UNIVERSITÄT LEIPZIG

Recommendation & Machine Learning

Implementierung

---

2. Testat

Autor	Matthias Frei
	Simon Ganz
Matrikelnummer	3742806
	3741513
Studiengang	M.Sc. Informatik
Bearbeitungszeitraum	SS 2016
Veranstaltung	Big Data Praktikum
Betreuer	Dr. Eric Peukert

Inhaltsverzeichnis

1 Einleitung **1**

2 Apache Spark **2**

3 Collaborative Filtering und ALS **3**

3.0.1 Umsetzung 3

4 Benutzerschnittstelle **5**

4.1 Client 5

4.1.1 Funktionalitäten 5

4.1.2 Technische Umsetzung 6

4.2 Server 7

4.2.1 Statische Dateien 8

4.2.2 Laden des Spark-Frameworks 8

4.2.3 Beschreibung der REST-Schnittstelle 8

5 Systemkommunikation **11**

5.1 Messwerte Laufzeit 12

Kapitel 1

Einleitung

In diesem Projekt soll ein „Movie Recommendation System“ mithilfe des Frameworks Spark[1] umgesetzt werden.
Das System soll dabei zunächst mit einer festen Anzahl an Nutzern, Filmen und Bewertungen trainiert werden, um anschließend Vorhersagen für Filme zu treffen, die einem bestimmten Nutzer gefallen könnten. Die hierfür verwendeten Daten werden von der Plattform Movielens[2] bezogen.
Die notwendigen Daten finden sich in einer CSV Datei mit dem Name `ratings.csv`. Jede Zeile besteht aus einer Nutzer-Id, einer Film-Id, einer Bewertung und einem Zeitstempel. Das System soll nun Bewertungen für die Nutzer-Film-Paare vorhersagen, die in dieser Datei nicht aufgelistet sind.
Diese Vorhersagen werden durch den Einsatz eines Machine-Learing-Algorithmus auf den gegebenen Daten generiert.

Im Folgenden werden das Framework Apache Spark und der in diesem Projekt verwendete Algorithmus kurz vorgestellt.

Kapitel 2

Apache Spark

Apache Spark ist ein „Cluster Computing Framework“ das aus mehreren einzelnen Komponenten besteht. Die Grundlage des Spark Systems wird dabei durch den „Spark Core“ gebildet. Er stellt die grundlegenden Funktionalitäten wie beispielsweise die Aufgabenverteilung, die Ein- und Ausgabe von Daten oder das Scheduling von Prozessen zur Verfügung.

Spark verwendet als Datenstruktur das sogenannten „Resilient Distributed Dataset“ (RDD) auf denen die entsprechenden Spark-Operationen ausgeführt werden. RDDs können dabei auf mehrere Rechner verteilt werden um parallele Berechnungen zu ermöglichen. Sie werden entweder aus externen Quellen gebildet oder als Ausgabe von verschiedenen Transformationsanwendungen erzeugt.

Neben dem „Spark Core“ stellt die Machine Learning Library (MLlib) einen wichtigen Bestandteil von Spark dar und ist für die Umsetzung des „Recommendation Systems“ essentiell. Bei der MLlib handelt es sich um eine Funktionsbibliothek, welche verschiedenen Machine-Learing-Algorithmen zur Verfügung stellt. Für das zu konstruierende System werden wir das „Collaborative Filtering“ verwenden, das im nächsten Abschnitt kurz erklärt wird.

Kapitel 3

Collaborative Filtering und ALS

„Collaborative Filtering“ ist eine weitverbreitete Technik für „Recommendation Systems“.
Diese Methode begründet sich durch die Annahme, dass falls zwei Personen A und B das selbe Interesse bezüglich einer Sache X haben, es wahrscheinlicher ist, dass Person A in einer anderen Sache Y mit B übereinstimmt, als mit einer anderen zufälligen Person. Im Fall des „Movie Recommendation System“ bedeutet das, dass einem Nutzer Filme eines anderen Nutzers gefallen könnten, wenn beide Nutzer bestimmte Filme ähnlich gut bewertet haben.

Um diese Filmvorschläge zu bestimmen müssen die leeren Einträge der Nutzer-Film-Matrix ausgefüllt werden. Diese Lücken entstehen wenn ein Nutzer noch keine Wertung für einen Film gesetzt hat. Die vorhergesagten Werte können dann als Empfehlungen an den Nutzer interpretiert werden. Die Umsetzung in Spark erfolgt durch den „Alternating Least Squares“ (ALS) Algorithmus. Dieser Algorithmus schätzt durch eine Faktorisierung der Nutzer-Film-Matix die fehlenden Einträge. Hierzu wird abwechselnd einer der beiden Matrizen festgehalten und das daraus resultierende quadratische Gleichungssystem gelöst. So können die fehlenden Einträge der Matrix iterativ berechnet werden. Als Eingabe fordert der Algorithmus die Nutzer-Film-Matrix, die zuvor aus einer CSV Datei eingelesen wird. Für die weiteren Parameter werden die default Werte verwendet. In Kapitel 5 wird ein kurzer Überblick über den Ablauf des Systems vom Einlesen bis zur Ausgabe der Ergebnisse gegeben.

3.0.1 Umsetzung

Zunächst müssen die gegebenen Daten aus den von MovieLens[2] bereitgestellten CSV-Dateien eingelesen werden (movies.csv, ratings.csv, links.csv). Die eingelesenen Dateien werden in gleichnamige HashMaps abgelegt, um später effizient auf die einzelnen Filminformationen zugreifen zu können.

Um nun Filmvorschlägen generieren zu können werden dafür n bewertete Film benötigt. Diese müssen im Format $movie\_id, rating$ vorliegen. Die Filme werden anschließend mit der $user\_id = 0$ in die Nutzer-Film-Matrix eingefügt, welche durch das JavaRDD *ratings* repräsentiert wird.

Über die Funktion *recommendMovies(int n)* können nun n Filmvorschläge generiert werden. Dafür wird zunächst ein *MatrixFactorizationModel* mit den default Werten erzeugt, welches als Eingabe die Nutzer-Film-Matrix fordert.
Durch *model.recommendProducts(0,n)* werden dann für den Nutzer mit der $Id = 0$ n Filmvorschläge erzeugt. Die Filmvorschläge haben die Form (movie_id, Empfehlungsquote des ALS) und müssen anschließend mit den Informationen aus den HashMaps angereichter werden, um die gewünschte Form (movie_id, movie_title, genre, rating, Empfehlungsquote des ALS, imdb-Id) zu erhalten.

Das Rating durch andere Nutzer wird dabei durch den Mittelwert der abgegebene Wertungen der einzelnen Nutzer bestimmt und in einer separaten HashMap gespeichert.

Um alle Filme bezüglich einem gegeben Genre abzufragen wurde außerdem eine HashMap erzeugt, welche für jedes mögliche Genre eine Liste von $movie\_ids$ führt.

Kapitel 4

Benutzerschnittstelle

Die Interaktion des Anwenders mit dem Recommendation System erfolgt über eine webbasierte Clientanwendung, welche Anfragen an einen Server im Hintergrund weiterleitet. Der Client basiert auf gängigen Webtechnologien, sodass ein Nutzer sie als normale Webseite/-anwendung öffnen kann. Die Clientapplikation kommuniziert mit dem Server über eine vorher definierte REST-Schnittstelle, auch REST-API genannt.
Der Server beantwortet die Anfrage durch seinen Datenbestand bzw. führt die zur Antwort nötigen Berechnungen darauf aus und gibt das Ergebnis zurück an die Webanwendung.
Der Datenaustausch erfolgt über HTTP im JSON-Format.

4.1 Client

4.1.1 Funktionalitäten

Der Benutzer hat in der Webanwendung Zugriff auf drei Ansichten.

- **Start**
  Die Seite „Start“ ist die Einstiegsseite. Es werden dem Nutzer neun zufällig ausgewählte Filme in zwei unterschiedlichen Darstellungen angezeigt.
  Die neun Filme werden als sog. „Slider“ im oberen Teil der Seite als auch als Aufzählung mit rundem Bild („Blasenform“) darunter angezeigt.

- **Recommendation**
  Auf dieser Seite kann der Nutzer zehn zufällig ausgewählte Filme mit Sternen zwischen 1 (schlecht) und 5 (gut) bewerten. Die Filme werden anschließend durch den Server analysiert und es werden dem Nutzer Vorschläge von weiteren Filmen angezeigt, die seinem Geschmack entsprechen könnten. Die Vorschläge basieren auf den Bewertungen anderer Nutzer, also umgangssprachlich auf dem „Nutzer denen XY gefällt, gefällt auch...“-Prinzip.

- **Genres**
  Die Seite „Genres“ ähnelt in ihrer Funktionsweise stark der der „Recommendation“-Seite.
  Sie bietet dem Anwender zu Beginn eine Übersicht aller verfügbaren bzw. katalogisierten Filmgenres an. Der Benutzer kann durch Anklicken ein Genre auswählen. Ihm werden im Anschluss, analog zur vorherigen Seite, zehn Film vorgelegt, die alle dem gewählten Genre entsprechen. Diese kann der Nutzer wieder mit 1-5 Sternen bewerten, um anschließend eine Auswahl an Filmvorschlägen zu erhalten.
  Der Nutzen dieser Seite liegt in der Möglichkeit der Beeinflussung des Recommendation-Algorithmuses. Bewertet man alle Filme einer Kategorie bzw. eines Genres extrem schlecht oder extrem gut, ist die Wahrscheinlichkeit vermutlich höher, dass diese in den Vorschlägen vermindert bzw. vermehrt auftreten werden. Allerdings basiert diese Vorgehensweise, bedingt durch den begrenzten Informationsgehalt der Datenquellen, wieder auf den Bewertungen anderer Nutzer. Es wird somit für diese Funktionalität angenommen bzw. vorausgesetzt, dass Anwender, die zum Beispiel einen Dokumentarfilm schlecht bewerten, auch andere Filme des gleichen Genres eher schlecht bewertet haben.

4.1.2 Technische Umsetzung

Die webbasierte Clientanwendung wurde mit gängigen und modernen Webtechnologien implementiert. Die Applikation basiert auf HTML5 in Kombination mit JavaScript. Für eine einfachere und leicht verständliche Implementierung werden zusätzlich die Frameworks jQuery und Bootstrap[3] eingesetzt. Als Designvorlage dient das kostenlose Bootstrap Demotemplate „Carousel“[4] (siehe Abbildung 4.1).
Zusätzliche Informationen über die vom Server ausgelieferten Filme fragt die Clientanwendung selbstständig durch eine Ajax-Anfrage bei der Filmdatenbank IMDB durch die kostenlose Schnittstelle `http://www.omdbapi.com` ab. Die von dort nachgeladenen Informationen dienen nur der Information des Anwenders und fließen in keinerlei Berechnungen von Ratings oder Vergleichbarem ein.

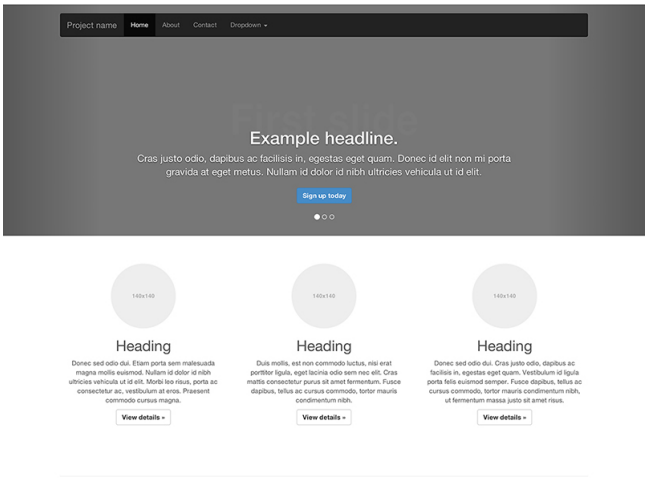

Abbildung 4.1: Bildschirmfoto des Carousel-Templates für Bootstrap

Um die Webanwendung simpel und überschaubar zu halten, ist der Aufbau einfach strukturiert. Durch Anklicken eines Menüpunktes in der Oberfläche wird eine zugehörige JavaScript-Funktionen aufgerufen, die den für die Seite nötigen Inhalt generieren und mittels jQuery in den gewünschten Container auf der Seite angehängt (z.B. `insertView_start()`).
Der gesamte Seiteninhalt befindet sich im Container *`#content-area`*.
Wird eine weitere Seite geladen oder Inhalt mittels einer Ajax-Anfrage vom Server nachgeladen, entfernt ggf. `clear_view()` den aktuellen Seiteninhalt und blendet einen Ladebalken ein. Sobald der erforderliche Inhalt eingetroffen ist, wird in der Callback-Funktion der Ajax-Anfrage die Funktion `remove_loading()` aufgerufen, die sämtliche Ladebalken entfernt.

4.2 Server

Die Serveranwendung ist mit der Programmiersprache Java implementiert. Die REST-Schnittstelle wird durch das Framework „Jersey“[5] zur Verfügung gestellt. Als Java-basierter Webserver wird Tomcat in Version 7 verwendet.

4.2.1 Statische Dateien

Tomcat wird durch standardmäßige Webservlets in der Klasse `com.movrec.Web_Files` angewiesen, alle Dateien des Ordners `/MovRecTwo/src/main/resources/web` auf Anfrage auszuliefern. Der Ordner kommt demnach dem sog. „Web-Root“ gleich. Um per URL an die Dateien zu gelangen, muss der Ordner `/files/` dem Dateinamen vorangestellt werden.

```
1 http://localhost:8080/MovRecTwo/files/custom/js/movrec.
    js
```

Codebeispiel 4.1: Beispiel für den Aufruf einer Datei aus dem Web-Root

Wird keine konkrete Datei angefordert, liefert das Servlet der Klasse `com.movrec.Web_Index` die Datei `index.html` aus dem Web-Root aus.

4.2.2 Laden des Spark-Frameworks

Wird die Serveranwendung gestartet, übernimmt die Klasse `com.movrec.Web_Init` die Initalisierung des Spark-Frameworks. Die erfolgreich erstelle Instanz wird anschließend über das „ServletContext“ global zur Verfügung gestellt, damit Spark durch API-Aufrufe verwendet werden kann.
Wird der Webserver beendet, kümmert sich die Klasse ebenfalls um die Beendigung von Spark.

4.2.3 Beschreibung der REST-Schnittstelle

Nachfolgend werden die einzelnen REST-APIs aufgeführt und deren Funktionsweise kurz erläutert.
Die Anfragen und Antworten sind jeweils mittels JSON formatiert.

- **getRandomMovies()**
  Die Funktion wird via HTTP-GET aufgerufen.
  Sie liefert neun zufällig ausgewählte Filme aus dem Datenbestand zurück.
  Die Rückgabe enthält Key-Value Paare. Als Key definiert ist die Movie-ID

aus dem Datenbestand. Das Value ist eine Liste mit den Werten Titel, Genre, IMDB-ID, Rating.
Beispiel: `"104177":["From One Second to the Next (2013)","Documentary","3108864","5.0"]`

- **getMoviesForRating()**
  Die Funktion wird via HTTP-GET aufgerufen.
  Sie liefert zehn zufällig ausgewählte Filme aus dem Datenbestand zurück. Die Rückgabe enthält Key-Value Paare. Als Key definiert ist die Movie-ID aus dem Datenbestand. Das Value ist eine Liste mit den Werten Titel, Genre und IMDB-ID
  Beispiel: `"7440":["Paper Clips (2004)","Documentary","0380615"]`

- **setMoviesForRating(@FormParam("ratingData") String ratingData)**
  Die Funktion wird via HTTP-POST aufgerufen.
  Als Eingabe erwartet sie Key-Value-Paare. Der Key entspricht der Movie-ID, das Value dem Rating des Nutzers zwischen 1 und 5.
  Nach erfolgter Berechnung liefert sie sechs Filmvorschläge zurück. Der Key entspricht auch hier der Movie-ID, das Value einer Liste mit Titel, Genres, Rating (anderer Nutzer), Empfehlungsquote des ALS-Algorithmuses und IMDB-ID.
  Beispiel: `"82":["Antonia's Line (Antonia)(1995)", "Comedy|Drama", "3.45", "4.090390537800163)", "0112379"]`

- **getAllGenres()**
  Die Funktion wird via HTTP-GET aufgerufen.
  Sie liefert alle vorkommenden Film-Genres aus dem Datenbestand zurück. Die Rückgabe enthält eine Liste mit den Genre-Bezeichnungen.
  Beispiel: `["Western","Documentary","Sci-Fi"]`

- **getMoviesForGenre(@FormParam("genre") String genre)**
  Die Funktion wird via HTTP-POST aufgerufen.
  Als Eingabe erwartet sie ein Key-Value-Paar. Der Key entspricht der Konstante „genre“, das Value den Namen des Genres.
  Die Rückgabe ist analog zu `getMoviesForRating()`.

Kapitel 5

Systemkommunikation

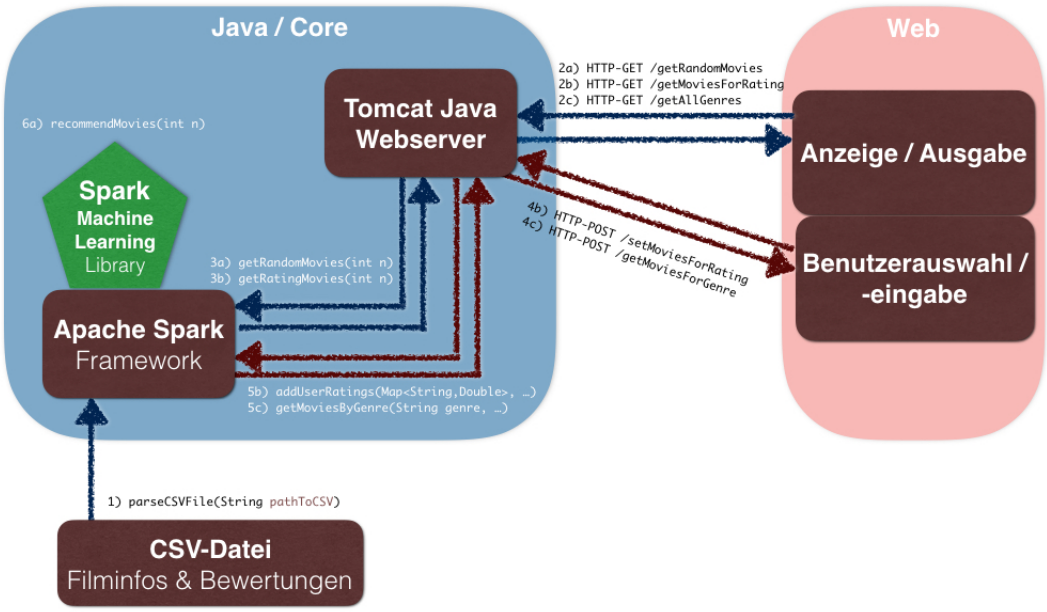

Abbildung 5.1: Kommunikation im Recommendation System

Abbildung 5.1 zeigt den geplanten Aufbau des Systems.
Grundlage ist ein auf Java basierender Kern, der den Zugriff auf Apache Spark und den Webserver ermöglicht. Die Machine Learning Bibliothek von Spark wird zu Beginn mit bestehenden Filmbewertungdsaten[2] trainiert. Diese werden aus einer CSV (Comma Seperated Value) Datei, welche aus dem Dateisystem gelesen wird, extrahiert.

Filminformationen und Bewertungen werden anschließend über einen in Java implementierten Webserver bereitgestellt. Der Nutzer kann sich diese somit in seinem Browser anzeigen lassen. Tätigt er eine Auswahl oder bewertet selber

Filme, werden diese Informationen an den Webserver übertragen, welcher sie wiederum an Spark weiterreicht. Hier werden nun anhand der Nutzerdaten passende Vorschläge für ihn generiert und wieder über den Webserver an den Nutzer ausgeliefert.

5.1 Messwerte Laufzeit

Im folgenden werden Messwerte bezüglich der Anzahl der verwendeten Cores und der Größe der initialen Datenmenge (MovieLens[2]) tabellarisch dargestellt.
Zur Messung wurde Spark wie folgt konfiguriert:

```
1 SparkConf conf = new SparkConf().setAppName("Movie
     Recommendation");
2 conf.setMaster("local[2]");
3 sc = new JavaSparkContext(conf);
```

Codebeispiel 5.1: Beispiel für Spark Setup mit zwei Cores

Die Messungen wurden unter Apple OSX 10.11.5 mit einem Intel Core i5 2,4 GHz und 8GB RAM durchgeführt.

Tabelle 5.1: Messwerte

Cores	ml-latest-small	ml-latest
1	12,986 sec * , 13,048 sec **	520,255 sec **
2	17,288 sec * , 16,255 sec **	367,254 sec **
4	27,456 sec * , 26,448 sec **	417,246 sec **

* Alle Filme mit einem Stern bewertet; ** Alle Filme mit fünf Sternen bewertet

Literaturverzeichnis

[1] The Apache Software Foundation: *Spark Overview*. https://spark.apache.org/docs/latest/index.html, Abruf: 24. Mai 2016 um 20:07

[2] GroupLens: *MovieLens*. http://grouplens.org/datasets/movielens/, Abruf: 24. Mai 2016 um 20:15

[3] The Bootstrap Core Team: *Bootstrap*. http://getbootstrap.com, Abruf: 24. Mai 2016 um 20:12

[4] The Bootstrap Core Team: *Carousel Template for Bootstrap*. http://getbootstrap.com/examples/carousel/, Abruf: 24. Mai 2016 um 20:19

[5] Oracle Corporation: *Jersey - RESTful Web Services in Java*. http://getbootstrap.com/examples/carousel/, Abruf: 19. Juli 2016 um 11:40

[6] The Apache Software Foundation: *Machine Learning Library (MLlib) Guide*. http://spark.apache.org/docs/latest/mllib-guide.html, Abruf: 24. Mai 2016 um 20:09