# Building a recommender system with R

The main task of a recommender system is to predict the users response to different options. GroupLens Research has collected and made available rating data sets from the MovieLens web site. A data set with 10 million rows is downloaded and split into a train (90%) and a 'unseen' test (10%) set for evaluation. This document is a deliverable of the first of two capstone projects in the 'Professional Certificate in Data Science' course hosted by Harvard University (HarvardX). I followed a CRISP-DM like process: a short background study about content-based versus collaborative-filtering, analyze and prepare data, configure machine learning model and evaluate the results. The evaluation statistic is root-mean-squared error (RMSE) and must be lower than 0.8775 to be rewarded with the maximum grade. This is achieved by applying a collaborative filtering method.

*Keywords*: recommendation, collaborative filtering, LIBMF, matrix factorization, R

**Problem description**

Train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set (unseen test data). The data used is the MovieLens Dataset with 10 million ratings. The download script with basic data wrangling (combine, split into train and test data) is provided by EDX.

**Methods**

I performed the following tasks to design and test a R scripts which predicts movie ratings:

- Background study on popular approaches for designing a recommender system.
- Improve the script provided by EDX so that data is downloaded only once when already available.
- Basic exploratory data analysis.
- I used the 'recosystem' package for Collaborative-Filtering.
- Parameter tuning, and model training and evaluation on the training data.
- Test the model on the 'unseen' test data.

**Popular approaches for designing a recommender system**

- **Content-Based systems** analyse the characteristics of items. Similarity of items is determined by measuring the similarity in their properties. Content-based filtering recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or tags given to the item by users. The user profile is represented with the same descriptors and built up by analyzing the content of items which have been seen by the user.

- **Collaborative-Filtering systems** analyse the relationship between users and items. The underlying assumption of the collaborative filtering approach is that if person A has the same opinion as person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

I've chosen collaborative-filtering approach, since the data, as specified within the capstone project, contains a limited set of meta data. For example, it does not contain user tags nor does it contain information about director, budget or cast of the movies.

**Matrix Factorization**

Collaborative-Filtering systems use an user-item matrix as a basis to identify similarity between relationships. In the simplified example below the system will assume that Bart is most likely to rate movie 3 with a 6 since he agrees with Nuray on other movies.

Table 1: User-item matrix

| User | Movie_1 | Movie_2 | Movie_3 |
|---|---|---|---|
| Bart | 5 | 1 | |
| Karel | 2 | 8 | |
| Nuray | 5 | 1 | 6 |

Other common names for this task include 'collaborative filtering', 'matrix completion' and 'matrix recovery'.

**The MovieLens Dataset**

GroupLens Research has collected and made available rating data sets from the MovieLens web site. Among others, a data set with 10 million rows which we split into a train (90%) and a 'unseen' test (10%) set for evaluation.

*Description*

The training dataset consists of 9000055 records and 6 columns with:

- 69878 anonymized user IDs.
- 10677 movies with title and genre, the real MovieLens id.
- 797 unique lists in the genres column.
- The ratings, made on a 5-star scale, with half-star increments. Hence 10 possible outcomes of our prediction algorithm.
- Timestamp of the rating.

The data spreads a timespam from 1995-01-09 11:46:49 until 2009-01-05 05:02:16.

Table 2: Example rows

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 1 | 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy \| Romance |
| 2 | 1 | 185 | 5 | 838983525 | Net, The (1995) | Action \| Crime \| Thriller |
| 4 | 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action \| Drama \| Sci-Fi \| Thriller |
| 5 | 1 | 316 | 5 | 838983392 | Stargate (1994) | Action \| Adventure \| Sci-Fi |
| 6 | 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action \| Adventure \| Drama \| Sci-Fi |
| 7 | 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children \| Comedy \| Fantasy |

Notes:

- The title contains the release year of the movie.
- The genre column contains multiple genres seperated with a pipe.
- The timestamp of the rating represented seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

*Missing values*

The dataset has 0 records with missing values.

Table 3: Percentage missing values per column

| userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |

*Suggestions for feature engineering when going content based*

A Collaborative-Filtering recommendation system uses a matrix of users, movies and ratings as values. Thus, feature engineering is not required. The engineering of the following derived features may add value to the data when building a Content-Based recommendation system:

- Date attributes: year, month, day of week, hour.
- Release year (included in the title text).
- Number of genres per movie.
- One-hot encoding of genres.
- Add mean, median and or mode of rating per genre, movie and user.
- Add number of ratings per movie.
- Add number of rating per user.

**Machine learning model based on LIBMF with R package 'recosystem'**

I used the R package 'recosystem' which is a wrapper of the open source LIBMF library developed by the Machine Learning Group at Nation Taiwan University. It is developed for recommender system based on 'collaborative filtering'.

*Main features of LIBMF*

The main features of LIBMF include:

- Providing solvers for real-valued matrix factorization, binary matrix factorization, and one-class matrix factorization.
- Parallel computation in a multi-core machine and other technical performance optimizations.
- Cross validation for parameter selection.
- Support disk-level training, which largely reduces the memory usages.

*Design decisions*

I made the following design decisions to optimize RMSE and to overcome some minor shortcomings:

- Instead of providing the original rating values I decided to convert the rating to integers after multiplying the original value by 2.
- The algorithm estimates the rating given by users to the movies rather than picking a rating out of the possible values ranging from 1 to 10. Hence, the outcome of the prediction must be rounded to the nearest integer for optimal RSME.
- The algorithm predicts values out of the range of the provided integer values. Predictions below 1 are adjusted to 1 and predictions above 10 are adjusted to 10.

*Process flow*

The final model consists out of the following steps:

- Feature selection: select user ID, movie ID and rating.
- Data conversion: convert all features to integers. Rating was multiplied with 2 to get it on a 1 to 10 scale.
- Determine optimal parameters with cross validation.
- Train model on 9.000k rows, which 90% of the provided data.
- Predict on 1000k unseen rows, which is 10% of the provided data.
- Correct predictions out of the 1 to 10 range.
- Convert ratings and predictions back to the original scale.
- Calculate RMSE: sqrt(mean((observed - predicted) ^ 2)

**Discussion**

- I did not (yet) analyze the predictions on topics like distribution, bias and overfitting.
- Training on the 9000k row training dataset took a long but reasonable amount of time on my laptop (Pentium I7, SSD, 8 GB). I did not investigate the scalability of the model.
- For reasons of technical performance I evaluated just a view settings for parameters optimization during cross validation. Result might improve a bit when more effort towards parameter tuning.
- Low absolute accuracy of around 25% out of 10 possible outcomes (ratings). The model will give a good estimate (RMSE of 0.7918), which can be used as an indication of user preferences towards an option (movie) but fails to be precise.

**Conclusion**

The main task of a recommender system is to predict the users response to different options. My mission was to grasp the concepts of a collaborative-filtering based recommender system and to provide an example in R. This document and the accompanying script show that it is relatively simple to create a basic system which predicts movie rating. The algorithm scores a RMSE of 0.7918. More than enough to be rewarded with the maximum grade.

Collaborative-filtering using LIBMF is easy to setup in both R and Python but has, as a method, the following cons:

- The so-called 'Continuous Cold Start' when the systems welcomes new movies and/or new users. By design it will not be able to provide reasonable predictions.
- Uses within a system need be active in providing ratings. Inactivity hides changes in their interests over time.
- The system needs to be retrained on the complete matrix after receiving new ratings. Hence, it has a performance penalty. One could consider training different models for different demographic and geographic segments.

A optimal solution would be an ensemble with business rule and more extensive content-based systems. Other common tips are:

- Apply a popularity based strategy. Trending products can be determined by trends and what's been popular recently, demographically, regionally, seasonal, or at that certain time of the day.
- Other user actions like showing interest, viewing or buying, are potentially more robust because most users do not provide product ratings. These actions can be translate to a binary indicator which can be used as input for a collaborative-filtering based recommender system.

**References**

*Resources consulted during background study*

I read, and used, the following online content during the background study about building a recommender system in R:

- http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
- https://www.csie.ntu.edu.tw/~cjlin/libmf/
- https://statr.me/2016/07/recommender-system-using-parallel-matrix-factorization/
- https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf
- https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f

*Resources consulted for technical design*

I recommend reading the following online posts which I used as a basis for the technical solution:

- https://cran.r-project.org/web/packages/recosystem/recosystem.pdf
- https://www.r-bloggers.com/recosystem-recommender-system-using-parallel-matrix-factorization/

*R packages used in software code*

The following R open source packages are used in the software code which is described, or used, in this document:

- **'tidyverse'**, A collection of R packages for data science. Includes 'dplyr' for data wrangling tasks. Ships with 'ggplot2' for data visualization.
- **'lubridate'**, Provides date functions, e.g. extract month and year from a date.
- **'Metrics'**, Calculates performance metrics for model validation.
- **'Caret'**, The caret package (short for Classification And Regression Training) contains functions to streamline the model training process for complex regression and classification problems. It provides a common interface to R packages.
- **'recosystem'**, Recommender system using Matrix Factorization, R wrapper of the 'libmf' library. LIBMF is a Matrix-factorization Library for Recommender Systems. It is open source and developped by the Machine Learning Group at National Taiwan University.