

NLTK-Buch Kapitel 3

Simon Clematide

`simon.clematide@uzh.ch`

Institut für Computerlinguistik
Universität Zürich

Programmiertechniken in die Computerlinguistik I

Übersicht

Konkordanzen

- Formatierungsausdrücke

- Stemmer

- Textindexklasse

Technisches

- Generatorausdrücke

- xrange

- Ausnahmen

- with

Lernziele

NLTK

- ▶ KWIC mit eigenen Klassen definieren
- ▶ Erstellung eines Index
- ▶ Porter-Stemmer für Englisch

Technisches

- ▶ Formatierungsausdrücke
- ▶ Generatoren als Ausdrücke
- ▶ Generatoren mit `yield` in Funktionsdefinitionen
- ▶ Ausnahmen (exceptions) behandeln
- ▶ Dateien öffnen mit dem `with`-Konstrukt

Motivation

Ziel: KWIC in Python

Eine Klasse programmieren, welche eine Konkordanz über einem gestemmtten Index anzeigt.

Beispiel-Output

```
>>> text.concordance('die', width=30)
BLACK KNIGHT : Then you shall die . ARTHUR : I command you
Camelot . He was not afraid to die , O brave Sir Robin .
Concorde , you shall not have died in vain ! CONCORDE : Uh
                2 : Oh , he ' s died ! FATHER : And I want
that ? MAYNARD : He must have died while carving it . LAUNCE
ARTHUR : Look , if he was dying , he wouldn ' t bother
```

Beispiel: Konkordanzprogramm über gestemmtten Wörtern

KWIC als Klasse: Datenstrukturen und Funktionalitäten

- ▶ **Text**: Folge von Wörtern
- ▶ **Index**: Abbildung von (gestemmttem) Wort zu allen Vorkommenspositionen im Text
- ▶ **Stemmer**: Stemming von Wortformen
- ▶ **KWIC-Anzeige**: Formatierung der Treffer im KWIC-Stil

Benötigte Kompetenzen

- ▶ Wie lassen sich (einfache) Klassen definieren?
- ▶ Definition eigener Regex-Stemmer oder Benutzung von NLTK-Stemmern
- ▶ Formatierung von zentriertem textuellem Output mit Format-Ausdrücken

Formatierung mit Hilfe von Format-Ausdrücken

- ▶ Flexiblere Kontrolle für Ausgabe von Zahlen und Strings ist erwünscht
- ▶ Formatierungsausdruck: 'STRINGTEMPLATE WITH FORMATS' % TUPLE
- ▶ Ein **Formatierungsausdruck** (*string formatting expression*) trennt Layout (Platzhalter %d, %f für Zahlen, %s für Strings) von den variablen Daten (Tupel)
- ▶ Anzahl Nachkommastellen ('%.2f'), Padding mit Leerzeichen ('% 4.2f'), linksbündig ('%-7s'), rechtsbündig('%7s')

```
>>> 'a string:%s and an integer:% 4d' % ('abc',3)
'a string:abc and an integer:   3'
```

```
>>> 'Padding a string:%-6s and a float:% 8.2f' % ('abc',3.175)
'Padding a string:abc      and a float:   3.17'
```

Formatierungsausdrücke

Überraschung

►1

```
>>> '%.1f' % 0.05
'0.1'
>>> '%.1f' % 0.15
'0.1'
>>> round(0.05,1)
0.1
>>> round(0.15,1)
0.1
```

💡 Schulregel mit aufzurundendem
.5 verzerrt systematisch (*bias*) ►2

Prozentzeichen schützen mit %

```
>>> '%.1f%%' % 0.15
'0.1%'
```

Variables Padding mit *

```
>>> width = 8
>>> '%*s' % (width, 'abc')
'      abc'
>>> '%-*s' % (width, 'abc')
'abc      '
```

Regex-Stemmer-Klasse definieren

Klasse mit optionalem Konstruktor-Argument



```
class RegexStemmer(object):  
    def __init__(self, r=r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'):  
        self._r = r  
  
    def stem(self, word):  
        m = re.match(self._r, word)  
        return m.group(1)
```

Initialisierung und Verwendung

```
regex_stemmer = RegexStemmer()  
regex_stemmer.stem('seeming')
```


Textindex als Klasse IndexedText definieren

```
class IndexedText(object):
    def __init__(self, stemmer, text):
        self._text = text
        self._stemmer = stemmer
        self._index = nltk.Index((self._stem(word), i)
                                for (i, word) in enumerate(text))
```

`enumerate(s)` generiert Paare (Position,Element) aus Sequenz

```
l = ['wenn', 'fliegen', 'hinter', 'fliegen', 'fliegen']
>>> list(enumerate(l))
[(0, 'wenn'), (1, 'fliegen'), (2, 'hinter'), (3, 'fliegen'), (4, 'fliegen')]
```

`nltk.Index(Pairs)` erzeugt invertierten Index aus (Element,Position)

```
>>> index = nltk.Index((w,i) for (i,w) in enumerate(l))
>>> index['fliegen']
[1, 3, 4]
```

Private und öffentliche Methode von IndexedText

Unterstrich markiert Privatheit: Nur für Benutzung **in** der Klasse

```
def _stem(self, word):  
    return self._stemmer.stem(word).lower()
```

Öffentliche Methode für Formatierung

```
def concordance(self, word, width=40):  
    key = self._stem(word)      # stemmed keyword  
    wc = width/4                # words of context  
    for i in self._index[key]:  
        lcontext = ' '.join(self._text[i-wc:i])  
        rcontext = ' '.join(self._text[i:i+wc])  
        ldisplay = '%*s' % (width, lcontext[-width:])  
        rdisplay = '%*s' % (width, rcontext[:width])  
        print ldisplay, rdisplay
```



Noch privater sind Attribute der Form o. `__NAME`.

Generatorausdrücke (*generator expressions*)

Listenkomprehension: Prinzip “Liste aller Dinge, die ...”

Baue die **Liste** aller kleingeschriebenen Wörter aus dem Brown-Korpus und erzeuge danach aus der Liste eine Menge!

```
set([w.lower() for w in nltk.corpus.brown.words()])
```

Generatorausdrücke: Prinzip “Der Nächste, bitte!”

Nimm ein kleingeschriebenes Wort nach dem andern und mache es zum Element der Menge!

```
set(w.lower() for w in nltk.corpus.brown.words())
```

Listenkomprehension vs. Generatorausdrücke

Generatorausdrücke statt Listenkomprehension

Im NLTK-Buch wird aus Effizienzgründen `set(w.lower() for w in text)` statt `set([w.lower() for w in text])` notiert.

- ▶ **Listenkomprehension** erzeugt im Arbeitsspeicher immer eine **Liste aller Elemente**.
- ▶ **Generatorausdrücke** sind **speichereffizient**. Sie übergeben ihre Element auf Verlangen einzeln der auswertenden Funktion (intern `g.next()`).
- ▶ Generatorausdrücke unterstützen darum **Längenmethode `len()`** nicht.
- ▶ Generatorausdrücke unterstützen **kein Slicing**: `1[:10]`.
- ▶ Mit `list(generator)` wird jeder Generator zur Liste.
- ▶ Speichereffizienz ist bei allen Funktionen optimiert, welche Daten vom Typ *iterable* verarbeiten: `max()`, `sum()`, `set()` usw.
- ▶ Generatoren sind nach 1 Durchgang erschöpft, d.h. aufgebraucht!

Generatorausdrücke und die Iteratorfunktion

next() ▶4

```
>>> quadrat = (i*i for i in [10,11])
>>> quadrat
<generator object <genexpr> at 0x16a6f80>
>>> type(quadrat)
<type 'generator'>
>>> quadrat.next()
100
>>> quadrat.next()
121
>>> quadrat.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

§ Die Ausnahme (*exception*) `StopIteration` erscheint, wenn der Generator erschöpft ist. Konsumenten von Generatoren müssen Ausnahme korrekt behandeln.

Generatorfunktionen mit yield

►5

```
def quadriere(iterierbar):  
    for i in iterierbar:  
        yield i*i
```

```
quadrat = quadriere([10,11])
```

```
>>> quadrat
```

```
<generator object quadriere at 0x103945500>
```

```
>>> type(quadrat)
```

```
<type 'generator'>
```

```
>>> quadrat.next()
```

```
100
```

```
>>> sum(quadrat)
```

```
121
```

```
>>> sum(quadriere([10,11]))
```

```
221
```

Rechenzeit und Speicherverbrauch messen

Programm mit Generatorausdrücken



```
import nltk, timeit, time, os
words = nltk.corpus.brown.words()

def test_generator():
    return set(w.lower() for w in words)

# Initialisiere Timer-Objekt
tg = timeit.Timer(test_generator)

# Timing von Generatorausdruck
print 'Timed generator (seconds):', tg.timeit(1)
```



Der Speicherverbrauch muss extern gemessen werden.

Rechenzeit und Speicherverbrauch messen

Programm mit Listencomprehension



```
import nltk, timeit, time, os
words = nltk.corpus.brown.words()

def test_listcomprehension():
    return set([w.lower() for w in words])

# Initialisiere Timer-Objekt
tl = timeit.Timer(test_listcomprehension)

# Timing von Listenkomprehension
print 'Timed list comprehension (seconds):', tl.timeit(1)
```



Der Speicherverbrauch muss extern gemessen werden.

Effizienz in Rechenzeit

Zufällige Auswahl von Elementen aus einem Bereich

►8

```
# Modul zur Zeitmessung von Python-Statements
```

```
import timeit
```

```
setup = 'import random'
```

```
# Konstruiere 2 Timer-Objekte
```

```
tr = timeit.Timer('random.sample( xrange(1000000),100)', setup)
```

```
tx = timeit.Timer('random.sample(xrange(1000000),100)', setup)
```

```
# Führe Timings je einmal durch und speichere Anzahl Sekunden
```

```
trsecs = tr.timeit(1)
```

```
txsecs = tx.timeit(1)
```

```
print "Aufgabe: Sample 100 Zahlen aus dem Bereich 0 bis 999999."
```

```
print "Zeit mit xrange:", txsecs, "Sekunden"
```

```
print "Zeit mit range:", trsecs, "Sekunden"
```

```
print "xrange ist etwa", trsecs/txsecs, "Mal schneller!"
```

Zufälliges Auswählen von Wörtern

Das Ziehen einer zufälligen Stichprobe (*sample*) aus einem Korpus.

►9

```
import nltk, random

corpus = nltk.corpus.nps_chat.words()

# for demonstration
for i in random.sample(xrange(len(corpus)),20):
    print corpus[i]

# as a reusable function with a generator return value
def sample_corpus1(text,size):
    return (text[i] for i in random.sample(xrange(len(text)),size))

# as a reusable function with a list return value
def sample_corpus2(text,size):
    return [text[i] for i in random.sample(xrange(len(text)),size)]
```

Häufige Exceptions

Ausnahmen können in jeder Stufe der Programmausführung auftreten!

SyntaxError

```
print 1 2
```

NameError

```
print a
```

ZeroDivisionError

```
1 / 0
```

IndexError

```
a = [1, 2, 3]  
a[3]
```

KeyError

```
a = {}  
a["test"]
```

RuntimeError

```
def x(): return x()  
x()
```

TypeError

```
sum(["1", "2", "3"])
```

Wie gehe ich mit Fehlern um?

```
x = raw_input()
```

Robuste Programmierung

- ▶ Wir wollen `x` in eine Zahl umwandeln, bei ungültiger Eingabe eine neue Eingabe verlangen.
- ▶ `float(x)` führt zu Programmterminierung
- ▶ `x.isdigit()` akzeptiert nur Teilmenge aller Zahlen
- ▶ Verkettung von Regeln möglich, aber umständlich

Ausnahmen (Exceptions)

- ▶ Ausnahmen können im Programm abgefangen werden, anstatt dass sie zur Terminierung führen.
- ▶ Oft eleganter, als Ausnahmen zu vermeiden.

Ausnahmen auffangen: *try*-Konstrukt

Syntax-Schema

```
try:  
    block1  
except E:  
    block2
```

Syntax-Schema mit finally

```
try:  
    block1  
except E:  
    block2  
finally:  
    block3
```

Erklärung

- ▶ Führe block1 aus.
- ▶ Wenn währenddessen eine Ausnahme vom Typ E auftritt, führe block2 aus
- ▶ Führe block3 auf jeden Fall am Schluss aus.



Ausnahmen sind ebenfalls Objekte und haben infolgedessen einen Typ!

Ausnahmen ignorieren

►10

```
while True:
    x = raw_input('Please type in a number: ')
    try:
        float(x)
        break
    except ValueError:
        pass
```

Leere Blöcke

- ▶ Blöcke müssen immer mindestens eine Anweisung enthalten
- ▶ pass für leere Blöcke (*no operation*, no(o)p)

Philosophien der Fehlerbehandlung: LBYL vs EAFP

LBYL

```
if w in freqs:  
    freqs[w] += 1  
else:  
    freqs[w] = 1
```

Look **b**efore **y**ou leap.



EAFP

```
try:  
    freqs[w] += 1  
except KeyError:  
    freqs[w] = 1
```

It's **e**asier to **a**sk for **f**orgiveness
than for **p**ermission.



Fehler auffangen: Wie spezifisch?

```
try:
    ...
    (ganz viel Code)
    ...
except:
    pass
```

Welche Exceptions soll man abfangen?

- ▶ Zu allgemeine except-Klauseln erschweren das Bemerken und Finden von Programmierfehlern.
- ▶ Setze try/except-Klauseln gezielt ein.
- ▶ Bestimme den Ausnahmentyp, der abgefangen werden soll.

with-Konstrukt für Datei-Handling

►11

- ▶ Das Betriebssystem erlaubt nicht, dass Hunderte von Dateien von einem Prozess geöffnet sind.
- ▶ Bei Prozessen, welche vielen Dateien lesen/schreiben, müssen die Dateien geschlossen werden.
- ▶ Das with-Konstrukt mit Datei-Objekten macht dies automatisch (was auch immer für Ausnahmesituationen beim Dateiverarbeiten entstehen).

```
filename = "with_open.py"

with open(filename, 'r') as f:
    for l in f:
        if l.rstrip() != '':
            sys.stdout.write(l)
```

Vertiefung

- ▶ Pflichtlektüre: Kapitel 3.1. bis und mit 3.6 aus dem NLTK-Buch

Verständnisfragen

- ▶ Woraus bestehen Formatierungsausdrücke?
- ▶ Was unterscheidet Generatorausdrücke von Listenkomprehension?
- ▶ Wie kann man Ausnahmen behandeln?
- ▶ Welche Möglichkeiten gibt es, die Effizienz in Rechenzeit und Arbeitsspeicher zu messen?
- ▶ Wieso ist das with-Konstrukt beim Öffnen von Dateien nützlich?

Liste der verlinkten Programme und Ressourcen I

Folie

►1 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/nltk3/round_floats.py	7
►2 Runden von floats: http://en.wikipedia.org/wiki/Rounding#Round_half_away_from_zero	7
►3 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/nltk3/stemmed_kwic.py	8
►4 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/generator/generators_next.py	13
►5 Programm: http://tinyurl.com/pcl1-hs14-generator-yield	14
►6 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/generator/timeit_generator.py	15
►7 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/generator/timeit_listcomprehension.py	16
►8 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/generator/random_sample_xrange_timeit.py	17
►9 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/generator/random_sample_xrange.py	18
►10 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/nltk3/float_raw_input.py	22
►11 Programm: http://www.cl.uzh.ch/siclemat/lehre/hs14/pcl1/lst/nltk3/with_open.py	25