

Einführung in die Computerlinguistik – Einführung in Python (3) Tokenizer und Textstatistik

Dozentin: Wiebke Petersen

12. Foliensatz

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung
- Runde Klammern im regulären Ausdruck zum Speichern des getroffenen Teilstrings: `re.search(r"([a-z]+)", string)`

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung
- Runde Klammern im regulären Ausdruck zum Speichern des getroffenen Teilstrings: `re.search(r"([a-z]+)", string)`
- Rückgabewert von `re.search()` bzw. `re.finditer()` erlaubt Zugriff auf die gespeicherten Klammerinhalte: `match.group(1)`

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung
- Runde Klammern im regulären Ausdruck zum Speichern des getroffenen Teilstrings: `re.search(r"([a-z]+)", string)`
- Rückgabewert von `re.search()` bzw. `re.finditer()` erlaubt Zugriff auf die gespeicherten Klammerinhalte: `match.group(1)`
- Suchen und Ersetzen mittels `re.sub(regex, replace, input)`

Kurzwiederholung der letzten Sitzung

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung
- Runde Klammern im regulären Ausdruck zum Speichern des getroffenen Teilstrings: `re.search(r"([a-z]+)", string)`
- Rückgabewert von `re.search()` bzw. `re.finditer()` erlaubt Zugriff auf die gespeicherten Klammerinhalte: `match.group(1)`
- Suchen und Ersetzen mittels `re.sub(regex, replace, input)`
- Zugriff auf Klammerinhalte beim Ersetzen mittels `\g<Gruppennummer>`

Dialogsystem: ELIZA (Weizenbaum 1966)

User: Men are all alike

ELIZA: IN WHAT WAY

User: They're always bugging us about something or other

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?

User: Well, my boyfriend made me come here

ELIZA: YOUR BOYFRIEND MADE YOU COME HERE

User: He says I'm depressed much of the time

ELIZA: I AM SORRY TO HEAR THAT YOU ARE DEPRESSED

Implementierungen:

- <http://www.hsweis.schule.neu-ulm.de/eliza/eliza3.html>
- <http://bs.cyty.com/menschen/e-etzold/archiv/science/rat.htm>
- <http://folk.uib.no/hhiso/eliza/index.htm>

Dialogsystem: ELIZA (Weizenbaum 1966)

① Wechsel von 1. zur 2. Person:

```
1 input = "I am sad about my work situation."  
2 input = re.sub(r"\bI('m| am)\b ", "you are ", input)  
3 input = re.sub(r"\b[Mm]y\b ", "your ", input)  
4 input = re.sub(r"\bmine\b ", "yours ", input)
```

② Generierung von Antworten:

```
1 output = re.sub(r".*\byou are (depressed|sad) .*", \  
2     "i am sorry to hear you are \g<1>", input)  
3 output = re.sub(r".*\byou are (depressed|sad) .*", \  
4     "why do you think you are \g<1>", input)  
5 output = re.sub(r".*\ball .*", "in what way", input)  
6 output = re.sub(r".*\balways .*", \  
7     "can you think of a specific example", input)
```

Übungseinheit

- Kopieren Sie das Skript `eliza.py` in ihr persönliches Python-Verzeichnis. Überlegen Sie sich, wie man das Programm verbessern könnte (z. B. durch die Hinzunahme weiterer Antwortmöglichkeiten), und nehmen Sie mindestens eine Verbesserung vor
- Wo liegen die Grenzen dieses Ansatzes?

Python: for-Schleife

- Die for-Schleife wird benutzt, wenn jedes Element einer Menge durchlaufen werden soll
- In Python ist dieses Konstrukt sehr vielseitig:
 - `for line in file:` durchläuft die Zeilen einer Datei (s.o.)

Python: for-Schleife

- Die for-Schleife wird benutzt, wenn jedes Element einer Menge durchlaufen werden soll
- In Python ist dieses Konstrukt sehr vielseitig:
 - `for line in file:` durchläuft die Zeilen einer Datei (s.o.)
 - `for i in (1,2,3):` durchläuft die Liste mit den Elementen 1, 2 und 3

Python: for-Schleife

- Die for-Schleife wird benutzt, wenn jedes Element einer Menge durchlaufen werden soll
- In Python ist dieses Konstrukt sehr vielseitig:
 - `for line in file:` durchläuft die Zeilen einer Datei (s.o.)
 - `for i in (1,2,3):` durchläuft die Liste mit den Elementen 1, 2 und 3
 - `for i in range(0,5):` durchläuft die Liste mit den Elementen von 0 bis 5 (exklusive)

Python: for-Schleife

- Die for-Schleife wird benutzt, wenn jedes Element einer Menge durchlaufen werden soll
- In Python ist dieses Konstrukt sehr vielseitig:
 - `for line in file:` durchläuft die Zeilen einer Datei (s.o.)
 - `for i in (1,2,3):` durchläuft die Liste mit den Elementen 1, 2 und 3
 - `for i in range(0,5):` durchläuft die Liste mit den Elementen von 0 bis 5 (exklusive)
 - `for char in "string":` durchläuft die einzelnen Zeichen einer Zeichenkette

Python: for-Schleife

- Die for-Schleife wird benutzt, wenn jedes Element einer Menge durchlaufen werden soll
- In Python ist dieses Konstrukt sehr vielseitig:
 - `for line in file:` durchläuft die Zeilen einer Datei (s.o.)
 - `for i in (1,2,3):` durchläuft die Liste mit den Elementen 1, 2 und 3
 - `for i in range(0,5):` durchläuft die Liste mit den Elementen von 0 bis 5 (exklusive)
 - `for char in "string":` durchläuft die einzelnen Zeichen einer Zeichenkette
- Um also einen Codeblock genau n mal auszuführen, schreibt man `for i in range(0,n):`

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st
 - `"string"[0:6]` ergibt string

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st
 - `"string"[0:6]` ergibt string
 - `"string"[2:4]` ergibt ri

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st
 - `"string"[0:6]` ergibt string
 - `"string"[2:4]` ergibt ri
 - `"string"[:2]` ergibt st (Default vor : ist 0)

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st
 - `"string"[0:6]` ergibt string
 - `"string"[2:4]` ergibt ri
 - `"string"[:2]` ergibt st (Default vor : ist 0)
 - `"string"[2:]` ergibt ring (Default nach : ist die String-Länge)

Python: String-Slices

- Ein für die Verarbeitung von Text sehr nützliches Konstrukt in Python sind die sog. String-Slices
- Mithilfe von String-Slices lassen sich einzelne Teile aus einem String extrahieren
 - `"string"[0]` ergibt s
 - `"string"[5]` ergibt g
 - `"string"[0:2]` ergibt st
 - `"string"[0:6]` ergibt string
 - `"string"[2:4]` ergibt ri
 - `"string"[:2]` ergibt st (Default vor : ist 0)
 - `"string"[2:]` ergibt ring (Default nach : ist die String-Länge)
- **Achtung:** eine Zuweisung an einen String-Slice ist nicht möglich, d.h. `name[0] = "A"` funktioniert nicht

Übungseinheit

- Schreiben Sie ein Programm, das alle Strophen des Liedes “Auf der Mauer, auf der Lauer” aus der ersten Strophe generiert (siehe “aufdermauer.txt”)
(http://de.wikipedia.org/wiki/Auf_der_Mauer,_auf_der_Lauer)
- Tipp: nutzen Sie dabei die folgenden Python-Konstrukte (andere Lösungen sind selbstverständlich auch möglich):
 - Reguläre Ausdrücke (Suchen & Ersetzen)
 - Dateien lesen und Zeile für Zeile auswerten
 - for-Schleife für jede Strophe
 - String-Slices
- Überlegen Sie sich vorher, wie die Nummer der Strophe mit dem Inhalt zusammenhängt (Wanze, tanzen) und wie sich dieses Verhältnis mit String-Slices abbilden lässt

Tokenisierung

- Segmentierung von Fließtext in Wörter.
- Segmentierung von Fließtext in Sätze (auch *sentence splitting*).

Tokenisierung

- Segmentierung von Fließtext in Wörter.
- Segmentierung von Fließtext in Sätze (auch *sentence splitting*).
- Reichen Punkte und Leerzeichen als Marker nicht aus?

Die Thomas Mann-Gesellschaft Düsseldorf e. V. lädt am Freitag, den 4. Dezember 2009, um 18 Uhr, zu einem Vortrag von Prof. Dr. Johannes Roskothen ein. Das Thema lautet: "Firma? Ruiniert. Familie? Ausgestorben. Häuser? Figurationen des Abstiegs in Thomas Manns erstem Roman "Buddenbrooks. Verfall einer Familie".

Tokenisierung

- Segmentierung von Fließtext in Wörter.
- Segmentierung von Fließtext in Sätze (auch *sentence splitting*).
- Reichen Punkte und Leerzeichen als Marker nicht aus?

Die Thomas Mann-Gesellschaft Düsseldorf e. V. lädt am Freitag, den 4. Dezember 2009, um 18 Uhr, zu einem Vortrag von Prof. Dr. Johannes Roskothen ein. Das Thema lautet: "Firma? Ruiniert. Familie? Ausgestorben. Häuser? Figurationen des Abstiegs in Thomas Manns erstem Roman "Buddenbrooks. Verfall einer Familie".

- Wörter: e. V. 4. Uhr, Häuser? "Buddenbrocks
- Sätze: Die Thomas Mann-Gesellschaft Düsseldorf e.

Tokenisierung

- Segmentierung von Fließtext in Wörter.
- Segmentierung von Fließtext in Sätze (auch *sentence splitting*).
- Reichen Punkte und Leerzeichen als Marker nicht aus?

Die Thomas Mann-Gesellschaft Düsseldorf e. V. lädt am Freitag, den 4. Dezember 2009, um 18 Uhr, zu einem Vortrag von Prof. Dr. Johannes Roskothen ein. Das Thema lautet: "Firma? Ruiniert. Familie? Ausgestorben. Häuser? Figurationen des Abstiegs in Thomas Manns erstem Roman "Buddenbrooks. Verfall einer Familie".

- Wörter: e. V. 4. Uhr, Häuser? "Buddenbrocks
- Sätze: Die Thomas Mann-Gesellschaft Düsseldorf e.
- In manchen Sprachen wie Chinesisch und Japanisch stehen keine Leerzeichen zwischen den Wörtern.

Satzsegmentierung

- ? und ! sind relativ sichere Satzsegmentierer (problematische Sätze: “Wann soll ich kommen?”, fragte er.)

Satzsegmentierung

- ? und ! sind relativ sichere Satzsegmentierer (problematische Sätze: "Wann soll ich kommen?", fragte er.)
- Punkte sind problematisch:
 - www.duesseldorf.de
 - Dr.
 - d.h.
 - 5.5.1955

Satzsegmentierung

- ? und ! sind relativ sichere Satzsegmentierer (problematische Sätze: "Wann soll ich kommen?", fragte er.)
- Punkte sind problematisch:
 - www.duesseldorf.de
 - Dr.
 - d.h.
 - 5.5.1955
- Lösungsansatz:
 - Erstellen eines binären Klassifizierers (regelbasiert und/oder statistisch)
 - Für jeden Punkt wird entschieden SATZENDE oder ANDERERPUNKT

Tokenisierung in Wörter

- Problem:
 - Klitika und Elisionen:
Peter's
I'm
 - Mehrwortausdrücke:
New York
Rock 'n' Roll

Tokenisierung von Chinesisch

- Im Chinesischen werden Wörter nicht durch Leerzeichen getrennt.
- Das durchschnittliche Wort ist 2,4 Zeichen lang.
- Standardalgorithmus zur Segmentierung: **Maximum Match / Greedy**

Gegeben eine Wortliste und ein String

1. Setze einen Pointer auf den Anfang des Strings
2. Finde das längste Wort in der Wortliste, das den String vom Pointer an matcht.
3. Setze den Pointer hinter das Wort in dem String. Gehe zurück zu 2.

Wenn das Verfahren scheitert, verwerfe das zuletzt gewählte Wort und wähle das nächstlängste (Backtracking).

Tokenisierung von Chinesisch

- Im Chinesischen werden Wörter nicht durch Leerzeichen getrennt.
- Das durchschnittliche Wort ist 2,4 Zeichen lang.
- Standardalgorithmus zur Segmentierung: **Maximum Match / Greedy**
 - Gegeben eine Wortliste und ein String
 - 1. Setze einen Pointer auf den Anfang des Strings
 - 2. Finde das längste Wort in der Wortliste, das den String vom Pointer an matcht.
 - 3. Setze den Pointer hinter das Wort in dem String. Gehe zurück zu 2.Wenn das Verfahren scheitert, verwerfe das zuletzt gewählte Wort und wähle das nächstlängste (Backtracking).
- Für das Englische funktioniert der Algorithmus nicht (Beispiel von Palmer)
 - Input: thetabledownthere
 - Output: Theta bled own there

Wörter zählen: Type vs. Token

- Aus wievielen Wörtern besteht der Text?
Zählen der **Token**: Anzahl von Wortformen
- Wieviele verschiedene Wörter kommen im Text vor? Zwei Wortformen eines Lemmas werden getrennt gezählt (z.B. sage, sagte).
Zählen der **Types**: Anzahl unterschiedlicher Wortformen
- Wie oft kommt ein bestimmtes Wort im Text vor?
Tokens pro Type: Vorkommenshäufigkeit von Wortformen

Wörter zählen: Type vs. Token

- Aus wievielen Wörtern besteht der Text?
Zählen der **Token**: Anzahl von Wortformen
- Wieviele verschiedene Wörter kommen im Text vor? Zwei Wortformen eines Lemmas werden getrennt gezählt (z.B. sage, sagte).
Zählen der **Types**: Anzahl unterschiedlicher Wortformen
- Wie oft kommt ein bestimmtes Wort im Text vor?
Tokens pro Type: Vorkommenshäufigkeit von Wortformen

Der Zaun, der den Garten umschließt,
den hat Otto für seinen Garten gebaut.

13 Tokens, 10 Types.

Gesetz von Zipf

- Das Gesetz von Zipf besagt, dass es für jeden Text eine Konstante k gibt, so dass $k \approx f(w) \cdot r(w)$ für jedes Wort w gilt. $f(w)$ ist die Frequenz von w und $r(w)$ ist der Rang von w in der Frequenzliste.

Gesetz von Zipf

- Das Gesetz von Zipf besagt, dass es für jeden Text eine Konstante k gibt, so dass $k \approx f(w) \cdot r(w)$ für jedes Wort w gilt. $f(w)$ ist die Frequenz von w und $r(w)$ ist der Rang von w in der Frequenzliste.
- Das Gesetz trifft zumindest außerhalb der Extrembereiche (sehr hohe bzw. sehr niedrige Frequenz) auf die meisten Texte recht gut zu.
- Das heißt, die Wahrscheinlichkeit des Auftretens eines Wortes ist umgekehrt proportional zu seinem Rang in der Frequenzliste.
- Die wichtigsten Grundaussagen sind:
 - Wenige Worte kommen häufig vor.
 - Viele Worte kommen selten vor.

Gesetz von Zipf

- Das Gesetz von Zipf besagt, dass es für jeden Text eine Konstante k gibt, so dass $k \approx f(w) \cdot r(w)$ für jedes Wort w gilt. $f(w)$ ist die Frequenz von w und $r(w)$ ist der Rang von w in der Frequenzliste.
- Das Gesetz trifft zumindest außerhalb der Extrembereiche (sehr hohe bzw. sehr niedrige Frequenz) auf die meisten Texte recht gut zu.
- Das heißt, die Wahrscheinlichkeit des Auftretens eines Wortes ist umgekehrt proportional zu seinem Rang in der Frequenzliste.
- Die wichtigsten Grundaussagen sind:
 - Wenige Worte kommen häufig vor.
 - Viele Worte kommen selten vor.
- Sprachdaten sind also sehr ungleich verteilt und es gibt viele seltene Wörter (problematisch für statistische Sprachmodelle)

Übungseinheit

- Schreiben Sie einen Tokenizer, also ein Programm, das Ihnen einen Text in seine Tokens (Wörter und Satzzeichen zerlegt). Geben Sie die gefundenen Tokens in einer Datei aus, wobei in jeder Zeile nur genau ein Token steht
- Wenn Sie noch Zeit haben, schreiben Sie ein Programm, das eine Datei öffnet und den Inhalt in eine andere Datei schreibt, wobei alle Vokale durch "a" ersetzt werden