Computer-based Content Analysis

Johannes Knopp, Cäcilia Zirn

Knowledge Representation and Knowledge Management Research Group
School of Business Informatics and Mathematics

06.10.2011
1. Warm-up

2. Today’s Goal: Using NLTK
Overview

1. Warm-up
   - 2010 World Cup
   - Regular Expressions in Python

2. Today’s Goal: Using NLTK
Overview

1. Warm-up
   - 2010 World Cup
   - Regular Expressions in Python

2. Today’s Goal: Using NLTK
### 2010 World Cup Analysis


- Analyzed the spellings of “goal”
- Searched all spelling variants

<table>
<thead>
<tr>
<th>Spellings</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>158636 gol</td>
<td>941 Gool</td>
</tr>
<tr>
<td>126669 goal</td>
<td>848 goooool</td>
</tr>
<tr>
<td>31722 Goal</td>
<td>844 GOOOOOOOL</td>
</tr>
<tr>
<td>24735 Gol</td>
<td>718 goooooool</td>
</tr>
<tr>
<td>19610 GOL</td>
<td>708 GOOOOOOOOOL</td>
</tr>
<tr>
<td>14317 GOAL</td>
<td>666 gooooooool</td>
</tr>
<tr>
<td>4178 gool</td>
<td>646 Goooooooool</td>
</tr>
<tr>
<td>2981 ggol</td>
<td>627 goall</td>
</tr>
<tr>
<td>2219 goll</td>
<td>589 GOOOOOOOOOOOL</td>
</tr>
<tr>
<td>1771 goool</td>
<td>588 GOOOOOOOOOOOL</td>
</tr>
</tbody>
</table>

⭐ Could you do that? How?
2010 World Cup Analysis


- Analyzed the amounts of “o”s
Today's Goal: Using NLTK
Overview

1. Warm-up
   - 2010 World Cup
   - Regular Expressions in Python

2. Today’s Goal: Using NLTK
Warm-up

Today's Goal: Using NLTK

Good old friend: RegEx

```python
>>> import re
>>> p = re.compile('g*a*l*')
>>> p.match('goooaaaall')
```

<table>
<thead>
<tr>
<th>Method/Attribute</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>match()</td>
<td>Determine if the RE matches at the beginning of the string.</td>
</tr>
<tr>
<td>search()</td>
<td>Scan through a string, looking for any location where this RE matches.</td>
</tr>
<tr>
<td>findall()</td>
<td>Find all substrings where the RE matches, and returns them as a list.</td>
</tr>
<tr>
<td>finditer()</td>
<td>Find all substrings where the RE matches, and returns them as an iterator.</td>
</tr>
</tbody>
</table>
>>> import re
>>> p = re.compile('g*o*a*l*')
>>> m = p.match("gooooaaaall")
>>> m.start()
>>> m.end()
Good old friend: RegEx

Groups are a very important concept.

```python
>>> p = re.compile('([^ goal]*)(goal)(.*)
>>> m = p.match("sdksjdj_gooooaaaallll_ksdjdsj")
>>> m.group(0)
'sdksjdj_gooooaaaallll_ksdjdsj'
>>> m.group(1)
'sdksjdj_goo'
>>> m.group(2)
'goooaaaallll'
>>> m.group(3)
'ksdjdsj'
>>> m.group(4)
Traceback (most recent call last):
  File "<stdin>"., line 1, in <module>
IndexError: no such group
```
Text (pre-)processing with NLTK

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
Warm-up

Today's Goal: Using NLTK

Motivation

Natural Language Processing with NLTK
- NER, Tokenizing, POS-Tagging
- Stemming, Lemmatizing
- Parsing
- Relation Extraction
- WordNet
Tasks NLTK might be helpful with

Some questions...

Imagine you have a large amount of text data and you want to know:

- Who died?
- Which company took over another company?
- Which companies went bankrupt?
- Which politician met whom where?
- ...

★ What are possible questions for an analysis in your domain?
Tasks NLTK might be helpful with

★ Which tools do you already know?

Tools...
Tasks NLTK might be helpful with

★ Which tools do you already know?

Tools...

- Named Entity Recognition
- Regular Expressions
Overview

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
Overview

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
Disgression: Getting familiar with NLTK

Natural Language Toolkit

Links

http://www.nltk.org/
http://www.nltk.org/book
http://text-processing.com/demo Online demo!

This is what you should use when programming: (API)

NLTK does not only provide tools for natural language processing, but comes with a large data set including various text corpora (some of them with rich annotation sets), grammars, and much more...
Disgression: Getting familiar with NLTK

```python
>>> from nltk.book import *

>>> texts()

>>> text7.concordance("dollar")

>>> text7.collocations()

>>> text7.common_contexts(["money", "dollar"])

>>> text7.similar("john")

>>> text7.dispersion_plot(["dollar", "America", "China", "yen"])
```

(output of commands omitted)
Disgression: Getting familiar with NLTK

```python
>>> text7.dispersion_plot(['dollar', 'America', 'China', 'yen'])
```
Disgression: Getting familiar with NLTK

```python
>>> set(text7)

>>> f = FreqDist(text3)

>>> f.plot(50)

>>> f.plot(50, cumulative=True)

>>> f.keys()

>>> f.values()

(output of commands omitted)
```
Disgression: Getting familiar with NLTK

★ Print all words that occur more than 50 times
Disgression: Getting familiar with NLTK

★ Print all words that occur more than 50 times

```python
>>> for word in f.keys():
    if (f.get(word) > 50):
        print(word)
```

(output of commands omitted)
Disgression: Getting familiar with NLTK

### Methods of FreqDist

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fdist = FreqDist(samples)</code></td>
<td>create a frequency distribution containing the given samples</td>
</tr>
<tr>
<td><code>fdist.inc(sample)</code></td>
<td>increment the count for this sample</td>
</tr>
<tr>
<td><code>fdist['monstrous']</code></td>
<td>count of the number of times a given sample occurred</td>
</tr>
<tr>
<td><code>fdist.freq('monstrous')</code></td>
<td>frequency of a given sample</td>
</tr>
<tr>
<td><code>fdist.N()</code></td>
<td>total number of samples</td>
</tr>
<tr>
<td><code>fdist.keys()</code></td>
<td>the samples sorted in order of decreasing frequency</td>
</tr>
<tr>
<td><code>for sample in fdist:</code></td>
<td>iterate over the samples, in order of decreasing frequency</td>
</tr>
<tr>
<td><code>fdist.max()</code></td>
<td>sample with the greatest count</td>
</tr>
<tr>
<td><code>fdist.tabulate()</code></td>
<td>tabulate the frequency distribution</td>
</tr>
<tr>
<td><code>fdist.plot()</code></td>
<td>graphical plot of the frequency distribution</td>
</tr>
<tr>
<td><code>fdist.plot(cumulative=True)</code></td>
<td>cumulative plot of the frequency distribution</td>
</tr>
<tr>
<td><code>fdist1 &lt; fdist2</code></td>
<td>test if samples in <code>fdist1</code> occur less frequently than in <code>fdist2</code></td>
</tr>
</tbody>
</table>
For Named Entity Recognition, we need some preprocessing steps.

1. Raw text (string)
2. Sentence segmentation
3. Tokenization
4. Part of speech tagging
5. Post-tagged sentences (list of lists of tuples)
6. Entity detection
7. Chunked sentences (list of trees)
8. Relation detection
9. Relations (list of tuples)
3 steps before NER

- **Sentence Tokenizing**
  Recognizing the border between sentences
  
  `Dr. Jekyll had the IP address 192.168.0.7. But not Mr. Hyde!`

- **Tokenizing**
  
  `[Dr][.] [Jekyll] [had] [the] [IP] [address] [192][.][168][.][0][.][7][.]`

- **POS-Tagging**
  Assigning each token its part-of-speech tag
  
  `Dr/NNP ./ Jekyll/NNP had/VBD the/DT IP/NNP address/NN 192/CD ./ 168/CD ./ 0/-NONE- ./ 1/CD`
>>> import nltk
>>> def ie_preprocess(document):
...     sentences = nltk.sent_tokenize(document)
...     sentences = [nltk.word_tokenize(sent) for sent in sentences]
...     sentences = [nltk.pos_tag(sent) for sent in sentences]
Those preprocessing steps are not only useful as a prerequisite for NER.

- Sentence Tokenizing, Tokenizing: Necessary for any other kind of automatic processing
- POS-Tagging: Topic of a text ist mostly determined by nouns, sometimes verbs

Background reading:
http://www.soehn.net/work/icl/tagging.pdf (Jan-Philipp Söhn, Uni Tübingen)
Sorry, still not there...

Chunking

- Named entities may consist of several tokens
  - New York
  - Monty Python
  - Her Royal Highness Princess Elizabeth Alexandra Mary of York
- We need to identify tokens belonging to one **noun phrase**
Finally!

NER and Chunking

- There is a **built-in** Named Entity Recognizer in NLTK which already contains a Chunker.
- You can write your own Chunker.
- You can train your own Named Entity Recognizer.

```python
>>> for sent in sentences:
    ...      print nltk.ne_chunk(sent, binary=True)
```
```python
>>> for sent in sentences:
    ...      print nltk.ne_chunk(sent)
```

(sentences must be tokenized and POS-tagged.)
Overview

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
Stemming & Lemmatizing

Stemming

A stemmer transforms inflected word forms back to their stem.

- tried → tri
- trying → tri
- try → tri
- womanizer → woman

Lemmatizing

Finds the correct lemma for a word.

- meeting → meeting
- better → good

There are different implementations of stemmers and lemmatizers, the results differ!
Stemming & Lemmatizing

Stemming

```python
>>> from nltk.stem.porter import PorterStemmer
>>> for sent in sentences:
    for word in sent:
        PorterStemmer().stem(word)
```

Lemmatizing

```python
>>> from nltk.stem.wordnet import WordNetLemmatizer
>>> for sent in sentences:
    for word in sent:
        WordNetLemmatizer().lemmatize(word)
```
Revisiting our pipeline

★ Where should we place the Stemming / Lemmatizing step?

```python
>>> sentences = nltk.sent_tokenize(document)
>>> sentences = [nltk.word_tokenize(sent) for sent in sentences]
>>> sentences = [nltk.pos_tag(sent) for sent in sentences]
>>> for sent in sentences:
...     print nltk.ne_chunk(sent)
```
Revisiting our pipeline

★ Where should we place the Stemming / Lemmatizing step?

```python
>>> sentences = nltk.sent_tokenize(document)
>>> sentences =
    [nltk.word_tokenize(sent) for sent in sentences]
>>> sentences =
    [nltk.pos_tag(sent) for sent in sentences]
>>> chunkedSentences =
    [nltk.ne_chunk(sent) for sent in sentences]
>>> stemmedWords = []
>>> for sentence in sentences:
...    stemmedWords.append(
[PorterStemmer().stem_word(word) for word in sentence])
```

We do not replace tokens by lemmatized tokens. We keep additional information in new, additional variables.
Resume
We are now able to ...
- Find a word in different flected forms
- Identify named entities
- Separate nouns from verbs, identify part-of-speech
- ...
Is this enough?
Warm-up

Today’s Goal: Using NLTK

Motivation

Natural Language Processing with NLTK
  - NER, Tokenizing, POS-Tagging
  - Stemming, Lemmatizing
  - Parsing
  - Relation Extraction
  - WordNet
Why parsing?

They called me short

I called them an ambulance
Why parsing?

Example

- You want to find all sentences in which a person is actively doing something, i.e. appears as subject of a sentence (not object)
  → For all situations where you need more knowledge about the syntax
Today's Goal: Using NLTK

Why parsing?

Example

- You want to find all sentences in which a person is actively doing something, i.e. appears as subject of a sentence (not object)

→ For all situations where you need more knowledge about the syntax
How does it work?

- A parser is analyzing the syntactic structure of a sentence.
- Either, the parser has to be trained on annotated data. It learns possible structures.
- Alternatively, there are parsers that take given grammar rules.
- There are different strategies for parsing (e.g. Chart-Parsing, Bottom-up-Parsing, ...)

Problem: Ambiguities

I shot an elephant in my pajamas.
My personal advice

- Try to avoid parsing. It is slow, complicated; and it is difficult to find appropriate training data / grammars as input.
- Most of the times, using Chunking, POS-Tagging and defining patterns / rules is enough.
Code snippet (Only working if grammar has been specified before!)

```python
>>> sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
>>> parser = nltk.ChartParser(groucho_grammar)
>>> trees = parser.nbest_parse(sent)
>>> for tree in trees:
...     print tree
...
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas)))))
```
Warm-up

Today’s Goal: Using NLTK

- Motivation

Natural Language Processing with NLTK
- NER, Tokenizing, POS-Tagging
- Stemming, Lemmatizing
- Parsing
- Relation Extraction
- WordNet
Searching for patterns

Efficient alternative to complete syntax parsing

Extract relations between named entities

1. First, define your own chunking rules:
   Analyze the type of chunks you expect to find.

2. Specify the patterns you are looking for with regular expressions

3. Use `nltk.sem.extract_rels(firstNE, secondNE, chunkedSentence, corpus = 'conll2002', pattern = yourRegEx)`

Takes some time to adapt the tools to your problem.
Searching for patterns

```python
>>> IN = re.compile(r'.*\bin\b(?!\b.+)ing)')
>>> for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
...     for rel in nltk.sem.extract_rels('ORG', 'LOC', doc,
...     corpus='ieer', pattern = IN):
...         print nltk.sem.show_raw_rtuple(rel)
[ORG: 'WHYY'] 'in' [LOC: 'Philadelphia']
[ORG: 'McGlashan &AMP; Sarrail'] 'firm in' [LOC: 'San Mateo']
[ORG: 'Brookings Institution'] ', the research group in' [LOC: 'Washington']
[ORG: 'Idealab'] ', a self-described business incubator based in' [LOC: 'Los Angeles']
[ORG: 'Open Text'] ', based in' [LOC: 'Waterloo']
[ORG: 'WGBH'] 'in' [LOC: 'Boston']
[ORG: 'Bastille Opera'] 'in' [LOC: 'Paris']
[ORG: 'Omnicom'] 'in' [LOC: 'New York']
[ORG: 'DDB Needham'] 'in' [LOC: 'New York']
[ORG: 'Kaplan Thaler Group'] 'in' [LOC: 'New York']
[ORG: 'BBDO South'] 'in' [LOC: 'Atlanta']
[ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta']
```

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
A useful resource: WordNet

What?

- WordNet is a thesaurus of the English language.
- Synonyms are collected in SynSets.
- Nouns have hypernyms and hyponyms.

```python
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
>>> wn.synset('car.n.01').lemma_names
['car', 'auto', 'automobile', 'machine', 'motorcar']
>>> motorcar = wn.synset('car.n.01')
>>> motorcar.hypernyms()
[Synset('motor_vehicle.n.01')]
>>> wn.synset('tree.n.01').part_meronyms()
```
Overview

1. Warm-up

2. Today’s Goal: Using NLTK
   - Motivation
   - Natural Language Processing with NLTK
     - NER, Tokenizing, POS-Tagging
     - Stemming, Lemmatizing
     - Parsing
     - Relation Extraction
     - WordNet
Today's Goal: Using NLTK

Repetition

Tools

- Using Regular Expressions in Python
- Sentence Tokenizer
- Tokenizer
- POS-Tagger
- Named Entity Recognizer
- Stemmer / Lemmatizer
- Parser
- Searching for patterns