far, we have neglected the **order** of words in the document

  "I can **not** *believe it –* **What** *a* **cool** *video!" vs.*
  "This video is **not cool** *–* **What** *a..."*

- A simple statistical approach are **n-grams**: Instead of segmenting text into single tokens, we segment it into subsequences of *n* tokens each!

Text Features: N-Grams

- So far, we have neglected the **order** of words in the document

  "I can **not** *believe it –* **What** *a* **cool** *video!" vs.*
  "This video is **not cool** *–* **What** *a..."*

- A simple statistical approach are **n-grams**: Instead of segmenting text into single tokens, we segment it into subsequences of *n* tokens each!

In the Example

- bag-of-words feature

  $\left\{ \text{(This: 1), (video: 1), (is: 1), (cool: 1), ...} \right\}$

- n-gram feature

  $\left\{ \text{(This video: 1), (video is: 1), (is not: 1), (not cool: 1), ...} \right\}$

Text Features: N-Grams

- So far, we have neglected the **order** of words in the document

  "I can ***not*** *believe it –* ***What*** *a* ***cool*** *video!" vs.*
  "This video is ***not cool*** *–* ***What*** *a..."*

- A simple statistical approach are **n-grams**: Instead of segmenting text into single tokens, we segment it into subsequences of n tokens each!

In the Example

- bag-of-words feature

$$\left\{ \textit{(This: 1), (video: 1), (is: 1), (cool: 1), ...} \right\}$$

- n-gram feature

$$\left\{ \textit{(This video: 1), (video is: 1), (is not: 1), (not cool: 1), ...} \right\}$$

- Problem: Features get (even more) **high-dimensional**!