
Einführung in die Informatik

Kodierung und Daten

Meik Teßmer

Inhalt der Veranstaltung

Lernziele:

- ▶ Fallstricke beim Umgang mit Daten
- ▶ Vorgehen bei der Datenanalyse

Situation Datenanalyse

- ▶ Aufgabe: Datensatz untersuchen
durchschnittliche Anzahl Wörter pro Zeile
- ▶ Datensatz: kommt per DVD
- ▶ keine zusätzlichen Angaben zu den Daten, nur: Datensätze sind zeilenweise angeordnet

Vorgehen?

Kodierung

- ▶ Rechner: kann nur 0 und 1 (=1 Bit)
- ▶ 8 Bit = 1 Byte
- ▶ binär dargestellte Zahlen: 1=00000001, 2=00000010 usw.
- ▶ Zeichenkodierung (Character Encoding): Zeichen werden im Rechner als Zahlen abgelegt
- ▶ ohne Angabe der Kodierung: Fehler können auftreten
- ▶ Beispiel: Zeichen an der Stelle 220
 - ▶ Kodierung ISO-8859-1: ü
 - ▶ Kodierung EBCDIC: }

Erste offizielle Kodierung: ASCII

- ▶ 1963: erste Version des American Standard Code for Information Interchange (ASCII)
- ▶ nutzt 7 Bit (=128 mögliche Zeichen)
- ▶ Tabelle besteht aus:
 - ▶ 33 nicht druckbaren Zeichen (Steuerzeichen)
 - ▶ 95 druckbare Zeichen (Zahlen, Alphabet, Satzzeichen)
- ▶ Beispiel: <http://de.wikipedia.org/wiki/ASCII>

Erweiterte Kodierungen

- ▶ Probleme: keine diakritischen Zeichen (ö, à, Æ)
 - ▶ keine anderen Zeichen (z.B. asiatische Zeichen)
 - ▶ Lösung: Hersteller entwickelten 8-Bit-Kodierungen (256 mögliche Zeichen)
 - ▶ Codepage 437 (DOS-Box im englischen Windows)
 - ▶ Codepage 850 (seit MS-DOS 3.3)
 - ▶ eine erste „offizielle“ Lösung: ISO-8859
 - ▶ ersten 128 Zeichen sind identisch zu ASCII
 - ▶ Rest: regionale Sonderzeichen (Teilnormen 1-16)
- siehe http://de.wikipedia.org/wiki/ISO_8859
- ▶ Westeuropäisch: ISO-8859-1 (auch Latin-1 genannt)

Unicode

- ▶ Lösung: Erweiterung auf bis zu 32 Bit
- ▶ jedes Zeichen steht an einem *Code Point* (oft hexadezimal)
- ▶ Beispiel: U+00DF ist der Code Point von ß
- ▶ dient nur zur Erfassung, nicht zur Darstellung
- ▶ Unicode Transformation Format (UTF)
 - ▶ UTF-8: UNIX-Systeme, E-Mail, WWW
 - ▶ UTF-16: Windows, OS X, Java, .NET

<http://de.wikipedia.org/wiki/Unicode>

Arbeiten mit Unicode

- ▶ Eingabe von Unicode-Zeichen:
 - ▶ Windows (Office): U+ Code Point im Hexadezimal-Format, dann Alt-c
 - ▶ Linux (GNOME etc.): Strg-Umschalttaste-u Code Point in Hexadezimal-Format
 - ▶ Vim-Editor: Strg-v u Code Point im Hexadezimal-Format
- ▶ Darstellung:
 - ▶ Webbrowser: unterstützen eigentlich alle UTF
 - ▶ Editoren: Kodierung lässt sich einstellen
 - ▶ Terminal (DOS-Box): hängt vom Betriebssystem ab

Zusammenfassung Kodierung

- ▶ neben den Daten muss die Kodierung bekannt sein
- ▶ je nach Kodierungs-„Brille“ können Daten unterschiedlich aussehen
- ▶ Anwendungen müssen die jeweilige Kodierung unterstützen
- ▶ weltweiter Datenaustausch ist möglich

Wie analysieren wir jetzt unsere Daten?

Zurück zur Datenanalyse

- ▶ Situation in Python (ab v3.x): Zeichenketten sind immer Unicode
- ▶ Tastatureingaben: abhängig von der Einstellung des Betriebssystems
- ▶ Dateien: Kodierung muss bekannt sein
sonst werden keine Zeichen, sondern nur Bytes
gelesen/geschrieben
→ Transformation muss manuell durchgeführt werden

Datenverarbeitung in Python

- ▶ Dateien verarbeiten: 2 mögliche Wege
 - ▶ Angabe der Kodierung direkt bei `open`
 - gibt Zeichenketten in Unicode zurück und erlaubt das Arbeiten wie bisher
 - ▶ als Binärdatei öffnen
 - *Bytes* werden gelesen/geschrieben und man muss selbst de-/kodieren
- ▶ Standard beim Öffnen: Text

Datenverarbeitung in Python

- ▶ Kodierung beim Öffnen mitgeben:

```
handle = open("dateiname", "r", encoding="utf-8")
```

→ gelesen werden immer Zeichenketten (`str()`)

- ▶ Vorteil: Code kann so bleiben wie bisher
- ▶ Nachteile:
 - ▶ Kodierung ist bei dieser Lösung fest eingebaut
 - ▶ Daten liegen in Textform vor; Binärdaten (z.B. MP3-Dateien)?
- ▶ <https://github.com/erikrose/chardet>: chardet-Paket schätzt die Kodierung einer Datei ab

Datenverarbeitung in Python: Weg 2

- ▶ Text- vs. Binärdateien: lesbar?

- ▶ Unterschied beim Öffnen

 - ▶ Text: `open("dateiname", "r")`

 - ▶ Binär: `open("dateiname", "rb")`

- ▶ binär gelesene Inhalte in Zeichenkette übersetzen:

```
inhalt = handle.read() # eingelesene Daten  
als_zeichenkette = inhalt.decode("utf-8")
```

- ▶ umgekehrter Weg:

```
inhalt = als_zeichenkette.encode("utf-8")
```

Datenverarbeitung: Code-Beispiel

Beispiel: als Text öffnen

```
handle = open("index.rst", "r", encoding="utf-8")
content = handle.read() # alles auf einmal einlesen
handle.close()
print(content)
```

Binär öffnen:

```
handle = open("index.rst", "rb", encoding="utf-8")
...
print(content.decode("utf-8"))
```

Zusammenfassung Kodierungen

- ▶ Wie liegen die Daten vor?
 - ▶ Text oder binär
 - ▶ Kodierung? UTF-8, ISO-8859-1 ...
- ▶ Konvertieren mit `decode()` und `encode()`

Allgemeine Regel:

Software sollte intern immer mit Unicode arbeiten, eingehende Daten so früh wie möglich dekodieren, und ausgehende Daten so spät wie möglich kodieren.

Datenanalyse

Vorgehensweise:

- ▶ Daten bereinigen/normalisieren
- ▶ Datenmenge reduzieren, wenn möglich
- ▶ Daten analysieren
- ▶ Interpretation der Analyseergebnisse

für jeden Schritt ggf. ein eigenes Werkzeug/Programm

Daten bereinigen

Testdaten:

Name;BLZ;Konto;Betrag;Verwendungszweck

Max Müller;48050161; 12312351; 23,20; Buch

Petra Schmidt;923 456 71; 52347890; 10,50; Beitrag

Hans Meier; 400 20012;99832392; 200,10; Schlafsack

Anne Schulz; 83724023;15673945; 342.80; Schuhe

Daten bereinigen

Bereinigen:

- ▶ Felder sollten keine Leerzeichen enthalten
- ▶ im Feld *Betrag* sollten entweder nur Punkte oder nur Kommata verwendet werden

→ Datensätze müssen bereinigt/normalisiert werden

Daten reduzieren/analysieren/interpretieren

Frage: Durchschnittlich ausgegebener Betrag?

→ Datensätze analysieren

Zusammenfassung

- ▶ Analyse kann nur auf „gleichförmigen“ Daten erfolgen
→ bereinigen/normalisieren
- ▶ je geringer die Datenmenge, desto besser
aber: keine relevanten Daten „wegreduzieren“
- ▶ auf Ressourcen (wesentlich: RAM) achten
→ (Zwischen-)Ergebnisse der Analyse in Dateien ablegen

Beispiel aus der Praxis

Analyse von Log-Dateien, bspw. Angriffsversuche

- ▶ Code: <https://github.com/darkseed/ssh-attack-visualisation>
- ▶ Video: <https://kura.gg/2012/03/02/visualised-24-hours-of-ssh-attacks-against-a-single-server/>

Ende