Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Simon Ging
Wintersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 9

Abgabe: Montag, 20.12.2021, 9:00 Uhr morgens

Aufgabe 9.1 (Text-Analyse; Punkte: 18; Datei: `analysis.py`)

In dieser Aufgabe geht es um die algorithmische Auswertung von Nachrichtentexten.

Nachrichten-Portale stellen häufig einen RSS-Feed bereit, der es erlaubt die Überschriften und Zusammenfassungen der aktuellsten Beiträge programmatisch abzufragen. In unserem Chat finden Sie im Ankündigungs-Raum (Kategorie *Teams*, Raum *2021WS-EiP*) einen Datensatz mit den Beschreibungen aller Artikel, die im Jahr 2020 auf *tagesschau.de* publiziert wurden. Jede Zusammenfassung ist dabei in einer einzelnen Datei abgelegt. Zum Beispiel enthält die Datei `00001.txt` den folgenden Text:

```
Subject: Liveblog: ++ Weitere Lockerungen in Schleswig-Holstein ++
From: "tagesschau" <>
Date: Wed, 15 Jul 2020 17:39:56 +0200
Content-Type: text/html; charset="utf-8"
Link: http://www.tagesschau.de/newsticker/liveblog-coronavirus-mittwoch-123.html

Ab kommender Woche sind in Schleswig-Holstein wieder Veranstaltungen mit
bis zu 500 Besuchern erlaubt. In den USA sind erstmals mehr als 67.000
Corona-Neuinfektionen binnen 24 Stunden gemeldet worden.
Alle Entwicklungen im Liveblog.
```

Wir wollen nun mit einfachen Methoden auswerten, welche Themen 2020 besonders relevant waren und welche Zusammenhänge zwischen verschiedenen Themen bestehen. Da die Sammlung 22087 Artikelbeschreibung enthält, wäre es sehr mühsam diese Auswertung von Hand durchzuführen. Stattdessen schreiben wir ein Pythonskript, welches versucht sich dieser Problemstellung algorithmisch zu nähern. Dabei bedienen wir uns den folgenden Metriken:

- Wir wollen wissen, wie häufig einzelne Worte in der Gesamtmenge der Artikelbeschreibungen vorkommen. Dies dient als Indikator für die Relevanz der Worte. Zum Beispiel ist zu erwarten, dass "Corona" in vielen Artikelbeschreibungen vorkommt.
- Wir wollen wissen, wie häufig zwei Worte in den selben Artikelbeschreibungen vorkommen. Dies dient als Indikator dafür, dass zwischen den Worten eine inhaltliche Beziehung vorhanden ist. Zum Beispiel ist zu erwarten, dass in Artikelbeschreibungen, die das Wort "Corona" enthalten, auch häufig das Wort "Masken" zu finden ist.

Das Ein- und Ausgabeverhalten des Pythonskripts soll dabei wie folgt sein (Benutzereingaben sind wie immer in blau hervorgehoben):

```
$ python3 analysis.py
Enter path of directory to evaluate: tagesschau/all

Read 22087 files.

Computing statistics...

Top 10 words:
  liveblog (5458 times)
  entwicklungen (5130 times)
  deutschland (2297 times)
  menschen (2197 times)
  coronavirus (1685 times)
  coronakrise (1600 times)
  zahl (1346 times)
  usa (1196 times)
  viele (1072 times)
  trump (1038 times)

> spd
The word 'spd' was found 263 times.
Top 10 correlations:
  spd grünen (86 times)
  spd cdu (75 times)
  spd union (58 times)
  spd grüne (58 times)
  spd linke (43 times)
  spd thüringen (36 times)
  spd hamburg (35 times)
  spd bürgerschaftswahl (31 times)
  spd sarrazin (29 times)
  spd liveblog (25 times)

> sarrazin
The word 'sarrazin' was found 24 times.
Top 10 correlations:
  sarrazin spd (29 times)
  sarrazin berliner (14 times)
  sarrazin umstrittenen (11 times)
  sarrazin partei (11 times)
  sarrazin streit (10 times)
  sarrazin rauswurf (8 times)
  sarrazin landesschiedskommission (8 times)
  sarrazin finanzsenator (8 times)
  sarrazin errungen (8 times)
  sarrazin darf (8 times)

> :quit
Good bye!
```

Da wir das Auslesen von Dateien noch nicht in der Vorlesung behandelt haben, stellen wir Ihnen die hierfür benötigte Funktionalität bereit: in der Datei `read_files.py`

finden Sie die Funktion `read_files`, die den Pfad zu einem Verzeichnis als Argument nimmt, jede Datei in diesem Verzeichnis durchläuft und den Inhalt der Dateien als Liste von Strings zurückgibt.[1] Der Verzeichnispfad ist relativ zu dem Verzeichnis, wo das Programm *ausgeführt* wird. In dem obigen Beispiel liegt das Verzeichnis `tagesschau` also im selben Verzeichnis wie das Pythonskript `analysis.py`. Stellen Sie sicher, dass in Ihrer Abgabe alle notwendigen `.py`-Dateien enthalten sind, aber *nicht* die Artikelbeschreibungen!

Implementieren Sie das eben beschriebene Programm wie folgt:[2]

(a) (1 Punkt) Wie im Einführungstext zu sehen ist, enthalten die Artikelbeschreibungen in den ersten Zeilen zusätzliche Metadaten, wie z.B. das Datum der Veröffentlichung. Wir möchten uns aber bei unserer Textanalyse auf die Zusammenfassungen beschränken.

Schreiben Sie eine Funktion `remove_header`, die den String einer Artikelbeschreibung als Argument nimmt, die Metadaten entfernt und den restlichen String zurückgibt.

In dem Datensatz zur Tagesschau sind die Metadaten immer genau durch eine freistehende Zeile von der Zusammenfassung getrennt. Sie können annehmen, dass dies für alle Datensätze der Fall ist.

(b) (1 Punkt) Um eine Artikelbeschreibung in eine Liste der enthaltenen Worte zu zerlegen, bietet es sich an die `split`-Methode des `str`-Typs zu verwenden. Diese Methode interpretiert beliebige Zweichenfolgeen als Worte, die durch mindestens ein Leerzeichen oder einen Zeilenumbruch voneinander getrennt sind. Beispiel:

```
>>> "Schulen schließen, Großveranstaltungen verbieten?\nDie Länderchefs haben...".split()
['Schulen', 'schließen,', 'Großveranstaltungen', 'verbieten?', 'Die',
 'Länderchefs', 'haben...']
```

Wie man an den Worten `'schließen,'` und `'verbieten?'` sieht, bekommt man dabei Probleme mit Satzzeichen. Als einfacher Lösungsansatz wollen wir deshalb jedes, durch `split` erzeugte Wort zunächst normalisieren bevor wir es weiterverwenden.

Schreiben Sie hierzu eine Funktion `normalize_word`, die einen String als Argument nimmt, alle Buchstaben in Kleinbuchstaben umwandelt, alle Zeichen entfernt die keine Buchstaben sind und den resultierenden String zurückgibt. Beispiel.

```
>>> normalize_word("Ver-BieTen?")
"verbieten"
```

Hinweis: Es bietet sich an die `str`-Methoden `isalpha` und `lower` zu verwenden.

[1] http://proglang.informatik.uni-freiburg.de/teaching/info1/2021/exercise/sheet09/read_files.py

[2] Die einzelnen Aufgabenteile können unabhängig voneinander gelöst werden mit Ausnahme von (c) und (h).

(c) (1 Punkt) Schreiben Sie eine Funktion `article_to_words`, die den Dateiinhalt einer Artikelbeschreibung als String-Argument nimmt, die Metadaten entfernt und die normalisierten Worte der Artikelbeschreibung zurückgibt. Verwenden Sie hierzu die Funktionen aus den beiden vorherigen Aufgabenteilen.

Desweiteren sollen nur Worte in der Ergebnisliste auftauchen, die nach der Normalisierung nicht in einer Menge von Füllworten enthalten sind. Die Menge der Füllworte finden Sie in der Datei `ignored_words.py`.[3]Dort sind Worte wie `"der"` oder `"die"` enthalten, die keine sinnvolle Information zu unseren Analyseergebnissen beitragen, aber sehr häufig in den Artikelbeschreibungen vorkommen.

Für die Artikelbeschreibung aus dem Einführungstext, soll sich die Funktion wie folgt verhalten:

```
>>> article_to_words(content_of_00001_txt)
['kommender', 'woche', 'schleswigholstein', 'veranstaltungen', 'besuchern',
 'erlaubt', 'usa', 'erstmals', 'coronaneuinfektionen', 'binnen', 'stunden',
 'gemeldet', 'worden', 'entwicklungen', 'liveblog']
```

(d) (2 Punkte) Jetzt wo wir die Worte der Artikelbeschreibungen auslesen können, wird es Zeit die erste Metrik zu implementieren: das Zählen der Worte. Hierfür verwenden wir Dictionaries.

Schreiben Sie eine Funktion `count_words`, die eine Liste von Worten als Argument nimmt und ein Dictionary zurückgibt, das jedes Wort der Liste auf die Anzahl seiner Vorkommnisse in der Liste abbildet. Beispiel:

```
>>> count_words(['trump', 'präsident', 'trump', 'amtsenthebung', 'trump'])
{'trump': 3, 'präsident': 1, 'amtsenthebung': 1}
```

(e) (2 Punkte) Da wir nicht nur die Worte aus einer einzelnen Artikelbeschreibung zählen wollen, sondern die Worte aus allen Artikelbeschreibungen, benötigen wir eine Möglichkeit, die Dictionaries zu kombinieren, die wir aus mehreren Aufrufen von `count_words` erhalten.

Schreiben Sie eine Funktion `merge_with_plus`, die zwei Dictionaries `d1` und `d2` als Argumente nimmt und die Einträge aus `d2` zu `d1` hinzufügt. Ist ein Schlüssel (Wort) aus `d2` bereits in `d1` vorhanden, so sollen die zugehörigen Werte (Anzahl Vorkommnisse) addiert werden. Beispiel:

```
>>> d = dict()
>>> merge_with_plus(d, {'trump': 2, 'präsident': 1, 'corona': 1})
>>> d
{'trump': 2, 'präsident': 1, 'corona': 1}
>>> merge_with_plus(d, {'trump': 3, 'präsident': 1, 'amtsenthebung': 1})
>>> d
{'trump': 5, 'präsident': 2, 'corona': 1, 'amtsenthebung': 1}
```

[3] http://proglang.informatik.uni-freiburg.de/teaching/info1/2021/exercise/sheet09/read_files.py

(f) (2 Punkte) Um uns die 10 häufigsten Worte anzeigen zu lassen, müssen wir die Einträge des Dictionary's sortieren. Da Dictionaries keine Ordnung haben, müssen wir die Einträge zunächst in eine Liste konvertieren.

Schreiben Sie eine Funktion `sort_by_value`, die ein entsprechendes Dictionary als Argument nimmt und eine sortierte Liste der Dictionary-Einträge zurückgibt. Die Liste soll dabei in absteigender Reihenfolge nach den Werten des Dictionary's sortiert sein.

Beispiel:

```
>>> sort_by_value({'trump': 3, 'präsident': 1, 'corona': 2})
[(3, 'trump'), (2, 'corona'), (1, 'präsident')]
```

Verwenden Sie zum Sortieren die `sort`-Methode für Listen. Siehe Dokumentation:

https://docs.python.org/3/library/stdtypes.html#list.sort

(g) (4 Punkte) Als nächstes widmen wir uns der zweiten Metrik: der Häufigkeit, mit der zwei Worte in den selben Artikelbeschreibungen auftreten.

Die einfachste Möglichkeit, diese Korrelationen darzustellen, wäre mit einem Dictionary, das Paare von Worten als Schlüssel verwendet und die Häufigkeit als Werte. Diese Repräsentation ist aber ungeeignet für unseren Anwendungsfall: wie im Einleitungstext beschrieben möchten wir nicht nur für Paare von Worten wissen, wie häufig diese zusammen vorkommen, sondern auch für ein einzelnes Wort, mit welchen anderen Worten es am häufigsten vorkommt. Wir stellen deshalb die Korrelationen als ein verschachteltes Dictionary dar: das Dictionary hat als Schlüssel das erste Wort und als Werte jeweils ein Dictionary welches das zweite Wort als Schlüssel hat und die Häufigkeit als Werte. Dies erlaubt es uns, für solch ein Dictionary `d` auf alle Korrelationen des Wortes über `d["word1"]` zuzugreifen, sowie auf die Korrelationshäufigkeit mit einem zweiten Wort über `d["word1"]["word2"]`.

Schreiben Sie eine Funktion `correlated_words`, die ein entsprechendes Dictionary und eine Liste von Worten einer Artikelbeschreibung als Argument nimmt und alle Kombinationen der Worte entsprechend in dem Dictionary einträgt. Wortkombinationen von einem Wort mit sich selbst sollen dabei ignoriert werden.

Beispiel:

```
>>> d = dict()
>>> correlated_words(d, ["trump", "präsident", "corona", "corona"])
>>> d
{'trump': {'präsident': 1, 'corona': 2},
 'präsident': {'trump': 1, 'corona': 2},
 'corona': {'trump': 2, 'präsident': 2}}
>>> correlated_words(d, ["trump", "präsident", "apfelsaft"])
>>> d
```

```
{'trump': {'präsident': 2, 'corona': 2, 'apfelsaft': 1},
 'präsident': {'trump': 2, 'corona': 2, 'apfelsaft': 1},
 'corona': {'trump': 2, 'präsident': 2},
 'apfelsaft': {'trump': 1, 'präsident': 1}}
```

(h) (5 Punkte) Verwenden Sie die Funktionen `read_files`, `article_to_words`, `count_words`, `merge_with_plus` und `correlated_words` um ein Programm zu schreiben, das sich wie im Einleitungstext beschrieben verhält:

- Zunächst soll der Benutzer dazu aufgefordert werden, den Pfad zu dem Verzeichnis der Artikeldateien einzugeben.
- Dann sollen die Dateien geladen werden.
- Dann soll aus dem Inhalt der Dateien die Worthäufigkeit und die Korrelationen berechnet werden.
- Dann sollen die 10 häufigsten Worte und wie oft diese auftreten in absteigender Reihenfolge ausgegeben werden.
- Zum Schluss soll der Benutzer so lange aufgefordert werden Worte einzugeben, bis das Wort `":quit"` eingegeben wurde. Für jedes Wort soll die Häufigkeit des Wortes und die 10 stärksten Korrelationen ausgegeben werden. Die Korrelationen sollen in absteigender Reihenfolge ausgegeben werden und für jede Korrelation soll ersichtlich sein wie oft diese auftritt.

Aufgabe 9.2 (Erfahrungen; 2 Punkte; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplatform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `6.5 h` steht dabei für 6 Stunden 30 Minuten.