Einführung in die Computerlinguistik – Tokenizer und Textstatistik mit Perl

Dozentin: Wiebke Petersen

3.12.2009

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.)
- Punkte sind problematisch:

  www.duesseldorf.de
  Dr.
  d.h.
  5.5.1955
- Lösungsansatz:
  - Erstellen eines binären Klassifizierers (regelbasiert oder auf 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 Wortlilste, 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

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

13 Tokens, 10 Types.

Beispiel: Kant, Kritik der reinen Vernunft

Die folgenden Zahlen wurden mit dem Übungsprogramm `textstat.pl` erhoben (siehe Hausaufgaben) und erheben keinen Anspruch auf Gültigkeit.

- Zahl der Token: 184942
- Zahl der Typen: 9907
- Token pro Typ (Durchschnitt): 18.6678106389422
- Die häufigsten 90 Wörter nehmen über die Hälfte des Gesamtkorpus ein (50.51%)
- Fast 70 % aller Typen kommen höchstens 3 mal im Text vor (68,93%). Die Wörter, die höchsten 3 mal vorkommen, nehmen immerhin 5,24% des Gesamttextes ein.
- Die 20 häufigsten Wörter alles Funktionswörter: der, die, und, in, ist, nicht, zu, als, von, sie, das, so, sich, den, dass, des, aber, es, auf, ein
- das erste Nomen, nämlich 'Vernunft', kommt erst auf Rang 28

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)

Perl: In- und Output in Dateien

Einlesen aus einer Datei

open(INPUT, "<file.ext"); öffnet die Datei `file.ext` zum Lesen.
Mit $line=<INPUT>; schreiben Sie die erste Zeile der Datei in die Variable `$line`.
Mit **close**(INPUT); schließen Sie die Datei.

Schreiben in eine Datei

open(OUTPUT, ">file.ext"); öffnet die Datei `file.ext` zum Schreiben.
Mit **print** OUTPUT "text"; schreiben Sie `text` in die Datei.
Mit **close**(OUTPUT); schließen Sie die Datei.

Perl: Übungseinheit (5)

- Schreiben Sie ein Programm, das eine Datei öffnet und zählt, wie oft in der Datei der unbestimmte Artikel vorkommt.
- Schreiben Sie einen Tokenizer, also ein Programm, dass Ihnen einen Text in seine Tokens (Wörter und Satzzeichen zerlegt).
- 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.
- Wenn Sie noch Zeit haben, schreiben Sie ein Programm, das aus der ersten Strophe des Liedes "Auf der Mauer auf der Lauer saß 'ne kleine Wanze" alle Strophen erzeugt und in eine neue Datei schreibt.

Perl: Hashes – assoziative Arrays

- Hashes sind Datenstrukturen, die irgendwo zwischen Mengen, Listen und Tabellen bzw. Datenbanken zu verorten sind.
- Die Elemente eines Hashes haben keine Ordnung, aber jedes Element hat einen Schlüssel (key) über den es identifiziert werden kann.
- Stellen Sie sich eine Strichliste vor, in der Sie zählen, wer wieviele Stimmen bekommen hat, dann ist der Kandidatenname der Schlüssel und die Stimmenzahl der Wert.
- Schlüssel dürfen nicht mehrfach vergeben werden.

Perl: Hashes – assoziative Arrays

```
1  #!perl -w
2  # mehrere Elemente des Hashes werden definiert
3  %familienHash = (
4  "Fred" => "Feuerstein",
5  "Barney" => "Geroellheimer",
6  "Betty" => "Geroellheimer",
7  );
8  # ein neues Hashelement wird definiert
9  $familienHash{"Otto"} = "Mustermann";
10 print "Wessen Nachname suchen Sie?";
11 $key=<>;
12 chomp($key);
13 print $familienHash{$key};
```

Perl: Hashes

```
1  #! perl -w
2  my %rand_hash = ();
3  for($i = 1; $i < 10000; $i++) {
4  my $random = int(rand(6));
5  $rand_hash{$random}++;
6  }
7  for($k = 0; $k <6; $k++){
8  print "Die Zahl $k wurde $rand_hash{$k} mal erzeugt.\n";
9  }
```

- Erzeugt zufällig 10000 Zahlen von 0 bis 5 und gibt an, welche Zahl wie oft erzeugt wurde.
- `$random = int(rand(6))` Belegt die Variable `$random` mit einer zufällig erzeugten Zahl zwischen 0 und 5.
- `my %rand_hash = ()` erzeugt einen Hash mit dem Namen `rand_hash`, der zunächst leer ist. Dieser Hash wird zur Zählung der erzeugten Zahlen verwendet (Key: Zahl zwischen 0 und 5, Wert: Frequenz der Erzeugung).
- `$rand_hash{$random}++` erhöht den Wert zu dem Schlüssel `$random` in dem Hash `%rand_hash` um 1.
- `for($i = 1; $i < 10000; $i++) { }` definiert eine Schleife, die 10000 mal durchlaufen wird. Dabei wird die Variable $1 zunächst mit dem Wert 0 belegt und dann in jedem Schritt um 1 erhöht (bis 9999).

Perl: Sortierter Zugriff auf Hashes (Schlüssel)

Sortiert den Hash nach Schlüsseln (ASCII-betisch):

```
1 foreach $key (sort keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Schlüsseln (absteigend ASCII-betisch):

```
1 foreach $key (reverse sort keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Schlüsseln (numerisch):

```
1 foreach $key (sort {$a<=> $b} keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Schlüsseln (absteigend numerisch):

```
1 foreach $key (reverse sort {$a<=> $b} keys %hash){
2     print $key;
3 }
```

ASCII: `http://unicode.e-workers.de/ascii.php`

Sortierter Zugriff auf Hashes (Werte)

Sortiert den Hash nach Werten (ASCII-betisch):

```
1 foreach $key (sort {$hash{$a} cmp $hash{$b}} keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Werten (absteigend ASCII-betisch):

```
1 foreach $key (reverse sort {$hash{$a} cmp $hash{$b}} keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Werten (numerisch):

```
1 foreach $key (sort {$hash{$a}<=> $hash{$b}} keys %hash){
2     print $key;
3 }
```

Sortiert den Hash nach Werten (absteigend numerisch):

```
1 foreach $key (reverse sort {$hash{$a}<=> $hash{$b}} keys %hash){
2     print $key;
3 }
```