Report on the machine learning movie recommendation system

Marc Roca-Musach

December 5, 2022

Introduction

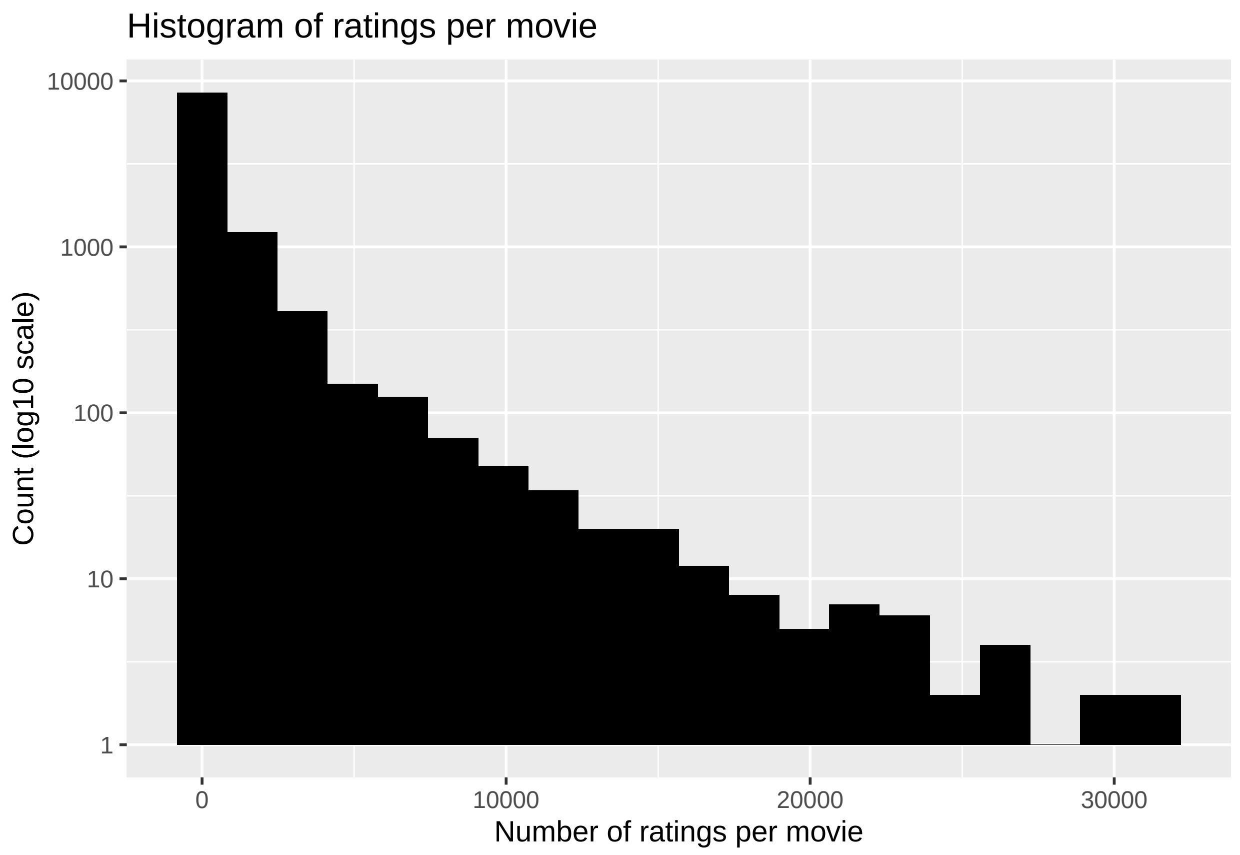
Online movie platforms become more and more popular due to the user's possibility to choose movies from a wide catalog at any time from anywhere. To simplify the browsing of the catalog, these platforms usually account for a recommendation system that proposes users some movies according to their preferences.

The aim of this report is to develop an algorithm for a recommendation system based the previous ratings of users. To achieve this goal, we account for some sample data classified by genres in the data-set MovieLens. This data-set was split into the `final_holdout_test` data-set and the `edx` data-set. The `final_holdout_test` contains 10% of the data, with 999999 rows, and will only be used at the very end to evaluate the performance of the proposed algorithm. The `edx` data-set, with 9000055 ratings, will be processed to generate the algorithm. Next figure shows some sample data from the first 6 rows of the `edx` data-set.

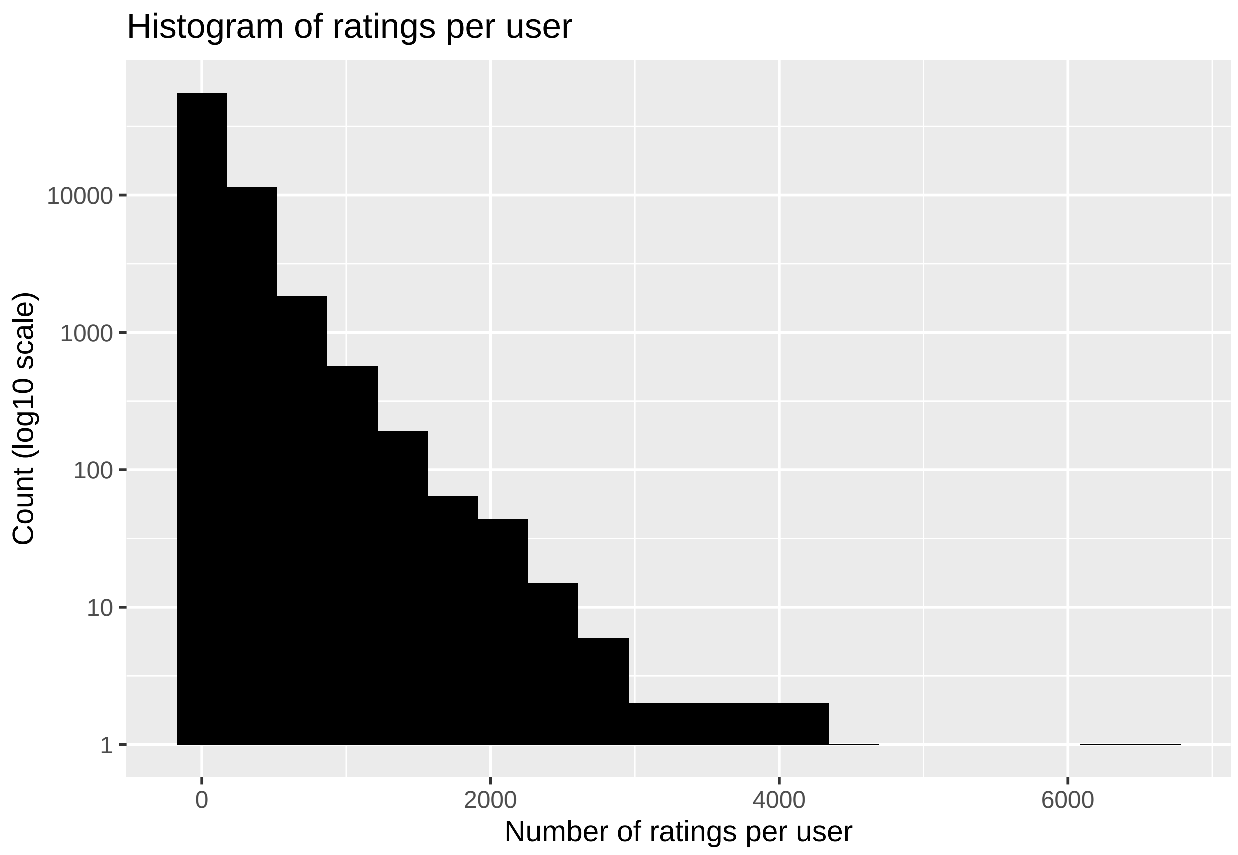
```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392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

The training data-set (called `edx`) contains 10677 movies, 69878 users, and 20 unique genres. The large size of the data-set makes using machine-learning algorithms impossible due to the lack of enough memory resources a regular computer. For this reason, the least square estimates are computed manually.

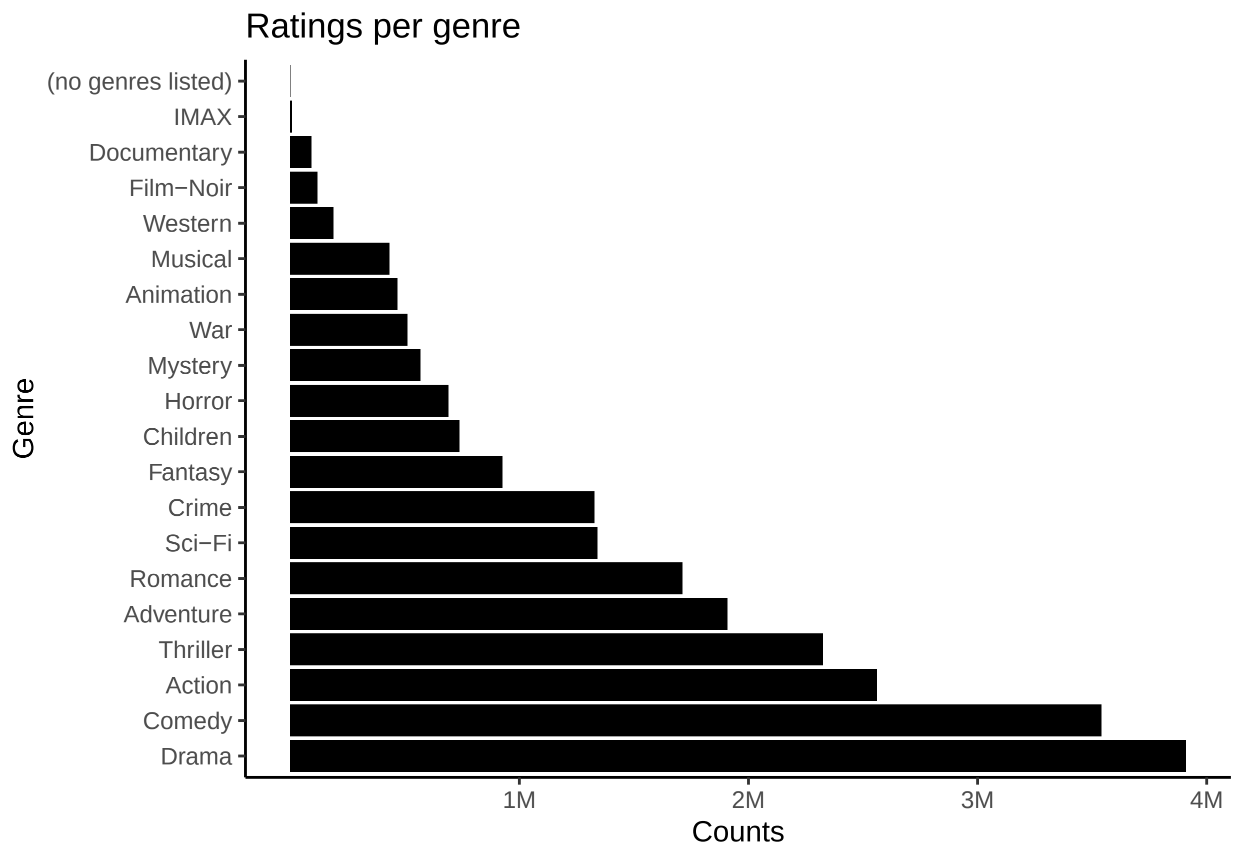
Inspecting the data, we can observe that some movies have more ratings than others.



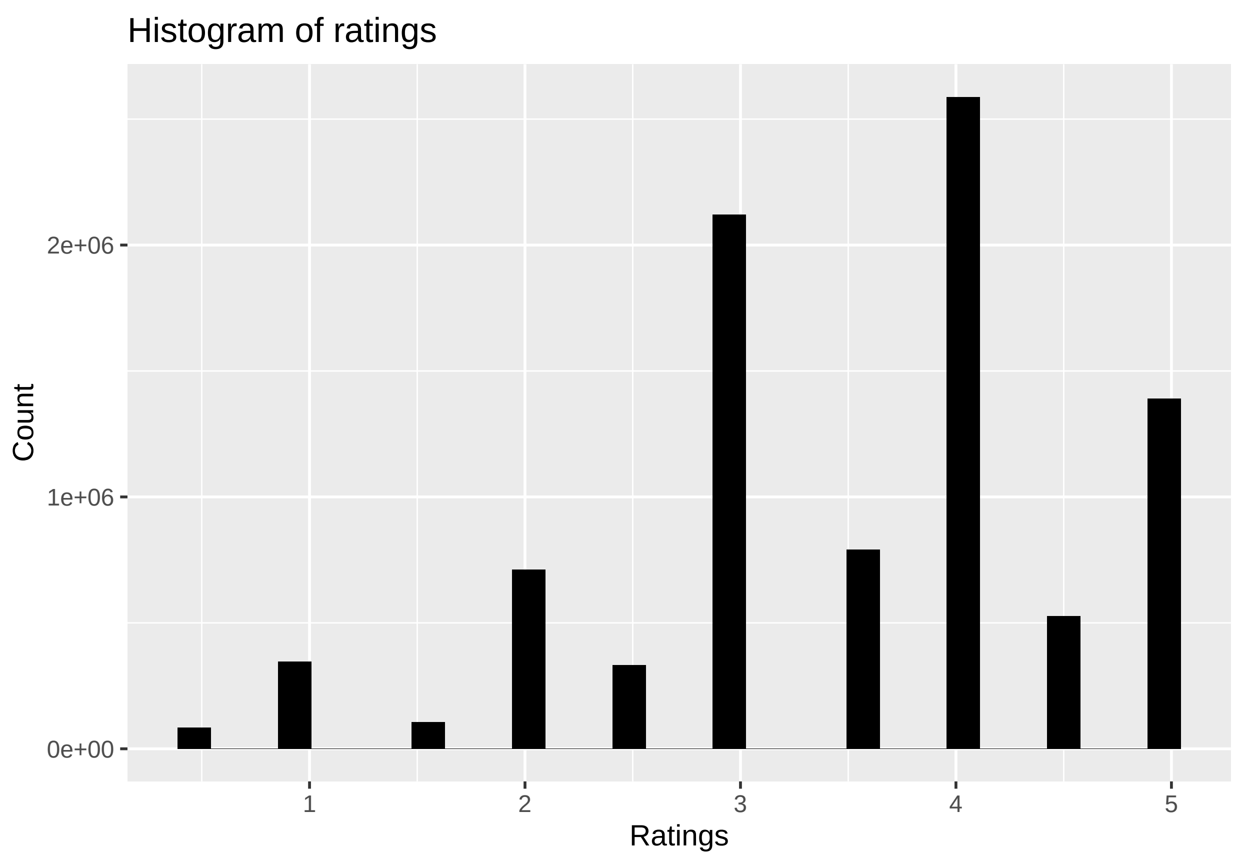
Also, some users tend to rate more films than others.



Moreover, some genres tend to be more rated than others.



To finish with, we can see that the ratings range between 0.5 and 5. For this reason, the algorithm is set up to replace all predicted values outside of this range to the nearest bound.



Finally, the RMSE will be calculated as an indicator of the accuracy of the proposed algorithm. The RMSE is calculated with the following formula:

$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$

Where

N is the number of movie-user combinations

$y_{u,i}$ is the rating for movie i by user u

$\hat{y}_{u,i}$ is the predicted rating for movie i and user u

Methods and analysis

The proposed algorithm takes the following considerations into account:

1. Some movies have higher ratings than others.
2. Some users tend to rate higher than others.
3. Some genres have higher ratings than others.
4. Movies with more ratings will obtain a better precision on the predicted rating.

All these considerations are processed below to obtain the final proposal for the algorithm.

0. Baseline case: Just the average

To begin with, we propose to evaluate a baseline where the rating is predicted regardless of the movie, user or genre. The estimate of the rating is the average of all ratings.

$Y_{u,i} = \mu + \epsilon_{u,i}$

It is processed with the following code:

```
mu_hat <- mean(train_set$rating)

# Calculate the RMSE for this method
mth_1 <- RMSE(final_holdout_test$rating, mu_hat)
```

In the data-set, the average of rating is $\mu = 3.512379$. With this simple case, the Root Mean Square Error (RMSE) is **1.0612018**.

1. The movie effect

When we assume that some movies have higher ratings than others, then the model must include the movie bias (b_i). We estimate this bias as the average of the ratings per movie.

$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$

It is processed with the following code:

```
mu_hat <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

predicted_ratings <- final_holdout_test %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu_hat + b_i) %>%
  .$pred

# Ensuring that the predicted ratings are not out of the bounds (0.5 <= pred <= 5)
predicted_ratings <- ifelse(predicted_ratings < 0.5,0.5,predicted_ratings)
predicted_ratings <- ifelse(predicted_ratings > 5,5,predicted_ratings)

# In case some rows doesn't have predicted ratings replace with the average
predicted_ratings[is.na(predicted_ratings)] <- mu_hat

# Calculate the RMSE for this method
mth_2 <- RMSE(final_holdout_test$rating, predicted_ratings)
```

Note that the predicted ratings are limited to the range between 0.5 and 5. Also, all the NA values are replaced with the average rating $\mu = 3.512379$. There are only few NA values that appear when a movie is at the `train_set`, but not at the `final_holdout_test` set.

Including this movie bias to the model, the Root Mean Square Error (RMSE) is **0.9439827**.

2. The user effect

When we assume that some users tend to rate higher than others, we can include in the model the user bias (b_i). We estimate this bias as the average of the ratings per user.

$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

It is processed with the following code:

```r
mu_hat <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

predicted_ratings <- final_holdout_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

# Ensuring that the predicted ratings are not out of the bounds (0.5 <= pred <= 5)
predicted_ratings <- ifelse(predicted_ratings < 0.5,0.5,predicted_ratings)
predicted_ratings <- ifelse(predicted_ratings > 5,5,predicted_ratings)

# In case some rows doesn't have predicted ratings replace with the average
predicted_ratings[is.na(predicted_ratings)] <- mu_hat

# Calculate the RMSE for this method
mth_3 <- RMSE(final_holdout_test$rating, predicted_ratings)
```

Including this user bias to the model, the Root Mean Square Error (RMSE) is **0.8655597**.

3. The genre effect

The genre of a movie also can affect its ratings. When we assume that some genres have higher ratings than others, we can include in the model the genre bias (b_g). In this model, the genre is considered the string provided by the MovieLens data-set, with no modifications. We estimate this bias as the average of the ratings per user.

$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$

It is processed with the following code:

```r
mu_hat <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

```r
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))

predicted_ratings <- final_holdout_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

# Ensuring that the predicted ratings are not out of the bounds (0.5 <= pred <= 5)
predicted_ratings <- ifelse(predicted_ratings < 0.5,0.5,predicted_ratings)
predicted_ratings <- ifelse(predicted_ratings > 5,5,predicted_ratings)

# In case some rows doesn't have predicted ratings replace with the average
predicted_ratings[is.na(predicted_ratings)] <- mu_hat

# Calculate the RMSE for this method
mth_4 <- RMSE(final_holdout_test$rating, predicted_ratings)
```

Including this genre bias to the model, the Root Mean Square Error (RMSE) is **0.8651437**.

4. Regularization

At this point, we want to introduce a variable to penalize those movies with fewer ratings, as the predicted rating for those movies will be less precise. We do so by calculating the regularized movie bias according to parameter lambda $\hat{b}_i(\lambda)$.

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

To verify the code for the regularization process, we firstly process the results using an arbitrary value of lambda $\lambda = 3$. This is the code for this process:

```r
lambda <- 3

mu_hat <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())

user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat)/(n()+lambda), n_i = n())
```

```r
genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu_hat)/(n()+lambda), n_i = n())

predicted_ratings <- final_holdout_test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

# Ensuring that the predicted ratings are not out of the bounds (0.5 <= pred <= 5)
predicted_ratings <- ifelse(predicted_ratings < 0.5,0.5,predicted_ratings)
predicted_ratings <- ifelse(predicted_ratings > 5,5,predicted_ratings)

# In case some rows doesn't have predicted ratings replace with the average
predicted_ratings[is.na(predicted_ratings)] <- mu_hat

# Calculate the RMSE for this method
mth_5 <- RMSE(final_holdout_test$rating, predicted_ratings)
```

After the regularization using the parameter $\lambda = 3$, the Root Mean Square Error (RMSE) is **0.8646999**.

Tuning the lambda To find the optimal value for lambda, we run the following tuning process that minimizes the value of the RMSE for the training data. Note that the tuning process is trained with the `train_set`, which is a portion of 90% of the `edx` data-set. The tuning process is then evaluated with a 10% portion of the `edx` data, NOT with the `final_holdout_test` data-set.

```r
# Tuning lambda value

# Note: the process is calculated repeatedly by increasing the precision on each
# run. This is done in order to reduce the time it requires to process.

i_precision <- c(1,0.1,0.02)
i_interval <- c(2,6)

lambda <- sapply(i_precision, function(precision){
  lambdas <- seq(i_interval[1], i_interval[2], precision)
  rmses_temp <- sapply(lambdas, function(l){
    b_i <- train_set %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu_hat)/(n()+l))
    b_u <- train_set %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))
    b_g <- train_set %>%
      left_join(b_i, by="movieId") %>%
      left_join(b_u, by="userId") %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - b_i - b_u - mu_hat)/(n()+l))
    predicted_ratings <-
      tuning_set %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      mutate(pred = mu_hat + b_i + b_u + b_g) %>%
      .$pred
    return(RMSE(tuning_set$rating, predicted_ratings))
  })

  # qplot(lambdas, rmses_temp)  # Just for debugging purposes

  lambda_temp <- lambdas[which.min(rmses_temp)]

  i_interval <<- c(lambda_temp-precision,lambda_temp+precision)

  return(lambda_temp)

  # Clean up
  rm(lambda_temp,rmses_temp,lambdas)
})

# Here we obtain the tuned lambda
lambda <- lambda[length(lambda)]

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())

user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat)/(n()+lambda), n_i = n())

genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu_hat)/(n()+lambda), n_i = n())

predicted_ratings <- final_holdout_test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

# Ensuring that the predicted ratings are not out of the bounds (0.5 <= pred <= 5)
predicted_ratings <- ifelse(predicted_ratings < 0.5,0.5,predicted_ratings)
predicted_ratings <- ifelse(predicted_ratings > 5,5,predicted_ratings)

# In case some rows doesn't have predicted ratings replace with the average
```

```r
predicted_ratings[is.na(predicted_ratings)] <- mu_hat

# Calculate the RMSE for this method
mth_6 <- RMSE(final_holdout_test$rating, predicted_ratings)
```

After the tuning of lambda, the optimal lambda value is $\lambda = 4.84$. The Root Mean Square Error (RMSE) obtained with this lambda value is **0.8646659**.

Results

The final model for the recommendation system considers the effect of movies, users and genres, as well as the regularization of the ratings for movies with not so many ratings. The table with the Root Mean Square Error (RMSE) obtained at each step of the design proces is presented below.

```
## # A tibble: 6 x 2
##   METHOD                                                   RMSE
##   <chr>                                                   <dbl>
## 1 Just the average (mu_hat = 3.5123789705421 )            1.06
## 2 Movie effect                                            0.944
## 3 Movie + User effect                                     0.866
## 4 Movie, User and Genres effect                           0.865
## 5 Reg. Movie, User and Genres Effect (lambda = 3 )        0.865
## 6 Reg. Movie, User and Genres Effect Tuned (lambda = 4.84 ) 0.865
```

The method with the best performance is the one that accounts for all effects and the regularization of the parameter lambda, with a Root Mean Square Error (RMSE) of **0.8646659**.

However, notice that the recommendation system here analyzed doesn't take into consideration other possibly relevant aspects such as the possible preference of certain users to some genres.

Conclusion

In this report, the results of developing an algorithm for a recommendation system based the previous ratings of users are presented. The best performing method in this report is the one taking into consideration the effect of movies, users and genres, as well as the regularization of the ratings for movies with not so many ratings. The obtained Root Mean Square Error (RMSE) is **0.8646659**.

However, this method is limited by the existence of ratings of a certain user. The data calculates the user bias according to the previous ratings of that user. This means that the user bias could not be calculated for new users that did not rate any movie yet.

To improve this method in the future, it would be possible to include the effect of preference of certain users to some genres and the prediction for new users with no rating history.