4.3 Textanalyse mit dem Computer

Leitidee

Zwei Sätze von Hand auszuzählen, geht gut. Aber ein echtes Sprachmodell lernt aus Millionen von Sätzen. Dort würdest du mit der Strichliste nie fertig. In diesem Abschnitt überlässt du das Zählen dem Computer: Du schreibst ein Python-Programm, das zu jedem Wort eines Textes die möglichen Folgewörter mit ihrer Häufigkeit ermittelt. Außerdem lernst du **SoekiaGPT** kennen, ein Lernprogramm im Browser, das dieselbe Analyse an größeren Textsammlungen vorführt.

Lernziele

Nach diesem Abschnitt kannst du

- begründen, warum größere Textkorpora rechnergestützt analysiert werden,
- beschreiben, wie n-Gramm-Häufigkeiten in einer Tabelle verwaltet werden,
- ein Python-Programm nutzen und erweitern, das zu einem Wort alle Folgewörter mit Häufigkeit und Wahrscheinlichkeit ermittelt,
- einen eigenen Textkorpus mit diesem Programm analysieren,
- mit SoekiaGPT n-Gramme in einer Textsammlung untersuchen.

4.3.1 Warum der Computer zählen muss

Im letzten Abschnitt hattest du Texte mit 9 bis 12 Wörtern. Schon da musstest du sorgfältig arbeiten, um kein Bigramm zu übersehen. Ein einziges Buch hat leicht 50 000 Wörter, also fast genauso viele Bigramme. Sprachmodelle werden mit Textsammlungen trainiert, die noch sehr viel größer sind.

Von Hand ist das nicht zu schaffen. Ein Programm erledigt dieselbe Arbeit dagegen in Sekundenbruchteilen und verzählt sich nie. Der Ablauf bleibt aber genau derselbe, den du schon kennst: Text in Wörter zerlegen, Bigramme bilden, zählen, Wahrscheinlichkeiten berechnen.

Merke

Bei größeren Textkorpora ist eine **Auszählung von Hand** nicht mehr praktikabel. Man nutzt Software, die die Häufigkeiten **automatisiert** ermittelt.

4.3.2 Ein größerer Korpus

Wir untersuchen einen Text aus fünf Zeilen über einen Hund und eine Katze. Damit er gut lesbar bleibt, soll er auch im Programm auf mehrere Zeilen verteilt sein.

Werkzeug

Einen Text über **mehrere Zeilen** schreibst du in Python zwischen **drei** Anführungszeichen `"""` am Anfang und am Ende. Die Zeilenumbrüche gehören dann zum Text dazu. Für `split()` ist ein Zeilenumbruch dasselbe wie ein Leerzeichen:

```python
text = """eins zwei
drei"""
print(text)
print(text.split())
```

AUSGABE

```
eins zwei
drei
['eins', 'zwei', 'drei']
```

4.3.3 Schritt 1: Alle Folgewörter eines Wortes sammeln

Von Hand hast du für ein Vorgängerwort alle Bigramme gesucht, die mit ihm beginnen, und dir jeweils das Folgewort notiert. Genau das macht die Funktion `folgewoerter`. Sie bekommt die Wortliste und ein Wort und liefert eine Liste mit **allen** Wörtern, die im Text direkt nach diesem Wort stehen – auch die doppelten.

```python
def folgewoerter(woerter, wort):
    folge = []
    for i in range(len(woerter) - 1):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

korpus = """
der hund läuft im park der hund spielt mit dem ball
die katze sitzt auf dem baum die katze beobachtet den hund
der hund bellt laut der hund läuft zur katze
die katze springt vom baum die katze rennt weg
der hund läuft ihr hinterher der hund ist müde
"""
woerter = korpus.split()
print(len(woerter), "Wörter")
print(folgewoerter(woerter, "der"))
print(folgewoerter(woerter, "dem"))
```

AUSGABE

```
49 Wörter
['hund', 'hund', 'hund', 'hund', 'hund', 'hund']
['ball', 'baum']
```

- **Zeilen 1–6:** Die Funktion geht mit `i` alle Positionen durch, auf die noch ein Wort folgt. Steht an Position `i` das gesuchte Wort, wird das Wort an Position `i + 1` in die Liste `folge` aufgenommen. Das ist das „Fenster“ aus zwei Wörtern, nur dass jetzt bloß die Fenster beachtet werden, die mit `wort` beginnen.
- **Zeilen 8–14:** Der Korpus als mehrzeiliger Text.
- **Zeile 15:** `split()` zerlegt den Text in eine Liste mit 49 Wörtern. Also gibt es 48 Bigramme.
- **Zeilen 17–18:** Nach „der“ steht im ganzen Text sechsmal „hund“ und sonst nichts. Nach „dem“ steht einmal „ball“ und einmal „baum“.

Die Länge dieser Liste ist genau die Anzahl der Bigramme, die mit dem Wort beginnen. Das ist die Zahl, die du für die Wahrscheinlichkeit im **Nenner** brauchst. Auch der Sonderfall „letztes Wort“ ist automatisch richtig, denn die Schleife schaut nur Positionen an, auf die noch ein Wort folgt.

4.3.4 Schritt 2: Häufigkeiten zählen

Die Liste `['ball', 'baum']` ist noch keine Tabelle. Für die Tabelle brauchen wir jedes Folgewort **einmal** und daneben seine Anzahl – wie bei der Strichliste. Dafür gibt es zwei kleine Hilfsfunktionen, die du in ähnlicher Form schon geschrieben hast:

```
1  def ohne_doppelte(liste):
2      ergebnis = []
3      for element in liste:
4          if element not in ergebnis:
5              ergebnis.append(element)
6      return ergebnis
7
8  def zaehle(liste, gesucht):
9      anzahl = 0
10     for element in liste:
11         if element == gesucht:
12             anzahl += 1
13     return anzahl
14
15 def zeige_tabelle(woerter, wort):
16     folge = folgewoerter(woerter, wort)
17     print("Mögliche Folgewörter nach", wort)
18     for kandidat in ohne_doppelte(folge):
19         print(f"  {kandidat}: {zaehle(folge, kandidat)}-mal")
20
21 zeige_tabelle(woerter, "der")
22 zeige_tabelle(woerter, "katze")
```

AUSGABE

```
Mögliche Folgewörter nach der
  hund: 6-mal
Mögliche Folgewörter nach katze
  sitzt: 1-mal
  beobachtet: 1-mal
  die: 1-mal
  springt: 1-mal
  rennt: 1-mal
```

- **Zeilen 1–6:** `ohne_doppelte` liefert eine neue Liste, in der jedes Element nur einmal vorkommt, und zwar in der Reihenfolge seines ersten Auftretens. Das sind die **Kandidaten** für das nächste Wort.
- **Zeilen 8–13:** `zaehle` zählt, wie oft `gesucht` in der Liste steht.
- **Zeilen 15–19:** `zeige_tabelle` setzt alles zusammen: Erst werden die Folgewörter gesammelt, dann wird für jeden Kandidaten die Anzahl ausgegeben.
- **Zeilen 21–22:** Nach „der“ folgt immer „hund“. Nach „katze“ folgen fünf verschiedene Wörter, jedes genau einmal. Dass auch „die“ dabei ist, liegt an „zur katze die katze“: Zwischen der dritten und vierten Zeile des Korpus gibt es keine Grenze, `split()` macht aus allem eine einzige Wortliste.

Stolperstein

Das Programm weiß nichts von Sätzen. Für die Analyse ist der Korpus eine lange Kette von Wörtern. Deshalb entstehen auch Bigramme über das Satzende hinweg, etwa „ball die“ oder „müde“ ganz am Ende ohne Nachfolger. Echte Sprachmodelle behandeln Satzzeichen und Satzgrenzen genauer. Für unser Modell vereinfachen wir das bewusst.

Versuch: Probiere es aus

Rufe `zeige_tabelle` für das Wort „müde“ auf. Warum wird nur die Überschrift ausgegeben? Rufe die Funktion auch mit einem Wort auf, das im Text gar nicht vorkommt, zum Beispiel „maus“.

4.3.5 Die Bigramm-Tabelle

Wenn man `zeige_tabelle` für jedes Wort des Korpus aufruft, entsteht die vollständige **Bigramm-Tabelle** des Modells. Ein Ausschnitt:

Kontext	mögliche Folgewörter (Anzahl)
der	hund (6)
die	katze (4)
hund	?
katze	sitzt (1), beobachtet (1), die (1), springt (1), rennt (1)
läuft	im (1), zur (1), ihr (1)
dem	ball (1), baum (1)
baum	die (2)

Das Wort in der ersten Spalte, nach dem die Folgewörter gesucht werden, heißt **Kontext**. Bei einem Bigramm-Modell ist der Kontext ein einzelnes Wort, bei einem Trigramm-Modell ein Wortpaar.

So eine Tabelle ist eine **Tabelle von Tabellen**: Zu jedem Kontext gehört eine kleine eigene Tabelle aus Folgewort und Anzahl. Man kann sie sich wie ein **Wörterbuch** vorstellen: Unter dem Stichwort „dem“ schlägt man nach und findet dort „ball: 1, baum: 1“.

In vielen Programmiersprachen gibt es genau dafür einen eigenen Datentyp, das **Dictionary** (englisch für Wörterbuch). In der Python-Schreibweise sieht ein Ausschnitt der Tabelle so aus:

```
{"der":  {"hund": 6},
 "dem":  {"ball": 1, "baum": 1},
 "baum": {"die": 2}}
```

Vor jedem Doppelpunkt steht ein Stichwort, dahinter der Eintrag dazu. Der Eintrag ist selbst wieder ein kleines Wörterbuch. Deshalb spricht man von einem **Dictionary von Dictionaries**.

Merke

- Die Ergebnisse werden meist **tabellarisch** verwaltet: Zu jedem **Kontext** (z. B. einem Wort oder Wortpaar) wird eine Liste möglicher Folgewörter mit ihrer jeweiligen Häufigkeit bzw. Wahrscheinlichkeit gespeichert.
- Eine gängige Datenstruktur dafür ist ein **Dictionary von Dictionaries**: `{kontext: {folgewort: anzahl}}`.

Tipp

Dictionaries brauchst du in diesem Skript nicht selbst zu programmieren. Unser Programm kommt mit Listen aus: Es speichert nicht die ganze Tabelle, sondern berechnet die Zeile für ein Wort jedes Mal neu, wenn sie gebraucht wird. Das ist etwas langsamer, liefert aber genau dieselben Zahlen.

Arbeitsauftrag 23: Eigene Analyse

Gib mit dem Programm für die Wörter „die“ und „hund“ jeweils aus, welche Folgewörter mit welcher Häufigkeit vorkommen. Ergänze damit die fehlende Zeile „hund“ in der Tabelle oben.

Ergebnis:

4.3.6 Schritt 3: Wahrscheinlichkeiten berechnen

Für die Wahrscheinlichkeit teilst du die Anzahl eines Folgewortes durch die Anzahl **aller** Bigramme, die mit dem Kontext beginnen. Diese Zahl ist die Länge der Liste `folgewoerter(woerter, wort)`. Für „hund“ gilt zum Beispiel $P(\text{hund} \rightarrow \text{läuft}) = \frac{3}{7} \approx 0{,}43$.

Das Programm soll das für alle Kandidaten eines Wortes ausrechnen. Dabei entstehen **zwei Listen, die zusammengehören** wie die zwei Spalten einer Tabelle: eine Liste mit den Kandidaten und eine mit ihren Wahrscheinlichkeiten. Das Element an Stelle 0 der einen Liste gehört zum Element an Stelle 0 der anderen Liste, und so weiter.

Index	**0**	**1**	**2**	**3**	**4**
Kandidaten	läuft	spielt	der	bellt	ist
Wahrscheinlichkeiten	?	?	?	?	?

Arbeitsauftrag 24: Wahrscheinlichkeiten berechnen

Schreibe eine Funktion `wahrscheinlichkeiten(woerter, wort)`, die für ein gegebenes Wort eine Liste mit den Wahrscheinlichkeiten seiner Folgewörter zurückgibt, und zwar in derselben Reihenfolge wie `ohne_doppelte(folgewoerter(woerter, wort))`. Die Wahrscheinlichkeit ist die Anzahl des Folgewortes geteilt durch die Gesamtanzahl aller Folgewörter zu diesem Wort. Runde auf zwei Nachkommastellen.

Teste die Funktion mit „der“, „die“ und „hund“ und fülle damit die Tabelle oben aus.

4.3.7 SoekiaGPT: Textanalyse im Browser

Für das Arbeiten mit größeren Textsammlungen gibt es Lernprogramme, die dir die n-Gramme und ihre Häufigkeiten anschaulich zeigen. Der Lehrplan empfiehlt **SoekiaGPT**. Es läuft im Browser: www.soekia.ch/GPT.

In SoekiaGPT kannst du eine vorgegebene **Textsammlung** auswählen (zum Beispiel Märchen); je nach Version kannst du auch eigene Texte hinzufügen. Das Programm analysiert die Sammlung und zeigt dir die gefundenen **n-Gramme mit ihren Häufigkeiten**. Gibst du einen Satzanfang ein, zeigt es dir die möglichen **Folgewörter** und setzt den Text Wort für Wort fort. Die genaue Gestaltung der Oberfläche kann sich von Version zu Version ändern. Lass dich davon nicht verwirren: Das Prinzip ist dasselbe wie in deinem Python-Programm.

Beide Zugänge ergänzen sich: SoekiaGPT zeigt das Verfahren anschaulich an größeren, echten Textsammlungen. Dein Python-Programm zeigt dir, was dabei „unter der Haube“ passiert.

Arbeitsauftrag 25: SoekiaGPT erkunden

Öffne SoekiaGPT im Browser.

1. Wähle eine der vorgegebenen Textsammlungen aus. Suche die Anzeige der n-Gramme und notiere drei Bigramme, die besonders häufig vorkommen.
2. Gib einen kurzen Satzanfang ein, der zur Textsammlung passt, und lass dir mögliche Folgewörter anzeigen. Notiere den Satzanfang und zwei vorgeschlagene Folgewörter.
3. Beschreibe in zwei bis drei Sätzen, welche Schritte deines Python-Programms du in SoekiaGPT wiedererkennst.

Meine Beobachtungen:

Gymnasium Ottweiler

4.3.8 Übungen

Arbeitsauftrag 26: Eigenen Textkorpus analysieren

Suche dir einen eigenen, etwas längeren Text: ein paar Sätze aus einem Buch, eine Zusammenfassung eines Liedtextes in eigenen Worten oder etwas frei Erfundenes. Schreibe ihn klein und ohne Satzzeichen in einen mehrzeiligen Text und analysiere ihn mit den Funktionen aus diesem Abschnitt.

Welches Wort hat die **meisten unterschiedlichen** Folgewörter? Schreibe ein Programm, das dieses Wort findet.

Tipp

So kannst du vorgehen: Gehe mit einer Schleife über alle **verschiedenen** Wörter des Textes. Berechne für jedes Wort, wie viele unterschiedliche Folgewörter es hat. Das größte Ergebnis findest du wie das Maximum in einer Liste. Der Anfang könnte so aussehen:

```
for wort in ohne_doppelte(meine_woerter):
    anzahl = len(ohne_doppelte(folgewoerter(meine_woerter, wort)))
```

4.3.9 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Für die Programmieraufgaben brauchst du die Funktionen `folgewoerter`, `ohne_doppelte` und `zaehle` und die Wortliste `woerter` des Hund-Katze-Korpus aus diesem Abschnitt. Schreibe deine Programme in derselben Datei darunter.

Arbeitsauftrag 27: Warum der Computer zählt ●○○

1. Ein Jugendbuch hat 60 000 Wörter. Bestimme, wie viele Bigramme es enthält.
2. Nenne zwei Gründe, warum man die Bigramme eines solchen Buches nicht von Hand auszählt.

Meine Antworten:

Gymnasium Ottweiler

Arbeitsauftrag 28: Eine Tabelle lesen ●○○

Ein Programm hat einen Korpus analysiert. Ein Ausschnitt seiner Bigramm-Tabelle in der Schreibweise `{kontext: {folgewort: anzahl}}`:

```
{"ich":   {"gehe": 3, "bleibe": 1},
 "gehe":  {"heute": 2, "morgen": 1},
 "heute": {"ins": 2}}
```

1. Nenne alle Kontexte in diesem Ausschnitt.
2. Nenne die Folgewörter von „gehe“ mit ihren Häufigkeiten.
3. Wie viele Bigramme beginnen mit „ich“?
4. Berechne die Wahrscheinlichkeit, dass auf „ich“ das Wort „gehe“ folgt.
5. Mit welcher Wahrscheinlichkeit folgt auf „heute“ das Wort „ins“?

Meine Antworten:

Arbeitsauftrag 29: Was liefert folgewoerter? ●○○

Bestimme **ohne Computer** die Ausgabe des Programms. Erkläre dann, warum zwei der Listen leer sind, und zwar aus verschiedenen Gründen.

```
1 schule = "wir lernen mathe wir lernen englisch wir chillen"
2 woerter_schule = schule.split()
3 print(folgewoerter(woerter_schule, "wir"))
4 print(folgewoerter(woerter_schule, "lernen"))
5 print(folgewoerter(woerter_schule, "chillen"))
6 print(folgewoerter(woerter_schule, "schule"))
```

Ausgabe und Erklärung:

Arbeitsauftrag 30: Eine Tabelle aufschreiben ●●○

Gegeben ist der Textkorpus:

> tom spielt fußball lea spielt tennis tom spielt tennis

1. Stelle die vollständige Bigramm-Tabelle in der Schreibweise `{kontext: {folgewort: anzahl}}` auf.
2. Berechne die Wahrscheinlichkeiten aller Folgewörter von „spielt“.

Meine Tabelle und meine Rechnung:

Gymnasium Ottweiler

Arbeitsauftrag 31: Fehler finden ●●○

Ben hat die Funktion `folgewoerter` aus dem Gedächtnis aufgeschrieben. Beim ersten Aufruf klappt alles, beim zweiten nicht:

```python
def folgewoerter_ben(woerter, wort):
    folge = []
    for i in range(len(woerter)):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

test = "der hund bellt der hund schläft".split()
print(folgewoerter_ben(test, "der"))
print(folgewoerter_ben(test, "schläft"))
```

AUSGABE

```
['hund', 'hund']
```

FEHLERMELDUNG

```
IndexError: list index out of range
```

1. Erkläre, warum der erste Aufruf funktioniert und der zweite zu einem Fehler führt.
2. Beschreibe, wie Ben den Fehler behebt.

Meine Erklärung:

Arbeitsauftrag 32: Das häufigste Wort ●●○

Bei der Textkörperanalyse kann man auch **Unigramme** zählen, also einzelne Wörter. Schreibe ein Programm, das im Hund-Katze-Korpus

1. ausgibt, wie viele verschiedene Wörter der Korpus enthält,
2. das Wort findet, das am häufigsten vorkommt, und es mit seiner Anzahl ausgibt.

Verwende `ohne_doppelte` und `zaehle`.

Tipp

Gehe mit einer Schleife über `ohne_doppelte(woerter)` und merke dir das bisher häufigste Wort und seine Anzahl – wie bei der Suche nach dem Maximum.

Arbeitsauftrag 33: Wahrscheinlichkeit für ein Wortpaar ●●●

Schreibe eine Funktion `wahrscheinlichkeit(woerter, wort, folgewort)`, die die Wahrscheinlichkeit zurückgibt, dass auf `wort` das Wort `folgewort` folgt, gerundet auf zwei Nachkommastellen. Hat `wort` keine Folgewörter (es kommt nicht vor oder nur als letztes Wort), soll die Funktion `0` zurückgeben.

Teste mit dem Hund-Katze-Korpus:

- „hund“ → „läuft“ (erwartet: $\frac{3}{7} \approx 0{,}43$),
- „dem“ → „baum“,
- „katze“ → „bellt“,
- „maus“ → „läuft“ und „müde“ → „ist“.

4.3.10 Das Wichtigste in Kürze

- Große Textkorpora werden nicht von Hand, sondern **automatisiert** mit Software analysiert.
- `folgewoerter(woerter, wort)` sammelt alle Wörter, die direkt nach `wort` stehen. Die Länge dieser Liste ist der Nenner für die Wahrscheinlichkeit.
- Mit `ohne_doppelte` und `zaehle` entsteht daraus eine Zeile der **Bigramm-Tabelle**: Kandidaten mit ihren Häufigkeiten.
- Das Wort, nach dessen Folgewörtern gesucht wird, heißt **Kontext**.
- Die Ergebnisse werden **tabellarisch** verwaltet. Eine gängige Datenstruktur dafür ist ein **Dictionary von Dictionaries** `{kontext: {folgewort: anzahl}}`. Unser Programm bildet die Tabelle mit Listen nach.
- **SoekiaGPT** zeigt n-Gramme, ihre Häufigkeiten und mögliche Folgewörter an größeren Textsammlungen.

Selbstcheck

- ☐ Ich kann begründen, warum man große Textkorpora mit dem Computer analysiert.
- ☐ Ich kann erklären, was die Funktion `folgewoerter` macht und warum ihre Länge der Nenner der Wahrscheinlichkeit ist.
- ☐ Ich kann mit dem Programm die Folgewörter eines Wortes mit Häufigkeit und Wahrscheinlichkeit ausgeben.
- ☐ Ich kann beschreiben, wie eine Bigramm-Tabelle aufgebaut ist und was „Kontext“ bedeutet.
- ☐ Ich kann einen eigenen Text mit dem Programm untersuchen.
- ☐ Ich kann in SoekiaGPT eine Textsammlung auswählen und mir n-Gramme und Folgewörter anzeigen lassen.