Capstone Movielens Project

R.K.Kulkarni

10/01/2022

Introduction

This project is based on movie rating data provided from the movielens competition set by Netflix in 2006 (as described at https://en.wikipedia.org/wiki/MovieLens). The purpose of this project is to find a way of accurately predicting a user's rating for a given movie based on previous ratings of movies seen by all users and the genre of the movie. Various algorithms can be used for such a project. A minimum rmse (root mean square of standard error) lower than 0.86490 is sought for this exercise in order to make a suitable algorithm choice.

The techniques used rely on the values of data fields (predictor features) to predict the value of another data field or outcome. The predictors need to be identified by this project to predict the outcome which is the user rating of a movie less than or equal to the target rmse. The predictor features are present with the dataset provided. The machine learning used is therefore *supervised*.

Methods

The following packages and libraries were installed and loaded: Tidyverse Caret Data.table

```
#Start timer
start_time <- Sys.time()
#Install packages if required
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")
#Load libraries
library(tidyverse)
library(caret)
library(data.table)
#Download movielens data file.
```

The following movielens data zip file was downloaded for processing.

```
#download movielens data zip file.
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

#set up ratings
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
#set up movies look up table
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
#output version of R for reference
R.Version()$version.string
```

```
## [1] "R version 4.0.3 (2020-10-10)"
```

```
v<-as.numeric(substr(R.Version()$version.string,11,13))
#populate movies according to version
if (v >= 4.0) {
  movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                             title = as.character(title),
                                             genres = as.character(genres))} else {
  movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                            title = as.character(title),
                                            genres = as.character(genres))}
#Create final table of base data to use.

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

Create data partitions for training and validation.

```
# Validation set will be 10% of MovieLens data
if (v <= 3.5) {
  set.seed(1)} else {
set.seed(1, sample.kind="Rounding")}

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Create validation set and display column headers for verification.

```
# 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)

#tidy-up
rm(dl, ratings, movies, test_index, temp, movielens, removed)

#while debugging display elapsed time, list column names of training set
#Sys.time() - start_time
#colnames(edx)
colnames(validation)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

Columns in data frame

- userId feature
- movieId feature
- rating outcome to predict.
- timestamp feature to investigate in the future.
- title possible feature?
- genres (pipe delimited) feature but depends on movieId

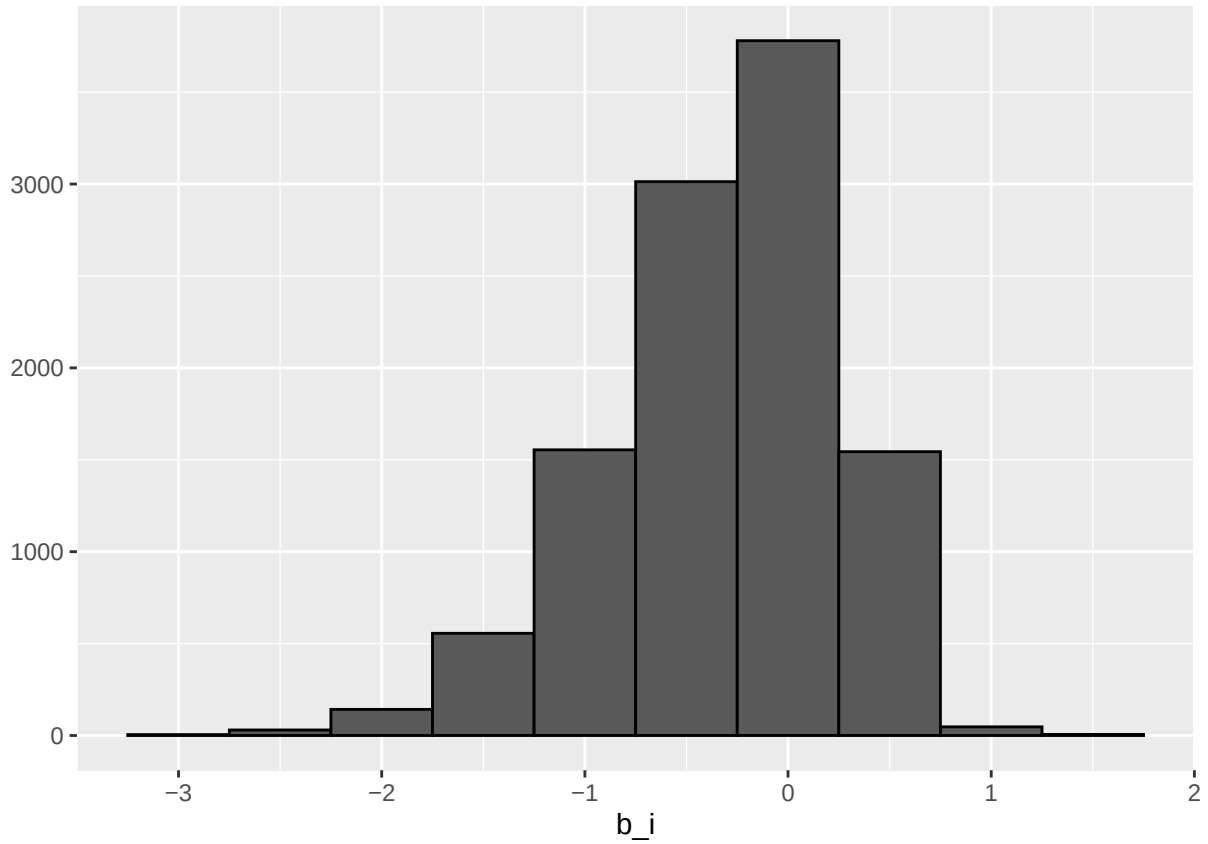
Exploring various methods RMSE lower than 1 and as low as possible is being sought. lm() cannot be used

because of the large dataset size.

```
#naive approach
mu <- mean(edx$rating)
#mu #3.512465

rmse_naive <- RMSE(validation$rating, mu)
#rmse_naive # naive 1.061205
```

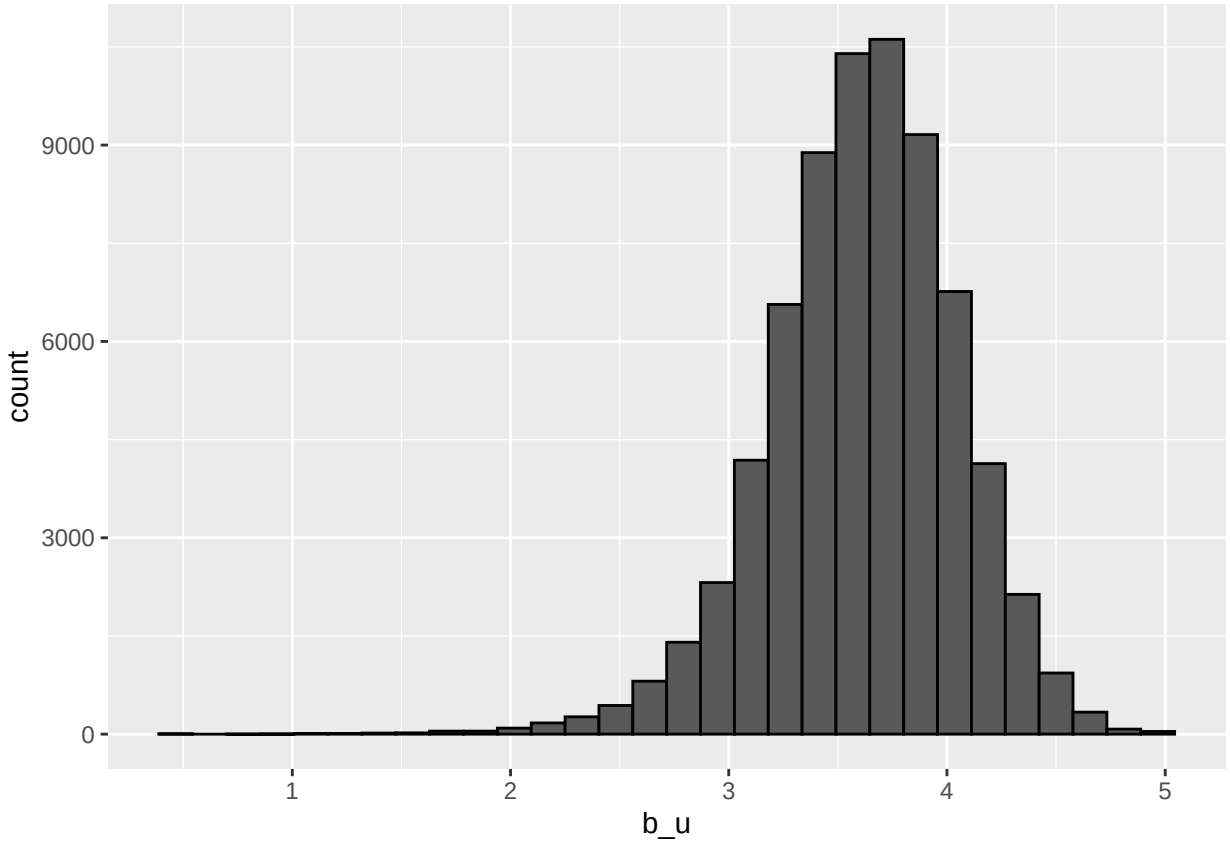
Compute rmse for movie effect on its own.

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Compute rmse of movie and user combined.

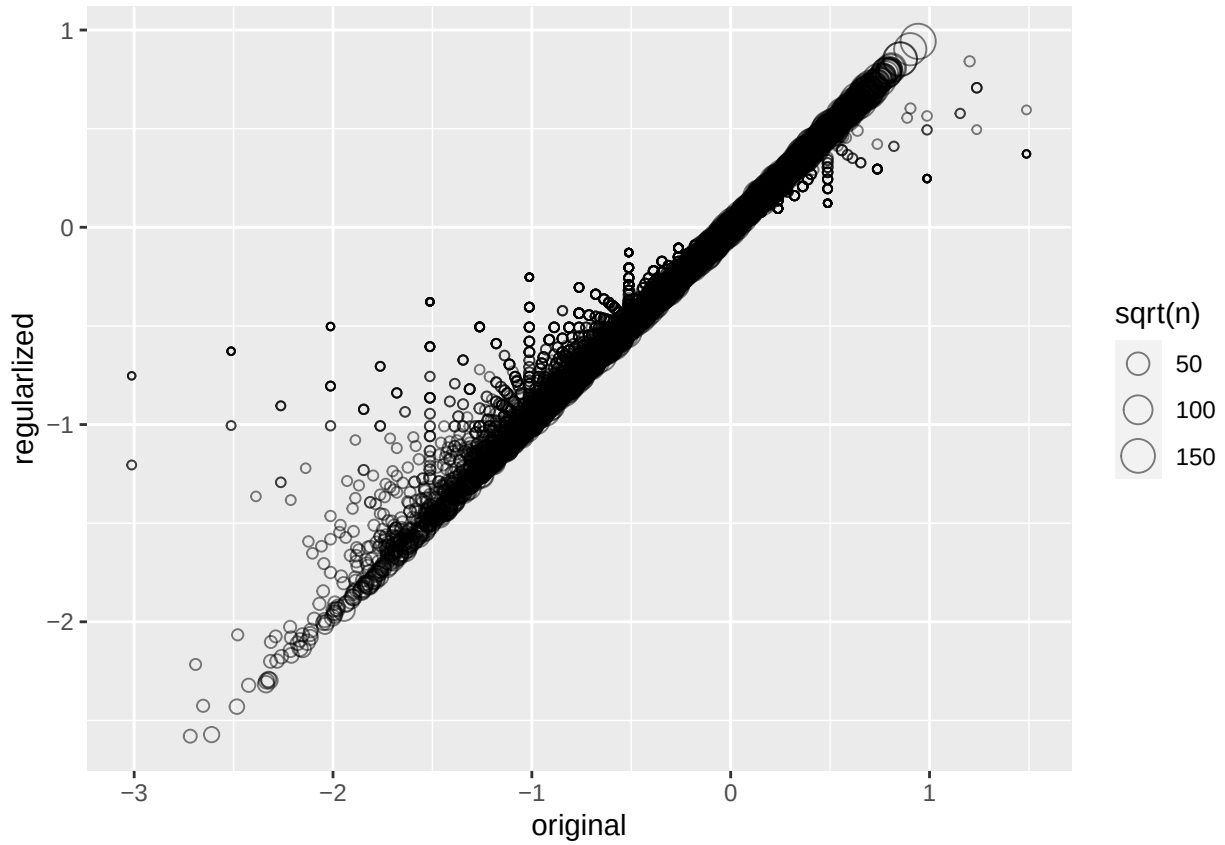
```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```

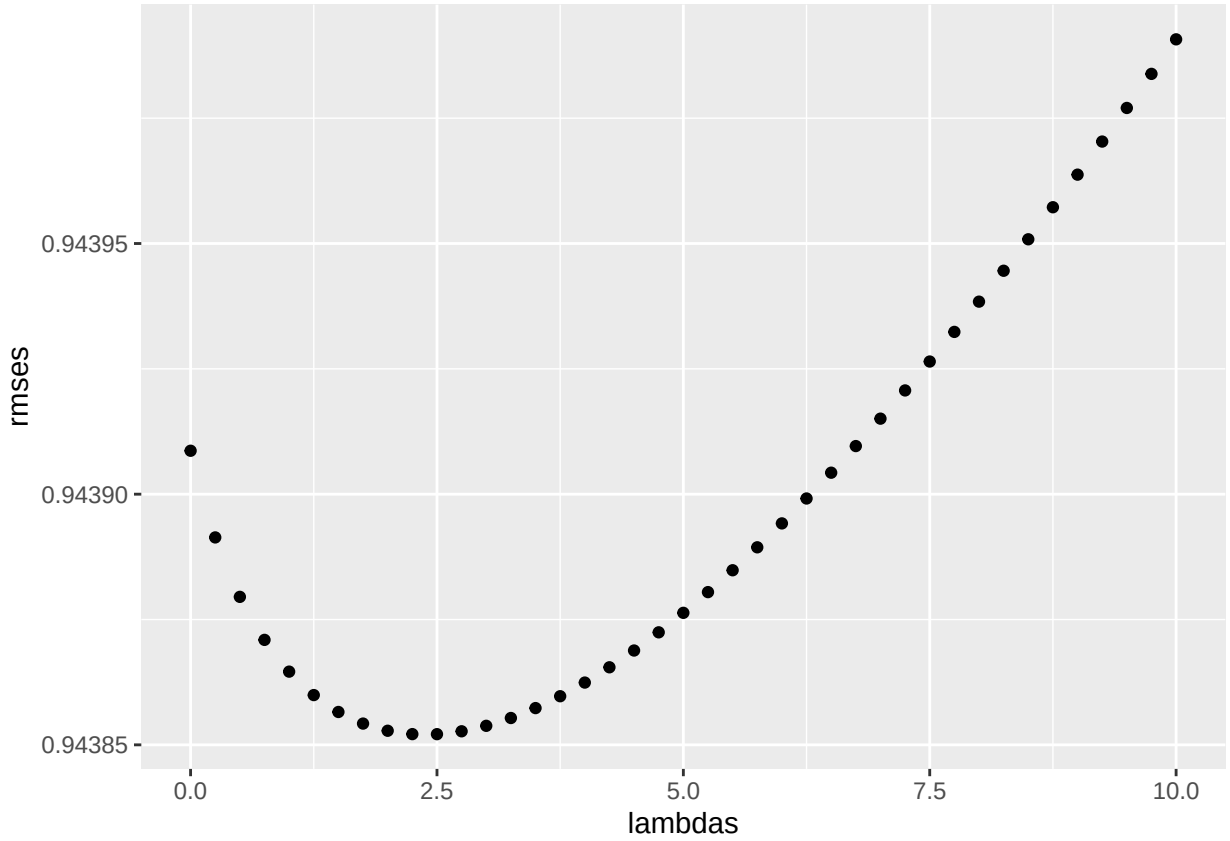
Compute regularised rmse. Calculating Lambda as described on P648 of Course Text. Lambda set to 3 as an initial attempt.

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Compute range of lambdas to help choose a better value of lambda to use.

```
# tune lambda to find best value

lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
rating_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(rating_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
qplot(lambdas, rmses)
```

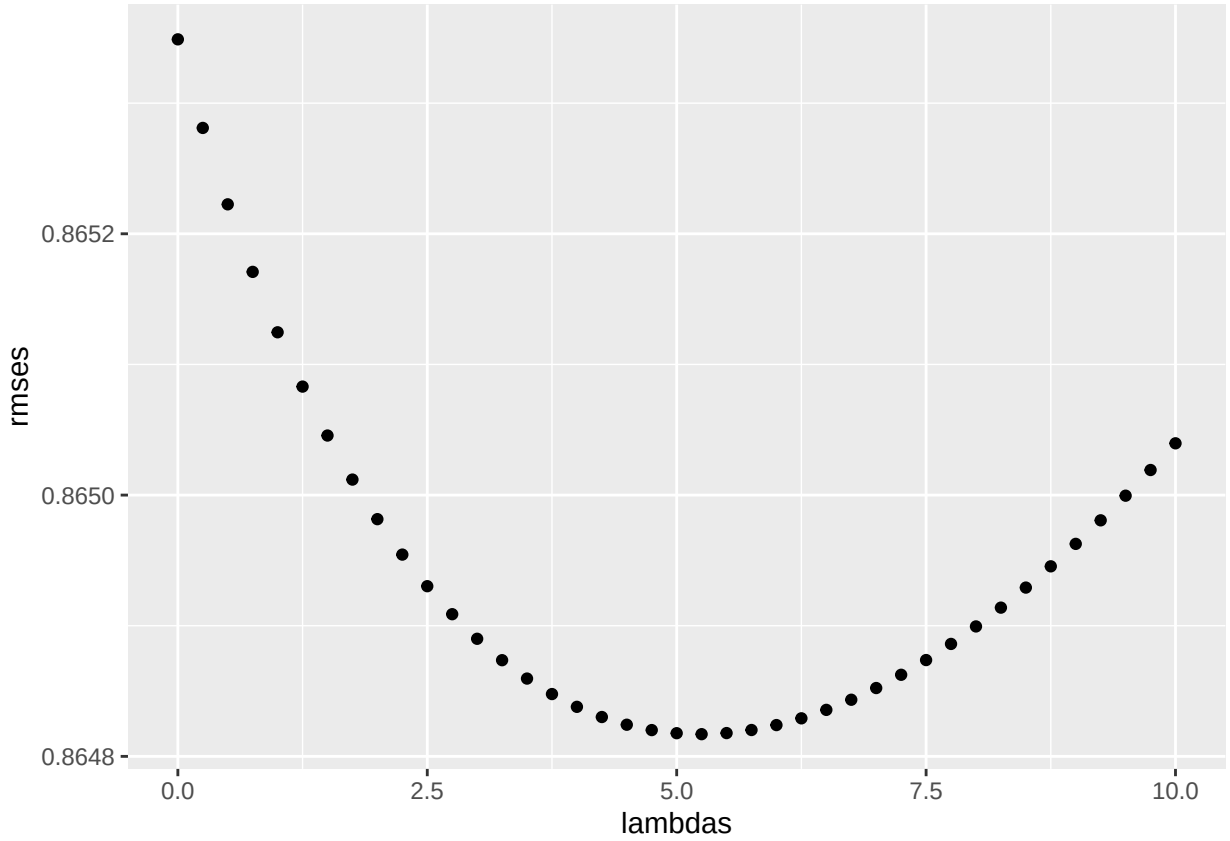


Compute minimal lambda and compute rmse for movie alone.

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Find range of lambdas to help choose lambda value for movie and user case.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
qplot(lambdas, rmses)
```



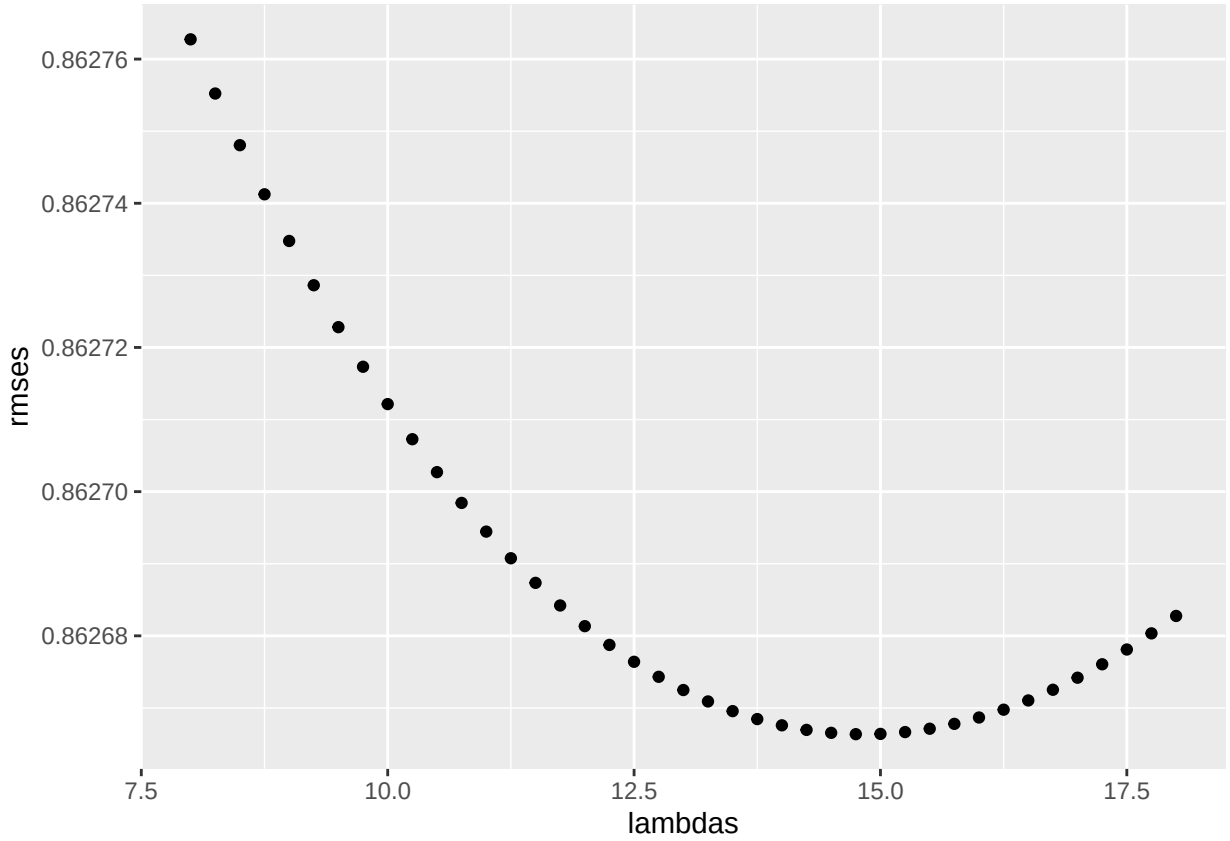
Find minimum lambda for movie+user and compute rmse.

```
lambda<-lambdas[which.min(rmses)] #minimum lambda movie+user  5.25

rmse_tuned_movie_user <- sapply(lambda, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
```

To further improve rmse - genre was included hence the previous steps were repeated using movie+user+genre. New test, train and validation sets were produced which had individual genres in each record. This meant that the pipe delimited genres field in the initial data sets had to be split.

```
#attempt genre
genre_train <- edx %>% separate_rows(genres, sep = "\\|")
```

```
genre_test <- validation %>% separate_rows(genres, sep = "\\|")
genre_edx <- edx %>% separate_rows(genres, sep = "\\|")
genre_validation <- validation %>% separate_rows(genres, sep = "\\|")

#lambdas <- seq(0, 10, 0.25) #min not found <10.0
lambdas <- seq(8, 18, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(genre_train$rating)
  b_i <- genre_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- genre_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- genre_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <-
    genre_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, genre_test$rating))
})
qplot(lambdas, rmses)
```




```
lambda_umg<-lambdas[which.min(rmses)] #minimum lambda movie+user  5.25

rmse_tuned_umg <- sapply(lambda_umg, function(l){
  mu <- mean(genre_train$rating)
  b_i <- genre_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- genre_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- genre_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <-
    genre_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, genre_test$rating))
})
#rmse_tuned_umg
```

```
#timestamp->date->recent rating measure as a predictor?
```

Results

The RMSEs found by the various methods were compared.

```
rmse_naive
```

```
## [1] 1.061202
```

```
rmse_movie
```

```
## [1] 0.9439087
```

```
rmse_movie_and_user
```

```
## [1] 0.8653488
```

```
rmse_regularised
```

```
## [1] 0.9438538
```

```
rmse_tuned_movie
```

```
## [1] 0.9438521
```

```
rmse_tuned_movie_user
```

```
## [1] 0.864817
```

```
rmse_tuned_umg
```

```
## [1] 0.8626664
```

The time elapsed is displayed for information.

```
#timestamp->date->recent rating measure as a predictor?

Sys.time() - start_time
```

```
## Time difference of 22.84199 mins
```

Conclusion

It was found that the rmses required regularization, and tuning of the lambda values used, in order to reduce their values below the required 0.86490 target. It was found that although a combination of movie and user could give an acceptable tuned rmse value (0.864817); using a combination of movie, user and genre could give an improved value (0.8626664). This project therefore recommends that a combination of movie, user and genre be used in the predictive algorithm.

Further work could be done to further augment and refine the features that can be used. This could be done by including new features like actor, director or author; in case a user preference can be detected and used. Also a user may prefer movies in a certain language (spoken or subtitled). Identifying whether a movie is a sequel of another (possibly based on title) could also be a feature to investigate.