PH125.9x Data Science: MovieLens Capstone Project

Rodrigo Dal Ben de Souza

December 28, 2021 - Last update: February 13, 2022

Contents

1 Introduction

This project is part of the edX-HarvardX PH125.9x Data Science course. Our goal is to use machine learning algorithms to build a movie recommendation system. Machine learning automatically generate data-driven insights and predictions about events. These algorithms are typically developed in two stages. First, using a subset of data with *known* values, we build a model to predict the known outcome. Second, we input our model with a similar subset of data, but this time with *unknown* values, and measure thee accuracy of our model in predicting these *unknown* values. How well our algorithm performs will tell us about the fit of our model to the data (i.e., how accurate it is).

To build our recommendation system, we will use movie ratings from the MovieLens 10M dataset, that contains 10 million movie ratings. This dataset will be divided into Training (90% of the data) and Test (10%) subsets. The Training subset, with *known* values, will be used during the algorithm development. The Test subset, with *unknown* values, will be used only when evaluating the models' accuracies. During evaluation, we will measure the Residual Mean Square Error (RMSE) as a measure of accuracy. Our target is a `RMSE < 0.86490`.

The key steps for building our system include: data preparation (cleaning and formatting), data exploration, visualization, modeling, and evaluation.

2 Method

2.1 Cleaning and formatting

2.1.1 Loading libraries

Here we install packages to be used during the project.
Obs. The evaluation of this code chunk is turned off, if you need to install any of the packages, turn it on.

```
# R version: 4.1.2
# packages - version
install.packages("tidyverse") # v1.3.1
install.packages("patchwork") # v1.1.1
install.packages("data.table") # v1.14.2
install.packages("caret") # v6.0-90
install.packages("here") # v1.0.1
```

Here we load the packages, create a vector with color blind friendly palette (`color_blind_colors`), and save our target RMSE.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(patchwork)
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(here)
```

```
## here() starts at /Users/rodrigo/Desktop/repositories/edx_data_science
```

```
color_blind_colors <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#
rmse_target <- 0.86490
```

2.1.2 Loading dataset

Here we use the code provided in the course to load and prepare the MovieLens 10M dataset.

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

# 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")

# 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)`
```

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

```
test_index <- caret::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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

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

2.2 Exploration & Visualization

Here we explore our data with summary statistics and visualizations. The Training subset is stored under the `edx` dataframe, the Test subset is stored under the `validation` dataframe. We will preserve our Test subset (`validation`) for model evaluation. So most summary statistics and visualizations will be made with the Training (`edx`) subset.

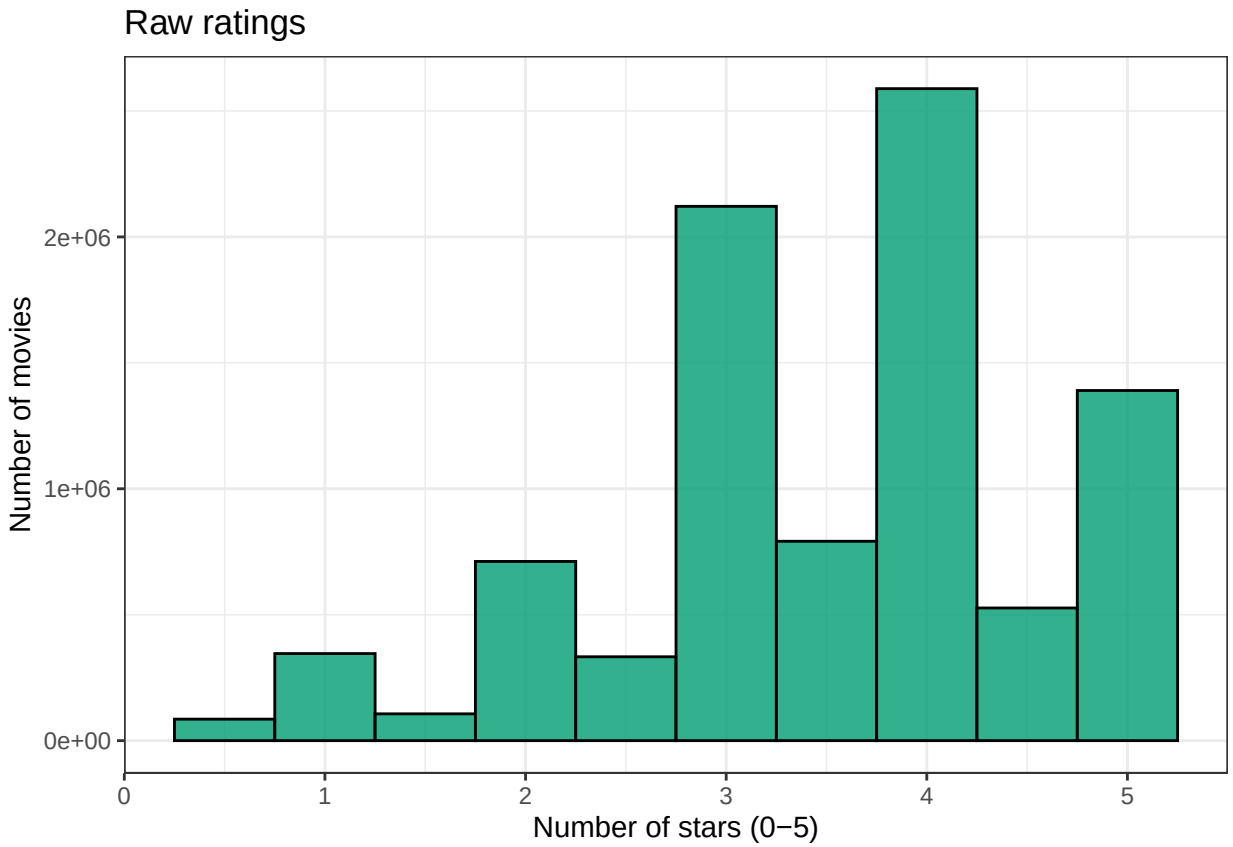
The Training subset is made of 6 columns, namely: userId, movieId, rating, timestamp, title, genres. It has 9000055 ratings or 90% of the data. Whereas the Test subset contains the same columns and is made of 999999 ratings, 10% of the data.

```
# unique movies & ratings
summ_movies <-
  edx %>%
  group_by(title) %>%
  summarise(n_ratings = n(),
            m_ratings = mean(rating),
            sd_ratings = sd(rating)
            ) %>%
  arrange(desc(n_ratings))

# unique users & ratings
summ_users <-
  edx %>%
  group_by(userId) %>%
  summarise(n_ratings = n(),
            m_ratings = mean(rating),
            sd_ratings = sd(rating)
            ) %>%
  arrange(desc(n_ratings))
```

The Training subset contains ratings on 10676 movies made by 69878 users. The movie that received the highest number of ratings was Pulp Fiction (1994), with 31362 ratings.

Overall, movies are frequently rated with either 3, 4, or 5 stars, and most users give full stars instead of fractions (e.g., 2.5 stars), see histogram below.

```
# number of ratings distributions
edx %>%
  ggplot(aes(x = rating)) +
  geom_histogram(binwidth = 0.5,
                 fill = color_blind_colors[4],
                 color = "black",
                 alpha = 0.8) +
  labs(title = "Raw ratings",
       x = "Number of stars (0-5)",
       y = "Number of movies") +
  theme_bw()
```

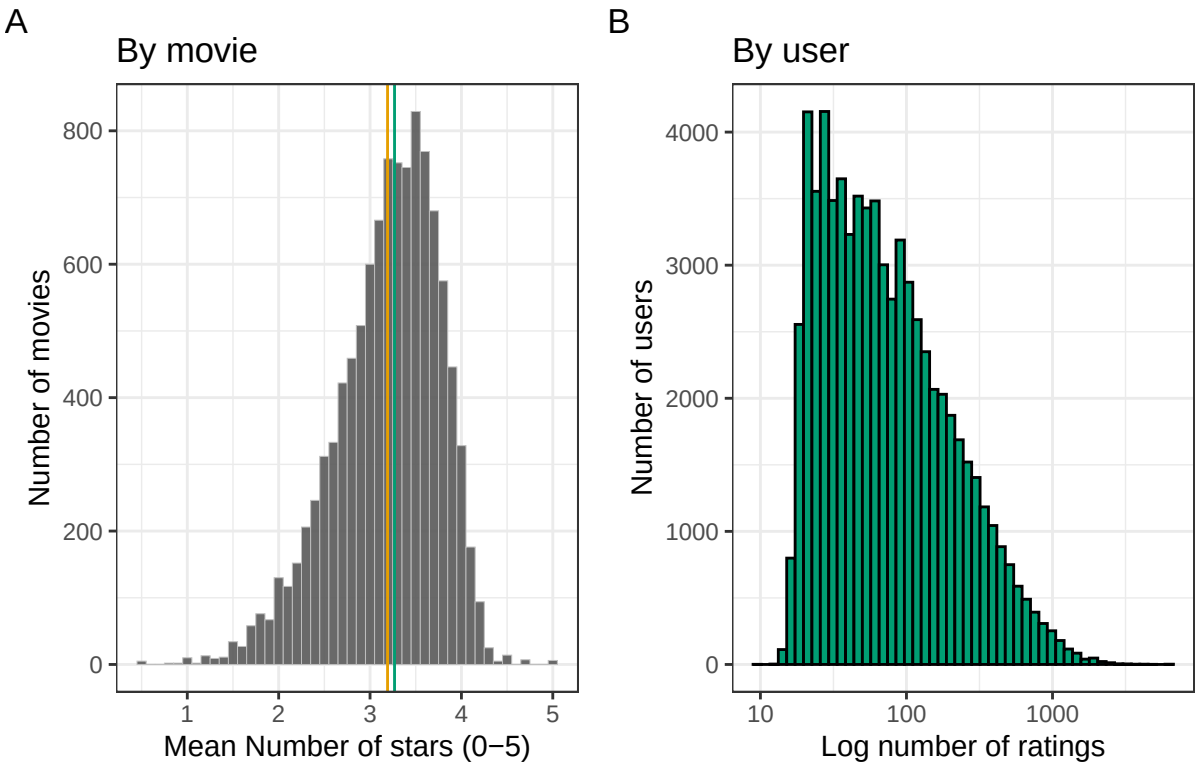


The histograms below indicate that the overall mean rating was $M = 3.19$ (yellow line, histogram A) and the overall median was $Median = 3.27$ (green line, histogram A). The summary of ratings by user indicate that on average users rated $M = 128.8$ movies. The distribution (histogram B) is positively skewed, most users rating hundreds of movies and a few users rating thousands of movies. These outliers can bias our models.

```
# ratings distributions by movie
hist_summary_ratings <-
  summ_movies %>%
  ggplot(aes(x = m_ratings)) +
  geom_histogram(binwidth = 0.1, color = "grey", size = 0, alpha = 0.9) +
  geom_vline(xintercept = mean(summ_movies$m_ratings), color = color_blind_colors[2]) +
  geom_vline(xintercept = median(summ_movies$m_ratings), color = color_blind_colors[4]) +
```

```
  labs(title = "By movie",
       x = "Mean Number of stars (0-5)",
       y = "Number of movies") +
  theme_bw()

# ratings distributions by user
hist_user_ratings <-
summ_users %>%
  ggplot(aes(x = n_ratings)) +
  geom_histogram(bins = 50,
                 fill = color_blind_colors[4],
                 color = "black") +
  scale_x_log10() +
  labs(title = "By user",
       x = "Log number of ratings",
       y = "Number of users") +
  theme_bw()

hist_summary_ratings + hist_user_ratings + plot_annotation(tag_levels = 'A', title = "Ratings distributi
```

Ratings distributions

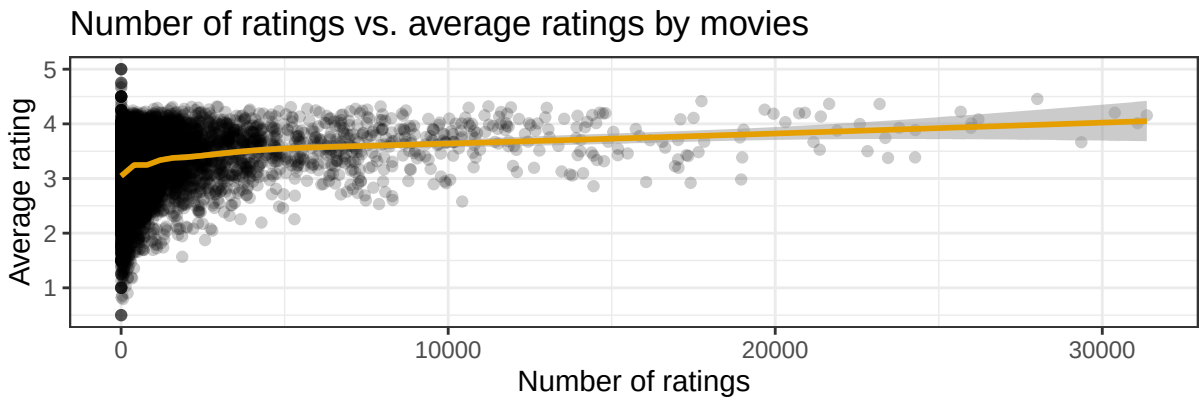


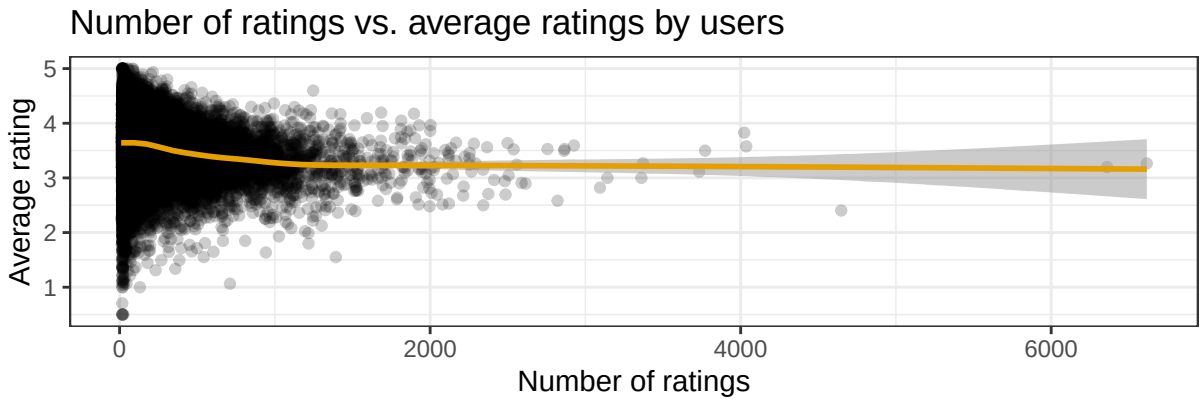
The graphs below indicate that movies that are rated more often usually have higher rates in comparison to movies that are not often rated. On the other hand, users that more frequently rate movies do not tend to give higher rates.

```
ratings_movies <-
  summ_movies %>%
  ggplot(aes(x = n_ratings, y = m_ratings)) +
```

```
  geom_point(alpha = 0.2) +
  geom_smooth(alpha = 0.5, color = color_blind_colors[2]) +
  labs(title = "Number of ratings vs. average ratings by movies",
       x = "Number of ratings",
       y = "Average rating") +
  theme_bw()

ratings_users <-
  summ_users %>%
  ggplot(aes(x = n_ratings, y = m_ratings)) +
  geom_point(alpha = 0.2) +
  geom_smooth(alpha = 0.5, color = color_blind_colors[2]) +
  labs(title = "Number of ratings vs. average ratings by users",
       x = "Number of ratings",
       y = "Average rating") +
  theme_bw()

ratings_movies / ratings_users
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```





Our models must account for ratings variability within movies and users. To do so we will build a series of regression models.

2.3 Models

We will build a sequence of five linear models to account for the effects of users and movies (predictors) on ratings (outcome):

1. Baseline model with the mean rating (μ) as the only predictor (plus a measurement error, ϵ): $Y_{u,m} = \mu + \epsilon_{u,m}$
2. User effects (b_u) as a predictor: $Y_{u,m} = \mu + b_u + \epsilon_{u,m}$
3. Movie effects (b_m) as a predictor: $Y_{u,m} = \mu + b_m + \epsilon_{u,m}$
4. User and movie effects (b_u, b_m) as predictors: $Y_{u,m} = \mu + b_u + b_m + \epsilon_{u,m}$
5. To account for outliers both in users effects (some users rate thousands of movies whereas others rate just a few, n) and movie effects (some movies get a lot more ratings than others, n), we will regularize (λ) our predictors (user and movie): $Y_{u,m} = \mu + b_{u,n,\lambda} + b_{m,n,\lambda} + \epsilon_{u,m}$

Finally, to evaluate the accuracy of each model we will use the Residual Mean Square Error (RMSE):

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

3 Results

Here we *fit* the models and *evaluate* the fit using RMSE for each model. We will add the RMSE for each model in a table to facilitate comparison across models.

3.1 Model 01: Baseline

We start by building a baseline model that predict movie rating (Y) based only on the expected mean rating (μ) for all users (u) and movies (m), plus a measurement error (ϵ):

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

```
# calculate the mean (Training subset)
mu <- mean(edx$rating)

# calculate the RMSE (Test subset)
rmse_baseline <- caret::RMSE(validation$rating, mu)

# create table with RMSE
rmse_scores <- tibble(Model = "Baseline",
                      RMSE = rmse_baseline,
                      `Below Target RMSE` = if_else(RMSE < rmse_target, "Yes", "No"))

rmse_scores %>% knitr::kable()
```

Model	RMSE	Below Target RMSE
Baseline	1.061202	No

The RMSE score resulting from the baseline model () indicate that it do not produce good predictions for movie ratings. Now we will we add another predictor to our model: user effects.

3.2 Model 02: User effects

Now we will add user effects (b_u) to account for variance arising from different rating patterns between users.

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

```
# calculate average deviance for each user
user_avg <-
  edx %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu))

# predict rating
pred_rating <-
  mu + validation %>%
  left_join(user_avg, by = "userId") %>%
  pull(b_u)

# save rmse
rmse_user_effect <- caret::RMSE(pred_rating, validation$rating)
rmse_scores <- bind_rows(rmse_scores,
                         tibble(Model = "User effects",
                                RMSE = rmse_user_effect,
                                `Below Target RMSE` = if_else(RMSE < rmse_target, "Yes", "No")))

rmse_scores %>% knitr::kable()
```

Model	RMSE	Below Target RMSE
Baseline	1.061202	No
User effects	0.978336	No

When accounting for user effects, our model produces an RMSE of 0.9783, a improvement of 0.0829 over our baseline model. Now we will add movie effects.

3.3 Model 03: Movie effects

Another source of variance in our data comes from movies ratings. For instance, some movies get higher and more frequent ratings than others. We will add movies' ratings to our **baseline model** to measure movie effects (b_m) on our model:

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

```r
# calculate average deviance for each user
movie_avg <-
  edx %>%
  group_by(movieId) %>%
  summarise(b_m = mean(rating - mu))

# predict rating
pred_rating <-
  mu + validation %>%
  left_join(movie_avg, by = "movieId") %>%
  pull(b_m)

# save rmse
rmse_movie_effect <- caret::RMSE(pred_rating, validation$rating)
rmse_scores <- bind_rows(rmse_scores,
                         tibble(Model = "Movie effects",
                                RMSE = rmse_movie_effect,
                                `Below Target RMSE` = if_else(RMSE < rmse_target, "Yes", "No")))

rmse_scores %>% knitr::kable()
```

Model	RMSE	Below Target RMSE
Baseline	1.0612018	No
User effects	0.9783360	No
Movie effects	0.9439087	No

When accounting for movie effects, our model produces an RMSE of 0.9439, a improvement of 0.1173 over our baseline model. Now we will combine user and movie effects in a single model.

3.4 Model 04: Movie and User effects

Now we will account for user (b_u) and movie (b_m) effects on the same model.

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

```r
# predict rating
pred_rating <-
  validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  mutate(pred = mu + b_m + b_u) %>%
  pull(pred)

# save rmse
rmse_user_movie_effect <- caret::RMSE(pred_rating, validation$rating)
rmse_scores <- bind_rows(rmse_scores,
                         tibble(Model = "User and movie effects",
                                RMSE = rmse_user_movie_effect,
                                `Below Target RMSE` = if_else(RMSE < rmse_target, "Yes", "No")))

rmse_scores %>% knitr::kable()
```

Model	RMSE	Below Target RMSE
Baseline	1.0612018	No
User effects	0.9783360	No
Movie effects	0.9439087	No
User and movie effects	0.8850398	No

When accounting for user and movie effects, our model produces an RMSE of 0.885, a improvement of 0.1762 over our baseline model. This combined model is a better fit to our data than previous models that accounted only for users' or movies' effects.

3.5 Model 05: Regularization dealing with bias

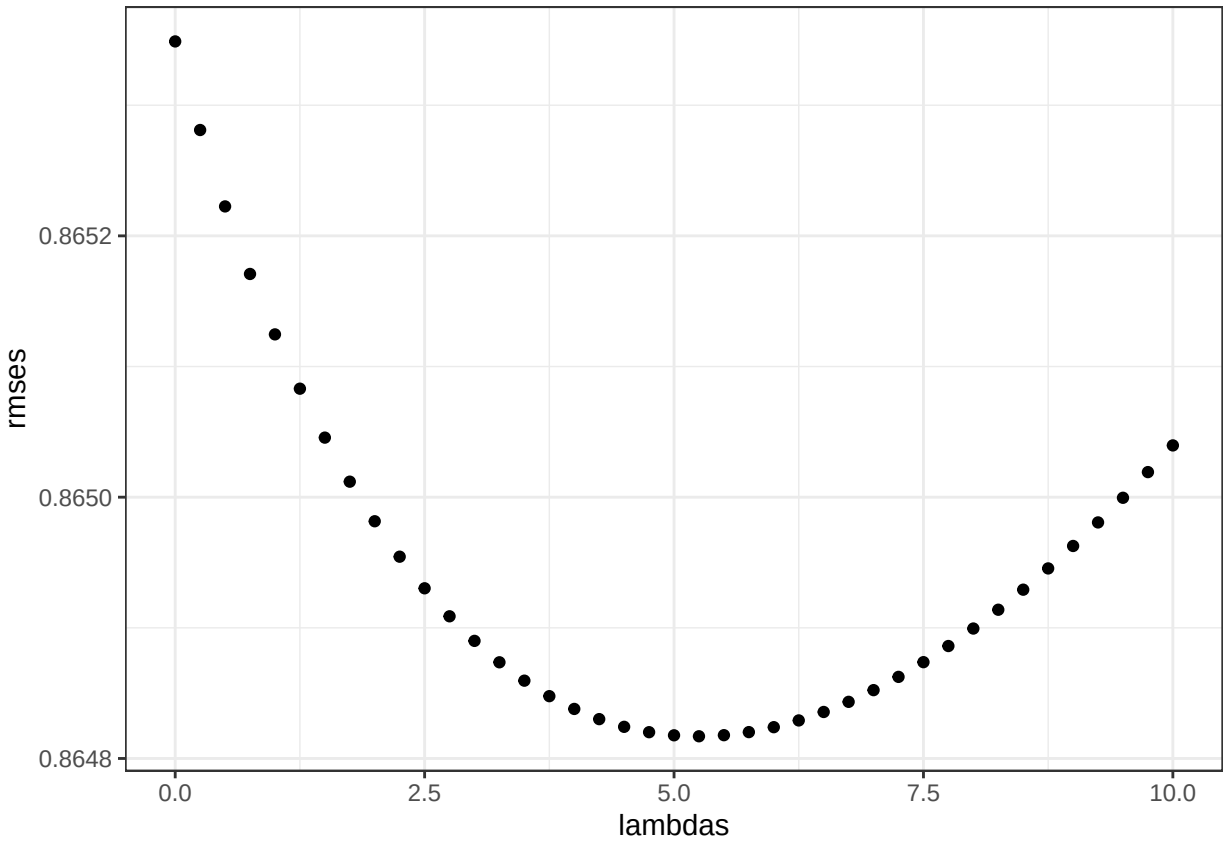
To reduce errors caused by outliers (n; see visualizations on the previous section) in user and movie effects (some users rate very few movies, some movies receive very few ratings), we can use a tuning parameter (λ) to penalize our regression model.

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

We will use cross validation (based on section 34.9.3 of our textbook) to find the tuning parameter score that return the smallest RMSE–which we will add to our table.

```
# create lambda
lambdas <- seq(0, 10, 0.25)

# find lambda that minimizes errors
rmses <- sapply(lambdas, function(l){

  # mu already calculated

  # calculate user effects
  b_movie <-
    edx %>%
    group_by(movieId) %>%
    summarise(b_movie = sum(rating - mu)/(n()+l))

  # calculate movie effects
  b_user <-
    edx %>%
    left_join(b_movie, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_user = sum(rating - b_movie - mu)/(n()+l))

  # predict ratings using the test dataset
  predicted_ratings <-
    validation %>%
    left_join(b_user, by = "userId") %>%
    left_join(b_movie, by = "movieId") %>%
    mutate(pred = mu + b_user + b_movie) %>%
    pull(pred)

  # return rmses based on lambdas
  return(caret::RMSE(predicted_ratings, validation$rating))
```

```
  })

# plot
qplot(x = lambdas, y = rmses) + theme_bw()
```




```
# find min rmse and best tunning parameter
min_rmse <- min(rmses)
lambda <- lambdas[which.min(rmses)]

# save rmse
rmse_scores <- bind_rows(rmse_scores,
                         tibble(Model = "User and movie effects regularized",
                                RMSE = min_rmse,
                                `Below Target RMSE` = if_else(RMSE < rmse_target, "Yes", "No"))
                         )

rmse_scores %>% knitr::kable()
```

Model	RMSE	Below Target RMSE
Baseline	1.0612018	No
User effects	0.9783360	No
Movie effects	0.9439087	No
User and movie effects	0.8850398	No
User and movie effects regularized	0.8648170	Yes

Regularization of both user and movies effects controlled for the influence of outliers in our data. Our regularized model produces an RMSE of 1, a improvement of 0.1964 over our baseline model. This model represent a better fit to our data than our previous models. It also allow us to achieve our target RMSE (`RMSE < 0.86490`)

4 Conclusion

After inspecting the data using descriptive statistics and visualizations, we created a machine learning algorithm to build an effective movie recommendation system. Using a sequence of linear models, we were able to account for the effects of users and movies as well as for outliers on both variables when predicting movie ratings. Our final model produced an RMSE = 0.864817, which is below the target RMSE (0.8649) for this project.