Mittwoch: Python für NLP

8 Python für NLP
- NLTK
- spaCy

Python für NLP

Was ist NLTK?

- Zusammenstellung von Python-Modulen für NLP-Lehre, -Forschung und -Entwicklung
- Code für Vielzahl von Aufgaben aus der Sprachverarbeitung
- 40 bekannte Korpora
- ausführliche Dokumentation inkl. Online-Buch

Ein einfaches Beispiel

- Einbindung in Python-Programme:

```
>>> text = 'NLTK is my favourite natural language toolkit.'
>>> import nltk
>>> nltk.WhitespaceTokenizer().tokenize(text)
['NLTK', 'is', 'my', 'favourite', 'natural', 'language',
 'toolkit.']
```

Was NLTK bietet

- Infrastruktur als Grundlage für NLP-Programme in Python:
  1. Grundlegende Klassen zur Repräsentation NLP-relevanter Daten
  2. Standard-Schnittstellen für übliche Aufgaben, z.B. Tokenisierung, Tagging, Parsing
  3. Demos, z.B. Parser, Chunker, Chatbots
  4. Ausführliche Dokumentation, Tutorien, Referenzen

Dokumentation

- Online-Buch: `http://www.nltk.org/book/`
- API-Dokumentation:
  `https://www.nltk.org/api/nltk.html`

Hilfe

- Online-Hilfe-Funktion:

```
>>> help(nltk.StemmerI)
class StemmerI(builtins.object)
 |  A processing interface for removing morphological affixes fro
 |  words.  This process is known as stemming.
 |
 |  Methods defined here:
 |
 |  stem(self, token)
 |      Strip affixes from the token and return the stem.
...
```

- Doc-Strings:

```
>>> print nltk.tag.BrillTagger.__doc__

    Brill's transformational rule-based tagger.  Brill
    taggers use an X{initial tagger} (such as
    L{tag.DefaultTagger}) to assign an intial tag
    sequence to a text; and then apply an ordered list
    of ...
```

Module

- NLTK ist in *Module* organisiert, die unterschiedliche Funktionalität bereitstellen.
- Wir schauen uns einige Module beispielhaft genauer an.

Modul `grammar` I

- Klassen zur Repräsentation kontextfreier Grammatiken (CFGs), Regeln, Nichtterminalzeichen, probabilistischer CFGs
- Funktionen zum Prüfen der Abdeckung, Induzieren probabilistischer CFGs, Parsen von Grammatiken, ...

Modul grammar II

```
>>> from nltk import grammar
>>> gr = grammar.CFG.fromstring("""
... S -> NP VP
... PP -> P NP
... NP -> Det N | NP PP
... VP -> V NP | VP PP
... Det -> 'a' | 'the'
... N -> 'dog' | 'cat'
... V -> 'chased' | 'sat'
... P -> 'on' | 'in'
... """)
>>> gr
<Grammar with 14 productions>
>>> gr.check_coverage("a cat chased a mouse".split())
ValueError: Grammar does not cover some of the input
words: u"'mouse'".
```

http://www.nltk.org/howto/grammar.html

Modul `corpus`

- Korpus-Reader für 40 NLTK-interne & -externe Korpora
- Beispiele für Funktionen einzelner Reader: `word()`, `sents()`, `paras()`, `tagged_words()`, `chunked_sents()`, `parsed_paras()`, `raw()`

```
>>> from nltk.corpus import brown
>>> print brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury',
 'said', ...]
```

Modul tag

- Klassen für verschiedene Taggermodelle: AffixTagger, BrillTagger, NGramTagger, HMM-Tagger

```
>>> from nltk import corpus, tag
>>> tagger = tag.UnigramTagger(corpus.brown.tagged_sents(),
...                            backoff=tag.DefaultTagger('NN'))
>>> tagger.tag(['The', 'dog', 'chewed', 'on', 'a', 'bone', '.'])
[('The', 'AT'), ('dog', 'NN'), ('chewed', 'VBD'), ('on', 'IN'),
('a', 'AT'), ('bone', 'NN'), ('.', '.')]
```

Modul `probability`

- Klassen zur Berechnung von Statistiken über Ergebnisse von Experimenten, unter anderem
  - FreqDist: Häufigkeitsverteilung
  - ConditionalFreqDist: Bedingte Häufigkeitsverteilungen, z.B.: Wie oft wird ein Wort mit welchem Tag versehen?

```
>>> from nltk import probability, tokenize
>>> probability.FreqDist(
...     tokenize.word_tokenize("this is a sentence"))
FreqDist({'this': 1, 'a': 1, 'is': 1, 'sentence': 1})
```

Beispiel: Satzgenerierung I

Beispiele entnommen aus Madnani (2007)

```
>>> from nltk.corpus import gutenberg
>>> from nltk.probability import ConditionalFreqDist
>>> from random import choice

# Create distribution object
>>> cfd = ConditionalFreqDist()

# For each token, count current word given previous word
>>> prev_word = None
>>> for word in gutenberg.words('austen-persuasion.txt'):
...     cfd[prev_word][word] += 1
...     prev_word = word
```

Beispiel: Satzgenerierung II

```
# Start predicting at the given word, say 'therefore'
>>> word = 'therefore'
>>> i = 1

# Find all words that can possibly follow the current word
# and choose one at random
>>> while i < 20:
...     print word
...     lwords = []
...     for next_word, count in cfd[word].items():
...         lwords.extend([next_word]*count)
...     follower = choice(lwords)
...     word = follower
...     i += 1

therefore it known of women ought . Leave me so well
placed in five altogether well placed themselves delighted
```

Weitere Module

Weitere Module sind unter anderem:

- classify
- metrics
- parse
- sem
- sentiment

Online demos

http://text-processing.com/demo/

Übung 11