Movielens Project

Jakub Below

February 2022

Introduction

This project's goal is to build a model capable of predicting new users' movie ratings based on the movielens data set. The main indicator of success is reaching the RMSE value of the model of 0.86490 or less.

In order to reach this goal, the author has explored the data set and constructed several predictive models based on his observations. The most effective one has been regularized and then evaluated against the validation set to confirm if the desired RMSE value has been achieved.

Data Preparation

The first step is to download the supplied data set for analysis.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# Note: this process could take a couple of minutes

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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")

# Validation set will be 10% of MovieLens data
```

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

Overview

Now, the data set can be reviewed to identify relevant features and trends.

```
edx <- edx %>% mutate(rating_y = year(as_datetime(timestamp)),
                released = str_match(title, "\\(([0-9][0-9][0-9][0-9])\\)")[,2],
                lapsed_y = as.integer(rating_y) - as.integer(released))
edx %>% select(-timestamp) %>% head(3) %>% kbl()
```

userId	movieId	rating	title	genres	rating_y	released	lapsed_y
1	122	5	Boomerang (1992)	Comedy|Romance	1996	1992	4
1	185	5	Net, The (1995)	Action|Crime|Thriller	1996	1995	1
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1996	1995	1

This data set consists of `9,000,055` rows and `9` columns with `10,677` distinct movies and `69,878` distinct users.

It also includes the following features:

- User ID
- Movie ID
- Movie rating
- Movie Title
- Movie Genres
- Timestamp (removed from the preview)
- Year the movie has been rated (calculated by the author)
- Year the movie has been released (calculated by the author)
- Years passed between the premiere and rating (calculated by the author)

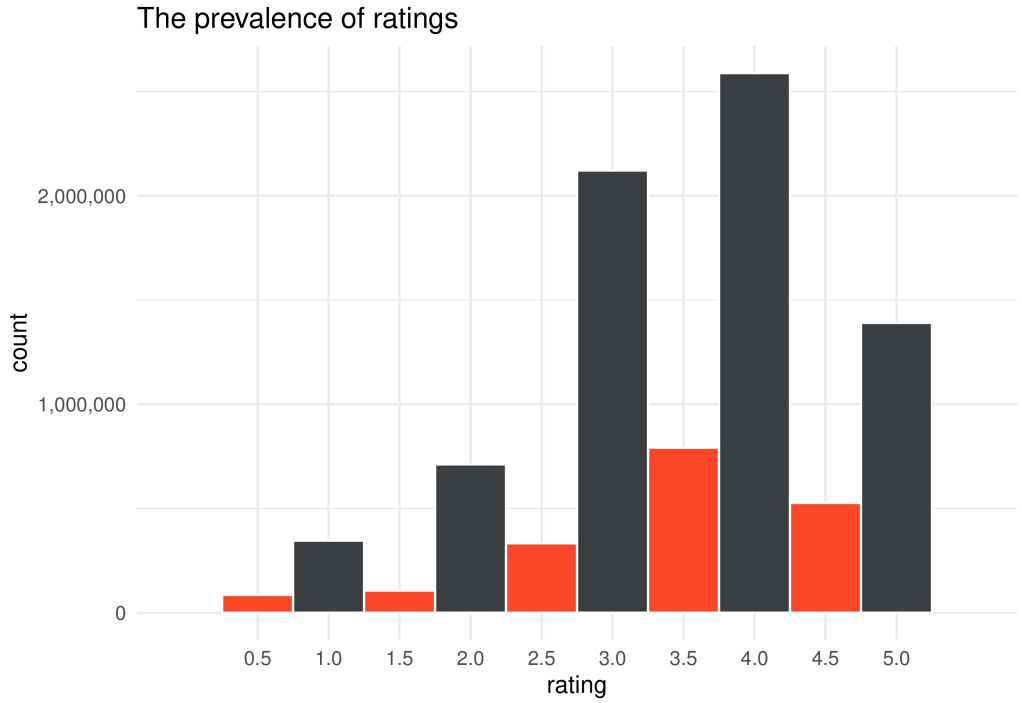
The data generally is in good shape and doesn't contain unexpected anomalies.

```
t(summary(edx)[c(1,3,4,6),]) %>% kbl()
```

userId	Min. : 1	Median :35738	Mean :35870	Max. :71567
movieId	Min. : 1	Median : 1834	Mean : 4122	Max. :65133
rating	Min. :0.500	Median :4.000	Mean :3.512	Max. :5.000
timestamp	Min. :7.897e+08	Median :1.035e+09	Mean :1.033e+09	Max. :1.231e+09
title	Length:9000055	Mode :character	NA	NA
genres	Length:9000055	Mode :character	NA	NA
rating_y	Min. :1995	Median :2002	Mean :2002	Max. :2009
released	Length:9000055	Mode :character	NA	NA
lapsed_y	Min. :-2.00	Median : 7.00	Mean :11.98	Max. :93.00

Analysis

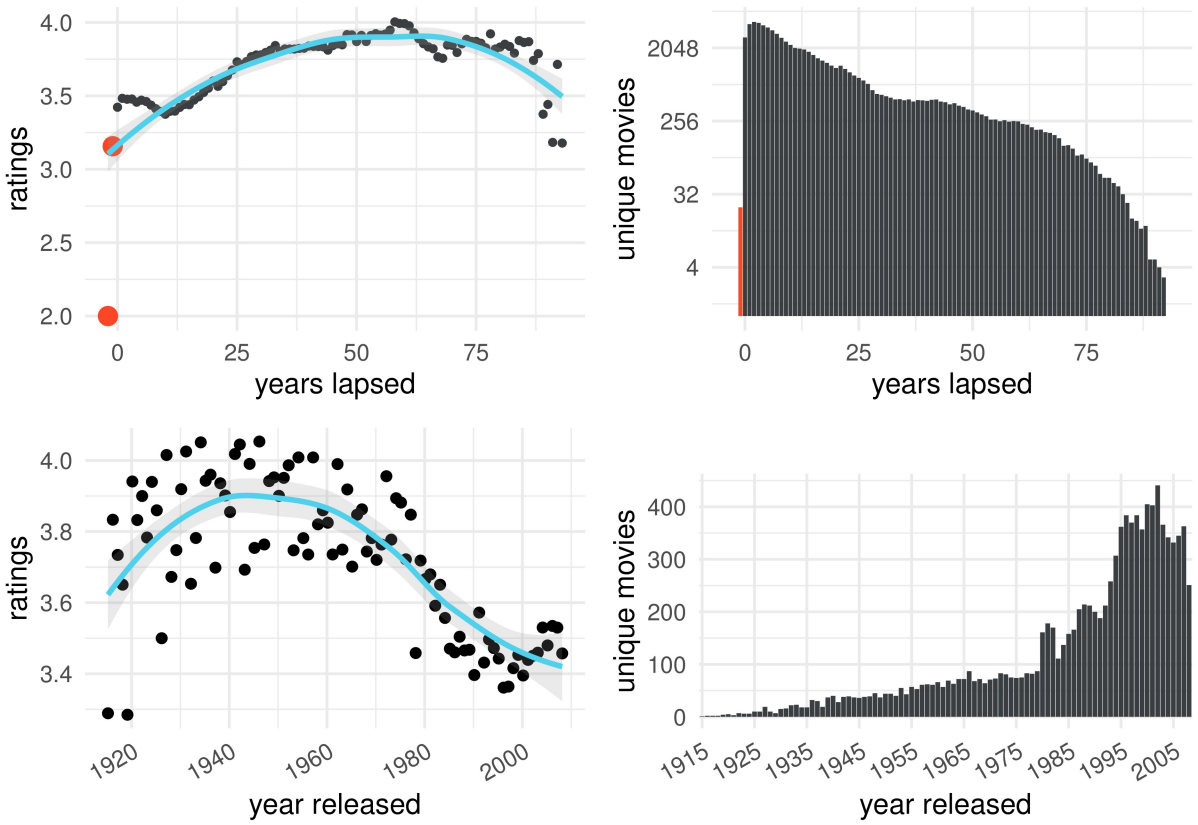
Users are more prone to grade a movie with a whole star rating than with half star rating (`79.52`% ratings are natural numbers). It's clearly visible with number of respective ratings plotted.



An important conclusion is that we only allow for full-star and half-star ratings. It's not important for constructing the model but if it would ever be used in real-life applications, it should be wrapped with a function like the following one to make sure it returns only viable ratings.

```
round_to_half <- function(x){round(x/0.5)*0.5}
```

An important question is if year the movie has been released, as well as time elapsed between premiere and the rating, affect ratings. I can be answered by visualizing key time trends.



The above charts clearly show that there is a time effect, although it's not as straightforward as one may assume.

- Top-left - The older the movie, the higher the average rating. There may be several reasons, e.g. people are more eager to watch good old movies and forget about the bad ones (see the Survivorship Bias). This effect wanes with older movies and completely disappears for the oldest ones - they may be dated and obscure for viewers. An interesting bias can be observed with first two data point highlighted in red. These are movies rated before they were screened. It may indicate that people who do not approve of the director or movie studio can rate the movie negatively without watching them.
- Top-right - The older the movie, the fewer the ratings. It may support the Survivorship Bias hypothesis (fewer but more popular movies are being watched).
- Bottom-left - This chart shows premiere year per movie, instead of years passed between release and rating. Older movies receive generally better ratings up to a certain point in which their ratings drop. Additionally, the deviation between ratings year-to-year increase as we move back in time. It also supports previous notions - older movies may be deemed classics but also very old movies may be seen as obscure.
- Bottom-right - This chart shows yet again that there are more recent movies reviewed than old movies. Initially, a chance to be reviewed increases with time since viewers have more occasions to catch-up with premiers but after several years this trend starts to revert and only well-known movies tend to be watched.

Generally, the more widely known the movie is, the higher ratings it gets. Movies with many reviews receive better ratings on average, which can mean that people tend to watch already popular flicks. The correlation though, is not high.

	n	rating
n	1.0000000	0.2114161
rating	0.2114161	1.0000000

The above comments may bring interesting insights to our model.

Preparations

The data set has already been transformed to include release year (scraped from the title column), the year the movie has been rated, and number of years between these two occurrences (it can be negative if people bombed the rating between the premiere screening). As a reminder, the following code has been used:

```
edx <- edx %>% mutate(rating_y = year(as_datetime(timestamp)),
              released = str_match(title, "\\(([0-9][0-9][0-9][0-9])\\)")[,2],
              lapsed_y = as.integer(rating_y) - as.integer(released))
```

As the data is clean, there is no NA values, and generally the quality is good, there is no need to extensive cleaning or manipulation of the data set.

```
print(sum(is.na(edx))) # no NA values
```

```
## [1] 0
```

Next, the data has been split into train set and test sets to create and evaluate models.

```
# keep 10% of records for evaluation
set.seed(47, sample.kind="Rounding")        # set seed to receive consistent outcomes
train_index <- createDataPartition(edx$rating, times=1, p=0.1, list=FALSE)
train_set <- edx[-train_index,]
test_set <- edx[train_index,]
```

Finally, a function to evaluate the models score has been declared. The measure used was the standard deviation of the residuals (RMSE).

```
RMSE = function(observed, predicted){
  sqrt(mean((observed - predicted)^2))
}
```

As mentioned before, the goal was to achieve the RMSE **lower than or equal to 0.86490**.

Methods

The start point is to take the expected value and use it as a prediction for each movie. The expected value is just the mean of all ratings in the training set (plus random error). The model took the following form.

$$y = \mu + \epsilon$$

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

# Evaluate the model
expected_value_rmse <- RMSE(test_set$rating, mu)

print(expected_value_rmse)
```

```
## [1] 1.060433
```

Any model worth its time have to be better than that and beat the average. For starters, a model including movie bias has been constructed. Since it is known that some movies are more popular than other, the author calculated the distance from each movie's average rating and the overall data set average. This model now includes the movie bias (b_m) for any given movie prediction (y_m).

$$y_m = \mu + b_m + \epsilon$$

```r
movie_mu <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))

linear_predictions <- test_set %>%
  # Join movie distances (biases)
  left_join(movie_mu, by="movieId") %>%
  # Make sure to fill NAs for movies not in the training set
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  # Predict by adding overall mean and movie bias
  mutate(predict_by_movie = mu + b_m)

movie_bias_rmse <- RMSE(test_set$rating, linear_predictions$predict_by_movie)

print(movie_bias_rmse)
```

```
## [1] 0.9435307
```

It's already better but still not close to the goal. Luckily, this model can be improved further when realized that some users are more critical than others. The new model will include the movie bias (b_m) and user bias (b_u) for any given movie prediction for given user (y_{mu}), thus changing the model to a new form.

$$y_{m,u} = \mu + b_m + b_u + \epsilon$$

```r
user_mu <- train_set %>%
  # Remove movie bias to only get user tendencies to be more or less strict
  left_join(movie_mu, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))

linear_predictions <- linear_predictions %>%
  left_join(user_mu, by="userId") %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u)) %>%
  mutate(predict_by_user = mu + b_u,
```

```
         predict_by_movie_and_user = mu + b_m + b_u)

user_bias_rmse <- RMSE(test_set$rating, linear_predictions$predict_by_user)
user_and_movie_rmse <- RMSE(test_set$rating, linear_predictions$predict_by_movie_and_user)

print(user_bias_rmse)
```

```
## [1] 0.9945983
```

```
print(user_and_movie_rmse)
```

```
## [1] 0.865455
```

Including user bias also improves the model even if slightly less than including just movie bias. However, when taking into account both biases, it yielded over 18% improvement from the initial average approach.

However, some genres are generally more popular than others. Therefore, it is prudent to attempt to add genre bias, resulting in the following model.

$$y_{m,u,g} = \mu + b_m + b_u + b_g + \epsilon$$

```
genres_mu <- train_set %>%
  left_join(movie_mu, by="movieId") %>%
  left_join(user_mu, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_m - b_u))

linear_predictions <- linear_predictions %>%
  left_join(genres_mu, by="genres") %>%
  mutate(b_g = ifelse(is.na(b_g), 0, b_g)) %>%
  mutate(predict_by_genres = mu + b_g,
         predict_by_user_movie_genres  = mu + b_m + b_u + b_g)

genres_bias_rmse <- RMSE(test_set$rating,
                         linear_predictions$predict_by_genres)
genres_user_movie_rmse <- RMSE(test_set$rating,
                               linear_predictions$predict_by_user_movie_genres)

print(genres_bias_rmse)
```

```
## [1] 1.058586
```

```
print(genres_user_movie_rmse)
```

```
## [1] 0.8651729
```

This time the improvement is nearly unnoticeable. A function could be defined to deconstruct the genres combination (e.g. "Action|Drama|Sci-Fi|Thriller") into respective genres and measures the sum of their biases but it's a computation-heavy solution for such a large data set and more importantly - doesn't yield satisfactory results. The author won't pursue this approach in this report but the following code can be used for such an approach for demonstration purposes.

```r
# create a table of genre biases
genre_mus <- train_set %>%
  select(genres, rating) %>%
  #(1 - split into respective biases)
  separate(col=genres, sep="[//|]", into=c("1","2","3","4","5","6","7","8")) %>%
  #(2 - make it tidy)
  pivot_longer(1:8, names_to="col_num", values_to="genre") %>%
  filter(!is.na(genre)) %>%
  group_by(genre) %>%
  # distance for each genre from the overall mean
  summarize(g_mu = mean(rating - mu))

check_genres_mu <- function(x){
  # This function takes in a row of data and checks for each genre in the genres column
  # (e.g. Drama|Comedy|Thriller) to sum all the biases for those genres
  b <- sapply(genre_mus$genre, function(g){                    # for each genre
    bias <- as.numeric(ifelse(str_detect(x$genres, g),         # if present in string
                              genre_mus %>% filter(genre==g) %>%   # take bias
                                .$g_mu, 0)
                       )
    bias                                                       # return all biases
  })
  rowSums(b)           # take the sum of all genres' biases for given movie
}

# apply the above function to the test set
genre_mu <- check_genres_mu(test_set)
```

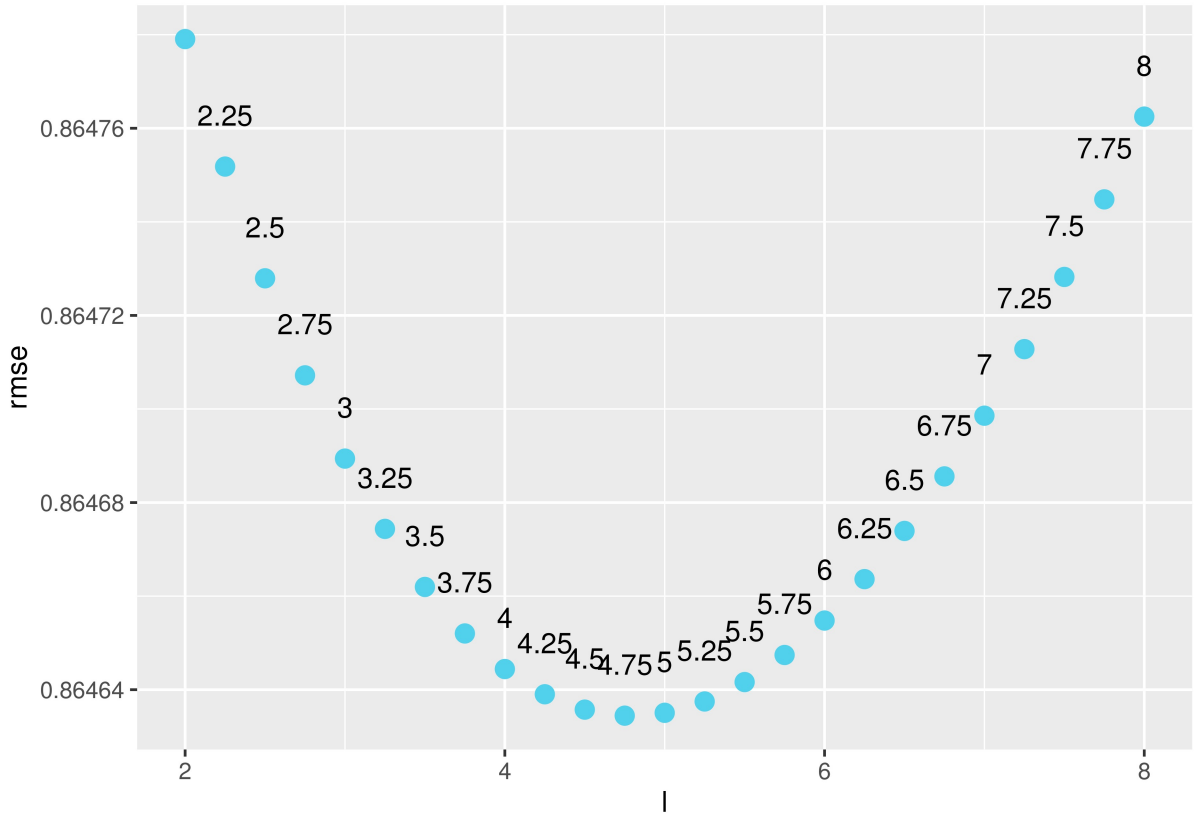
The outcome of the model factoring in movie, user, and genres combination biases is already close to the goal defined at the beginning (0.86517 vs **0.86490**).

Having the model chosen, it can be tuned for better results. It boils down to experimenting with lambda (λ) parameter, which will penalize biases for categories without many records (users or movies with few ratings). This is calculated by dividing a sum of biases for the given category by the sum of observations plus the lambda value.

$$\frac{\sum(bias)}{n+\lambda}$$

```r
lambda <- seq(2,8, 0.25)

tune_lambda <- sapply(lambda, function(l){
  # This function will apply given lambda to movie bias and check RMSE for predictions
  movie_mu <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n() + l))       # test bias with a given lambda

  user_mu <- train_set %>%
    left_join(movie_mu, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n() + l))

  genres_mu <- train_set %>%
    left_join(movie_mu, by="movieId") %>%
```

```
    left_join(user_mu, by="userId") %>%
    group_by(genres) %>%
    summarize(b_gs = sum(rating - mu - b_m - b_u)/(n() + l))

  predictions <- test_set %>%
    left_join(movie_mu, by="movieId") %>%
    left_join(user_mu, by="userId") %>%
    left_join(genres_mu, by="genres") %>%
    replace(is.na(.), 0) %>%
    mutate(predict = mu + b_u + b_m + b_gs)

  RMSE(test_set$rating, predictions$predict)
})

data.frame(l=lambda, rmse = tune_lambda) %>%          # plot the results
  ggplot(aes(l, rmse)) +
  geom_point(color="#53d1ee", size=3) +
  geom_text(aes(label=l), vjust = -2, size = 4)
```



When plotted, it becomes apparent that the best lambda is 4.75, with the following outcome.

```
regularized_genres_user_movie_rmse <- min(tune_lambda)
print(regularized_genres_user_movie_rmse)
```

```
## [1] 0.8646344
```

Results

Finally, all the models can be compared against each other.

method	RMSE
Regularized Genre & User & Movie	0.8646344
Genre & User & Movie	0.8651729
User and Movie	0.8654550
Movie	0.9435307
User	0.9945983
Genres	1.0585860
Expected value	1.0604325

The regularized model utilizing user, movie, and genres biases meet the requirements. As the final step, it has to be evaluated using the validation data set.

Validation

A function that will return the RMSE has already been defined. The train data set has already been used to decide on which model should be selected base on its performance. Having that that, the author has to additionally load a separate data set that only serves for final evaluation of selected model. This allows for avoiding the risk of overfitting.

```
FINAL_MODEL <- function(train_set, test_set, l=4.5){
  # This function will apply given lambda to biases and check RMSE for predictions
  movie_mu <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n() + l))

  user_mu <- train_set %>%
    left_join(movie_mu, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n() + l))

  genres_mu <- train_set %>%
    left_join(movie_mu, by="movieId") %>%
    left_join(user_mu, by="userId") %>%
    group_by(genres) %>%
    summarize(b_gs = sum(rating - mu - b_m - b_u)/(n() + l))

  predictions <- test_set %>%
    left_join(movie_mu, by="movieId") %>%
    left_join(user_mu, by="userId") %>%
    left_join(genres_mu, by="genres") %>%
    replace(is.na(.), 0) %>%
    mutate(predict = mu + b_u + b_m + b_gs) %>%
    pull(predict)

  RMSE(test_set$rating, predictions)
}

# Pass the validation set to the function to evaluate the final model against it
```

```r
final_rmse <- FINAL_MODEL(train_set, validation)

# Final RMSE
print(final_rmse)
```

```
## [1] 0.8648431
```

Finally, the author can answer the question: does this model yield expected RMSE lower than **0.86490**?

```r
final_rmse < 0.86490
```

```
## [1] TRUE
```

Conclusions

The overall goal of this project was to predict new ratings for movies based on the given data set containing information on already rated movies, their titles, genres, etc. More specifically, the aim was to produce a predictive model with RMSE of at least 0.86490 or less.

The author downloaded the aforementioned data set and presented an overview of available data and exploratory analysis covering its most important trends and characteristics. After testing several models and tuning them with the regularization techniques, the final model has proven to achieve the main goal as confirmed with the validation data set.

An important notion is that for any model to be implemented in real-life applications, the result has to be rounded to the nearest half so that the outcome matches the rating scale (half-star or full-star ratings between 0.5 and 5.0).

The final model has utilized data on movie ID, user ID, genres, and ratings. The data set includes a large number of records, which limits further exploration. The model could be potentially refined with the use of date of rating and the date of release since they appear to have an impact on the outcome. More robust models could also create clusters of similar users to leverage their preferences to tweak the final prediction, however, this will also likely be more expensive computation-wise.