Data Science: Capstone - MovieLens Recommendation Model Report

by Edin Ceman

14th February 2021

1. Introduction

The following report will give you an overview of my *Data Science: Capstone MovieLens* project. In the **Introduction** section, background information on the *MovieLens Collection* is provided. This is followed by a description of the used data set, the goal of the project and the key steps that were performed. In the **Methods & Analysis** section, the modeling approach, process and methods will be explained. The **Results** section will then present and discuss the final results of the recommendation model. Finally, the **Conclusion** section will summarize the outcomes and discuss it's limitations. Please note that for the report at hand and the depicted R code, all comments have been removed for the purpose of clarity and formatting. The commented code and further details can be found in the corresponding R script.

1.1 The MovieLens Collection

The *MovieLens* dataset is a publicly available source of information containing user ratings for a large collection of movies over a defined period of time. This data was collected and made available by GroupLens Research on the web site (http://movielens.org). There are different kinds of sizes of the *MovieLens* data. While there is also a full-size version of the data set available, in this course we use the smaller "10m" version to facilitate the modeling process while also reducing the necessary computing power.

1.2 Dataset

The "10m" dataset from *MovieLens* is a smaller version of the full-size data set and contains 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users of the online movie recommending service *MovieLens*. From the data set, the following information are extracted as part of the *Data Science: Capstone* course:

- **userId:** This is an integer representing a unique user ID. *MovieLens* users were selected at random for inclusion. Their IDs have been anonymized.
- **movieId:** This is a numeric representing a unique movie ID.
- **rating:** This is a numeric representing the user rating for a movie. It can display the following discrete values (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0).
- **timestamp:** This is an integer representing the exact time when the rating for a certain movie by a certain user was made.
- **title:** This is a character value displaying the movie's full title.
- **genres:** This is a character representing the genre of the movie. Movies can be in multiple genres, in this case they are all listed and separated with "|".

1.3 Goal

The goal is to build a movie recommendation model based on the provided *MovieLens* data set. Following that, the aim of the recommendation model is to predict a rating for a certain user and a certain movie at a certain point in time based on the information and patterns from the *MovieLens* data set. To achieve this goal, our model captures different effects or biases in the data set to identify the parameters that shape a user rating. This information can then be used to provide movie recommendations to a certain user.

1.4 Key Steps

To build the movie recommendation model, the following key steps were performed:

- Data Initiation: First, the "10m MovieLens" data set is downloaded, and the relevant information is extracted, transformed and loaded into R. The downloaded data set is then split into an *edx* set (90%) for the purpose of developing, training and testing, and a final hold-out *validation* data set (10%) for the final evaluation of the recommendation model.
- Data Preparation: Next, the *edx* data set is checked for NAs, filtered and enriched with additional features for the model design. The *edx* set is then split again into a train (80%) and a test (20%) set to develop, train and evaluate the recommendation model.
- Model Design: In this step, the recommendation model is built using the train set, and evaluated with the test set. The model is designed such that it captures certain biases or effects in the train data. Additionally, regularization is applied to account for overfitting. The Root Mean Squared Error (RMSE) is used to evaluate the model performance and to select the optimal and final model parameters.
- Model Evaluation: The final model is then evaluated using the final hold-out validation set and the optimized parameters. Predictions are calculated for the validation set and those are used for the final evaluation of the model performance.
- Results: The final model results are presented and discussed.
- Conclusion: Lastly, the report results are summarized, and limitations as well as potential future work is discussed.

2. Methods & Analysis

This section explains the approach for initiating and preparing the data set as well as the designing, building and evaluation of the recommendation model. The data initiation and data preparation sections describe how the *MovieLens* data is loaded and prepared. Based on that, the section *Model Design* describes the approach, rationale and the methods used to build the recommendation model. The *Model Evaluation* section describes the evaluation and optimization process and the metrics used to determine the model performance.

2.1 Data Initiation

The relevant code to download and create the *edx* and *validation* set has been kindly provided in the edx course "Data Science: Capstone". The provided code downloads the "10m MovieLens" data from source, extracts and transforms the relevant information into R. It should be noted that the **movieId** is transformed into a numeric data type, the **title** and **genres** are transformed into a character data type. This data set is then split into a set with the name *edx* and a set with the name *validation*. The *edx* set contains 90% (or 9,000,055 rows) of the 10m MovieLens data set and shall be used to build, train and optimize the recommendation model. The *validation* set contains 10% (or 999,999 rows) from the "10m" data set and represents the **final hold-out** data set that is only used in the final evaluation of the recommendation model. It is important to mention that the validation set is **not used** for the development and training of the model.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2.2 Data Preparation

The *edx* data set is used for the purpose of developing and training the recommendation model. Initially, the *edx* set is checked for NAs, i.e. if there are missing values that need to be filled. The *edx* set however does not have NAs, hence no filling of NAs has to be conducted. Next, the data set is enriched with additional information for the model development. The feature **date** is created from the **timestamp** since we want to include a *Time Effect* in the model later on. Additionally, the *edx* set is filtered to include only users that have at least 15 movie ratings. The rationale for the features is provided in the model design section. The latter step reduces the size of the *edx* set to 8,999,671 rows since not every user has more than 15 ratings. Finally, the prepared *edx* data set **edxT** is then split again into a train set (80%) (or 7,199,736 rows) and a test set (20%) (or 1,799,889 rows) to be used in the model development and parameter optimization.

```r
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")

library(lubridate)
library(ggplot2)

print(sapply(edx, function(x) sum(is.na(x))))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```r
edxT <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit = "day"))
dim(edxT)
```

```
## [1] 9000055       7
```

```r
edxT <- edxT %>% group_by(userId) %>% filter(n() >= 15) %>% ungroup()
dim(edxT)
```

```
## [1] 8999671       7
```

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
testIndex <- createDataPartition(y = edxT$rating, times = 1, p = 0.2, list = FALSE)
trainSet <- edxT[-testIndex, ]
testSet <- edxT[testIndex, ]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
testSet <- testSet %>%
  semi_join(trainSet, by = "movieId") %>%
  semi_join(trainSet, by = "userId")
```

2.3 Model Design

The recommendation model we want to build is called **Regularized Recommendation Model with Movie, Restricted User, Time & Genre Effect**. The goal of the model is to predict a rating for a certain movie, by a certain user at a certain point in time using the *MovieLens* data. The model is designed such that it captures different biases or effects in the training data set to identify the **biases** or **effects** that influence or shape the resulting rating. For the model, we are going to include a *movie*, *user*, *time* and *genre* bias. Those will be explained later in this section. For each of the selected effects, the concept of **regularization** is used to account for **overfitting**. This issue arises when a model is too specific, i.e. such that it performs well on a well known data set, but rather poorly on an unknown data set.

To apply the concept of regularization, we introduce a penalty parameter **lambda** to each bias. The penalty parameter is applied to penalize the effect of a bias such that it's influence on the predicted rating is adjusted. The penalty parameter can be considered as a tuning parameter. Hence, we will define a range of lambdas, and repeat the model training and evaluation against the test set to find the optimal lambda. From my trials, I narrowed down the optimal lambda range to be somewhere between 4.5 to 5.5 in 0.1 steps resulting in only 10 iterations to avoid excessive computing time. We could also conduct a finer penalty parameter search, e.g. 1 to 10 in 0,001 steps which would result in 9,000 iterations to optimize our algorithm. However this approach would most probably result in overfitting and massively extend computing time. Hence, we will stick to the selected sequence from 4.5 to 5.5 in 0.1 steps.

To build the recommendation model, we will use the *train* set and evaluate it with the *test* set that we have created in the previous section. To find our optimal lambda, we will use the *Root Mean Squared Error* (RMSE) to evaluate the model performance for each iteration. The RMSE is a widely used performance measure and is constructed such that it calculates the standard deviation of the *residuals*, i.e. the difference between the true rating and the predicted rating. The RMSE is always non-negative due to the residuals getting squared, and it therefore also penalizes large deviations disproportionally. Hence, it is sensitive to outliers. In terms of evaluation, the smaller the RMSE is, the better is the performance of our model.

We construct our recommendation model using different biases or effects in the data set. A *bias* or an *effect* can be considered a feature or explanatory variable that captures a certain pattern that influences a user's movie rating. As a starting point, some of the biases have been selected on the basis of the information that was provided in the edx module "Data Science - Machine Learning", and on top of that, the recommendation model has been extended, refined, and developed further based on additional insights.

The recommendation model at hand considers different biases or effects in the data, i.e. more specifically the training data. Each bias or effect is constructed such that it captures the difference or "distortion" of an effect in relation to the overall average rating (called *avg*). This difference (= bias) is averaged and penalized with lambda to avoid overfitting. For example, the *Movie Effect* is capturing the averaged and penalized difference of ratings belonging to a certain movie rating in relation to the overall average *avg*. Every additional effect is building on top of that, i.e. for example the subsequent *User Effect* is capturing the averaged and penalized difference of a certain user and his ratings in relation to the overall average *avg* and the *Movie Effect*. This leads to the insight that the order of effects that we use to construct our recommendation model has an impact on the model outcome and performance. The earlier an effect is in the sequence, the more weighty it is by design. I have chosen this order since it produced the best results. Hence, the model has the following effects:

- **Movie Effect:** This parameter captures the effect that a certain movie has on the rating, while adjusting for the overall average rating and regularizing it. Assumption: The movie itself has an effect on the rating.
- **User Effect:** This parameter captures the effect that a certain user has on the rating, while adjusting for the overall average rating and the movie effect and regularizing it. Assumption: The user's behavior and taste has an effect on the rating. We have filtered the *edx* data set in the *Data Preparation* section to users with at least 15 ratings. The assumption is that users who rate more often have more experience / more data points and thus enable more reliable insights into their behavior.
- **Time Effect:** This parameter captures the effect of the date (time) on the rating, i.e. the time when a rating was made, while adjusting for the overall average, the movie effect, the user effect and regularizing it. Assumption: The date when a rating was made has an effect on the rating.
- **Genre Effect:** This parameter captures the effect the genre of the movie has on the rating, while adjusting for the movie, user, and time effect and regularizing it. Assumption: The genre that a movie is belonging to has an effect on the rating.

The code below builds the model and conducts a search for the optimal penalty parameter lambda based on the train and test set. Predictions are calculated for the test set as the sum of *avg*, *Movie Effect*, *User Effect*, *Time Effect* and *Genre*

Effect for each row in the test set. Please note that we generate an additional feature called *movie_avg* that represents the specific movie average in contrast to the overall average *avg*. The movie average is used in the prediction process to fill NAs or missing predictions with a finer measure than the overall average. Due to our applied filtering in the *Data Preparation* section (only users with more than 15 ratings) and selected biases (especially time and genre), some NAs or missing values are present in the predictions since there is no information available for all movie, user, genre, time combinations. Hence, we replace those NAs with either 1. the movie average (more precise) or 2. with the overall average in case the movie average is not available too. Additionally, we observe that computed predictions are numerical and can therefore be out of the "allowed" rating range from 0.5 to 5.0 (see *1.2 Data Set* section). To avoid penalties in the RMSE, we trim these values to the nearest allowed rating.

Finally, each iteration of a lambda is evaluated using the RMSE.

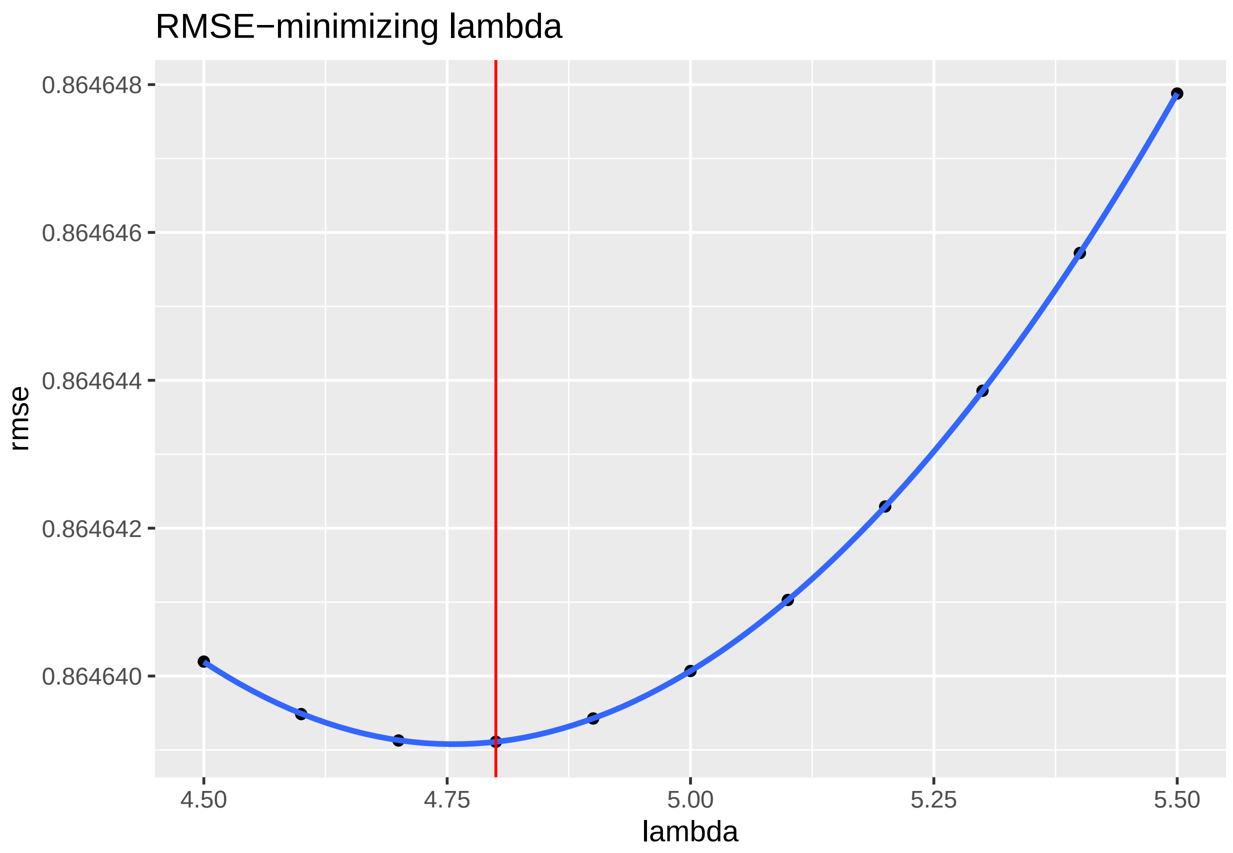
```r
lambdas <- seq(4.5, 5.5, 0.1)

RMSE <- function(true, predicted){
  sqrt(mean((true - predicted)^2, na.rm = TRUE))
}

rmses <- sapply(lambdas, function(l){
  avg <- mean(trainSet$rating)
  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))
  b_movie <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg)/(n()+l))
  b_user <- trainSet %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - avg - b_movie)/(n()+l))
  b_time <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    group_by(date) %>%
    summarize(b_time = sum(rating - avg - b_movie - b_user)/(n()+l))
  b_genre <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    left_join(b_time, by='date') %>%
    group_by(genres) %>%
    summarize(b_genre = sum(rating - avg - b_movie - b_user - b_time)/(n()+l))
  predicted_ratings <- testSet %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_movie, by = "movieId") %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_time, by = "date") %>%
    left_join(b_genre, by = "genres") %>%
    mutate(pred = avg + b_movie + b_user + b_time + b_genre) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred
  return(RMSE(predicted_ratings, testSet$rating))
})
```

The following figure shows the result of the parameter search:

```
optResults <- data.frame(lambda = lambdas, rmse = rmses)
print(optResults %>% ggplot(aes(lambda, rmse)) +
  ggtitle("RMSE-minimizing lambda") +
  geom_point() +
  geom_smooth() +
  geom_vline(xintercept = lambdas[which.min(rmses)], color = "red"))
```



The optimal lambda or penalty parameter is the value that minimizes the RMSE. In this case it is 4.8.

```
lambda <- lambdas[which.min(rmses)]

print(lambda)
```

```
## [1] 4.8
```

We can use this insight for the subsequent model evaluation.

2.4 Model Evaluation

From the previous section, we have identified the optimal penalty parameter lambda to be **4.8**. We can now re-run the recommendation model one more time with the optimal parameter to evaluate our model with the training and test data.

```r
trainingRmse <- sapply(lambda, function(l){
  avg <- mean(trainSet$rating)
  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))
  b_movie <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg)/(n()+l))
  b_user <- trainSet %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - avg - b_movie)/(n()+l))
  b_time <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    group_by(date) %>%
    summarize(b_time = sum(rating - avg - b_movie - b_user)/(n()+l))
  b_genre <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    left_join(b_time, by='date') %>%
    group_by(genres) %>%
    summarize(b_genre = sum(rating - avg - b_movie - b_user - b_time)/(n()+l))
  predicted_ratings <- testSet %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_movie, by = "movieId") %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_time, by = "date") %>%
    left_join(b_genre, by = "genres") %>%
    mutate(pred = avg + b_movie + b_user + b_time + b_genre) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred
  return(RMSE(predicted_ratings, testSet$rating))
})

print(trainingRmse)
```

```
## [1] 0.8646391
```

We can observe that the recommendation model produces a **training RMSE** of **0.8646391** with the optimal penalty parameter based on the train and test set. Please note that this is not the final RMSE using the validation set. The final evaluation of the recommendation model will be conducted in the next section.

3. Results

This section will perform the final evaluation of the recommendation model. Subsequently, the model results will be presented and discussed. To evaluate the model results, we will use the RMSE as before.

The following code performs the final evaluation of the recommendation model using the final hold-out validation set.

```r
finalRmse <- sapply(lambda, function(l){
  avg <- mean(trainSet$rating)
  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))
  b_movie <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - avg)/(n()+l))
  b_user <- trainSet %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - avg - b_movie)/(n()+l))
  b_time <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    group_by(date) %>%
    summarize(b_time = sum(rating - avg - b_movie - b_user)/(n()+l))
  b_genre <- trainSet %>%
    left_join(b_movie, by='movieId') %>%
    left_join(b_user, by='userId') %>%
    left_join(b_time, by='date') %>%
    group_by(genres) %>%
    summarize(b_genre = sum(rating - avg - b_movie - b_user - b_time)/(n()+l))
  predicted_ratings <- validation %>%
    mutate(date = as_datetime(timestamp), date = round_date(date, unit = "day")) %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_movie, by = "movieId") %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_time, by = "date") %>%
    left_join(b_genre, by = "genres") %>%
    mutate(pred = avg + b_movie + b_user + b_time + b_genre) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred
  return(RMSE(predicted_ratings, validation$rating))
})
```

```r
print(finalRmse)
```

```
## [1] 0.8646854
```

We receive a **final RMSE** of **0.8646854** on the validation set. We can see that the final RMSE has slightly worsened in comparison to the RMSE when using the test data set in the previous section. This was expected since the search for the optimal parameter lambda was conducted on the test set and therefore performed slightly worse on the validation set. This might be a minor overfitting issue. However, the difference is marginal.

4. Conclusion

The report at hand described the key steps and results of my *Capstone Project*. Initially, the *MovieLens Project* and the used data set have been explained and background information was provided. Following that, the goal and the key steps of the *Capstone Project* have been described. Afterwards, the modeling process as well as the approach for building and evaluating the recommendation model have been explained and a reasoning for the chosen design has been given. Finally, the results have been evaluated, presented and discussed. The recommendation model has a final RMSE of 0.8646854 on the validation set.

The limitations of this project are that we have used a restricted *MovieLens* data set (10m) instead of the full-scope data set. The full-scope set might have revealed further effects or insights into the data. Another limitation - even though we used a reduced data set - is the limitation that is put by the limited computing power of a regular notebook or desktop pc. In my case, e.g. it did not allow for the application of more advanced and computationally intensive modeling techniques in a reasonable time, e.g. using ensemble methods with Random Forests, AdaBoost or GamLoess. Future work can focus on the full-scope *MovieLens* data set, and even join additional publicly available data to the existing data to gather further insights. In addition, the usage of cloud services, e.g. *Microsoft Azure*, *Google Cloud Platform* or *Amazon AWS* could enable the application of more advanced and computationally intensive models in a reasonable time frame.