1_nlp_crawling_schlaak_weise

August 23, 2020

Von Pascal Schlaak, Tim Weise - Natural Language Processing (SoSe 20)

1 Web Crawling

Die Rohdaten für die Sprachverarbeitung stellt in unserem Projekt die International Movie Database (IMDB): Diese führt auf ihrer Seite eine "Top 250" der am besten bewerteten Filme auf.

Die Herausforderung besteht demnach in der Transformation der auf den Websiten veröffentlichten Daten in ein aggregiertes und strukturiertes Datenformat.

Für die grundlegenden Informationen zu den jeweiligen Filmen (Titel, Jahr, durchschnittliche Bewertung) sind hierfür alle Informationen auf den Übersichtslisten zu finden, die wiederum in Seiten à 50 Filme veröffentlicht sind.

Relevant für die Sprachverarbeitung sind jedoch in erster Linie die Synopsen, da diese besonders große Textmengen liefern: Diese finden sich jeweils in Detailseiten, welche einem festen URL-Schema folgen.

1.0.1 Scrapy als Web Crawler

Als Framework für das Web Crawling kommt **Scrapy** zum Einsatz, das als Python-basiertes Framework den Vorteil mit sich bringt, dass die gesamte Codebasis auf derselben Programmiersprache und somit demselben Tooling aufbaut.

1.0.2 Architektur

Scrapy baut auf sog. **Spiders** auf, die als in sich geschlossene Einheiten Websiten nach den in ihnen definierten Anweisungen crawlen.

Die Scrapy-Engine erwartet diese Scraperdefinitionen in Form von Implementierungen der Spider-Basisklasse. Diese müssen ein `name`-Feld haben, mittels dessen der Crawler im Aufruf aus der CLI eindeutig adressiert werden kann. Darüber hinaus muss mit dem Feld `start_urls` eine Liste an Einstiegspunkten für den Crawler definiert werden. An diese URLs wird beim Start des Crawlers ein HTTP-Request gesendet und die HTML-Seiten-Antwort an den default-Callback zurückgegeben. Um Elemente auf den Seiten zu selektieren kommen CSS- bzw. XPath-Selektoren zum Einsatz:

1.1 Spider-Implementierung

Auf der Startseite der IMDB Top 250 befinden sich die Filme in Listenelementen jeweils in einem `<div>`-Element mit der Klasse `lister-item-content`. Diese Listenelemente lassen sich

entsprechend mittels `response.css("div.lister-item-content")` als Python-Liste extrahieren. Wir wollen aus den Listenelementen jedoch das Erscheinungsjahr, den Titel, die durchschnittliche Bewertung und den Link zur Synopsis extrahieren. Hierfür iterieren wir über die Python-Liste, wenden weitere Selektoren auf die Einzelelemente an und geben die Ergebnisse mit `.get()` als String zurück. Es zeigt sich, dass die Unterseiten mit den Synopsen dem Schema `<unterseitenlink>/plotsummary` folgen, wobei der Link zur Unterseite aus der Listenelement-Kopfzeile extrahiert werden kann.

```
for movie in response.css("div.lister-item-content"):
    synopsis_link = movie.css(
      ".lister-item-header a::attr(href)")[0].get() + "plotsummary/"

    main_movie_info = {
      "title": movie.css(".lister-item-header a::text")[0].get(),
      "date": movie.css(".lister-item-header .lister-item-year::text")[0].get().replace('(', '
      "rank": movie.css(".lister-item-header .lister-item-index::text")[0].get().replace('.',
    }
```

Da render ein Generator ist, wird mit `yield` der nächste Wert geliefert, welcher über die `response.follow`-Methode bezogen wird. Dabei wird dem Link der Synopsis gefolgt und sobald die HTML-Response vorliegt die Callback-Funktion `parse_synopsis_page` ausgeführt, welche den Synopsistext extrahiert. Um die bislang extrahierten Felder des aggregierten JSON-Objekts nicht zu verlieren, werden diese dem Callback als `meta`-Parameter übergeben.

```
yield response.follow(synopsis_link, callback=self.parse_synopsis_page, meta={"main_movie_info"
```

Aus den Unterseiten mit den Synopsen muss ein XPath-Selektor eingesetzt werden, um den Text als Gesamtes, also auch aus Textknoten in Kindelementen wie a-Tags zu extrahieren.

Das Python-Dictionary wird schließlich im Key “synopsis” um den extrahierten Text ergänzt.

```
  def parse_synopsis_page(self, response):
    complete_movie_info = response.meta["main_movie_info"].copy()
    synopsis_text_nodes = response.xpath(
      '//ul[@id="plot-synopsis-content"]/li//text()').getall()
    complete_movie_info["synopsis"] = "".join(synopsis_text_nodes)
    yield complete_movie_info
```

Zurück in der parse-Funktion wird in der Schleife noch der CSS-Selektor für den Button zur nächsten Seite aufgerufen. Liefert dieser ein Ergebnis, enthält die Seite einen solchen Button und somit eine weitere Seite mit 50 Listenelementen und dem Link wird gefolgt, anderenfalls ist die Schleife beendet.

```
next_page = response.css("a.lister-page-next.next-page::attr(href)").get()
if next_page is not None:
  next_page = response.urljoin(next_page)
  yield scrapy.Request(next_page, callback=self.parse)
```

1.1.1 Ausführen des Spiders und Rückgabe als JSON-Datei

Scrapy lässt sich mit einer eigenen CLI ausführen. Der Befehl für den Spider `movies` lautet

```
scrapy crawl movies -o movies.json
```

1.1.2 Navigation

- Weiter zu: Datenvalidierung
- Zurück zur Übersicht

[ ]: