MovieLens_RecommendationSystem

Chhaya Mudgal

12/2/2020

Introduction and Project Outline

Recommender Systems are systems that give recommendations to the user based on ratings available. It requires large amount of data set which is filtered, processed and trained. It looks at the different features available in the data look at the usage to make suggestions. There are different algorithms that can be used for building recommender Systems. 1) Collaborative Filtering, it is of 2 types a) Item Based b) User Based. 2) Content Based 3) Classification Model. In each outcome there are different set of predictors.

Project Problem- This is a Movie Lens Project to build a movie recommender system using the dataset provided in the assignment. This will require to train the data with different algorithms and compare the accuracy of the algorithm against the validation set. Following steps are taken to build a recommender system.

1) Load Data
2) Explore and Visualize data
3) Prepare Data.
4) Evaluate Algorithms.
5) Make Predictions and Present Results.

Load Data

Load Data and Install Library Packages

Using the script provided in the course Download data set Install necessary library packages Create edx Data Set and Validation Set (final hold-out test set)

Note: this process could take a couple of minutes

```r
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)
library(tibble)
#library(Metrics)
### MovieLens 10M dataset:
### https://grouplens.org/datasets/movielens/10m/
### http://files.grouplens.org/datasets/movielens/ml-10m.zip
```

```r
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)`
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)
```

Prepare Data

Create training and test sets to assess the accuracy of the models.

90 percent of edx data will be training and 10% will be test data set

```r
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

## Make sure userId and movieId in test set are also in train set
```

```r
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)

rm(test_index, temp, removed)
```

Explore and Visualize Data

In order to build model we first need to look at the data.

Dimensions of the data set

Lets find out the total number of columns and rows in the edx data set.

```r
dim(edx)
```

```
## [1] 9000055       6
```

Peek at the first 5 rows of the data

We peek at the dataset and find that the column names in the dataset are:

UserId, movieId, Rating, Timestamp, Title and Genre.

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

Summarize edx data

```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:9000055     Length:9000055    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
## 
```

Genres

The data set contains 797 different combinations of genres. Here is the list of the first six.

```r
edx_genres <- edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  head()
edx_genres
```

```
## # A tibble: 6 x 2
##   genres                                                n
##   <chr>                                             <int>
## 1 (no genres listed)                                    7
## 2 Action                                            24482
## 3 Action|Adventure                                  68688
## 4 Action|Adventure|Animation|Children|Comedy         7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy   187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX       66
```

Ratings

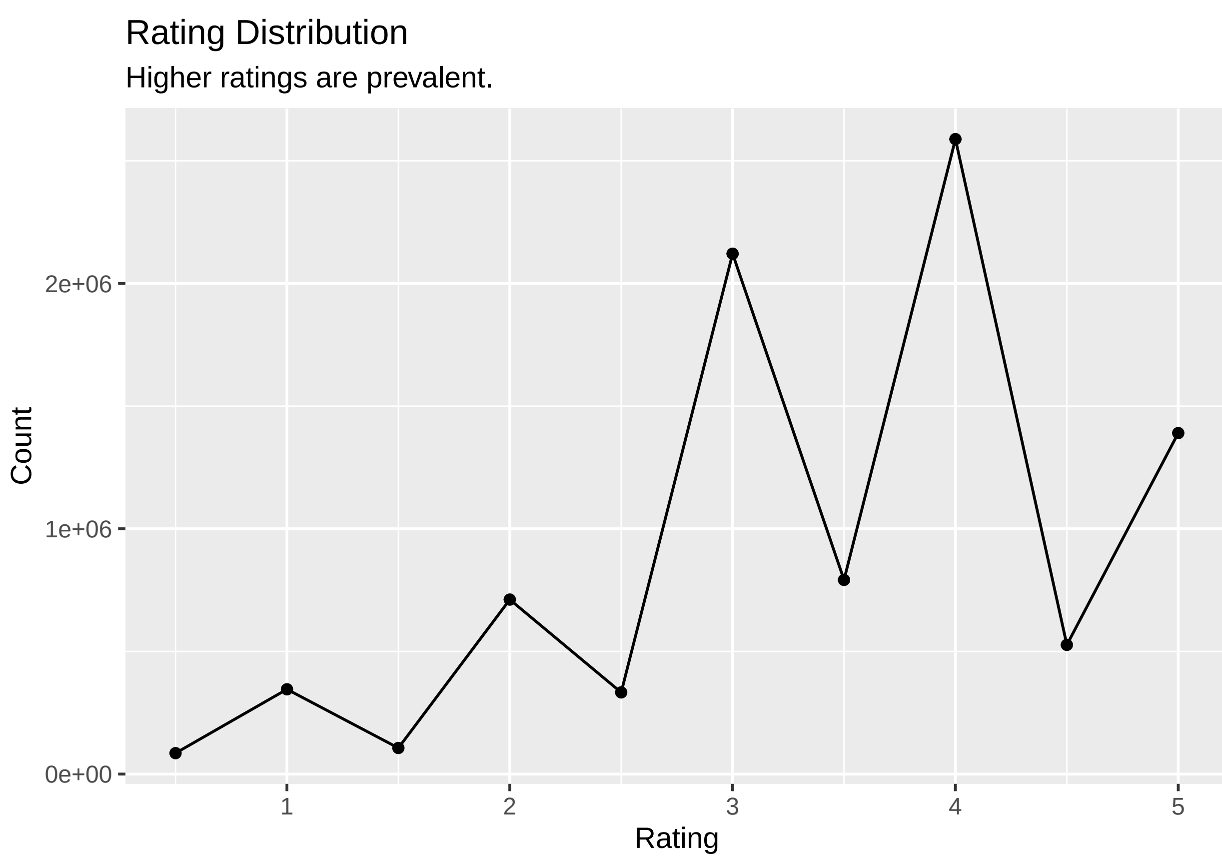
```r
edx_ratings <- edx %>% group_by(rating) %>% summarize(n=n())
edx_ratings
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
```

```
## 3     1.5  106426
## 4     2     711422
## 5     2.5  333010
## 6     3    2121240
## 7     3.5  791624
## 8     4    2588430
## 9     4.5  526736
## 10    5    1390114
```

Visualize Data

```
edx %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_line() +
  geom_point() +
  ggtitle("Rating Distribution", subtitle = "Higher ratings are prevalent.") +
  xlab("Rating") +
  ylab("Count")
```



Evaluate Algorithms

Loss Function It is a means to evaluate how specific algorithm behaves for a given data.If predictions deviates too much from actual results, loss function R will be a very large number. Optimization function help to reduce the error in prediction.

Define Mean Absolute Error (MAE)

Mean absolute error, is the average of sum of absolute differences between predictions and actual observations.

```
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}
```

Define Mean Squared Error (MSE)

Mean square error is the average of squared difference between predictions and actual observations.

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

Define Root Mean Squared Error (RMSE)

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

Simple Assumption Based Model

Model assumes same ratings for all users. If we predict all unknown ratings with mu_i we obtain the following RMSE:

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

```
## [1] 3.512456
```

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

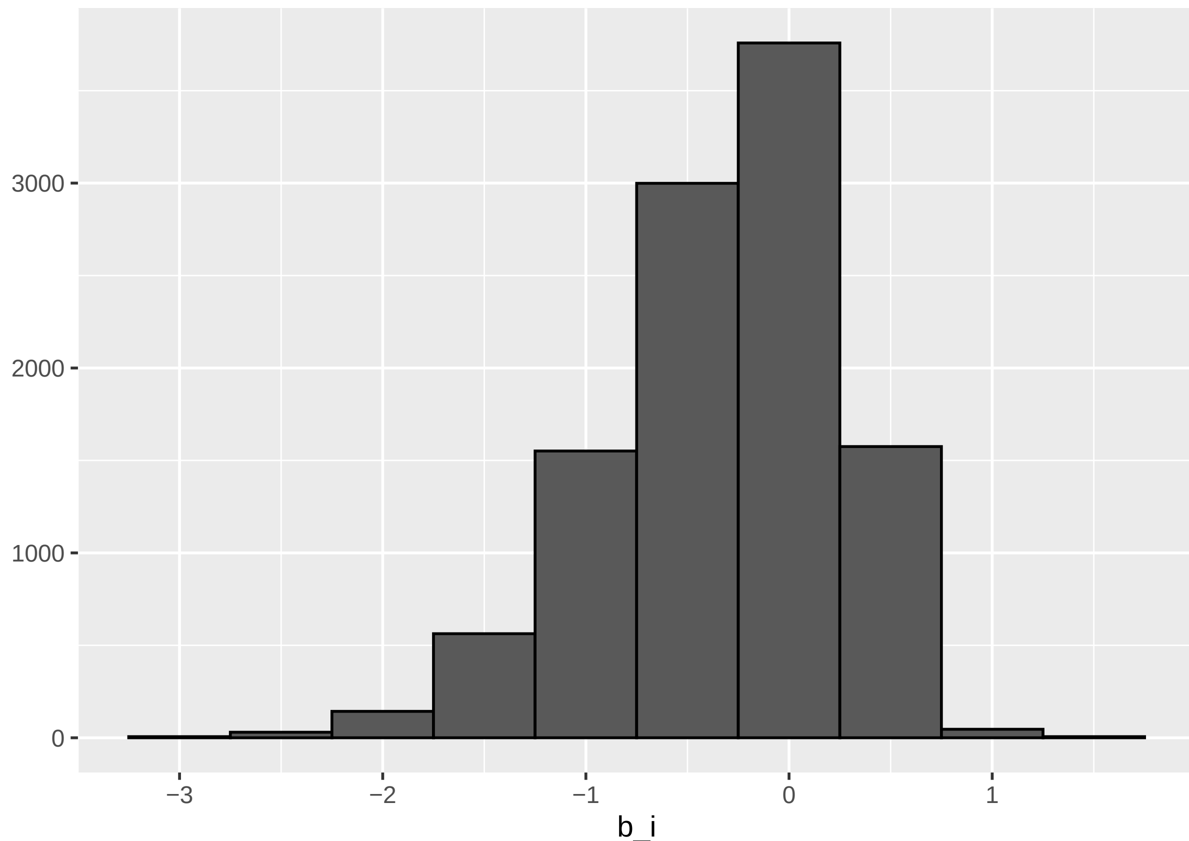
```
## [1] 1.060054
```

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

Including Movie Effect to the model

Augment our previous model by adding the term b_i to represent average ranking for movie

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

qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



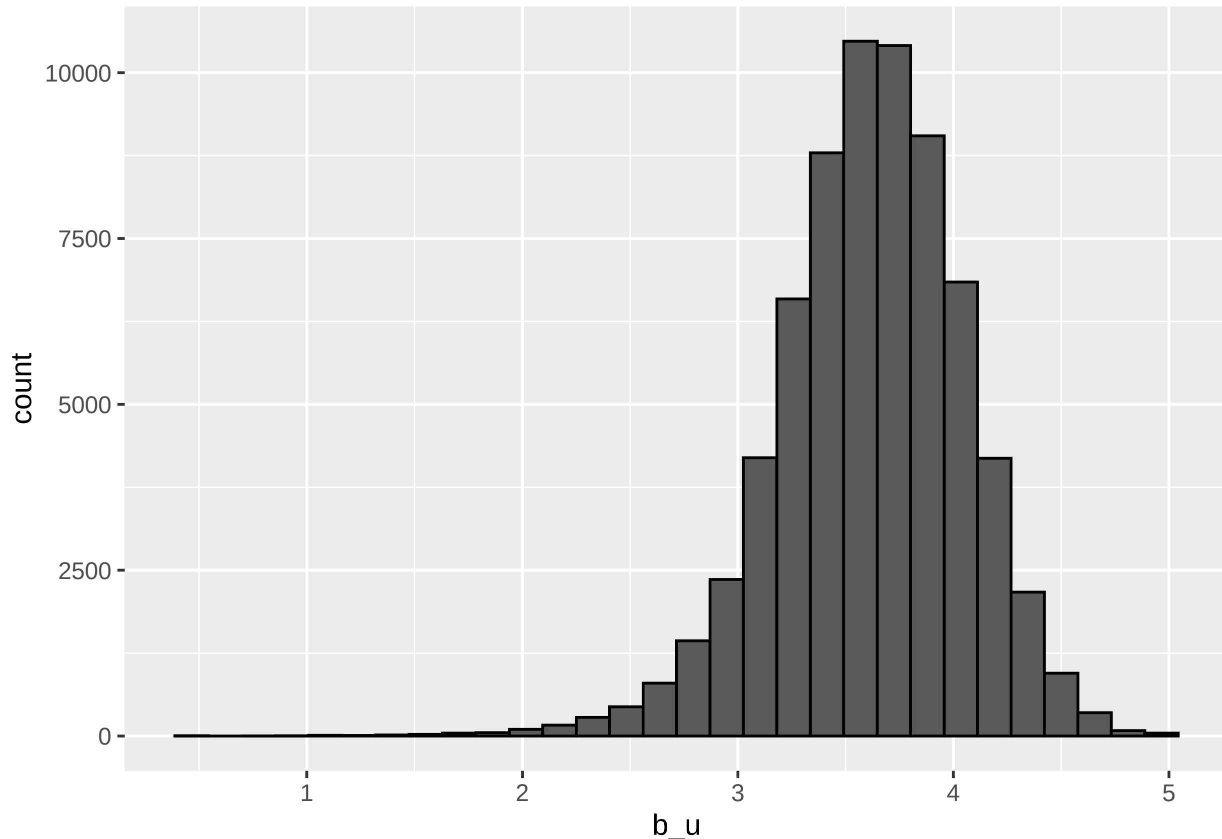

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.9429615
```

Including User Effect

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
```

```
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```




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


predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.8646843
```

Regularization

A technique to solve over fitting.

User and Movie effects are regularized adding a penalty factor lambda, which is a tuning parameter. We define a ### number of values for lambda and use the regularization function to pick the best value that minimizes the RMSE.

```
test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "From Justin to Kelly (2003)"      "Shawshank Redemption, The (1994)"
##  [3] "Shawshank Redemption, The (1994)" "Godfather, The (1972)"
##  [5] "Godfather, The (1972)"            "Godfather, The (1972)"
##  [7] "Godfather, The (1972)"            "Usual Suspects, The (1995)"
##  [9] "Schindler's List (1993)"          "Schindler's List (1993)"
```

```
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()


movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10)  %>%
  pull(title)
```

```
##  [1] "Besotted (2001)"
##  [2] "Hi-Line, The (1999)"
##  [3] "Accused (Anklaget) (2005)"
##  [4] "Confessions of a Superhero (2007)"
##  [5] "War of the Worlds 2: The Next Wave (2008)"
##  [6] "SuperBabies: Baby Geniuses 2 (2004)"
##  [7] "Disaster Movie (2008)"
##  [8] "From Justin to Kelly (2003)"
##  [9] "Hip Hop Witch, Da (2000)"
## [10] "Criminals (1996)"
```

```
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1] 1 1 1 1 1 1 4 2 4 4
```

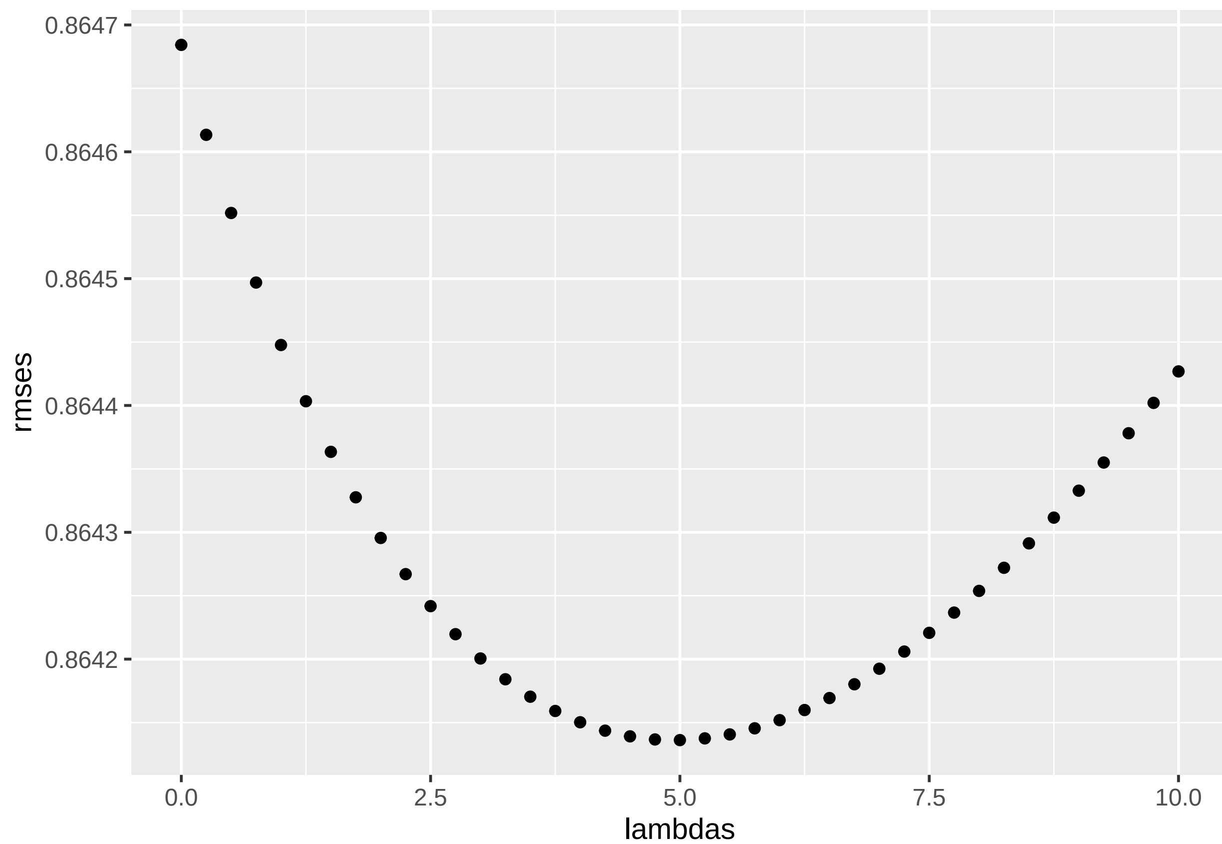
```
train_set %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)
```

```
## [1]   1   1   1   1   2  47  30 183  11   1
```

Lambda - a tuning parameter

```r
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return(RMSE(predicted_ratings, test_set$rating))
})
```

```r
qplot(lambdas, rmses)
```




```
min(rmses)
```

```
## [1] 0.8641362
```

```
lambda <- lambdas[which.min(rmses)]
```

Run Model with Min Lambda value

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

# Movie effect (bi)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect (bu)
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Prediction
y_hat_reg <- test_set %>%
```

```r
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# result table
result <- tibble(Method = "Model with bi and bu with tuned lambda",
                           RMSE = RMSE(test_set$rating, y_hat_reg),
                           MSE  = MSE(test_set$rating, y_hat_reg),
                           MAE  = MAE(test_set$rating, y_hat_reg))

# Regularization made a small improvement in RMSE.
result
```

```
## # A tibble: 1 x 4
##   Method                                  RMSE   MSE   MAE
##   <chr>                                  <dbl> <dbl> <dbl>
## 1 Model with bi and bu with tuned lambda 0.864 0.747 0.669
```

Result and Conclusion

Running the model against the validation set created earlier using lambda for min RMSE value

```r
mu_edx <- mean(edx$rating)

# Movie effect (bi)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

# User effect (bu)
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

# Prediction
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  pull(pred)

# Result
result <- tibble(Method = "Regularize Model run for edx vs validation set",
                           RMSE = RMSE(validation$rating, y_hat_edx),
                           MSE  = MSE(validation$rating, y_hat_edx),
                           MAE  = MAE(validation$rating, y_hat_edx))

# Show the RMSE improvement
result
```

```
## # A tibble: 1 x 4
##   Method                                          RMSE   MSE   MAE
##   <chr>                                          <dbl> <dbl> <dbl>
## 1 Regularize Model run for edx vs validation set 0.865 0.748 0.669
```

Comparison Chart

RMSE improved from initial estimation from mean. The result after regularization with using value of lambda corresponding to min RMSE are close when compared with validation set.

Method	RMSE
Average	1.060054
Movie effect	0.9421695
Movie and user effects	0.8646843
Regularized effect training set	0.8641362
Regularized effect validation set	0.865