NLTK-Buch Kapitel 2: Lexikalische Ressourcen

Simon Clematide
simon.clematide@uzh.ch

Institut für Computerlinguistik
Universität Zürich

Programmiertechniken in die Computerlinguistik I

Übersicht

Lernziele I

NLTK

- Zugriff auf lexikalische Ressourcen
- Effizientes Filtern von Stoppwörtern auf Textkorpora
- Lexika mit komplexer Datenstruktur

Technisches

- Eigene Klassen definieren: Spezialisierte Objekte designen
- Ober- und Unterklassen verstehen
- Konstruktorfunktion `__init__()` von Klassen verstehen
- Definieren von eigenen Methoden und Attributen (Instanzvariablen)

Wortlisten als Lexika

Definition (Wortlisten)

Die einfachste Form von Lexika sind Wortlisten.
Als Rohtext-Datei typischerweise 1 Wort pro Zeile und sortiert.

Stoppwortlisten (*stopwords*) in NLTK ▸1

```
stopwords_en = nltk.corpus.stopwords.words('english')

print len(stopwords_en), stopwords_en[::20]
# >>> 127 ['i', 'herself', 'was', 'because', 'from', 'any', 't']
```

Hinweis: Spezialsyntax `[::20]` gibt jedes 20. Element zurück.

Rechnen mit Stoppwortlisten

Was berechnet `foo()`? Was wäre ein guter Funktionsname? ▸2

```
import nltk
stopwords_en = nltk.corpus.stopwords.words('english')

# Was berechnet foo()?
def foo(text):
    """ Hier fehlt Dokumentation... """
    bar = [w for w in text if w.lower() not in stopwords_en]
    return len(bar)/len(text)*100.
```

- Wie kann man besser dokumentieren?
- Wie kann man effizienter berechnen?

Anteil von echten Inhaltswörtern bestimmen

Wie kann man die Interpunktionstoken eliminieren? ▸3

```
import re

def delete_punctuation(s):
    """ Return string with all punctuation symbols of iso-latin 1 deleted. "
    p = r'[!"#%&\x27`()*,-./:;?@[\]_{}\xa1\xab\xb7\xbb\xbf]'
    return re.sub(p,'',s)

def content_word_percentage(text):
    """ Return the percentage of content words in a list of English tokens.
    content_words = [w for w in text
                     if delete_punctuation(w) != ''
                     and w.lower() not in stopwords_en_set]
```

CMU (Carnegie Mellon University) Pronouncing Dictionary

```
File Format: Each line consists of an uppercased word,
a counter (for alternative pronunciations), and a transcription.
Vowels are marked for stress (1=primary, 2=secondary, 0=no stress).
E.g.: NATURAL 1 N AE1 CH ER0 AH0 L

The dictionary contains 127069 entries.  Of these, 119400 words are assigned
a unique pronunciation, 6830 words have two pronunciations, and 839 words have
three or more pronunciations.  Many of these are fast-speech variants.

Phonemes: There are 39 phonemes, as shown below:

    Phoneme Example Translation    Phoneme Example Translation
    ------- ------- -----------    ------- ------- -----------
    AA      odd     AA D           AE      at      AE T
    AH      hut     HH AH T        AO      ought   AO T
...
$ grep -w  RESEARCH  /Users/siclemat/nltk_data/corpora/cmudict/cmudict
RESEARCH 1 R IY0 S ER1 CH
RESEARCH 2 R IY1 S ER0 CH
```

Wie soll man solche Information in Python als Daten repräsentieren?

CMU (Carnegie Mellon University) Pronouncing Dictionary

Strukturierte Lexikoneinträge

CMU besteht aus Paaren von Lemma und Listen von phonetischen Kodes.

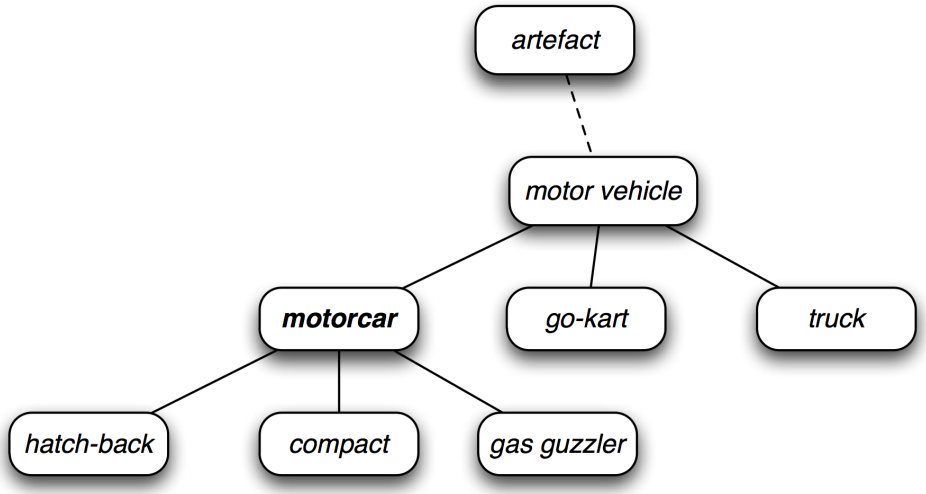
Filtern von Lexikoneinträgen ▸4

```
import nltk

entries = nltk.corpus.cmudict.entries()

print entries[71607]
# ('love', ['L', 'AH1', 'V'])]

# Finde alle Wörter auf -n, welche als -M ausgeprochen werden.
print [ word for (word,pron) in entries
          if  pron[-1] == 'M'
          and word[-1] == 'n' ]
```

WordNet: Ein Netz von Bedeutungsbeziehungen[1]

Wie lässt sich die Bedeutung eines Worts angeben?

- Klassische Charakterisierung: Umschreibung, Definition
- Relationale lexikalische Semantik = Bedeutungsbeziehungen
- Angabe von Synonymen, Hypernymen, Hyponymen, Antonymen usw., welche ein Netz (Hierarchie) von verknüpften Bedeutungen ergeben



Quelle: `http://www.nltk.org/images/wordnet-hierarchy.png`

[1] `http://wordnetweb.princeton.edu/perl/webwn`

WordNet: Komplexe lexikalische Datenstruktur[2]

Speziell zugeschnittene Datenstruktur benötigt

- Zugriff auf Bedeutungen (*synsets*) und Wörter (*lemmas*)
- Navigation im Wortnetz entlang der semantischen Relationen (Oberbegriffe, Unterbegriffe, Gegenbegriffe)
- Berechnen von semantischer Verwandtschaft (Nähe, Bezüge) im Netz

WordNet in NLTK

▸5

```
import nltk
from nltk.corpus import wordnet as wn
# Welche Bedeutungen hat das Wort "car"?
print wn.synsets('car')
# Definition einer Bedeutung
print wn.synset('car.n.01').definition()
# Alle hyponymen Bedeutungen eines Lemmas berechnen
print wn.synset('car.n.01').hyponyms()
```

[2] Siehe http://www.nltk.org/book/ch02.html#fig-wn-hierarchy

Spezialisierte Objekte entwerfen

Objekte unterstützen die Abstraktion

Objekte erlauben es, Daten und die dazugehörigen Methoden an einer "Adresse" anzusprechen

Listenobjekte: `list`

- Was sind die Daten?
- Was sind die Methoden?

Verteilungshäufigkeiten: `nltk.FreqDist`

- Was sind die Daten?
- Was sind die Methoden?

Eigene Klassen definieren

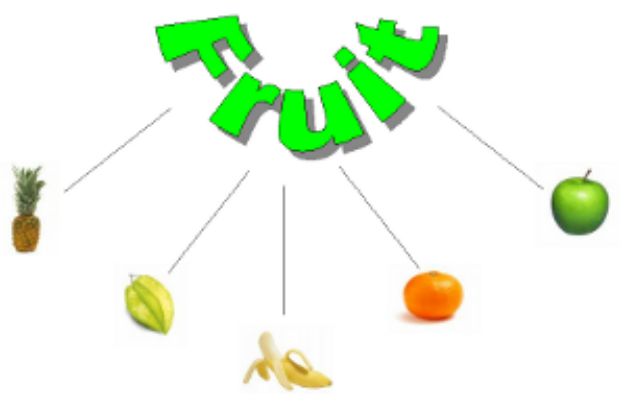
Mit Hilfe von Klassendefinitionen können spezialisierte Objekte entworfen werden, welche auf eine ganz bestimmte Anwendung zugeschnitten sind!

Klassenhierarchien: Oberklassen und Unterklassen

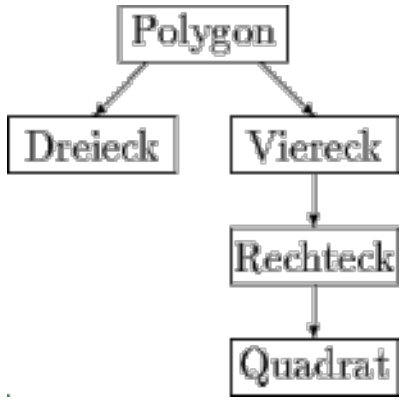
Gemeinsame Eigenschaften und Fähigkeiten von Objekten

Was verbindet oder unterscheidet die Objekte der verschiedenen Klassen?

Flache Hierarchie



Verschachtelte Hierarchie



Beziehung zwischen einer Klasse und ihrer Oberklasse

Alle Äpfel sind Früchte. Jedes Quadrat ist ein Polygon.

Klärung: Instanzen vs. Unterklassen

Instanzen (`isinstance(Object,Class)`)

Relation zwischen einem Objekt und seiner Klasse (Typ)!

```
>>> isinstance([1,2,3], list)
True
>>> isinstance([1,2,3], dict)
```

Unterklassen (`issubclass(Upperclass,Lowerclass)`) ▸6

Relation zwischen 2 Klassen/Typen!

```
>>> issubclass(nltk.FreqDist, dict)
True
>>> issubclass(nltk.FreqDist, collections.Counter)
True
>>> issubclass(collections.Counter, dict)
True
```

Die 7 Wahrheiten über Klassen in Python

- Klassen spezifizieren und implementieren die Eigenschaften (=Attribute) und Funktionen (=Methoden) von Objekten.
- Klassen abstrahieren gemeinsame Eigenschaften und Funktionalitäten.
- Klassen sind in Unter-/Oberklassen (*superclass/subclass*) organisiert (Vererbung).
- Vererbung heisst, dass Eigenschaften/Methoden einer Oberklasse defaultmässig auch in der Unterklasse zur Verfügung stehen.
- Die Methoden können in der Unterklasse aber auch umdefiniert werden (Flexibilität).
- Jede selbstdefinierte Klasse muss eine Oberklasse haben.
- Eine selbstdefinierte Klasse kann auch mehrere Oberklassen haben, d.h. Mehrfachvererbung ist möglich.

Die Eigentümlichkeiten der obersten Klasse `object`

Die Oberklasse aller Klassen in Python heisst `object`.

Die Klasse `object` ist trotz ihres Namens eine Klasse!

```
>>> help(object)
Help on class object in module __builtin__:
class object
 |  The most base type
```

Objekte (Instanzen) der Klasse `object`

```
>>> o = object()
>>> type(o)
<type 'object'>
```

Gibt es eine Oberklasse der Klasse `object`?

```
>>> issubclass(object, object)
True
```

Klassen definieren: Case-insensitive Strings

Motivation: Konsistenter Umgang mit Zeichenketten, wo Gross-/Kleinschreibung keine Rolle spielt.

Definition der Klasse, der Konstruktorfunktion und einer Methode ▸7

```
class Istr(object):                  # Unterklasse von object
    """Case-insensitive string class"""
    def __init__(self, s):           # Konstruktor-Funktion
        self._is = s.lower()         # self ist Instanzparameter
                                     # _is ist Instanzvariable
    def endswith(self, s):           # Methode endswith(s)
        return self._is.endswith(s.lower())
```

Instantiierung eines Objekts und Methodenaufruf

```
s = Istr('ABC') # Konstruktion eines Objekt der Klasse Istr
s.endswith('c') # Methoden-Aufruf
```

Zusammenhang von Definition und Verwendung

Klassendefinition

```
class Istr(object):
```

Konstruktordefinition

```
def __init__(self,s):
 self._is = s.lower()
```

Methodendefinition

```
def find(self,s):
 ls = s.lower()
 return self._is.find(ls)
```

Objektinstantiierung

```
s = Istr('ABC')
```

Methodenaufruf

```
s.find('bC')
```

Instanzvariablen

Jede Objekt-Instanz kann Attribute mit individuellen Werten in sich tragen.

Normalerweise werden diese Instanzvariablen beim Konstruieren des Objekts mit einem Wert belegt.

Instanzvariablen

```
>>> s = Istr('ABC')
>>> print s._is
abc
>>> s2 = Istr('XYZ')
>>> print s2._is
xyz
```

Grade der Öffentlichkeit: Namenskonvention

- Öffentliche Instanzvariablen ohne Unterstrich am Anfang:
  Überall frei benutzbar!
- Private Instanzvariablen beginnen mit Unterstrich:
  Sollen nur innerhalb der Klassendefinition verwendet werden!
  Grund: Datenabstraktion: Interne Implementation kann ändern, ohne das Klassenbenutzung sich ändert muss

Fazit Objektorientierte Programmierung (OOP)

Kernkonzepte nach `http://en.wikipedia.org/wiki/Object-oriented_programming`

- Datenkapselung I: Bündeln von Datenstrukturen und zugehöriger Funktionalität unter einer Adresse (=Objekt)
- Datenkapselung (Abstraktion) II: Klare Schnittstelle, welche Attribute und Methoden für öffentliche und welche für private (objektinterne) Zwecke nutzbar sind
- Klassenzugehörigkeit: Objekte sind Instanzen einer Klasse
- Vererbung: Unterklassen können Attribute/Methoden von ihren Oberklassen erben
- Dynamische Bindung: Welche Methode (d.h. Methode von welcher (Ober-)klasse) ein Objekt benutzt, wird erst beim Aufruf der Methode festgelegt anhand der *method resolution order*.
- Selbst-Parameter (*self*): Platzhalter für das Instanzobjekt in der Definition einer Klasse

Vertiefung

- Pflichtlektüre: Kapitel 2.1. bis und mit 2.5 aus [Bird et al. 2009]
- Gutes deutschsprachiges Tutorat
  `http://www.python-kurs.eu/klassen.php`

Verständnisfragen

- Wieso sind selbstdefinierte Klassen nützlich?
- Was beinhalten die 7 Wahrheiten zu Klassen in Python?
- Inwiefern unterscheidet sich die Instanzrelation von der Unterklassenrelation?
- Wie definiert man Klassen in Python?
- Wozu dient der Parameter `self`?
- Was versteht man unter einer Instanzvariablen?
- Wozu dient die Methode `__init__()`?

Liste der verlinkten Programme und Ressourcen I

Literaturangaben I

- Bird, Steven, E. Klein und E. Loper (2009).
  Natural Language Processing with Python. O'Reilly.