NLTK-Buch Kapitel 1

Simon Clematide
simon.clematide@uzh.ch

Institut für Computerlinguistik
Universität Zürich

Programmiertechniken in die Computerlinguistik I

Übersicht

Lernziele

NLTK

- NLTK installieren, kennenlernen und selbst anwenden
- Korpuslinguistische Funktionen kennenlernen
- Gutenberg-Korpora als Sequenz von Tokens einlesen

Technisches

- Was sind Module und Packages? Wie kann man sie importieren?
- Was ist Listenkomprehension? Wie funktioniert sie?
- Was ist bei der Definition von Funktionen zu beachten? Was bewirkt das Statement `return`?
- Was sind globale und lokale Namensräume?

NLTK (Natural Language Toolkit)

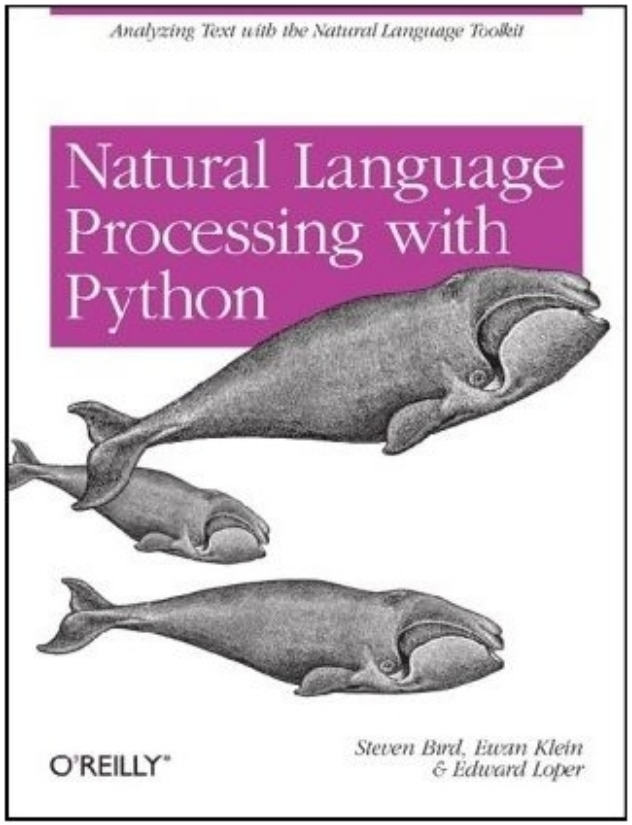
NLTK-Framework `http://www.nltk.org`

- Sammlung von Open-Source-Python-Modulen für die Sprachverarbeitung (Natural Language Processing, NLP)
- Ressourcen: frei verfügbare Korpora, Treebanks, Lexika ...
- Applikationen: Tokenizer, Stemmer, Tagger, Chunker, Parser, Semantik, Alignierung...
  (oft eher Toy-Implementationen für Lehrzwecke; andere Toolkits sind industrial-strength `http://spacy.io`)
- Module für Evaluation, Klassifikation, Clustering, Maschinelles Lernen (Schnittstellen zu State-of-the-Art-Bibliotheken)
- API (Application Programming Interface) für WordNet und Lexika

Installationsanleitungen für Win, Mac, Linux

`http://www.nltk.org/install.html`

Bird et. al (2009): Natural Language Processing in Python[1]



- Praktische Einführung in NLP mit Hilfe von NLTK 3
- Anwendungsorientiert, keine vertiefte Einführung in Python-Konzepte
- Kursbuch für PCL I (und auch II)
- 2. Edition (nur online!) benutzt Syntax von Python 3 (kompatibel mit futurisiertem Python 2)
- 1. Edition (NLTK 2!) als PDF abgelegt im Materialordner
- Weiteres NLTK-basiertes Buch mit vielen NLP-Rezepten: [Perkins 2010]

[1] `http://www.nltk.org/book` ; 1. Edition online unter `http://www.nltk.org/book_1ed`

Modulübersicht

Language processing task	NLTK modules	Functionality
Accessing corpora	nltk.corpus	standardized interfaces to corpora and lexicons
String processing	nltk.tokenize, nltk.stem	tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	nltk.tag	n-gram, backoff, Brill, HMM, TnT
Classification	nltk.classify, nltk.cluster	decision tree, maximum entropy, naive Bayes, EM, k-means
Chunking	nltk.chunk	regular expression, n-gram, named-entity
Parsing	nltk.parse	chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	nltk.sem, nltk.inference	lambda calculus, first-order logic, model checking
Evaluation metrics	nltk.metrics	precision, recall, agreement coefficients
Probability and estimation	nltk.probability	frequency distributions, smoothed probability distributions
Applications	nltk.app, nltk.chat	graphical concordancer, parsers, WordNet browser, chatbots
Linguistic fieldwork	nltk.toolbox	manipulate data in SIL Toolbox format

Verzeichnisstruktur vom NLTK 3

```
$ tree -F /Library/Python/2.7/site-packages/nltk/
/Library/Python/2.7/site-packages/nltk
|-- VERSION
|-- __init__.py
|-- __init__.pyc
|-- app
|   |-- __init__.py
|   |-- __init__.pyc
|   |-- chartparser_app.py
|   |-- chartparser_app.pyc
|   |-- chunkparser_app.py
|   |-- chunkparser_app.pyc
...
|-- util.py
...
24 directories, 648 files
```

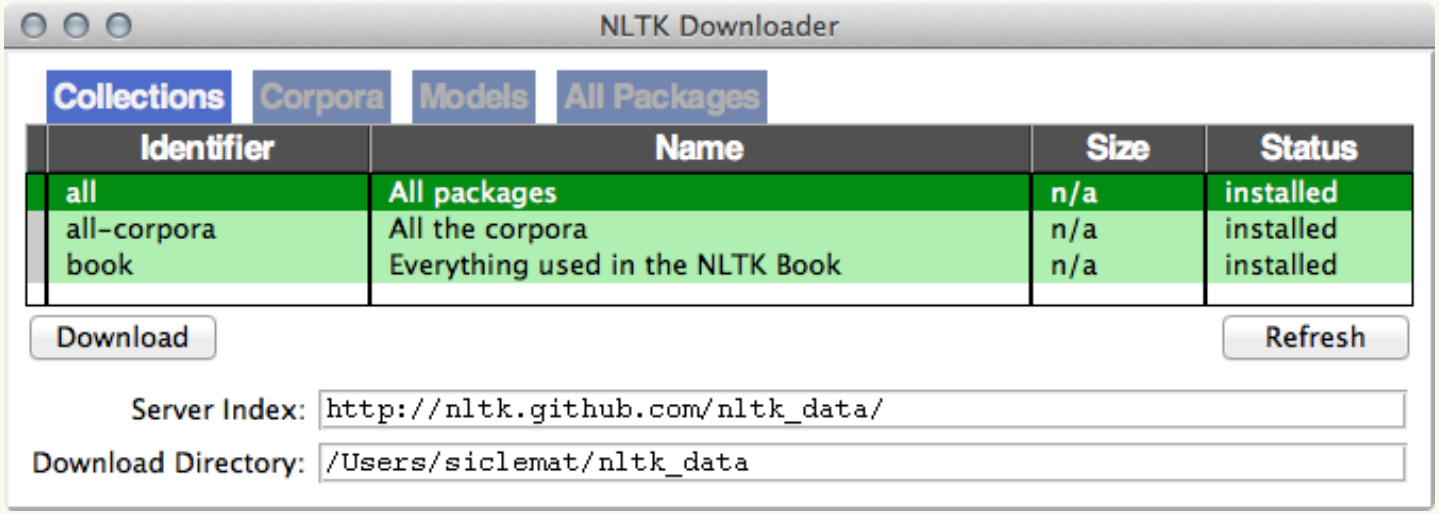
- Module: Dateien mit Python-Quellkode: `util.py`
- Maschinenunabhängig kompilierter Bytekode: `chunkparser_app.pyc`
- Packages: Verzeichnisse wie `nltk` oder `app` mit `__init__.py`

NLTK-Data: Korpora und weitere Ressourcen

Installation der Ressourcen-Sammlung `nltk_data`

Um die Beispiele im Buch ausführen zu können, muss das Verzeichnis `nltk_data` runtergeladen werden.
Nur die Kollektion `book` ist notwendig für PCL I.

```
>>> import nltk
>>> nltk.download()
```



Module importieren

Anweisung: `import` *Module* ▸1

```
# Importiere Modul book aus Package nltk
import nltk.book

# Objekte und Funktionen aus nltk.book können nur in
# vollqualifizierter Punktnotation bezeichnet werden.
print "Zweites Wort aus text1:", nltk.book.text1[1]

# Objekte und Funktionen können nicht direkt bezeichnet werden:
print text1[1]
```

Alle Objekte und Funktionen aus Modulen importieren

Anweisung: `from Module import *` ▸2

```
# Lade Modul book aus Package nltk und
# importiere alle Objekte und Funktionen ins aktuelle Modul
from nltk.book import *

# Objekte und Funktionen aus nltk.book können ohne
# Modulpräfixe bezeichnet werden.
print "Zweites Token aus text1:", text1[1]

# Die vollqualifizierter Punktnotation geht dann nicht
print "Zweites Wort aus text1:", nltk.book.text1[1]
```

Python 2 futurisieren

Wichtige Direktiven zur Kompatibilität von Python 2 mit Python 3

```
from __future__ import print_function, unicode_literals, division
```

Python 2: print-Statement

```
>>> print u'sähe:', count
```

Python 3: print-Funktion ▶

```
>>> print('sähe:', count)
```

Python 2: Ganzzahldivision

```
>>> print 1/3
0
```

Python 3: Division

```
>>> print(1/3)
0.3333333333333333
```

Eine Tour durch Kapitel 1

- Repräsentation von Text-Korpora als Objekt vom Typ Text (im Wesentlichen als Liste von String-Token)
- KWIC (*Keyword in context*): Konkordanzen erstellen und anzeigen
- Vorkommensähnlichkeit (*similarity*): Welche unterschiedlichen Wörter erscheinen häufig in ähnlichen Kontexten?
- Häufigkeitsverteilungen (*frequency distribution*) berechnen: Wie oft kommt welche Wortform vor?
- Statistische Kollokationen (*collocations*): Welche Wortpaare kommen viel häufiger zusammen vor als zufällig zu erwarten wäre?
- Dispersion-Plot (Korpuslinguistik): An welchen Stellen in einem Korpus kommt ein Wort vor? [Baker et al. 2006]

Visualisierungen mit Plotting benötigen separate Diagramm-Bibliothek matplotlib (Download via `matplotlib.org`).

Listenkomprehension (list comprehension)

Mathematische Mengenkomprehension

Die Menge aller Quadratzahlen aus der Grundmenge der Zahlen von 0 bis 9.

$$\{x^2 \mid x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$

In Python mit Listen als Mengen:

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Syntax

```
[ x**2 for x in range(10) ]
```

Bildungsvorschrift

Grundmenge, deren Werte x durchläuft

Listen ohne Listenkomprehension

Auftrag: Baue eine zu Liste `a` äquivalente Liste `b` auf, ohne Listenkomprehension zu benützen, dafür aber mit einer normalen For-Schleife?

```
a = [x**2 for x in range(10)]
```

Listenkomprehension mit Bedingungen ▶

Syntaxschema (aus [Bird et al. 2009, 19])

$\{w \mid w \in V \,\&\, P(w)\}$

`[w for w in V if p(w)]`

Die Liste aller Elemente w aus V, für die die Eigenschaft $P(w)$ wahr ist.

Filtern von Vokabularlisten ▸3

```
from nltk.book import *

words = set(text1)

longwords = [w for w in words if len(w)>15]
```

`set(text1)` erzeugt Menge aller Listenelemente aus `text1`.

Funktionen definieren und aufrufen

Definition der einstelligen Funktion `foo()` ▸4

```
def foo(a):
    """ Return foo of a """
    result = 0
    for item in a:
        result += item
    return result
```

`return`-Statement ergibt Rückgabewert, d.h. den Funktionswert.
Die Parameter der Funktion (Platzhalter für Argumente) stehen in Klammern. Die erste Zeichenkette im Rumpf ist der Doc-String.

Funktionsaufruf (call)

```
c = foo([5,10,23])
```

Mehrere `return`-Anweisungen

Effekt der return-Anweisung ▸5

```
def describe_number(n):
    if n > 1000000:
        return "LARGE"
    elif n > 1000:
        return "Medium"
    else:
        return "small"
    print "Never printed!"
```

- Verarbeitung der `return`-Anweisung beendet die Ausführung der Funktion.
- Beliebige Objekte können als Funktionswert zurückgegeben werden, auch Listen.

Wann und wozu sind Funktionen gut?

```
def foo(a):
    """ Return foo of a """
    result = 0
    for item in a:
        result += item
    return result
```

Heilmittel gegen Spaghettikode

- Abstraktion: Eine Funktion kann einige Zeilen Kode bezeichnen, welche oft gebraucht werden.
- Schnittstelle: Die Parameter einer Funktion machen den Kode an ganz bestimmten Stellen variabel (= Parametrisierung).
- Klarheit: Eine gute Funktion hat eine klar beschreibbare (=spezifizierbare) Funktionalität.

Skopus (Erreichbarkeit) von Variablennamen

Modulweit erreichbare globale Variablen (*globals*)

- (Variablen-)Namen, die in einem Modul zugewiesen werden, sind danach im ganzen Modul erreichbar.
- Modul `foo` ist Python-Quellkode aus Datei `foo.py`.

Funktionsweit erreichbare lokale Variablen (*locals*)

- Parameter `a` und `b` einer Funktion `foo(a,b)`, sind nur innerhalb der Funktion `foo()` erreichbar.
- (Variablen)-Namen, die in einer Funktion definiert werden, sind nur innerhalb der Funktion erreichbar.

Introspektion

Die eingebauten Funktionen `globals()` und `locals()` geben die zum Aufrufzeitpunkt definierten Namen aus.

Globale und lokale Variablennamen

Derselbe Name kann global und lokal unterschiedliche Werte haben.

Auszug aus globals_and_locals.py ▶ ▸6

```
a = "Globale Variable"

def foo(a):
    print "In Funktion: a =", a
    return a

c = foo("Lokale Variable")

print "In Modul: a =", a
```

Lokale Variablennamen nur lokal!

Erreichbarkeit aus Funktionsdefinitionen ▶ ▸7

```
g = "Globale Variable"

def foo():
    a = g
    return a

def bar():
    b = a
    return b

foo()
bar()
```

Was passiert?

Wie und wo entstehen überall Variablennamen?

Sind die entstehenden Namen lokal oder global?

Vertiefung

- Pflichtlektüre: Kapitel 1.1. bis und mit 1.4 aus [Bird et al. 2009] online lesen `http://nltk.org/book`
- Enthält nochmals anschauliche Repetition zu vielen bisher behandelten Themen (Listen, Zeichenketten, Bedingungen)

Liste der verlinkten Programme und Ressourcen I

Folie

Literaturangaben I

- Baker, Paul, A. Hardie und T. McEnery (2006).
  A glossary of corpus linguistics. Edinburgh University Press, Edinburgh.
- Bird, Steven, E. Klein und E. Loper (2009).
  Natural Language Processing with Python. O'Reilly.
- Perkins, Jacob (2010).
  Python Text Processing with NLTK 2.0 Cookbook. Packt Publishing.