MovieLens Recommendations

HarvardX Data Science Professional Certificate: Capstone Project 1

Ricardo P. Mendes

Contents

1 Introduction

Recommendation systems are very important applications of machine learning today, and are used to recommend products, movies, songs, and all types of items to users. They are key to help users stay in the site and increase revenue.

The MovieLens dataset consists of 10 million movie ratings, which have been split into a training and a validation set.

The purpose of this project is to select an algorithm that can predict ratings with a root mean square error of less than 0.86490 compared to the actual ratings in the validation set.

This report presents:

1. An analysis of the data in the MovieLens dataset.
2. Training and testing of some algorithms to determine the best one.
3. Results obtained in the validation set.
4. A conclusion with a brief summary, the limitations of the selected algorithm and the potential for future work.

Note: This report was generated using R Markdown in RStudio.

2 Creating the training and validation sets

A standard code has been provided by the course creators to download the dataset and split it in two parts:

- Training set: represented by the **edx** variable.
- Validation set: represented by the **validation** variable.

```r
# Create edx set, validation set (final hold-out
# test set) Note: this process could take a
# couple of minutes

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)

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

dl <- tempfile()
download.file("https://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")

# if using R 3.6 or earlier: movies <-
# as.data.frame(movies) %>% mutate(movieId =
# as.numeric(levels(movieId))[movieId], title =
# as.character(title), genres =
# as.character(genres))

# if using R 4.0 or later:
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)
```

```r
# Saving the data to the disk
save(edx, file = "edx.RData")
save(validation, file = "validation.RData")
```

```r
# Loading libraries

library(tidyverse)
library(ggplot2)
library(dplyr)
library(knitr)
library(caret)
library(kableExtra)
```

```r
# Configuring the theme for the plots
plots_theme <- theme_classic() + theme(plot.title = element_text(color = "#0099F8",
    size = 16, face = "bold"), axis.title = element_text(size = 10))
```

3 Dataset Overview

The dataset contains the following columns: userId, movieId, rating, timestamp, title, genres

These are the first 5 rows:

```r
# Tabulate class of variables and first 5 rows
# included in edx dataset
rbind((lapply(edx, class)), head(edx)) %>%
    kable(caption = "edx dataset: variable class and first 5 rows",
        align = "cccll1", booktabs = T, format = "latex",
        linesep = "") %>%
    row_spec(1, hline_after = T) %>%
    kable_styling(full_width = FALSE, position = "center",
        latex_options = c("scale_down", "hold_position"))
```

Table 1: edx dataset: variable class and first 5 rows

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

These are the dataset dimensions: 9,000,055 rows and 6 columns.

4 Analysis

4.1 Data Exploration and Visualization

4.1.1 Ratings

```
ratings <- edx %>%
    group_by(rating) %>%
    summarise(count = n())
frequent_ratings <- ratings %>%
    arrange(desc(count)) %>%
    head(2) %>%
    select(rating) %>%
    arrange(rating) %>%
    pull()
```

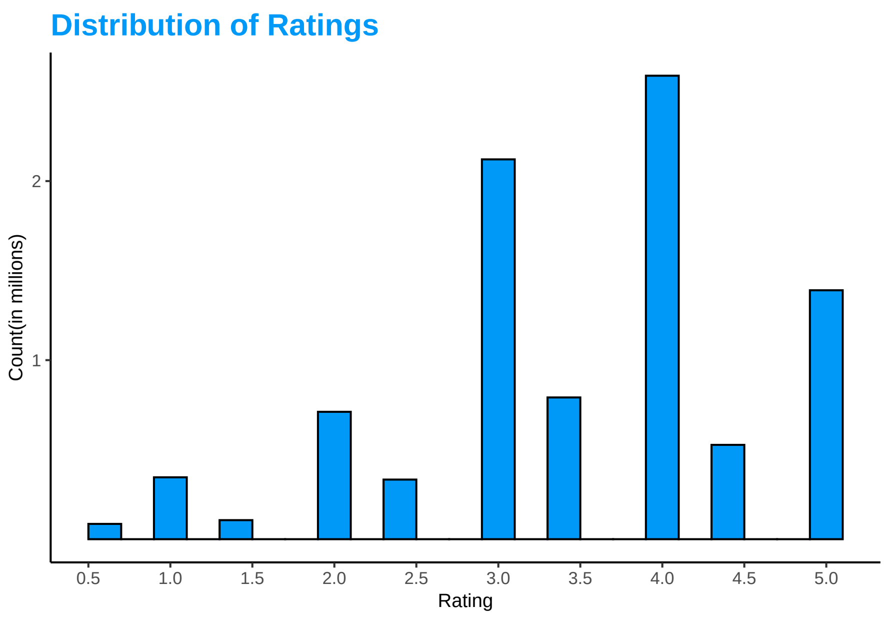
As shown below, ratings range from 0.5 to 5, with 3 and 4 being the most frequent values.

```
# Number of ratings
ratings %>%
    arrange(rating) %>%
    kable(col.names = c("Rating", "Count"), caption = "Distribution per Rating",
        align = "cc", booktabs = T, format = "pandoc",
        linesep = "") %>%
    row_spec(1, hline_after = T, font_size = 3) %>%
    kable_styling(full_width = FALSE, position = "center",
        latex_options = c("scale_down", "hold_position"))
```

Table 2: Distribution per Rating

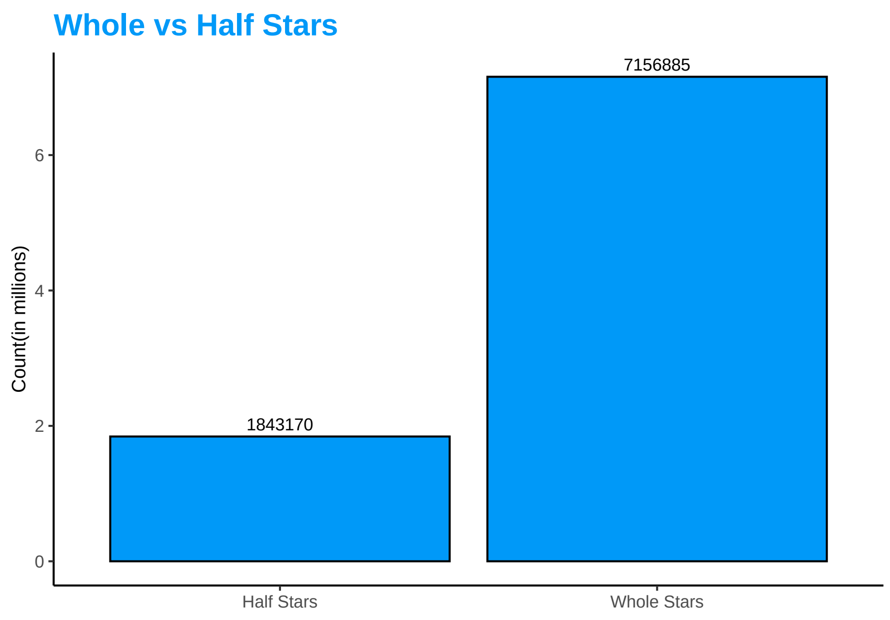
Rating	Count
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

```
# Plotting the ratings in a histogram
edx %>%
    ggplot(aes(rating)) + geom_histogram(binwidth = 0.2,
    color = "#000000", fill = "#0099F8") + scale_y_continuous(breaks = c(1e+06,
    2e+06), labels = c("1", "2")) + scale_x_continuous(breaks = seq(0.5,
    max(edx$rating), 0.5)) + labs(title = "Distribution of Ratings",
    x = "Rating", y = "Count(in millions)") + plots_theme
```



The chart below compares the number of full stars versus the number of half starts given:

```
# Most given ratings
wholeStars <- edx %>%
    group_by(rating) %>%
    summarise(count = n()) %>%
    filter(rating %in% c(1, 2, 3, 4, 5)) %>%
    summarise(total = sum(count)) %>%
    pull()
halfStars <- edx %>%
    group_by(rating) %>%
    summarise(count = n()) %>%
    filter(rating %in% c(0.5, 1.5, 2.5, 3.5, 4.5)) %>%
    summarise(total = sum(count)) %>%
    pull()

stars <- data.frame(Type = c("Whole Stars", "Half Stars"),
    Values = c(wholeStars, halfStars))

stars %>%
    ggplot(aes(Type, y = Values/1e+06)) + geom_bar(stat = "identity",
    color = "#000000", fill = "#0099F8") + geom_text(aes(label = Values),
    size = 3, vjust = -0.5) + labs(title = "Whole vs Half Stars",
    y = "Count(in millions)") + plots_theme + theme(axis.title.x = element_blank())
```



4.1.2 Movies

4.1.2.1 Number of Movies This is the number of unique movies in the dataset: 10677

4.1.2.2 Movies per Genre There are 20 different genres in the dataset.

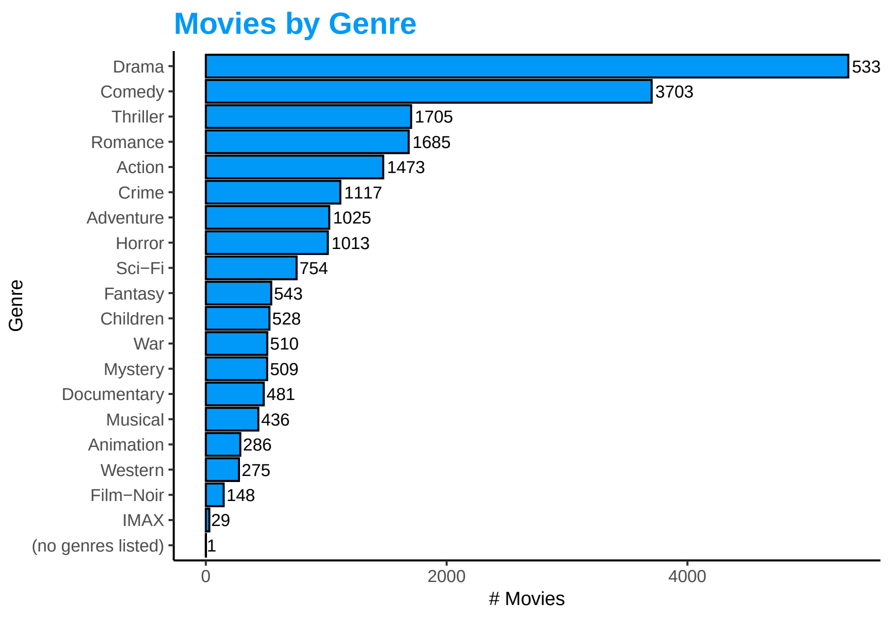
The table and chart below show the number of movies in each genre. Please note that each movie can be classified as more than one genre.

```
# Number of movies per genre
movies_genre <- edx %>%
    select(movieId, genres) %>%
    distinct() %>%
    separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarise(count = n())

movies_genre %>%
    kable(col.names = c("Genre", "# Movies"), caption = "Movies per Genre",
        format = "latex", linesep = "") %>%
    kable_styling(full_width = FALSE, position = "center",
        latex_options = "hold_position")
```

```
# Plotting the genres in a histogram
movies_genre %>%
    ggplot(aes(genres, x = reorder(genres, count),
        y = count)) + geom_bar(stat = "identity", color = "#000000",
    fill = "#0099F8") + coord_flip() + geom_text(aes(label = count),
    vjust = 0.5, hjust = -0.1, size = 3) + labs(title = "Movies by Genre",
    x = "Genre", y = "# Movies") + plots_theme
```

Table 3: Movies per Genre

Genre	# Movies
(no genres listed)	1
Action	1473
Adventure	1025
Animation	286
Children	528
Comedy	3703
Crime	1117
Documentary	481
Drama	5336
Fantasy	543
Film-Noir	148
Horror	1013
IMAX	29
Musical	436
Mystery	509
Romance	1685
Sci-Fi	754
Thriller	1705
War	510
Western	275



```r
# Separate individual genres and ranking them by
# the total number of ratings in the edx dataset
edx %>%
    separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarise(count = n(), rating = round(mean(rating),
        2)) %>%
    arrange(desc(count)) %>%
    kable(col.names = c("Genre", "No. of Ratings",
        "Ave. Rating"), caption = "Individual genres ranked by number of ratings",
        align = "lrr", booktabs = TRUE, format = "latex",
        linesep = "") %>%
    kable_styling(full_width = FALSE, position = "center",
        latex_options = "hold_position")
```

Table 4: Individual genres ranked by number of ratings

Genre	No. of Ratings	Ave. Rating
Drama	3910127	3.67
Comedy	3540930	3.44
Action	2560545	3.42
Thriller	2325899	3.51
Adventure	1908892	3.49
Romance	1712100	3.55
Sci-Fi	1341183	3.40
Crime	1327715	3.67
Fantasy	925637	3.50
Children	737994	3.42
Horror	691485	3.27
Mystery	568332	3.68
War	511147	3.78
Animation	467168	3.60
Musical	433080	3.56
Western	189394	3.56
Film-Noir	118541	4.01
Documentary	93066	3.78
IMAX	8181	3.77
(no genres listed)	7	3.64

```
# Get the movie with the greatest number of
# ratings
edx %>%
    group_by(movieId, title) %>%
    summarise(count = n()) %>%
    arrange(desc(count)) %>%
    head() %>%
    kable(col.names = c("Movie ID", "Movie Title",
        "Ratings"), caption = "Ratings per Movie",
        format = "pandoc", linesep = "") %>%
    kable_styling(full_width = FALSE, position = "center",
        latex_options = "hold_position")
```

4.1.2.3 Top Rated Movies

Table 5: Ratings per Movie

Movie ID	Movie Title	Ratings
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212

4.1.3 Users

This is the number of unique users in the dataset: 69878

4.2 Insights

4.3 Creating the Test Set

At the beginning, part of the data was reserved for the final validation (the **validation** dataset). However, during development, there is the need to evaluate the different algorithms by performing cross-validation and refinements without the risk of overtraining. Therefore, the **edx** dataset must be split into training (80%) and test (20%) sets.

To perform this division, we must ensure that the test set contains only users and movies that are also in the training set. Moreover, the removed data must be put back to the training set to maximize the amount of data available for training purposes.

```r
# Creating training and testing sets from edx
set.seed(1234, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
    p = 0.2, list = FALSE)
train_set <- edx[-test_index, ]
temp <- edx[test_index, ]

# Ensure userId and movieId are also in the
# training set
test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

# Add rows removed from test set back into train
# set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
# Remove temporary files to tidy environment
rm(test_index, temp, removed)
```

4.4 Calculating the Error

The error measurement used here is the residual mean square error (RMSE). It compares the ratings predicted by the algorithm with the actual ratings in the test set. This is the formula for RMSE:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,m}\left(\hat{y}_{u,m} - y_{u,m}\right)^2}$$

In this formula:

- $y_{u,m}$: the actual rating provided by the user u for movie m
- $\hat{y}_{u,m}$: the predicted rating user u would give for movie m
- N: total number of user/movie combinations

As stated before, the objective of the project is to achieve RMSE < 0.86490.

```r
rmse_objective <- 0.8649
```

4.5 Modeling

In this section, some algorithms are tested to determine which one yields the better performance.

4.5.1 Average Model

The simplest way to predict ratings is to always predict the same value, regardless of the user or the movie. That value should be the average rating, represented by μ. Therefore, the formula for this first algorithm is:

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

From the dataset, we obtain the average rating by doing $\hat{\mu}$ = mean(train_set$rating)

```r
# Calculate the average rating across all movies
# in the training set
mu_hat <- mean(train_set$rating)
# Calculate RMSE for this model
simple_avg_rmse <- RMSE(test_set$rating, mu_hat)
```

The results are:

- Average: 3.5125648
- RMSE: 1.0609196

4.5.2 Random Forest

The previous data analysis showed that ratings vary greatly and it is natural to expect that different factors contribute for the ratings. The Random Forest model is a good candidate since it considers all the factors and their variations to build decision trees.

```r
library(randomForest)
library(doParallel)

predictors_train <- train_set %>%
    select(userId, movieId, timestamp, title, genres)
predictors_test <- test_set %>%
    select(userId, movieId, timestamp, title, genres)

cluster <- makePSOCKcluster(7)
registerDoParallel(cluster)

rf_result <- randomForest(x = predictors_train, y = train_set$rating,
    xtest = predictors_test, ytest = test_set$rating,
    do.trace = TRUE)
print(rf_result)

stopCluster(cluster)
```

The code above was run until the MSE on the test set started to increase, indicating overfitting. The purpose was to find out the number of trees needed to get the best MSE. This was the result:

Tree	Out-of-bag MSE	Out-of-bag %Var(y)	Test Set MSE	Test Set %Var(y)
1	1.0790	96.02	1.4820	131.71
2	1.0300	91.67	1.2060	107.13
3	1.0150	90.35	1.1380	101.11
4	1.0000	88.99	1.0990	97.64
5	0.9951	88.54	1.0760	95.63
6	0.9782	87.02	1.0620	94.36
7	0.9635	85.73	1.0500	93.26
8	0.9502	84.54	1.0430	92.69
9	0.9433	83.92	1.0360	92.08
10	0.9341	83.10	1.0310	91.61
11	0.9272	82.49	1.0280	91.33
12	0.9217	82.00	1.0270	91.20
13	0.9166	81.55	1.0240	90.97
14	0.9131	81.24	1.0210	90.69
15	0.9100	80.96	1.0200	90.66
16	0.9073	80.72	1.0190	90.53
17	0.9046	80.48	1.0160	90.29
18	0.9022	80.27	1.0150	90.19
19	0.9003	80.10	1.0130	89.98
20	0.8986	79.95	1.0120	89.89
21	0.8968	79.78	1.0110	89.85
22	0.8956	79.68	1.0100	89.70
23	0.8945	79.58	1.0090	89.63
24	0.8933	79.48	1.0070	89.47
25	0.8922	79.38	1.0060	89.39
26	0.8911	79.28	1.0050	89.30
27	0.8902	79.20	1.0060	89.35
28	0.8893	79.12	1.0050	89.25
29	0.8885	79.05	1.0040	89.20
30	0.8877	78.98	1.0040	89.18

Tree	Out-of-bag MSE	Out-of-bag %Var(y)	Test Set MSE	Test Set %Var(y)
31	0.8870	78.92	1.0040	89.18
32	0.8864	78.86	1.0030	89.13
33	0.8857	78.80	1.0020	89.00
34	0.8853	78.76	1.0020	89.01
35	0.8848	78.72	1.0010	88.90
36	0.8843	78.68	1.0010	88.91
37	0.8839	78.64	1.0000	88.87
38	0.8835	78.60	0.9995	88.80
39	0.8831	78.56	0.9997	88.82
40	0.8826	78.52	0.9988	88.74
41	0.8823	78.49	0.9986	88.72
42	0.8819	78.46	0.9988	88.73
43	0.8816	78.43	0.9984	88.71
44	0.8812	78.40	0.9984	88.70
45	0.8809	78.37	0.9977	88.64
46	0.8806	78.34	0.9974	88.62
47	0.8803	78.32	0.9973	88.60
48	0.8800	78.30	0.9973	88.60
49	0.8798	78.27	0.9974	88.62
50	0.8796	78.25	0.9978	88.65
51	0.8793	78.23	0.9978	88.65
52	0.8791	78.21	0.9977	88.64
53	0.8788	78.19	0.9980	88.66
54	0.8786	78.17	0.9975	88.62
55	0.8784	78.15	0.9976	88.63

The random forest model reached MSE = 0.9973 (RMSE = 0.9986) on the test set before starting overfitting (the MSE increased with more than 48 trees). Unfortunatelly, it is still far from the desired RMSE.

4.5.3 Linear Regression

On a different approach, let us check if the linear regression can achieve better results. The average model does not consider the other data we have on each row and how they affect the rating.

These are the other information we have:

- The user who rated the movie
- The movie that has been rated
- When the user rated that movie
- The genres to which the movie belongs

Each of these factors is represented by a bias factor in the model equation. Let's consider each of them.

4.5.3.1 User Effect Since different people have different opinions and tastes, we must take into account who is rating a movie. To do that, we add the user bias b_u to the model equation:

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

We could do that by running:

```r
lm_fit <- lm(train_set$rating ~ as.factor(userId),
    data = train_set)
```

However, since there are thousands of users, the **lm()** function would take a long time to complete and the system would run out of memory. To work around this problem, we can assume that

$$\hat{b_u} = \hat{Y}_{u,i} - \hat{\mu}$$

Let's check if we get better predictions:

```r
user_avg <- train_set %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat))

lr_predicted_u <- test_set %>%
    left_join(user_avg, by = "userId") %>%
    mutate(predicted = mu_hat + b_u) %>%
    .$predicted

lr_u_rmse <- RMSE(lr_predicted_u, test_set$rating)
```

By considering the user, we obtain an RMSE of 0.9797569.

4.5.3.2 Movie Effect Since each user would rate movies differently (i.e. a person does not give the same rating to all movies they watch), we must factor in the movie that is being rated. To do that, we will add the movie bias b_m to the equation:

$$Y_{u,m} = \mu + b_u + b_m + \epsilon_{u,m}$$

Let's check what happens with the RMSE now:

```r
movie_avg <- train_set %>%
    left_join(user_avg, by = "userId") %>%
    group_by(movieId) %>%
    summarize(b_m = mean(rating - mu_hat - b_u))

lr_predicted_m <- test_set %>%
    left_join(user_avg, by = "userId") %>%
    left_join(movie_avg, by = "movieId") %>%
    mutate(predicted = mu_hat + b_u + b_m) %>%
    .$predicted

lr_um_rmse <- RMSE(lr_predicted_m, test_set$rating)
```

By considering both the user and the movie, we obtain an RMSE of 0.88397.

4.5.3.3 Genre Effect It is natural that people tend to prefer some genres over others. To check if that is an important factor, we must add the b_g term to the equation:

$$Y_{u,m} = \mu + b_u + b_m + b_g + \epsilon_{u,m}$$

Let's check the impact on RMSE:

```
genre_avg <- train_set %>%
    left_join(user_avg, by = "userId") %>%
    left_join(movie_avg, by = "movieId") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu_hat - b_u - b_m))

lr_predicted_g <- test_set %>%
    left_join(user_avg, by = "userId") %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(genre_avg, by = "genres") %>%
    mutate(predicted = mu_hat + b_u + b_m + b_g) %>%
    .$predicted

lr_umg_rmse <- RMSE(lr_predicted_g, test_set$rating)
```

By considering the user, the movie and the genre, we obtain an RMSE of 0.88397.

4.5.3.4 Date Effect

To make the **timestamp** column useful, we must first convert it to a date. Let's use only the year of the date so that we can group ratings by year.

```
library(lubridate)
train_set$year <- year(as_datetime(train_set$timestamp))
test_set$year <- year(as_datetime(test_set$timestamp))
```

Now we have an additional column with the year:

```
head(train_set)
```

userId	movieId	rating	timestamp	title	genres	year
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1996
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1996
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1996
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1996
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1996
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War	1996

To consider the year effect, we must add the b_y term to the equation:

$$Y_{u,m} = \mu + b_u + b_m + b_g + b_y + \epsilon_{u,m}$$

Let's check if the RMSE improves:

```
year_avg <- train_set %>%
    left_join(user_avg, by = "userId") %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(genre_avg, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = mean(rating - mu_hat - b_u - b_m -
        b_g))

lr_predicted_y <- test_set %>%
    left_join(user_avg, by = "userId") %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(genre_avg, by = "genres") %>%
    left_join(year_avg, by = "year") %>%
    mutate(predicted = mu_hat + b_u + b_m + b_g + b_y) %>%
    .$predicted

lr_umgy_rmse <- RMSE(lr_predicted_y, test_set$rating)
```

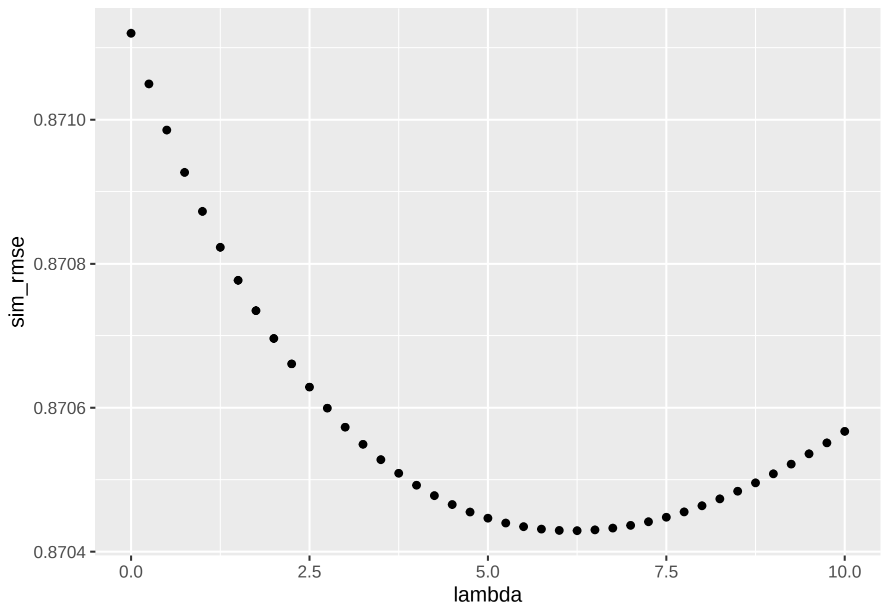
By considering the user, the movie, the genre, and the year, we obtain an RMSE of 0.8831119.

4.5.3.5 Regularization To improve the RMSE, we can apply regularization to the estimates, so that large estimates formed using small sample sizes are penalized. That will eliminate large errors, thus improving the RMSE. The regularization is done by applying a factor (λ):

$$\frac{1}{N}\sum_{u,m}(Y_{u,m} - \mu - b_u - b_m - b_g - b_y)^2 + \lambda(\sum_u b_u^2 + \sum_m b_m^2 + \sum_g b_g^2 + \sum_y b_y^2)$$

To find out the best value for λ, we need to simulate some values:

```
lambda <- seq(0, 10, 0.25)

sim_rmse <- sapply(lambda, function(l) {
    b_u <- train_set %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - mu_hat)/(n() +
            l))

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

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

    b_y <- train_set %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_m, by = "movieId") %>%
        left_join(b_g, by = "genres") %>%
        group_by(year) %>%
        summarize(b_y = sum(rating - mu_hat - b_u -
            b_m - b_g)/(n() + l))

    predicted <- train_set %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_m, by = "movieId") %>%
        left_join(b_g, by = "genres") %>%
        left_join(b_y, by = "year") %>%
        mutate(predicted = mu_hat + b_u + b_m + b_g +
            b_y) %>%
        .$predicted

    return(RMSE(predicted, train_set$rating))
})

qplot(lambda, sim_rmse)
```



Picking the **lambda** with the minimum RMSE:

```
lambda <- lambda[which.min((sim_rmse))]
```

The ideal **lambda** is 6.25.

We can now perform an assessment on the test dataset:

```
b_u <- train_set %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat)/(n() + lambda))

b_m <- train_set %>%
    left_join(b_u, by = "userId") %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat - b_u)/(n() +
        lambda))

b_g <- train_set %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_hat - b_u - b_m)/(n() +
        lambda))

b_y <- train_set %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu_hat - b_u - b_m -
        b_g)/(n() + lambda))

predicted <- test_set %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "year") %>%
    mutate(predicted = mu_hat + b_u + b_m + b_g + b_y) %>%
    .$predicted

lr_reg_rmse <- RMSE(predicted, test_set$rating)
```

The RMSE after regularization is 0.8809226. There was no big improvement in RMSE using regularization, though.

4.5.3.6 Matrix Factorization

The last attempts to improve RMSE have not worked as expected, namely considering the genre and date, and performing regularization. Therefore, let's get back to the model that considers only the variation caused by the user and the movie:

$$Y_{u,m} = \mu + b_u + b_m + \epsilon_{u,m}$$

That model does not take into account the fact that groups of movies are rated similarly and groups of users have similar rating patterns. Those patterns can be discovered through residuals, that are calculated using this formula:

$$r_{u,i} = Y_{u,m} - \hat{\mu} - \hat{b_u} - \hat{b_m}$$

Let's calculate the residuals in both training and testing datasets:

```
# Calculating the user effect again without
# regularization
b_u <- train_set %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat))

# Calculating the movie effect again without
# regularization
b_m <- train_set %>%
    left_join(b_u, by = "userId") %>%
    group_by(movieId) %>%
    summarize(b_m = mean(rating - mu_hat - b_u))

# Calculating the residuals on the training set
train_set <- train_set %>%
```

```r
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    mutate(residual = rating - mu_hat - b_u - b_m)

# Calculating the residuals on the testing set
test_set <- test_set %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    mutate(residual = rating - mu_hat - b_u - b_m)
```

To help the usage of matrix factorization, we are going to use the **recosystem package**, which can be obtained by running the following code:

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

library(recosystem)
```

Once we have the package installed, we need to provide the data in the expected format:

```r
# Providing training data
train_reco <- data_memory(user_index = train_set$userId,
    item_index = train_set$movieId, rating = train_set$residual,
    index1 = T)

# Providing test data
test_reco <- data_memory(user_index = test_set$userId,
    item_index = test_set$movieId, rating = test_set$residual,
    index1 = T)

# Creating the model
recommender <- Reco()
```

Then, we need to optimize the parameters to minimize the RMSE.

```r
# Setting the seed to always obtain the same
# result
set.seed(100)

# Tune the parameters
reco_parameters <- recommender$tune(train_reco, opts = list(dim = c(10,
    20, 30), costp_l1 = 0, costq11 = 0, lrate = c(0.05,
    0.1, 0.2), nthread = 2))

# Shows the parameters
print(reco_parameters$min)
```

The code above was run, but since it takes a long time to execute, these are the final results:

$dim [1] 30

$costp_l1 [1] 0

$costp_l2 [1] 0.01

$costq_l1 [1] 0

$costq_l2 [1] 0.1

$lrate [1] 0.05

$loss_fun [1] 0.804404

Now that we have the optimal parameters, we can train the model using the training data.

```r
# Setting the seed to always obtain the same
# result
set.seed(100)

suppressWarnings(recommender$train(train_reco, opts = c(dim = 30,
    costp11 = 0, costp12 = 0.01, costq11 = 0, costq12 = 0.1,
    lrate = 0.05, verbose = FALSE)))
```

After that, we can make predictions using the test data and check the resulting RMSE:

```r
reco_test_predictions <- recommender$predict(test_reco,
    out_memory()) + mu_hat + test_set$b_u + test_set$b_m

# If any prediction went beyond 5, set it to 5
index <- which(reco_test_predictions > 5)
reco_test_predictions[index] <- 5

# If any prediction is less than 0.5, set it to
# 0.5
index <- which(reco_test_predictions < 0.5)
reco_test_predictions[index] <- 0.5

# Calculate the RMSE
lr_matrix_rmse <- RMSE(reco_test_predictions, test_set$rating)
```

The RMSE obtained is 0.8393085.

Since the RMSE is under the expected 0.8649 value, let's now calculate the final RMSE on the validation (edx) dataset.

```r
# Calculating the user effect again without
# regularization
b_u <- edx %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat))

# Calculating the movie effect again without
# regularization
b_m <- edx %>%
    left_join(b_u, by = "userId") %>%
    group_by(movieId) %>%
    summarize(b_m = mean(rating - mu_hat - b_u))

# Calculating the residuals on the edx set
edx <- edx %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    mutate(residual = rating - mu_hat - b_u - b_m)

# Calculating the residuals on the validation set
validation <- validation %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_m, by = "movieId") %>%
    mutate(residual = rating - mu_hat - b_u - b_m)

# Providing training data
edx_reco <- data_memory(user_index = edx$userId, item_index = edx$movieId,
    rating = edx$residual, index1 = T)

# Providing test data
validation_reco <- data_memory(user_index = validation$userId,
    item_index = validation$movieId, rating = validation$residual,
    index1 = T)

# Creating the model
final_recommender <- Reco()

# Setting the seed to always obtain the same
# result
set.seed(100)

# Training the model using the edx set and the
# same parameters obtained on the tuning phase
suppressWarnings(final_recommender$train(edx_reco,
    opts = c(dim = 30, costp11 = 0, costp12 = 0.01,
        costq11 = 0, costq12 = 0.1, lrate = 0.05, verbose = FALSE)))

reco_final_predictions <- final_recommender$predict(validation_reco,
    out_memory()) + mu_hat + validation$b_u + validation$b_m

# If any prediction went beyond 5, set it to 5
index <- which(reco_final_predictions > 5)
reco_final_predictions[index] <- 5

# If any prediction is less than 0.5, set it to
# 0.5
index <- which(reco_final_predictions < 0.5)
reco_final_predictions[index] <- 0.5

# Calculate the RMSE
final_rmse <- RMSE(reco_final_predictions, validation$rating)
```

The RMSE obtained on the validation set was 0.8358525.

5 Conclusion

The best solution was obtained by performing a linear regression considering the user and the movie being rated. The other data that could have an impact on predictions was considered by calculating the residuals and modeling them using matrix factorization.

Using the **edx** data set for training, we achieved an RMSE of 0.8358525 on the **validation** set.

Due to the dataset size, it was not possible to explore other approaches, such as:

- combining the predictors (e.g., users and genres to find the favorite genres for each user) and use the Loess approach
- computing similarity between items
- using neural networks

Future works could include those alternative approaches, but that would require running them on the cloud, where processing power and memory constraints are not a limiting factor as in a personal laptop.