Prediction of Movie Ratings on the MovieLens 10M Data Set

Eric Gabriel

December 10th, 2020

Contents

Abstract

In this report, we performed prediction of movie ratings based on the MovieLens 10M data set. After an exploratory data analysis (EDA), we decided use two different classes of models for obtaining the predictions. First, we created a regularized linear regression model that accounts for all effects reported in the EDA. Second, we trained an approach based on matrix factorization. Evaluating these two model classes on the test set by means of the residual mean squared error (RMSE), we report a RMSE of 0.8646 and 0.7827 for linear regression and matrix factorization, respectively. We conclude, that the matrix factorization model clearly outperforms the linear regression models and provide an outlook for future work.

Introduction

This report is part of the first project submission of the course **HarvardX PH125.9x "Data Science: Capstone"**.

The goal of this report is to generate and compare models that are able to predict users' movie ratings. The generation of predictions in this context is also often referred to as collaborative filtering (CF). These predictions could be used, e.g., for a movie recommendation system. To do so, several models are trained based on the same fixed training data set. Afterwards, their performance is evaluated and compared on a validation data set.

Problem Statement

The problem to be solved is the prediction of movie ratings on a given data set (MovieLens 10M) that will be tackled using different models that operate on a certain set of input features. More specifically, the problem at hand is a regression task with supervised learning.

MovieLens Data Set

For training and evaluating the generated models, the 10M version of the MovieLens data set (GroupLens 2009) is used. A script provided in the HarvardX PH125.9x course is used to retrieve the data set and to split the available ratings into a training set and a hold-out test data set. For comparability, the provided script ensures the same training/test split for all students.

Training Data

The training data set consists of 9,000,055 observations of 6 variables. The variables or features are:

- **userId**: integer
- **movieId**: numeric
- **rating**: numeric
- **timestamp**: integer
- **title**: character
- **genres**: character

```r
# Summarize the training data set
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
# Print the first observations of the training data set
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

Each observation corresponds to a **rating** that has been given to a movie (**movieId**, with corresponding **title** and **genres**, where multiple genres are separated by a vertical bar) by a user (**userId**) at a certain **timestamp**.

Test Data

The hold-out test data set consists of 999,999 observations of the same variables as in the training data set.

```r
# Summarize the hold-out test data set
summary(validation)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
# Print the first observations of the test data set
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                    title
## 1:                                  Dumb & Dumber (1994)
## 2:                                  Jurassic Park (1993)
## 3:                                     Home Alone (1990)
## 4:                                        Rob Roy (1995)
## 5:                                 Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                    genres
## 1:                                 Comedy
## 2:       Action|Adventure|Sci-Fi|Thriller
## 3:                        Children|Comedy
## 4:               Action|Drama|Romance|War
## 5:                            Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

This test set will be used for the evaluation of the prediction models by comparing the generated predictions $\hat{y}$ and the actual ratings y.

Exploratory Data Analysis

In this section, an exploratory data analysis is conducted in order to gain further insights about the MovieLens training data set. These insights and characteristics of the data set will then be used to choose the appropriate data pre-processing steps as well as models that are most suitable for the problem at hand.

Unique Movies and Users

```r
# Find the number of unique movies
length(unique(edx$movieId))
```

```
## [1] 10677
```

```r
# Find the number of unique users
length(unique(edx$userId))
```

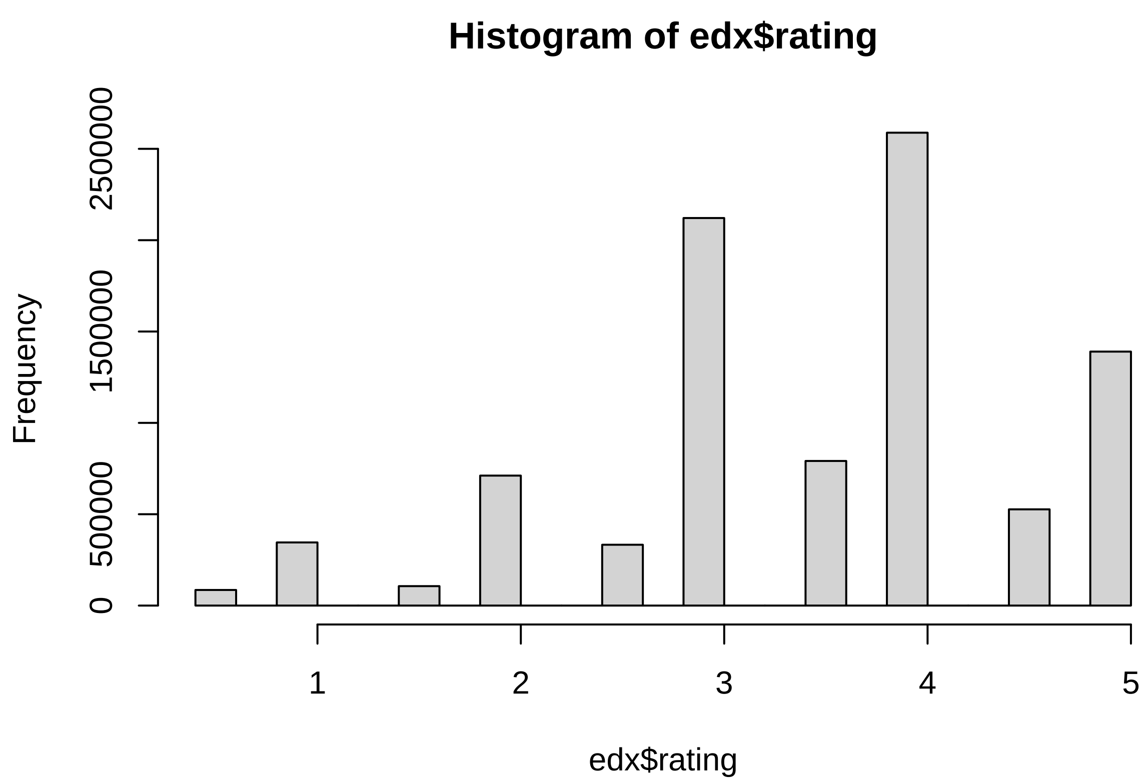
```
## [1] 69878
```

The training data set consists of ratings that were given to **10,677** unique movies by **69,878** unique users.

Ratings

As presented in the training summary, the mean movie rating is **3.512** (Median: 4). The following plots shows the histogram of given ratings in the training data set.

```r
# Plot the histogram of all available ratings in the training set
hist(edx$rating)
```

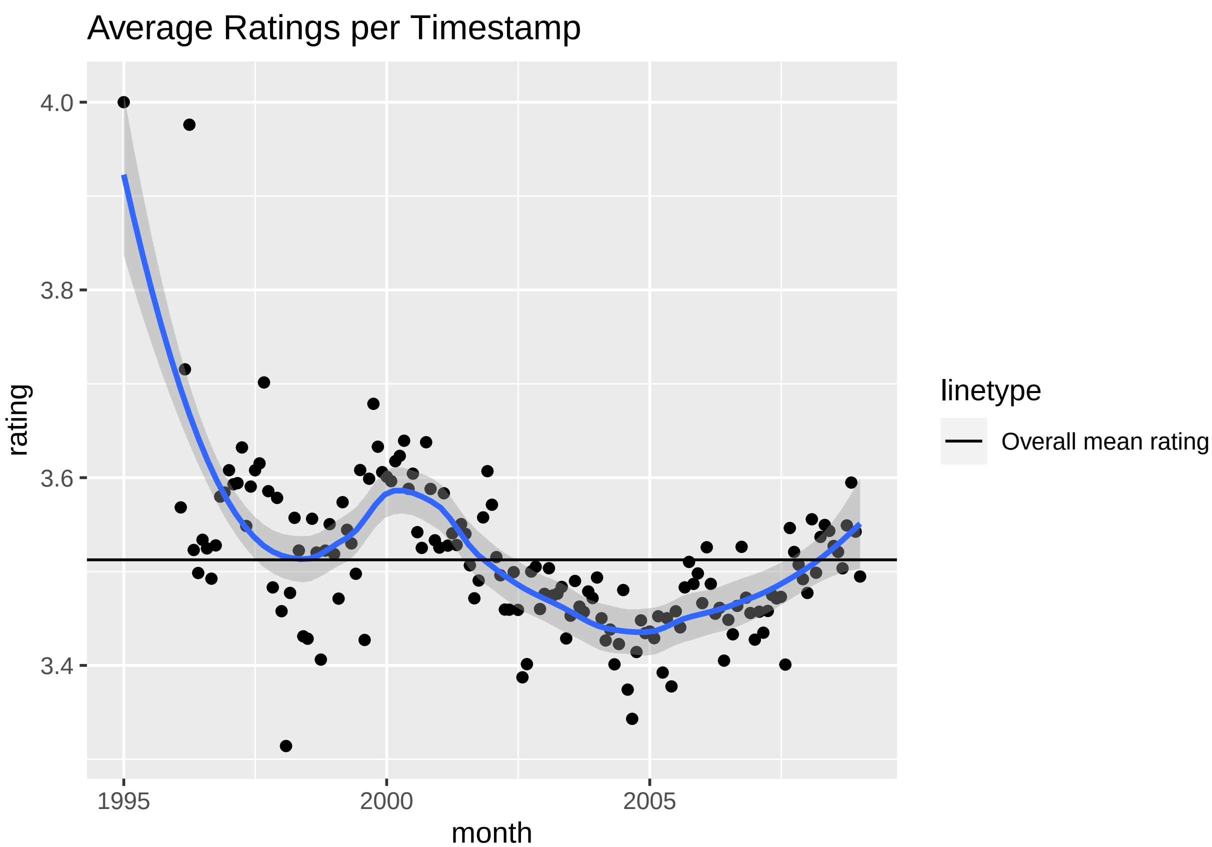


The histogram shows that ratings are given ranging from a minimum of 0.5 to a maximum of 5.0 in steps of 0.5.

Timestamps

In this section, we will evaluate if a time effect on the movie ratings can be determined from the available training data. To do so, we will use the respective month as rounding unit for the timestamp.

```
# Plot the average rating of each timestamp rounded to the nearest month against
# all available months, plot the loess regression curve and compare to the
# overall mean rating
edx %>%
  mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
  group_by(month) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(month, rating)) +
  geom_point() +
  geom_smooth(method="loess", span = 0.4) +
  geom_hline(aes(yintercept = mean_train, linetype = "Overall mean rating")) +
  labs(title = "Average Ratings per Timestamp")
```



From this plot, we can observe that the timestamp might have an impact on the movie rating.

Genres

```
# Find and display all unique genres
genre_names <- unique(unlist(str_split(edx$genres, "\\|")))
genre_names
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

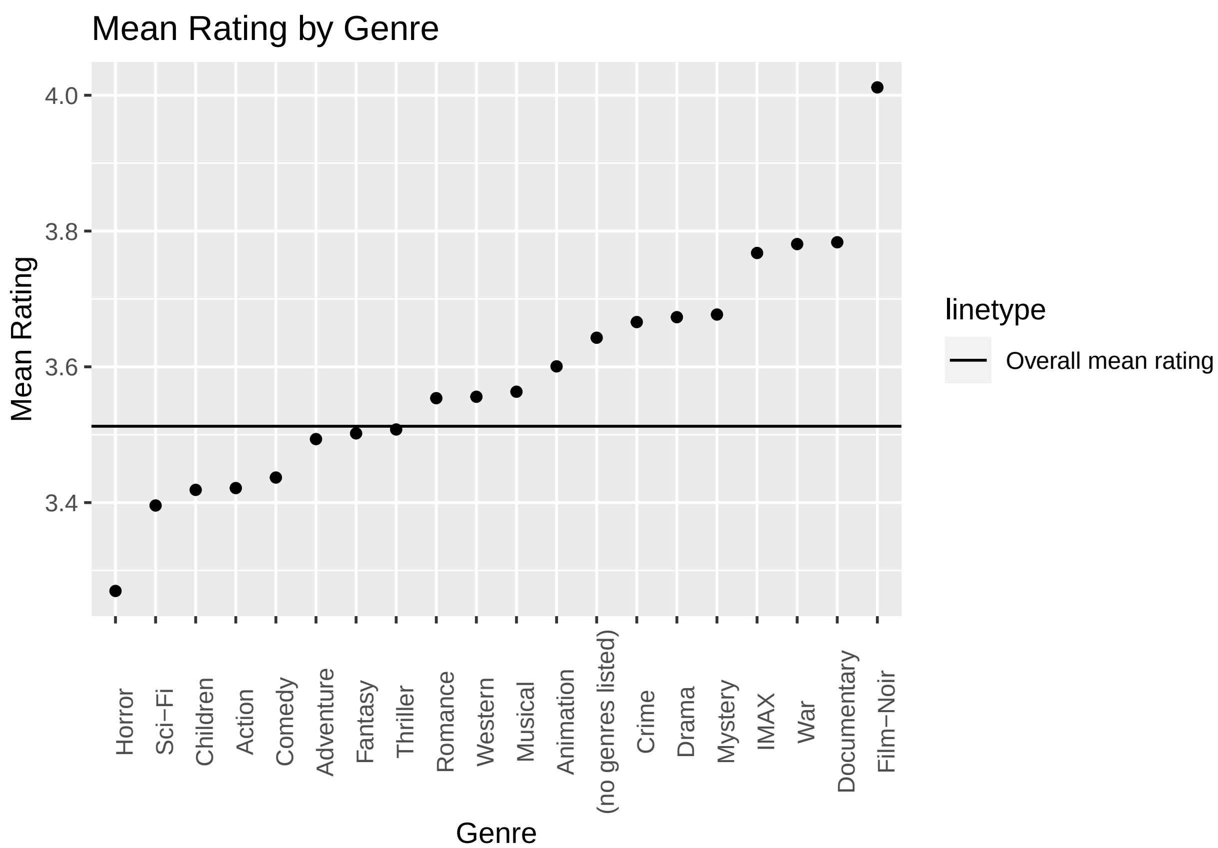
The training data set contains 20 unique genres. As seen in the training data section, there is often more than one genre that is assigned to a movie.

```
# Compute the mean number of individual genres per rating
edx %>%
  mutate(number_of_genres = str_count(genres, "\\|") + 1) %>%
  summarise(mean_number_of_genres = mean(number_of_genres))
```

```
##   mean_number_of_genres
## 1              2.596809
```

On average, a rating of any movie has **2.6** corresponding genres. The following plot shows the average rating of each individual genre compared to the overall average rating plotted as a horizontal line:

```
# Compute the mean rating for each individual genre
genre_means <- sapply(genre_names, function(g){
  edx %>%
    filter(str_detect(genres, g)) %>%
    summarise(mean_rating = mean(rating)) %>%
    select(mean_rating)
})

# Create a data frame containing the genre names and mean ratings
genre_details_df <- as.data.frame(genre_names)
genre_details_df$genre_mean <- as.numeric(genre_means)

# Plot the mean rating of the individual genres in ascending order
genre_details_df %>%
  ggplot(aes(x = reorder(genre_names, genre_mean), y = genre_mean)) +
  geom_point() +
  geom_hline(aes(yintercept = mean_train, linetype = "Overall mean rating")) +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(title = "Mean Rating by Genre", x = "Genre", y = "Mean Rating")
```



We can see that there are clear differences in the average rating based on the individual genres. Film-Noir and Documentary are the top genres, whereas Horror and Sci-Fi movies have the worst average rating.

```r
# Summarize the number of ratings per individual genre in the training set
genre_numbers <- sapply(genre_names, function(g){
  edx %>%
    filter(str_detect(genres, g)) %>%
    summarise(total = n()) %>%
    pull(total)
})
# Show number of rating per genre
genre_numbers
```

```
##             Comedy            Romance             Action              Crime
##            3540930            1712100            2560545            1327715
##           Thriller              Drama             Sci-Fi          Adventure
##            2325899            3910127            1341183            1908892
##           Children            Fantasy                War          Animation
##             737994             925637             511147             467168
##            Musical            Western            Mystery          Film-Noir
##             433080             189394             568332             118541
##             Horror        Documentary               IMAX (no genres listed)
##             691485              93066               8181                  7
```

From these numbers of ratings, we see that most genres in the training set have smaple sizes that are large enough. Only *(no genres listed)* has only seven ratings.

Pre-Processing

The EDA shows that the training and test data split of the MovieLens 10M data set is already clean, e.g., no nans are in the data and it is already in a tidy format. Thus, no further pre-processing is required.

Methods

Based on the exploratory data analysis, we will use two different classes of models for the prediction of user ratings. The first class of models is based on linear regression, where we will start with a simple regression model and then account for different effects observed in the exploratory data analysis. The second model is a recommendation system based on matrix factorization.

Evaluation Metric

For the final evaluation of the predictions, we will use the residual mean squared error (RMSE) using the prediction $\hat{y}_{u,i}$ and the actual rating $y_{u,i}$ for all movies i and users u in the test set:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})} \tag{1}$$

Regression Models

Mean Rating

As the simplest regression model, we calculate the overall mean rating of the training data set and will use this as predictions for the test set. This approach can be written as:

$$Y_{u,i} = \mu_{train} + \epsilon_{u,i} \tag{2}$$

where $Y_{u,i}$ denotes the rating of movie i by user u, μ_{train} denotes the average rating of the training set, and $\epsilon_{u,i}$ is an independent error for movie i and user u that is sampled from the same distribution with zero mean.

```
# Get mean rating of training data
mean_train <- mean(edx$rating)

# Compute RMSE of y_hat = mean_training
compute_rmse(validation$rating, mean_train)
```

```
## [1] 1.061202
```

Movie Effect

Next, we will account for the movie effect by adding a bias term b_i for each movie i:

$$Y_{u,i} = \mu_{train} + b_i + \epsilon_{u,i} \tag{3}$$

The bias estimates $\hat{b}_i$ for all movies can be computed by minimizing the least squares as follows

$$\hat{b}_i = Y_{u,i} - \mu_{train} \tag{4}$$

```
# Compute estimates of the movie bias on the training set
movie_effect <- edx %>%
  group_by(movieId) %>%
```

```r
  summarize(b_movie = mean(rating - mean_train))

# Add computed movie bias estimates to the test set and compute y_hat
# according to: Y_{u,i} = \mu_{train} + b_i
y_hat_model2 <- validation %>%
  left_join(movie_effect, by='movieId') %>%
  mutate(y_hat_model2 = mean_train + b_movie) %>%
  .$y_hat_model2

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(validation$rating, y_hat_model2)
```

```
## [1] 0.9439087
```

User Effect

The user effect is incorporated by adding another bias term b_u for each user u:

$$Y_{u,i} = \mu_{train} + b_i + b_u + \epsilon_{u,i} \tag{5}$$

The bias estimates $\hat{b}_u$ for all users can be computed as follows

$$\hat{b}_u = Y_{u,i} - \mu_{train} - \hat{b}_i \tag{6}$$

```r
# Compute estimates of the user bias on the training set
user_effect <- edx %>%
  left_join(movie_effect, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mean_train - b_movie))

# Add computed movie and user bias estimates to the test set and compute y_hat
# according to: Y_{u,i} = \mu_{train} + b_i + b_u
y_hat_model3 <- validation %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  mutate(y_hat_model3 = mean_train + b_movie + b_user) %>%
  .$y_hat_model3

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(validation$rating, y_hat_model3)
```

```
## [1] 0.8653488
```

Time Effect

In this model, we will account for the time effect by adding the bias term b_t for each month:

$$Y_{u,i} = \mu_{train} + b_i + b_u + b_t + \epsilon_{u,i} \tag{7}$$

The bias estimates for all months $\hat{b}_t$ can be computed as follows after having rounded the timestamp of the rating to the respective month:

$$\hat{b}_t = Y_{u,i} - \mu_{train} - \hat{b}_i - \hat{b}_u \tag{8}$$

```r
# Compute estimates of the time bias on the training set
time_effect <- edx %>%
  mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  group_by(month) %>%
  summarize(b_month = mean(rating - mean_train - b_movie - b_user))

# Add computed movie, user and time bias estimates to the test set and compute
# y_hat according to: Y_{u,i} = \mu_{train} + b_i + b_u + b_t
y_hat_model4 <- validation %>%
  mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  left_join(time_effect, by='month') %>%
  mutate(y_hat_model4 = mean_train + b_movie + b_user + b_month) %>%
  .$y_hat_model4

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(validation$rating, y_hat_model4)
```

```
## [1] 0.8653153
```

Genre Effect

In this model, the genre effect is incorporated by adding a bias terms b_g for each genre that is assigned to movie i:

$$Y_{u,i} = \mu_{train} + b_i + b_u + b_t + \sum_{g \forall g_i} b_g + \epsilon_{u,i} \tag{9}$$

The bias estimates $\hat{b}_g$ for all individual genres can be computed as follows. First, we determine the set of individual genres G. Then, for each $g \in G$, we estimate the bias for genre g using:

$$\hat{b}_g = Y_{u,i} - \mu_{train} - \hat{b}_i - \hat{b}_u - \hat{b}_t \tag{10}$$

```r
# Get unique list of genre names
genre_names <- unique(unlist(str_split(edx$genres, "\\|")))

# Compute effect per individual genre
genre_effects <- sapply(genre_names, function(g){
  edx %>%
    filter(str_detect(genres, g)) %>%
    mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
    left_join(movie_effect, by='movieId') %>% # Add movie effect by movieId
    left_join(user_effect, by='userId') %>% # Add user effect by userId
    left_join(time_effect, by='month') %>% # Add time effect by month
    summarize(b_genre = mean(rating - b_movie - mean_train - b_user - b_month)) %>%
    pull(b_genre)
})

# Compute sum of genre effects (of all assigned genres, respectively) for all
# rows in the validation set
```

```r
genre_effect_sum <- sapply(seq(1:nrow(validation)), function(i){
  sum(genre_effects[str_split(validation[i,]$genres, "\\|", simplify = TRUE)])
})

# Create extended validation set with b_genre
validation_extended <- validation
validation_extended$b_genre <- genre_effect_sum

# Add computed movie, user, time and genre bias estimates to the test set and
# compute y_hat according to: Y_{u,i} = \mu_{train} + b_i + b_u + b_t + b_g
y_hat_model5 <- validation_extended %>%
  mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  left_join(time_effect, by = "month") %>%
  mutate(prediction = mean_train + b_movie + b_user + b_month + b_genre) %>%
  .$prediction

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(y_hat_model5, validation$rating)
```

```
## Movie, user, time and genre effect
##                          0.8652335
```

Regularization

To further improve the predictions, we will use regularization for the movie and user effects. This technique adds a penalty term that penalizes large estimates of b_i and b_u that occur due to small sample sizes, i.e., a very small number of ratings given to a movie or by a user, respectively. Small sample sizes can lead to large estimates of b_i and b_u, but in fact these estimates are just noisy. Using regularization, large estimates of b_i and b_u coming from small sample sizes are shrunken towards 0 w.r.t. the factor λ.

Instead of minimizing the least squares only, a penalty term is added as follows:

$$\sum_{u,i}(y_{u,i} - \mu_{train} - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2) \tag{11}$$

The regularized bias estimates $\hat{b}_i$ are computed using:

$$\hat{b}_i(\lambda) = \frac{1}{n_i + \lambda}\sum_{u=1}^{n_i}(Y_{u,i} - \mu_{train}) \tag{12}$$

where n_i is the number of movies.

The regularized bias estimates $\hat{b}_u$ are computed using:

$$\hat{b}_u(\lambda) = \frac{1}{n_u + \lambda}\sum_{i=1}^{n_u}(Y_{u,i} - \mu_{train} - \hat{b}_i) \tag{13}$$

where n_u is the number of users.

From these two equations we can see that bias estimates from large sample sizes, i.e., large n_i and n_u, are less affected by λ, whereas bias estimates from small sample sizes will be shrunken towards zero.

In order to select λ, we will use 10-fold cross-validation on the training set. We will use a maximum value of λ corresponding to the 10th percentile of the ratings per movie.

```
# Get number of ratings per movie
ratings_per_movie <- edx %>%
  group_by(movieId) %>%
  summarize(count = n())

# Print the 10th percentile of the number of ratings
quantile(ratings_per_movie$count, probs = 0.1)
```

```
## 10%
##  10
```

Thus, we will determine the best value for $\lambda \in [2, 10]$ w.r.t. 10-fold cross-validation on the training set:

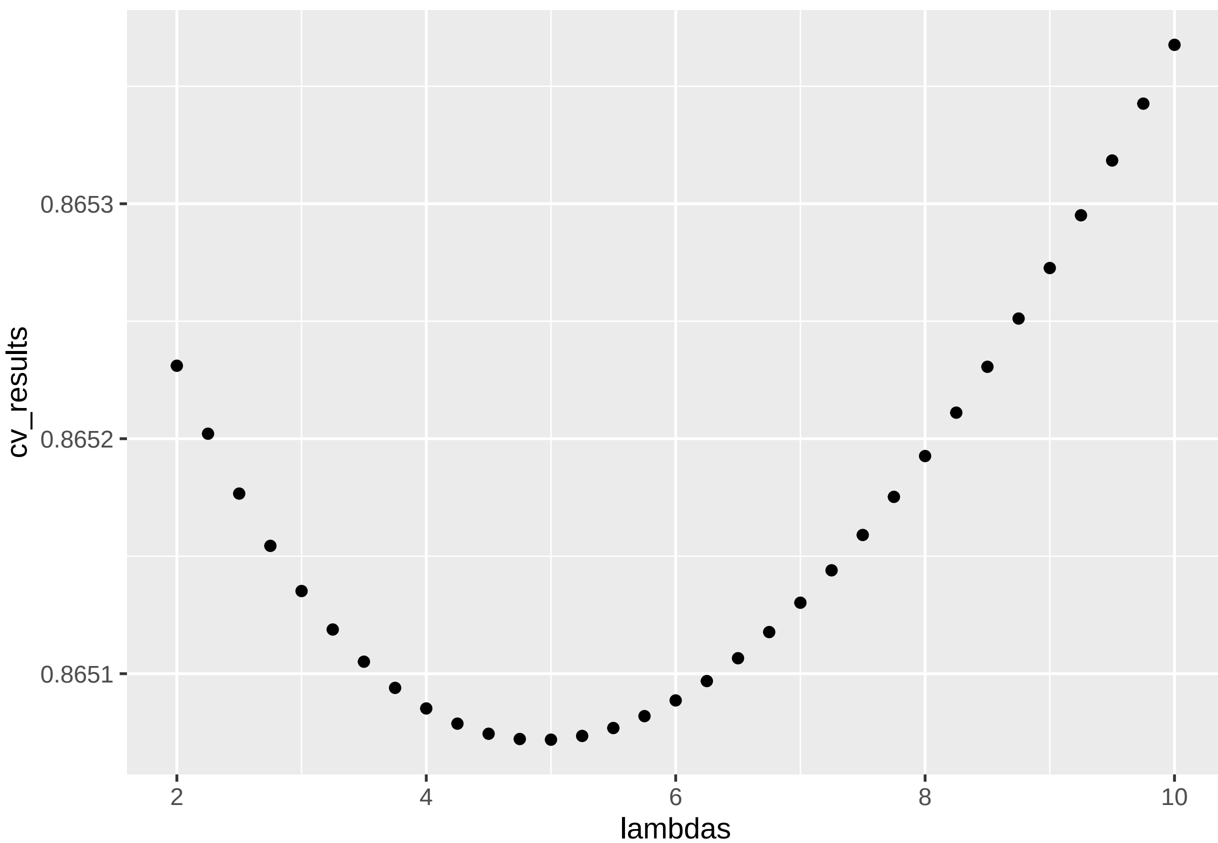
```
# Create the sequence of values for lambda
lambdas <- seq(2, 10, 0.25)

# Set the seed before creating training/validation data splits of the training
# data to get reproducible results (for versions of R > 3.5)
set.seed(2020, sample.kind="Rounding")

# Create 10 training/validation sets (90%/10%) from the training data for CV
cv_folds <- createDataPartition(edx$rating, times = 10, p = 0.9)

# For each lambda, find b_i & b_u, followed by prediction & testing
# ATTENTION: the code below takes some time
cv_results <- sapply(lambdas, function(l){

  rmse_results <- sapply(cv_folds, function(train_indices){
    # Fill training and validation data sets from the training data for current
    # cross-validation fold
    cv_train <- edx[train_indices,]
    temp <- edx[-train_indices,]

    # Make sure userId and movieId in validation set are also in training set
    cv_test <- temp %>%
    semi_join(cv_train, by = "movieId") %>%
    semi_join(cv_train, by = "userId")

    # Add rows removed from validation set back into training set
    removed <- anti_join(temp, cv_test)
    cv_train <- rbind(cv_train, removed)

    # Compute mean rating of the training set of the current CV fold
    mean_train <- mean(cv_train$rating)

    # Estimate b_i with regularization using current lambda
    b_movie <- cv_train %>%
      group_by(movieId) %>%
      summarize(b_movie = sum(rating - mean_train)/(n()+l))

    # Estimate b_u with regularization using current lambda
    b_user <- cv_train %>%
      left_join(b_movie, by="movieId") %>%
```

```
      group_by(userId) %>%
      summarize(b_user = sum(rating - b_movie - mean_train)/(n()+l))

    # Compute y_hat of validation set of current CV fold
    y_hat <-
      cv_test %>%
      left_join(b_movie, by = "movieId") %>%
      left_join(b_user, by = "userId") %>%
      mutate(prediction = mean_train + b_movie + b_user) %>%
      .$prediction

    # Compute the RMSE between y_hat and actual ratings
    compute_rmse(y_hat, cv_test$rating)
    })

  # Return mean RMSE of the 10 CV folds for current lambda
  mean(rmse_results)

})
```

```
# Select best lambda based on minimum RMSE
best_lambda <- lambdas[which.min(cv_results)]
best_lambda
```

```
## [1] 5
```

```
# Plot mean RMSE of 10-fold CV for each value of lambda
qplot(lambdas, cv_results)
```



Based on the RMSE results of the 10-fold cross-validation on the training set, we will recompute the bias estimates $\hat{b}_i$ and $\hat{b}_u$ with λ set to 5, and then report the RMSE of the complete model:

```
# Recompute estimates of the movie bias on the training set using best lambda
b_movie <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = sum(rating - mean_train)/(n()+best_lambda))

# Recompute estimates of the user bias on the training set using best lambda
b_user <- edx %>%
  left_join(b_movie, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_user = sum(rating - b_movie - mean_train)/(n()+best_lambda))

# Add (re)computed movie, user, time and genre bias estimates to the test set
# and compute y_hat according to: Y_{u,i} = \mu_{train} + b_i + b_u + b_t + b_g
y_hat_model6 <-
  validation_extended %>%
  mutate(month = round_date(as_datetime(timestamp), unit = "month")) %>%
  left_join(b_movie, by = "movieId") %>%
  left_join(b_user, by = "userId") %>%
  left_join(time_effect, by = "month") %>%
  mutate(prediction = mean_train + b_movie + b_user + b_month + b_genre) %>%
  .$prediction

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(y_hat_model6, validation$rating)
```

```r
rmses["Movie, user, time and genre effect with regularization"]
```

```
## Movie, user, time and genre effect with regularization
##                                                0.8647274
```

Referring to the exploratory data analysis about the ratings, we observed that the minimum rating is 0.5 and the maximum rating is 5.0.

```r
# Get the number of predicted ratings that are out of the target range
sum(y_hat_model6 < 0.5 | y_hat_model6 > 5)
```

```
## [1] 1682
```

From this output, we see that 1682 predicted ratings are out of the target range. As the last improvement for the linear regression models, we will cut-off the ratings out of range.

```r
# Cut-off predictions outside of the target range
y_hat_model6_cutoff <- ifelse(y_hat_model6 < 0.5, 0.5, y_hat_model6)
y_hat_model6_cutoff <- ifelse(y_hat_model6_cutoff > 5, 5, y_hat_model6_cutoff)

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(y_hat_model6_cutoff, validation$rating)
```

```
## [1] 0.864616
```

Matrix Factorization

Matrix factorization (MF) is a well-known and effective approach for recommendation systems, e.g., (Koren, Bell, and Volinsky 2009). In this approach, the movie and user IDs (i and u, respectively) are used to span the rows and columns of a large matrix – the so-called rating matrix – and only those entries are filled for which a rating $r_{u,i}$ is available in the training data. Thus, the resulting matrix is very sparse.

Given this rating matrix R with m distinct users and n distinct movies, MF aims at finding two dense factor matrices P and Q – each in their own latent space that has k dimensions –, where $P \in \mathbb{R}^{k \times m}$ and $Q \in \mathbb{R}^{k \times n}$ such that $r_{u,i} \simeq p_u^T q_i$. Here, p_u denotes the u-th column of P and q_i denotes the i-th column of Q. Thus, p_u and q_i are both k-dimensional vectors.

Thus, each latent space consists of k latent factors (one per dimension) which cover abstract information or preferences about users (e.g., liking or disliking a certain genre or actor) and movies (e.g., blockbusters vs. art-house movies etc.), respectively.

The optimization problem of finding P and Q is usually done using stochastic gradient descent (SGD) (Koren, Bell, and Volinsky 2009).

To train a matrix factorization model on our training set, we use the R package *recosystem*, which provides an R wrapper for the LIBMF (Chin et al. 2015a)(Chin et al. 2015b)(Chin et al. 2016). This library provides a very efficient implementation of the matrix factorization approach similar to (Koren, Bell, and Volinsky 2009) as well as a fast and parallelized version of SGD for training. To select the best training options, we perform a grid search on k as well as the learning rate using the *tune* function.

```
# Create Reco model
reco = Reco()

# Create training data from edx training set (from memory)
training_data <- data_memory(user_index = edx$userId,
                             item_index = edx$movieId,
                             rating = edx$rating)

# Find best options for training (ATTENTION: takes a while)
opts = reco$tune(training_data, opts = list(dim = seq(5, 30, 5),
                                            lrate = seq(0.05, 0.3, 0.05),
                                            costp_l1 = 0,
                                            costq_l1 = 0,
                                            nthread = 4,
                                            niter = 20))
# Save best training options
best_tune <- opts$min

# Train Reco model using the training data
reco$train(training_data, opts = c(best_tune, nthread = 1, niter = 20))
```

Having trained the model on the training set using the best tuning parameters, we can make predictions of the ratings on the hold-out test set:

```
# Create test data from hold-out test set (from memory)
test_data <- data_memory(user_index = validation$userId,
                         item_index = validation$movieId,
                         rating = validation$rating)

# Make predictions for the test set using the trained Reco model
predictions <- reco$predict(test_data = test_data, out_pred = out_memory())

# Compute RMSE of y = validation$rating and y_hat
```

```r
compute_rmse(validation$rating, predictions)
```

```
## Matrix Factorization
##            0.7829288
```

Finally, we can again cut-off predictions smaller than 0.5 or larger than 5 as for the linear regression models.

```r
# Cut-off predictions outside of the target range
predictions_cutoff <- ifelse(predictions < 0.5, 0.5, predictions)
predictions_cutoff <- ifelse(predictions_cutoff > 5, 5, predictions_cutoff)

# Compute RMSE of y = validation$rating and y_hat
compute_rmse(validation$rating, predictions_cutoff)
```

```
## Matrix Factorization with cutoff
##                        0.7826978
```

Results

In this section, we present an overview of the reported results. The following table contains all final RMSE results that were presented in the previous section:

	RMSE on Test Set
Mean Rating	1.0612018
Movie effect	0.9439087
Movie and user effect	0.8653488
Movie, user and time effect	0.8653153
Movie, user, time and genre effect	0.8652335
Movie, user, time and genre effect with regularization	0.8647274
Movie, user, time and genre effect with regularization and cutoff	0.8646160
Matrix Factorization	0.7829288
Matrix Factorization with cutoff	0.7826978

Discussion

Using only the mean rating of the training set for predictions, the RMSE is 1.0612. A large improvement can be seen when accounting for user- and movie-specific biases, where the RMSE is 0.8653. When including biases for the month of the rating as well as the assigned genres and when using regularization and a cut-off of predictions that were out of range, we were able to achieve a RMSE of 0.8646.

For the best grading regarding the RMSE, it was required to achieve a RMSE that is smaller that 0.8649. As reported in the RMSE result table, this is already fulfilled using linear regression with regularization.

As a second class of models, we used a matrix factorization approach similar to (Koren, Bell, and Volinsky 2009) using the R package *recosystem*. We performed a grid search to find the best training parameters using the *tune* function. This function performs a 5-fold cross validation on the training set using RMSE as default loss function. According to the best RMSE of 0.792 on the training set, we chose the number of dimensions of the latent space k to be 30 and the learning rate to be 0.1.

Having trained the model on the training set using the aforementioned training parameters, we achieved a RMSE of 0.7829 on the hold-out test set. This shows that the matrix factorization model clearly outperforms the linear regression models.

Comparing the best training RMSE 0.792 and the test RMSE of 0.7827, we still do not see the effect of over-fitting to the training data. Thus, it might be possible to achieve even better predictions on the test set, e.g., when using a larger k.

Conclusion

In this report, we have performed prediction of movie ratings on the MovieLens10M data set, which is also referred to as collaborative filtering (CF) and used in movie recommendation systems. We used two classes of models to obtain predictions: linear regression and matrix factorization.

Using a linear regression model that accounts for movie-, user-, time- and genre-specific biases and when using regularization, we were able to achieve a RMSE of 0.8646 on the test set when additionally cutting off predictions that were outside of the target range.

Using matrix factorization, we achieved a RMSE of 0.7827 on the test set which shows a much better performance than with linear regression.

Regarding future work, it might be useful to select a larger grid for searching a better configuration of the training parameters for the matrix factorization approach, e.g., a larger k. Additionally, one could examine the largest remaining errors for each model class and choose appropriate tuning parameters or other extensions to further improve the prediction performance.

References

Chin, W.-S., B.-W. Yuan, M.-Y. Yang, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. 2016. "LIBMF:A Library for Parallel Matrix Factorization in Shared-Memory Systems." *JMLR*.

Chin, W.-S., Y. Zhuang, Y.-C. Juan, and C.-J. Lin. 2015a. "A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems." *ACT TIST*.

———. 2015b. "A Learning-Rate Schedule for Stochastic Gradient Methods to Matrix Factorization." *PAKKD*.

GroupLens. 2009. "MovieLens 10M Dataset." 2009. http://grouplens.org/datasets/movielens/10m/.

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *Computer* 42: 261–63.