NLTK - The Natural Language Toolkit

Armin Schmidt (armin.sch@gmail.com)

Resourcen-Vorkurs, SS 08 Uni Heidelberg

Plan für den 2. April 08, Teil 2

1. Einführung
   (a) Wer, Wie, Was ist NLTK?
   (b) Was bietet NLTK?
   (c) Wo finde ich Hilfe?
   (d) Wie kann ich beitragen?
   (e) FAQ
2. Struktur und Inhalt
   (a) Übersicht über die Module
   (b) Bsp. 1: Satzgenerierung mit Word-Predictor
   (c) Bsp. 2: Part-of-Speech-Tags
   (d) Bsp. 3: Word-Association
3. Übungen

Einführung - Wer, Wie, Was?

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

Einführung - Wer, Wie, Was?

- Projekt-Administratoren: Steven Bird (University of Melbourne), Edward Loper (University of Pennsylvania), Ewan Klein (University of Edinburgh)
- viele weitere Entwickler
- Beiträge von Studenten & Forschern aus aller Welt

Einführung - Wer, Wie, Was?

- Einbindung in Python-Programme:

```
>>> text = '''NLTK, the Natural Language Toolkit, is a suite of program
... modules, data sets and tutorials supporting research and teaching in
... computational linguistics and natural language processing.'''
>>> import nltk
>>> nltk.LineTokenizer().tokenize(text)
['NLTK, the Natural Language Toolkit, is a suite of program', 'modules,
data sets and tutorials supporting research and teaching in', 'computatio
linguistics and natural language processing.']
```

- schneller: `from ... import ...` - Konstrukt

Einführung - Was bietet NLTK?

- Infrastruktur als Grundlage für NLP-Programme in Python:
  1. Grundlegende Klassen zur Representation 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

Einführung - Was bietet NLTK?

- Online-Buch:
  1. Grundlagen: Textverarbeitung, Tokenization, Tagging, Lexikons, Language Engineering, Text Classification
  2. Parsing: Phrase Structure, Trees, Grammars, Chunking, Parsing
  3. Fortgeschrittene Themen: Feature-Based Grammar, Unification, Semantics, Linguistic Data Management

Einführung - Wo finde ich Hilfe?

- Online-Hilfe-Funktion:

```
>>> help(nltk.FreqDist)
class WordnetStemmer(nltk.stem.api.StemmerI)
 |  A stemmer that uses Wordnet's built-in morphy function.
 |
 |  Method resolution order:
 |      WordnetStemmer
 |      nltk.stem.api.StemmerI
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Create a new wordnet stemmer.
...
```

Einführung - Wo finde ich Hilfe?

- Mailing-Listen: `nltk-announce, nltk-devel, nltk-users`
- NLTK-Homepage: http://nltk.org, inkl. Tutorien, Code-Beispielen, Chatroom
- NLTK-Dokumentation: http://nltk.org/index.php/Documentation, inkl. Buch, API-Dokumentation

Einführung - Wie kann ich beitragen?

- Open-Source-Gemeinschaftsprojekt
- Email an User Group, Bugreports, Feature vorschlagen
- Patch einreichen, Rezepte für's Kochbuch, Doku-Übersetzung
- Liste mit Projektvorschlägen unter http://nltk.org/index.php/Projects, z.B.:
  - Develop a morphological analyser for a language of your choice
  - Develop a coreference resolution system
  - Develop a program for unsupervised learning of phonological rules, using the method described by Goldwater and Johnson
  - Implement a dependency parser

Einführung - FAQ

- Welche Lizenz benutzt NLTK? - GNU Public License
- Wie wird NLTK-Entwicklung unterstützt? - Gelegentliche Funds für Studenten, die an einem bestimmten Projekt arbeiten oder ein Praktikum suchen
- Wissenschaftliche Artikel über NLTK? - siehe http://scholar.google.com.au/scholar?q=NLTK
- Was ist der Unterschied zw. NLTK und NLTK-Lite? - NLTK-Lite ist einfacher, schneller und stellt weniger Ansprüche an der Programmierer. Wo möglich, werden Standard-Python-Objekte statt angepasster NLP-Versionen genutzt. Wenn fertig, wird es die gleiche Funktionalität wie NLTK liefern.

Struktur und Inhalt - Übersicht Module

- NLTK-Api unter: http://nltk.org/doc/api/
- Dokumentation aller Module, Klassen, Funktionen, Variablen

Struktur und Inhalt - Übersicht Module

Modul *cfg*

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

Struktur und Inhalt - Übersicht Module

Modul *classify*

- Klassifikation von: Wörtern, Sätzen, Dokumenten, etc. anhand von Merkmalen (”Feature-Sets”)

```
>>> # Define a feature extraction function.
>>> def document_features(document):
...     return dict([('contains-word(%s)'%w,True) for w in document])

>>> # Classify each Gutenberg document.
>>> for file in nltk.corpus.gutenberg.files():
...     doc = nltk.corpus.gutenberg.tokenized(file)
...     print doc_name, nltk.NaiveBayesClassifier.classify(
        document_features(doc))
```

- Trainieren & Anwenden versch. Klassifizierer: DecisionTree, NaiveBayes, Weka, Maxent ...

Struktur und Inhalt - Übersicht Module

Modul *corpus*

- Korpus-Readers für 40 NLTK-interne & -externe Korpora
- Bsp. 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', ...]
```

- Paket `nltk.corpus.reader` stellt versch. Reader zur Verfügung (`TaggedCorpusReader`, `PlaintextCorpusReader`) → können auch auf fremde Korpora übertragen werden

Struktur und Inhalt - Übersicht Module

Modul *draw*

- Submodule:
  - nltk.draw.cfg': Visualisierungstools für CFGs.
  - nltk.draw.plot': Einfaches Toll zum Plotten von Funktionen.
  - nltk.draw.tree': Graphisches Darstellen von Baumstrukturen.
- Klassen zum Explorieren von Parsern:
  - nltk.draw.chart
  - nltk.draw.rdparser
  - nltk.draw.srparser

Struktur und Inhalt - Übersicht Module

Modul *probability*

- Klassen zur Berechnen von Statistiken über Ergebnisse von Exeperimenten, z.B.
  - FreqDist: Häufigskeitsverteilung
  - DictionaryProbDist: Häufigkeiten direkt anhand eines Lexikons spezifiziert
  - GoodTuringProbDist: Good-Turing-Schätzung einer Verteilung

```
>>> fdist = FreqDist()
>>> for word in tokenize.whitespace(sent):
...     fdist.inc(word.lower())
```

Struktur und Inhalt - Übersicht Module

Andere Module

- chunk
- cluster
- containers
- data
- featstruct
- inference
- sem
- stem
- tokenize
- wordnet

Bsp. 1: Satzgenerierung mit Word-Predictor

(Beispiele entnommen aus Madnani (2007): http://www.umiacs.umd.edu/ nmadnani/pdf/crossroads.pdf)

```
>>> 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'):
. . . cfd[prev_word].inc(word)
. . . prev_word = word
# Start predicting at the given word, say 'therefore'
>>> word = 'therefore'
>>> i = 1
```

Bsp. 1: Satzgenerierung mit Word-Predictor

```
# Find all words that can possibly follow the current word
# and choose one at random
>>> while i < 20:
. . . print word,
. . . lwords = cfd[word].samples()
. . . 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
```

Bsp. 2: Part-of-Speech-Tags

```
>>> from nltk.corpus import brown
>>> from nltk.probability import FreqDist, ConditionalFreqDist
>>> fd = FreqDist()
>>> cfd = ConditionalFreqDist()
# for each tagged sentence, get the (token, tag) pair and update
# both count(tag) and count(tag given token)
>>> for text in brown.items:
. . . for sentence in brown.tagged_sents(text):
. . . for (token, tag) in sentence:
. . . fd.inc(tag)
. . . cfd[token].inc(tag)
>>> fd.max() # The most frequent tag is ...
'NN'
```

Bsp. 2: Part-of-Speech-Tags

```
>>> wordbins = [] # Initialize a list to hold (numtags,word) tuple
# append each (n(unique tags for token),token) tuple to list
>>> for token in cfd.conditions():
. . . wordbins.append((cfd[token].B(), token))
. . .
# sort tuples by number of unique tags (highest first)
>>> wordbins.sort(reverse=True)
>>> print wordbins[0] # token with max. no. of tags is ...
(12, 'that')
```

Bsp. 2: Part-of-Speech-Tags

```
>>> male = ['he','his','him','himself'] # masculine pronouns
>>> female = ['she','hers','her','herself'] # feminine pronouns
>>> n_male, n_female = 0, 0 # initialize counters
# total number of masculine samples
>>> for m in male:
. . . n_male += cfd[m].N()
. . .
# total number of feminine samples
>>> for f in female:
. . . n_female += cfd[f].N()
. . .
>>> print float(n_male)/n_female # calculate required ratio
3.257
```

Bsp. 3: Word-Association

```
>>> from nltk.corpus import brown, stopwords
>>> from nltk.probability import ConditionalFreqDist
>>> cfd = ConditionalFreqDist()
# get a list of all English stop words
>>> stopwords_list = stopwords.words('english')
# define a function that returns true if the input tag is some form of no
>>> def is_noun(tag):
. . . return tag.lower() in ['nn','nns','nn$','nn-tl','nn+bez',
            'nn+hvz', 'nns$','np','np$','mp+bez','nps', 'nps$',
            'nr','np-tl','nrs','nr$']
```

Bsp. 3: Word-Association

```
# count nouns that occur within a window of size 5 ahead of other nouns
>>> for text in brown.items:
...     for sentence in brown.tagged_sents(text):
...         for (index, tagtuple) in enumerate(sentence):
...             (token, tag) = tagtuple
...             token = token.lower()
...           if token not in stopwords_list and is_noun(tag):
...                 window = sentence[index+1:index+5]
...                 for (window_token, window_tag) in window:
...                     window_token = window_token.lower()
...                     if window_token not in stopwords_list and \
                                                 is_noun(window_tag):
...                         cfd[token].inc(window_token)
```

Bsp. 3: Word-Association

```
>>> print cfd['bread'].max()
cheese
>>> print cfd['life'].max()
death
>>> print cfd['man'].max()
woman
>>> print cfd['woman'].max()
world
>>> print cfd['boy'].max()
girl
>>> print cfd['girl'].max()
trouble
```