MovieLens 10M

Chirui GUO

2024-12-06

MovieLens Capstone Project

Introduction

Summary

This project is inspired by The Netflix Prize which was a groundbreaking competition launched by Netflix in October 2006 to improve its movie recommendation system. The goal of this competition was to significantly enhance the accuracy of Netflix's recommendation algorithm, Cinematch, by predicting user ratings for movies based on previous ratings.

MovieLens 10M Dataset

In this Capstone we use MovieLens 10M as our dataset. The MovieLens 10M dataset is a widely used benchmark dataset created by the GroupLens Research Lab at the University of Minnesota. It contains 10 million movie ratings and 100,000 tag applications, making it an excellent resource for developing and evaluating recommendation systems, data analysis, and machine learning models.

Provided data preprocessing codes

HarvardX PH125.9x provided these data preprocessing codes for the raw dataset. We can easily get the training set and testing set through this code.

```
##########################################
# Create edx and final_holdout_test sets
##########################################

# 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(ggtext)) install.packages("ggtext", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(ggtext)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
```

```
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings |>
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies |>
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp |>
  semi_join(edx, by = "movieId") |>
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

You really doesn't need to run these code every time. You can save these data in files.

```
# write_rds(edx, "edx.rds")
# write_rds(final_holdout_test, "final_holdout_test.rds")
```

Methods

Data transforming

```
## load the library
# library(tidyverse)
# library(caret)
# library(ggtext)

# load the edx and final_holdout_test
# edx <- read_rds("edx.rds")
# final_holdout_test <- read_rds("final_holdout_test.rds")
```

Though edx provided some data preprocessing code for us, but it's not enough for further analysis. We should do more for data transforming.

Firstly, Transforming `userId`, `movieId` and `genres` into **factor** type makes our data analysis easier.

Secondly, use `str_extract` and `str_trim` to split movies realse years (`year`) from `title`.

Lastly, convert `timestamp` to **POSIXct** type

```
### Transforming `userId`, `movieId` and `genres` into factor type
### split movies realse years (`year`) from `title`
### convert `timestamp` to **POSIXct** type
edx <- edx |>
  mutate(
    userId = as.factor(userId),
    movieId = as.factor(movieId),
    year = as.numeric(str_extract(title, "(?<=\\()\\d{4}(?=\\))")),
    title = str_trim(str_remove(title, "\\(\\d{4}\\)")),
    genres = as.factor(genres),
    timestamp =  as.POSIXct(timestamp, origin = "1970-01-01")
  ) |>
  select(userId, movieId, rating, year, title, genres, timestamp)

final_holdout_test <- final_holdout_test |>
  mutate(
    userId = as.factor(userId),
    movieId = as.factor(movieId),
    year = as.numeric(str_extract(title, "(?<=\\()\\d{4}(?=\\))")),
    title = str_trim(str_remove(title, "\\(\\d{4}\\)")),
    genres = as.factor(genres),
    timestamp =  as.POSIXct(timestamp, origin = "1970-01-01")
  ) |>
  select(userId, movieId, rating, year, title, genres, timestamp)
```

Exploratory Analysis

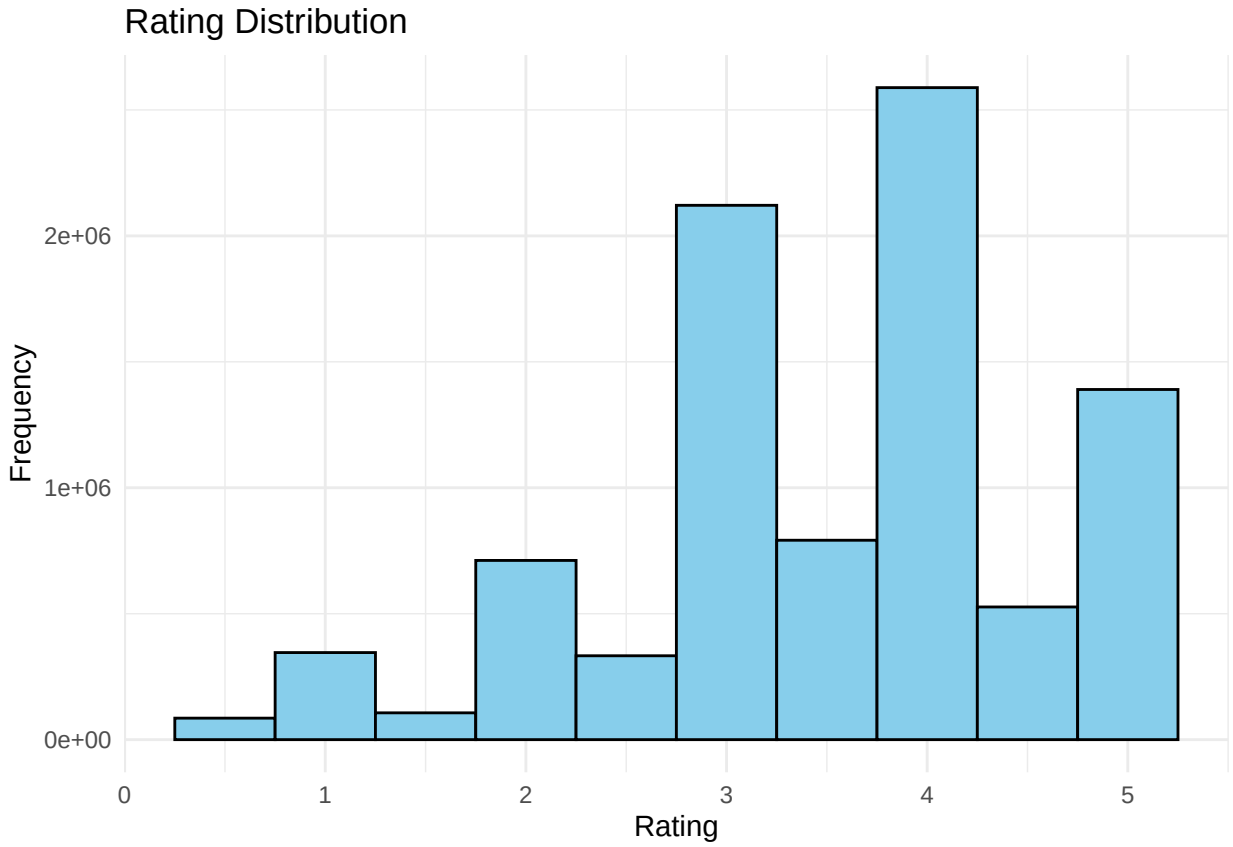
The distribution of ratings is concentrated around 4, and people tend to give integer number scores to movies.

```
summary(edx$rating)
```

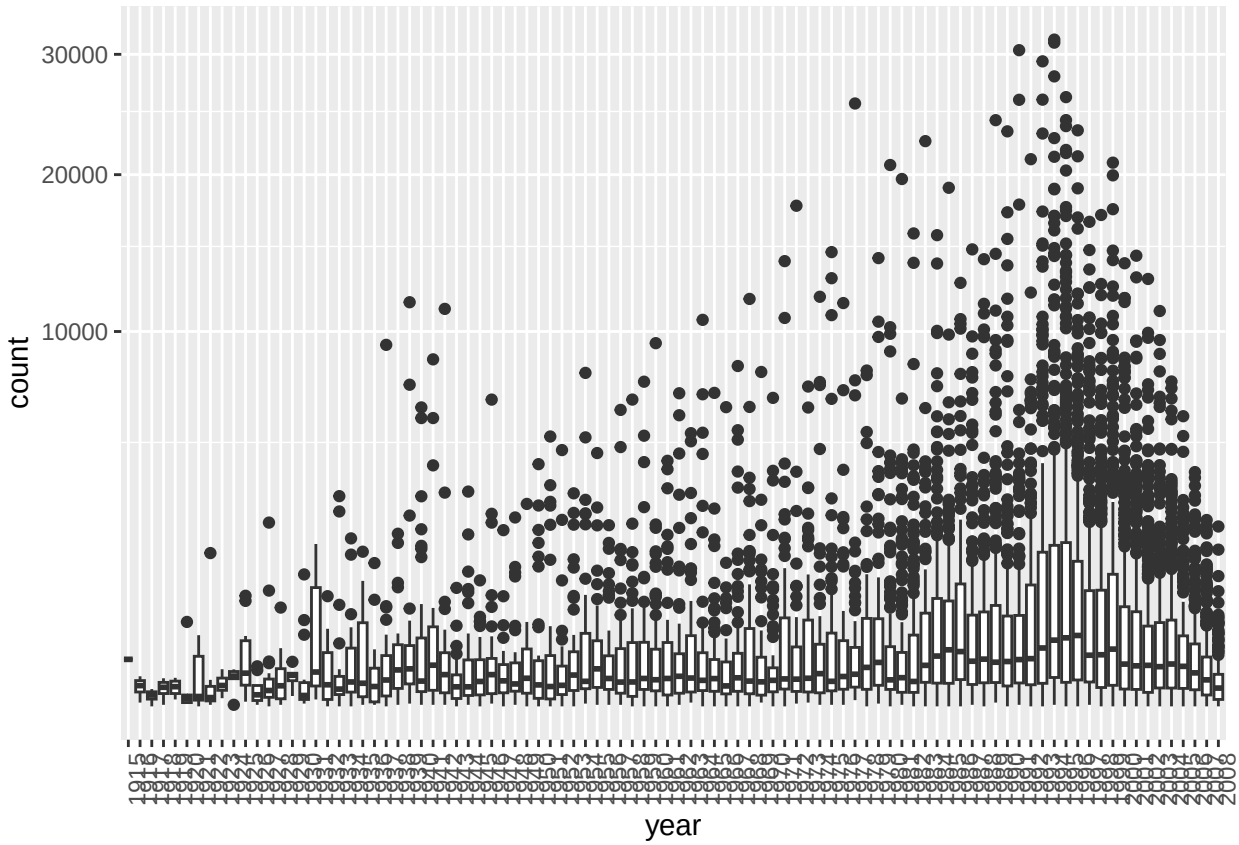
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

```
edx |>
  ggplot(aes(x = rating)) +
```

```
  geom_histogram(
    fill = "skyblue",
    color = "black",
    binwidth = 0.5
  ) +
  labs(
    title = "Rating Distribution",
    x = "Rating",
    y = "Frequency"
  ) +
  theme_minimal()
```




```
edx |>
  group_by(movieId) |>
  summarise(count = n(), year = as.character(first(year))) |>
  ggplot(aes(year, count)) +
  geom_boxplot() +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We see that, on average, movies that came out after 1995 get more ratings. We also see that with newer movies, starting in 1995, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

The model

We begin with a model that posits a uniform rating for every movie and user, attributing any variations to random fluctuations. If μ denotes the actual rating applicable to all movies and users, and ε signifies independent errors drawn from a common distribution centered around 0, then:

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

In this scenario, the least squares estimate of μ, which minimizes the root mean squared error, is determined by calculating the average rating of all movies from all users. We can improve our model by adding a term, b_i that represents the average rating for movie i.

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

We can further improve our model by adding b_u the user-specific effect:

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

To evaluate various models or to assess our performance against a baseline, we will utilize root mean squared error (RMSE) as our loss function. RMSE can be understood in a manner akin to standard deviation.

If we assume that N is the number of user-movie combinations, $y_{u,i}$ is the rating for movie i which gives by user u, and $\hat{y}_{u,i}$ is our prediction, then **RMSE** is defined as follows:

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

the code is:

```
RMSE <- function(true_ratings, predicted_ratings){
     sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

```
edx <- edx |>
  select(userId, movieId, rating)
# Create the index
test_index <- createDataPartition(edx$rating, times = 1, p = .2, list = F)
# Create Train set
train <- edx[-test_index, ]
# Create Test set
test <- edx[test_index, ]

# Remove The same movieId and usersId appears in both set.
test <- test |>
  semi_join(train, by = "movieId") |>
  semi_join(train, by = "userId")
```

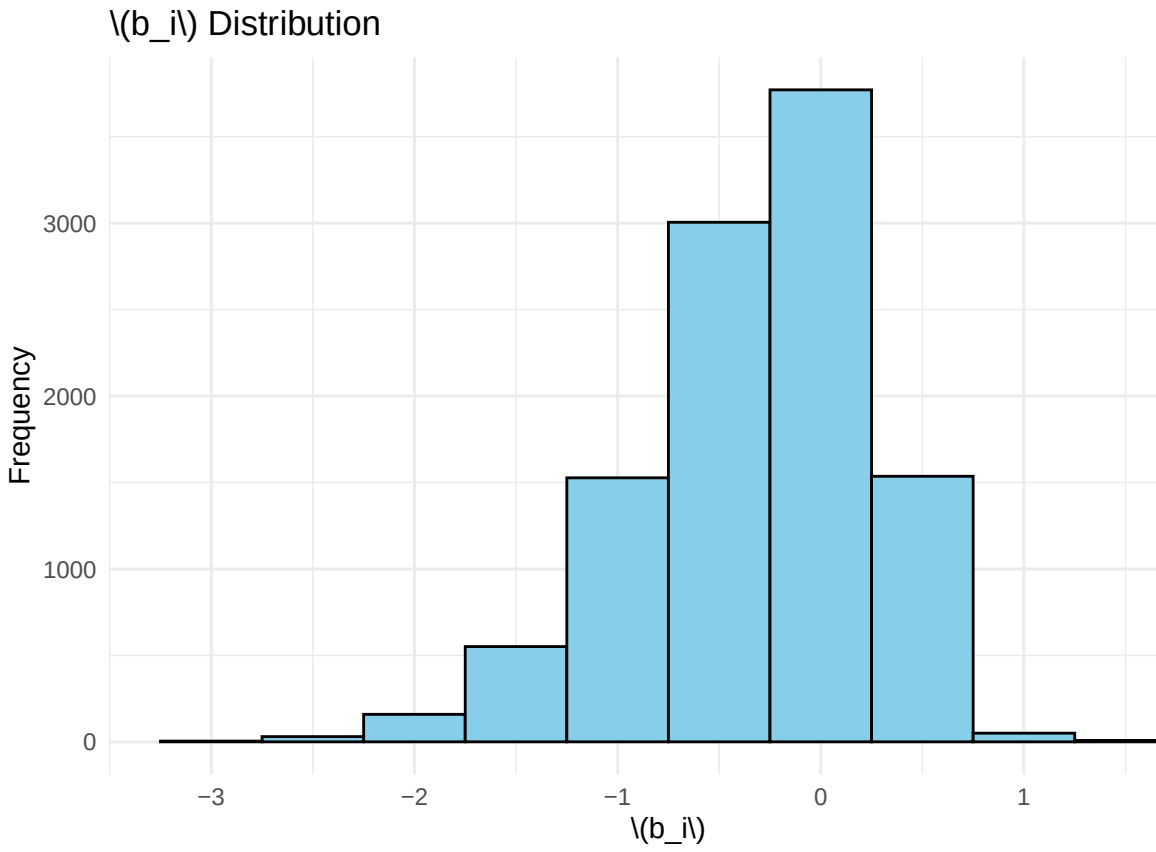
Train and Test set

```
mu <- mean(train$rating)
naive_train_rmse <- RMSE(train$rating, mu)
rmse_results <- tibble(method = "Just the average", dataset="train", RMSE = naive_train_rmse)

naive_test_rmse <- RMSE(test$rating, mu)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Just the average",
                                     dataset="test",
                                     RMSE = naive_test_rmse ))
```

Naive Model

```
# Compute the b_i for movie effect
B_i <- train |>
     group_by(movieId) |>
     summarize(b_i = mean(rating - mu))

# plot the distribution of b_i
B_i |>
  ggplot(aes(b_i)) +
  geom_histogram(bins = 10,
                 fill = "skyblue",
                 color = "black", ) +
  labs(
    title = "$b_i$ Distribution",
```

```
    x = "$b_i$",
    y = "Frequency"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_markdown(),
    axis.title.x = element_markdown(),
    axis.title.y = element_markdown()
  )
```

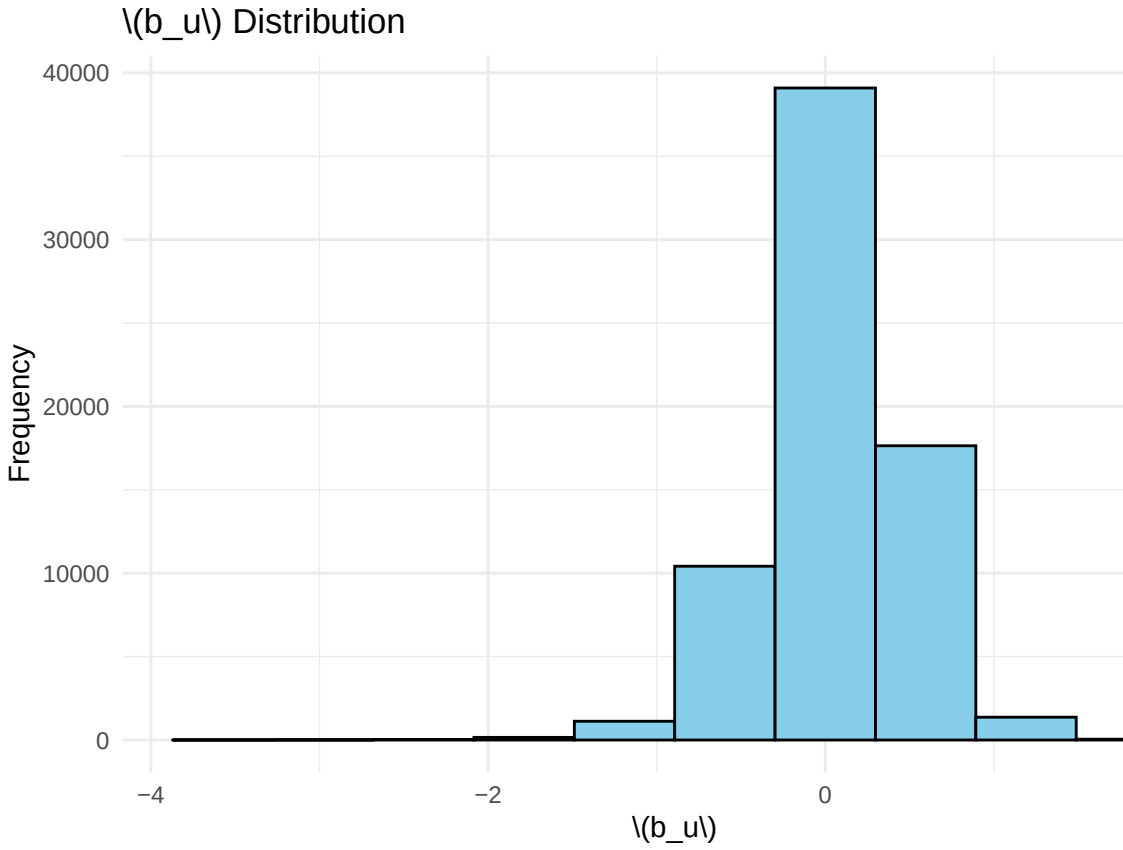


Movie Effects Model

```
# use the model to predict rating on train set
predicted_ratings <- mu + train |>
     left_join(B_i, by='movieId') |>
     with(b_i)
# compute rmse on train set
M_train_rmse <- RMSE(predicted_ratings, train$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 dataset="train",
                                 RMSE = M_train_rmse))

# use the model to predict rating on test set
predicted_ratings <- mu + test |>
     left_join(B_i, by='movieId') |>
     with(b_i)
# compute rmse on test set
M_test_rmse <- RMSE(predicted_ratings, test$rating)
```

```r
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 dataset="test",
                                 RMSE = M_test_rmse))
```

```r
B_u <- train |>
     left_join(B_i, by='movieId') |>
     group_by(userId) |>
     summarize(b_u = mean(rating - mu - b_i))

B_u |>
  ggplot(aes(b_u)) +
  geom_histogram(bins = 10,
                 fill = "skyblue",
                 color = "black", ) +
  labs(
    title = "$b_u$ Distribution",
    x = "$b_u$",
    y = "Frequency"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_markdown(),
    axis.title.x = element_markdown(),
    axis.title.y = element_markdown()
  )
```



Movie and User Effects

```
# use the model to predict rating on train set
predicted_ratings <- train |>
     left_join(B_i, by='movieId') |>
     left_join(B_u, by='userId') |>
     mutate(pred = mu + b_i + b_u) |>
     with(pred)
# compute rmse on train set
MU_train_rmse <- RMSE(predicted_ratings, train$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effect Model",
                                     dataset="train",
                                     RMSE = MU_train_rmse ))

# use the model to predict rating on test set
predicted_ratings <- test |>
     left_join(B_i, by='movieId') |>
     left_join(B_u, by='userId') |>
     mutate(pred = mu + b_i + b_u) |>
     with(pred)
# compute rmse on test set
MU_test_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effect Model",
                                     dataset="test",
                                     RMSE = MU_test_rmse ))
```

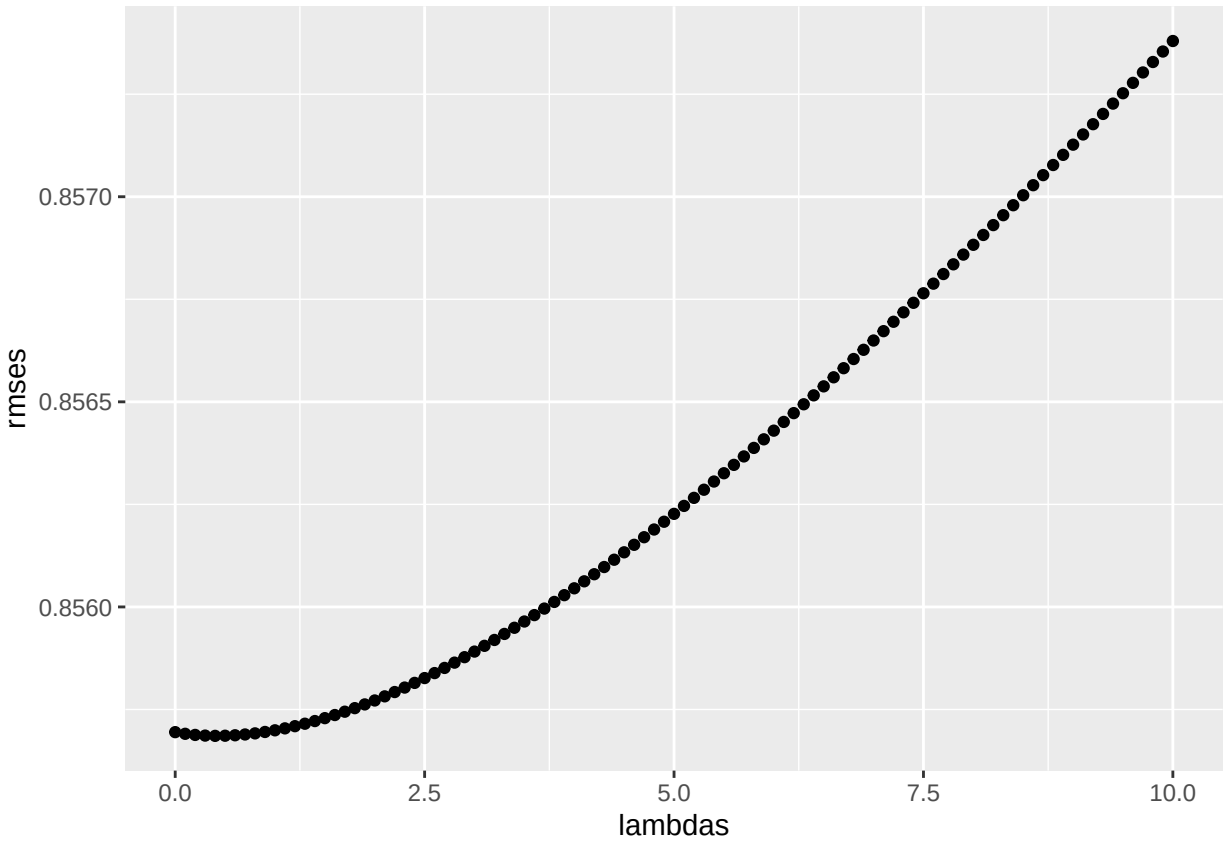
Regularization

To enhance our outcomes, we will implement regularization. This technique limits the overall variability of effect sizes by imposing penalties on large estimates derived from small sample sizes.

We can use regularization to estimate the **movie effect** and **user effect**. We will now minimize this equation:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2)$$

```
## the lambda's range
lambdas <- seq(0, 10, 0.1)

## define pipeline of finding appropriate lambda
pipeline <- function(l, dataset){
     mu <- mean(dataset$rating)
     B_i <- dataset |>
          group_by(movieId) |>
          summarize(b_i = sum(rating - mu)/(n()+l))
     B_u <- dataset |>
          left_join(B_i, by="movieId") |>
          group_by(userId) |>
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     predicted_ratings <-
          dataset |>
          left_join(B_i, by = "movieId") |>
          left_join(B_u, by = "userId") |>
          mutate(pred = mu + b_i + b_u) |>
          with(pred)
     return(RMSE(predicted_ratings, dataset$rating))
}

# use train set to find the appropriate lambda
rmses <- sapply(lambdas, function(l){pipeline(l, dataset=train)})
rmses <- tibble(lambdas = lambdas, rmses = rmses)
rmses |>
  ggplot(aes(lambdas, rmses)) +
  geom_point()
```




```
out <- rmses |>
  filter(rmses==min(rmses))
lambda_0 <- out |> pull(lambdas)
MUR_train_rmse <- out |> pull(rmses)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effect Model with Regularization",
                                 dataset="train",
                                 RMSE = MUR_train_rmse))

# use lambda_0 to calculate rmse on test set
MUR_test_rmse <- pipeline(lambda_0, dataset=test)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effect Model with Regularization",
                                 dataset="test",
                                 RMSE = MUR_test_rmse))
```

Results

Movie and User Effects model with regularization On final_holdout_test

We can use lambda_0 to calculate rmse on valid set

```
## use lambda_0 to calculate rmse on valid set
MUR_valid_rmse <- pipeline(lambda_0, dataset=final_holdout_test)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effect Model with Regularization",
                                 dataset="valid",
                                 RMSE = MUR_valid_rmse))
```

```
rmse_results
```

```
## # A tibble: 9 x 3
##   method                                             dataset  RMSE
##   <chr>                                              <chr>   <dbl>
## 1 Just the average                                   train   1.06
## 2 Just the average                                   test    1.06
## 3 Movie Effect Model                                 train   0.942
## 4 Movie Effect Model                                 test    0.944
## 5 Movie and User Effect Model                        train   0.856
## 6 Movie and User Effect Model                        test    0.866
## 7 Movie and User Effect Model with Regularization train   0.856
## 8 Movie and User Effect Model with Regularization test    0.841
## 9 Movie and User Effect Model with Regularization valid   0.826
```

In the last, We can observe that the optimal RMSE is achieved by Movie and User Effect Model with regularization on both the training set and the test set.

We can conclude that the RMSE obtained from the Movie and User Effect model with regularization on the validation set represents our final model. This RMSE is achieved when $\lambda = 0.4$, resulting in a value of **0.82561**.

Conclusion

The variables `userId` and `movieId` possess sufficient predictive power to allow us to estimate how a user will rate a movie. This indicates that we can provide better movie recommendations tailored to individual users of the streaming service, potentially encouraging them to spend more time on the platform.

The RMSE of **0.82561** is quite acceptable, especially considering the limited number of predictors used. However, both Movie and User effects are strong enough to effectively predict the rating a specific user will assign to a movie.