Symbolische Programmiersprache: Einführung in Python

Software-Projekt
Konzept-Linker basierend auf Wikipedia Links

Benjamin Roth & Annemarie Friedrich
CIS LMU München

Wintersemester 2016/2017

Achtung – Kleinere Updates in der Aufgabenstellung sind wahrscheinlich! (z.B. um Unklarheiten auszuräumen.) Bitte regelmäßig auf der Kurswebseite nachsehen und bei Unklarheiten sofort (per e-Mail) nachfragen.

In diesem Projekt geht es darum einen Konzept-Linker zu implementieren, der für eine gegebene Zeichenkette die wahrscheinlichste Wikipediaseite zurückgibt. Basieren wird unser Linker auf den Wikipedia-internen Links, genauer gesagt den darin enthaltenen Paaren von verlinktem Text und der Zielseite.
In einem Wikipedia-Artikel werden Links z.B. so markiert:
`Barack Obama worked as a [[Civil and political rights|civil rights]] attorney.`
Hier wird der Text *"civil rights"* zu dem Wikipedia Artikel zu *"Civil and political rights"* gelinkt. Eine Übersicht über alle möglichen Link-Formen finden Sie unter
`https://www.mediawiki.org/wiki/Help:Links#Internal_links`.

Aufgabe 1: Auf der Vorlesungsseite finden Sie die Datei `wikilinks_en.txt.zip`, die sämtliche Linkvorkommen der englischen Wikipedia enthält. Jedes Linkvorkommen ist in eine separate Zeile geschrieben, also z.B.:
`[[Civil and political rights|civil rights]]`
Beachten Sie, dass der gleiche Link mehrmals vorkommen kann.

Die erste Aufgabe besteht darin, aus den verschiedenen Linkformaten den genauen Zielartikelnamen und den gelinkten Text ("Linktext") zu erhalten. Schreiben Sie Zielartikelnamen und Linktext, getrennt durch Tabulator, Zeile für Zeile in eine neue Datei.

Aufgabe 2: **(a)** Schreiben Sie ein Python-Programm, das zählt wie oft eine bestimmte Artikelnamen-Linktext-Kombination vorkommt. Kommt die obige Kombination z.B. 127 mal vor, sollte die Ausgabedatei unter anderem folgende Zeile enthalten:

`127 Civil and political rights civil rights`
Die erste Spalte enthält die Anzahl der Vorkommen, die zweite Spalte den Wikipedia-Artikelnamen und die dritte Spalte den dazugehörigen Linktext. Wikipedia-Artikelnamen können wiederum mehrfach vorkommen.

(b) Schreiben Sie ein Programm, das dieselbe Ausgabe erzeugt wie in (a), aber nur diejenigen Wikipedia-Artikel enthält, die insgesamt mindestens 200 Mal Ziel eines Links sind. **Verwenden Sie, wenn angegeben, diese kleinere Link-Statistik (die nur die häufiger verlinkten Artikel enthält), ansonsten die volle Linkstatistik.**

Anmerkung: Die Link-Statistik ist relativ groß. Falls Sie Schwierigkeiten haben alle Daten im Speicher zu halten, sortieren Sie die Datei in einem Zwischenschritt mit einem externen Tool, z.B. Linux `sort`. Benutzen Sie in diesem Fall die Option `-S` um mehr Hauptspeicher für die Sortierung zu nutzen, also z.B. `sort -S 5G` für 5 Gigabyte.

Aufgabe 3: Schreiben Sie nun ein Programm, welches für Stringeingaben den am häufigsten vorkommenden Wikipediaartikelnamen zurückliefert. Die Stringeingaben entsprechen hier direkt den Linktexten, die in der Lösung von Aufgabe 2 in der dritten Spalte stehen. Wenn kein passender Linktext für eine Stringeingabe gefunden wird, soll auch keine Ausgabe erfolgen (Hinweis: Exceptions benutzen).

Das Programm soll in zwei Modi laufen: Erstens in einem interaktiven Modus, bei dem der Nutzer Strings eingeben kann, und die Wikipediaartikelnamen direkt angezeigt werden. Das Programm soll beendet werden, wenn der Nutzer als Anfrage einen leeren String eingibt.

Zweitens in einem Batchmodus, wo für eine Eingabedatei mit Strings eine Ausgabedatei mit dem jeweils häufigsten Artikelnamen angezeigt wird (je Zeile für Zeile, mit Leerzeilen falls kein Artikelname gefunden wird). Lassen Sie das Programm auf `names_line_by_line.txt` laufen, experimentieren Sie im interaktiven Modus und notieren Sie interessante Beobachtungen. Der Batchmodus soll gestartet werden, falls zwei passende Kommandozeilenargument angegeben werden, beispielsweise so:
`python3 aufgabe_3.py full-linkStats.bz2 names_line_by_line.txt names_line_by_line_results.txt`
Der interaktive Modus hingegen soll gestartet werden, falls keine Kommandozeilenargumente angegeben werden.

Notieren Sie in Ihrer Dokumentation, wie Ihr Programm aufgerufen werden muss, um zum jeweiligen gewünschten Ergebnis zu kommen.

Anmerkungen: **Diese Aufgabe soll mit der vollen Linkstatistik funktionieren.** Falls die Größe der Link-Statistiken ein Problem für Sie darstellt, können Sie in einer Map als Schlüssel Integerrepräsentationen (*Hashes*) der Strings anstelle der Strings selber verwenden, um Platz zu sparen. Falls Sie das tun, beschreiben Sie bitte kurz was sogenannte Hash-Kollisionen sind, wie sie zu Fehlern führen

können, und dokumentieren Sie das Programm entsprechend.
Es ist möglich, dass die Link-Statistik Fälle enthält, in denen der Linktext ein leerer String ist. Ignorieren Sie diese Zeilen in Ihrer Implementierung (benutzen Sie Exceptions).

Aufgabe 4: Verbinden Sie Ihren Linker nun mit dem Stanford Named Entity Tagger und NLTK (`nltk.tag.stanford.StanfordTagger`). Schreiben Sie ein Program, welches für einen beliebigen Text die typischen Vorverarbeitungsschritte (Sentence Splitting, Tokenization) bis einschließlich dem Erkennen der Named Entities durchführt, und für alle erkannten Named Entities den häufigsten Wikipedia-Artikelnamen findet. Als Ausgabe soll das Programm die Links im Text anzeigen (wie in der Wikipedia, bzw. im Beispiel von Aufgabe 1).

Wie in Aufgabe 3 soll das Programm in 2 Modi laufen: Erstens, in einem Interaktiven Modus, bei dem der Benutzer einen Satz eingibt, und der Satz mit den verlinkten Named Entities direkt angezeigt wird. Zweitens, ein Modus, bei dem die Verlinkung in einer beliebigen Textdatei vorgenommen wird, und das Ergebnis in eine Ausgabedatei geschrieben wird.

Beispiel für den interaktiven Modus:

"Eingabe": `Trump met Obama in the White House.`

"Ausgabe": `[[Donald Trump|Trump]] met [[Barack Obama|Obama]] in the [[White House]].`

Lassen Sie das Programm auch auf `text_for_linking.txt` laufen, und untersuchen Sie die Qualität. Was können Sie feststellen?

Diese Aufgabe soll mit der vollen Linkstatistik funktionieren.

Aufgabe 5: Vermutlich ist Ihnen aufgefallen, dass die Ranking-Methode in Aufgabe 3 extrem simpel ist und nicht immer zu einem zufriedenstellenden Ergebnis führt. Die Suche nach *Johnson* beispielsweise gibt immer den Artikel für den Komponisten *Johnson (composer)* aus, da der Linktext *Johnson* am häufigsten auf diesen Wikipedia-Artikel verweist. Angenommen, wir interessieren uns für eine Person, von der wir wissen, dass sie Fußballspieler war und *Johnson* hieß und möchten gezielt nach einem passenden Artikel suchen – in diesem Fall kommen wir mit der einfachen Ranking-Methode von oben nicht zum Ziel. In dieser Aufgabe implementieren wir einen besseren Ranking-Algorithmus mit Hilfe des Vektorraum-Modells.

(a) In diese Teilaufgabe soll ein Index, bestehend aus *term frequencies* und *inverse document frequencies* erstellt werden. Informieren Sie sich, welche Informationen diese beiden Statistiken jeweils beinhalten.

- Betrachten Sie jeden Wikipedia-Artikel als ein Dokument. Der Inhalt des Dokuments besteht aus **allen Linktexten** (jedoch nicht den Artikelnamen!), die auf den Artikelnamen verweisen. Hierbei sollen mehrfach vorkommende Linktexte auch mehrfach berücksichtigt werden.

– Wenden Sie folgende Vorverarbeitungsschritte auf den Inhalt der Dokumente an: (1) konvertieren Sie alle Zeichen in Kleinbuchstaben; (2) entfernen Sie alle Satzzeichen, die im Set `string.punctuation` enthalten sind, und außerdem runde Klammern; (3) splitten Sie die Linktexte mit Hilfe von Leerzeichen; (4) stemmen Sie die so entstandenen Tokens mit Hilfe des PorterStemmers (`nltk.stem.porter`).

– Berechnen Sie die *inverse document frequency* (idf) für jedes oben gefundene Token (term) nach der Formel:

$$idf(term) = log(\frac{number\_of\_all\_documents+1}{number\_of\_documents\_that\_contain\_term+1})$$

Um den Logarithmus zu berechnen, benutzen Sie bitte die Funktion `math.log(x)`. Schreiben Sie den so entstandenen Index in eine Datei namens `idf.csv`. Die erste Spalte enthält den jeweiligen Term, die zweite Spalte die *idf*; Spalten sollen durch Tabulatoren getrennt sein.

– Berechnen Sie außerdem die *term frequency* (tf) für die Terme pro Dokument nach der Formel:

$$tf(term, doc) = \frac{number\_of\_times\_term\_occurs\_in\_doc}{maxOccurrences}$$

Hierbei ist *maxOccurrences* die Häufigkeit des häufigsten im Dokument vorkommenden Terms. Schreiben Sie diese Information in einem geeigneten Format in eine Datei namens `tf.csv`.

– **Anmerkung**: Das Erstellen dieses Index dauert in unserer Referenz-Implementierung weniger als eine Minute. Eine Zeile der Datei `tf.csv` sieht beispielsweise so aus:

```
Royal Hospital Chelsea     'chelsea':  0.9606986899563319, 'hospit':
1.0, 'at':  0.004366812227074236, 'royal':  1.0, 'governor':
0.1572052401746725, 'of':  0.013100436681222707
```

Eine Zeile aus dem Index `idf.csv` sieht beispielsweise so aus (ohne Smoothing[1]):
`governor 5.898957823881563`

(b) In dieser Teilaufgabe benutzen wir die in (a) erstellen Indizes um Suchanfragen zu bearbeiten. Eine Suchanfrage (*query*) besteht aus einer Liste von Wörtern, z.B. *Johnson soccer*. Ihr für diese Aufgabe erstelltes Programm soll wiederum in den beiden in Aufgabe 3 beschriebenen Modi laufen können.

– Wenden Sie dieselben Vorverarbeitungsschritte wie in Teilaufgabe (a) auf die Suchanfrage an.

[1] $idf(term) = log(\frac{number\_of\_all\_documents}{number\_of\_documents\_that\_contain\_term})$ ⇒ in der Praxis sollte es keinen Term geben, der in allen Dokumenten vorkommt (auch nicht in der Entwicklungsdatei `donald_sample.csv`, wenn Sie die "Dokumente" für jeden Artikel ausschließlich aus den Linktexten erstellen, s.o.). Falls Ihre Lösung Smoothing anwendet, ist das kein Problem, vermerken Sie das einfach in Ihrer Dokumentation.

- Finden Sie alle für eine Query relevanten Dokumente: alle Dokumente, deren Inhaltsterme sich mit den Termen der Query überlappen.
- Stellen Sie sowohl die Suchanfrage als auch jedes relevante Dokumente als einen Vektor dar, wobei die Dimensionen des Vektors den Termen entsprechen und den Wert $tf(term, doc) * idf(term)$ zugewiesen bekommen.
- Berechnen Sie die Ähnlichkeit zwischen der Query und jedem relevanten Dokument nach der Formel:

$$sim(\vec{q}, \vec{d}) = \frac{\sum_t q_t * d_t}{|\vec{q}| * |\vec{d}|}$$

  Im Nenner wird das Skalarprodukt zwischen dem Queryvektor und dem Dokumenten-Vektor berechnet, im Nenner wird der Betrag der jeweiligen Vektoren berechnet. Informieren Sie sich auf Wikipedia, wie dies jeweils mathematisch berechnet wird. Tipp: $|\vec{q}|$ ist für eine bestimmte Suchanfrage konstant und kann deshalb ignoriert werden. Ranken Sie die Wikipedia-Artikel nach diesen Ähnlichkeitswerten und geben Sie so für jede Query den wahrscheinlichsten Artikel zurück. Für die Suchanfrage *Johnson soccer* ist das *Eddie Johnson (American soccer)*.
- **Weitere Tipps**: Wenn Sie die Ähnlichkeit $sim(\vec{q}, \vec{d})$ berechnen, sind die Dimensionen aller Terme, die nicht in der Query vorkommen, 0 (da die term frequency im "Dokument" Query 0 ist). Es genügt daher, wenn Sie in Ihrer Implementierung jeweils die Dimensionen, die den Wörtern der Suchanfrage entsprechen, betrachten. (Tipp: Dictionaries verwenden.)
- Der in (a) erstellte Index (d.h. die Dateien `tf.csv` und `idf.csv`) sollen hier *nicht* erneut berechnet werden, sondern am Anfang nur eingelesen werden. Dies kann für das komplette Corpus einige Minuten dauern, entwickeln Sie daher unbedingt mit Hilfe der kleineren Entwicklungsdatei.

Für diese Aufgabe können Sie die kleinere Linkstatistik verwenden.

Aufgabe 6: Erweitern Sie nun das Programm aus Aufgabe 4 durch den Ranking-Algorithmus aus Aufgabe 5, um den Kontext bei der Verlinkung zu berücksichtigen.

- Anstatt des häufigsten Wikipedia-Artikelnamens lassen Sie das Programm in einem ersten Schritt für jede erkannte Named Entity die drei häufigsten Artikelnamen zurückliefern.
- Wählen Sie aus diesen Artikelnamen den besten anhand der TF-IDF Methode aus. Als Suchanfrage (Query) verwenden Sie den Kontext der jeweiligen Named Entity Mention, z.B. ein Wortfenster von 5 Wörtern in jede Richtung, oder das gesamte Dokument.
- Experimentieren Sie mit verschiedenen Parametern und lassen Sie das Programm auch auf `text_for_linking.txt` laufen, und untersuchen Sie die Qualität. Was können Sie feststellen?

Für diese Aufgabe können Sie die kleinere Linkstatistik verwenden.

Aufgabe 7: Haben Sie Ideen, wie sie den (zugegebenermaßen immer noch nicht perfekten) Ranking-Algorithmus verbessern könnten? Sie können hier auch gerne zusätzliche Daten verwenden, z.B. Volltext-Daten aus einem Wikipedia-Dump. Setzen Sie diese in Ihrem Code um (als konfigurierbare Optionen oder in einer neuen Datei).

Abgabe / Präsentation:

- Checken Sie Ihre Lösung wie die Übungen in das Gitlab-Projekt Ihres Teams ein (kontaktieren Sie ggf. die Tutoren).
- Achten Sie bitte darauf, dass die Dateien alle einheitlich benannt sind, direkt im Projekt Ordner liegen und es einen Ordner (data) für die Linkstatistiken etc. gibt. Der Inhalt des `Projekt`-Ordners soll folgendermaßen aussehen:

  ```
  aufgabe_1.py
  aufgabe_2a.py
  aufgabe_2b.py
  aufgabe_3.py
  aufgabe_4.py
  aufgabe_5a.py
  aufgabe_5b.py
  aufgabe_6.py
  aufgabe_7.py
  data/
  documentation.pdf
  ```

  Dabei sollten Sie **nicht** für jede Aufgabe den "bisherigen" Code kopieren, sondern nach Möglichkeit z.B. in Aufgabe 6 das Modul `aufgabe_4` **importieren** und die dort definierten Funktionen und Klassen wiederverwenden.

- Als Teil der Lösung schreiben Sie bitte eine Dokumentation (**maximal 5 Seiten**) im PDF-Format, die die Struktur Ihres Codes und Ihrer Ein- und Ausgabedateien erklärt und außerdem zeigt, wie die einzelnen Module aufgerufen werden können. Antworten Sie in dieser Dokumentation auch auf oben gestellte Fragen und berichten Sie genau, welche Schwierigkeiten aufgetreten sind oder welche Teilaufgaben evtl. noch nicht gelöst wurden.
  - Die Dokumentation soll uns ermöglichen, die Struktur Ihres Codes zu verstehen, wie man Ihr System benutzen sollte, und Ihre Ideen für Aufgabe 7 erklären.
  - Formatierung: Einfacher Zeilenabstand, Schriftgröße 10 und 2cm Rand.
- Vermerken Sie in Ihrer Dokumentation und auch in Kommentaren in all Ihren Python-Dateien, welches Team-Mitglied welche Teilaufgaben gelöst hat. Schreiben

Sie zum Beispiel in den docstring jeder einzelnen Methode, wer (ein oder mehrere Personen) die jeweiligen Funktionen implementiert hat.

- Jedes Team muss außerdem eine Kurzpräsentation über den eigenen Lösungsansatz, und eventuell aufgetretene Probleme (sowie deren Lösung) halten.
  - Gesamtlänge der Präsentation: max. 5 Minuten pro Team, jedes Team-Mitglied muss einen Aspekt Vortragen.
  - Es folgen ca. 10 Minuten Fragen
  - Inhalt der Präsentation soll sein:
    - Überblick über Gesamtarchitektur.
    - Probleme die Aufgetreten sind, und wie sie gelöst wurden.
    - Eigene Beobachtungen und Ideen.
  - Schicken Sie Ihre Vortragsfolien vorab an `sp1617@cis.uni-muenchen.de`.

Zusätzliche Hilfestellung:

- Das Modul `bz2` könnte hilfreich sein, falls Sie die Daten innerhalb Ihres Python-Codes on-the-fly extrahieren möchten: `https://docs.python.org/3.6/library/bz2.html`