MovieLens project(Recommendation system)

MOVIELENS PROJECT:

INTRODUCTION:

The objective of this project is to build a movie recommendation system to predict the rating a user would be giving a specific movie. The prediction is done based on the taste/nature of the user in giving the rating and the likelyhood of a specific movie getting rated higher or lower in general. We are using the movielens data set(which has aound 10 million movie ratings) for this project. We split the data set to edx(training set) and validation(test set) data sets for developing/training and testing the model respectively. All the users in the training set should be present in test set and all the movies in the training set should be present in the test set and vice versa. There are ~9 million records in edx data set and ~1 million records in the validation data set We removed the users and movies that are present in only one of the data set inorder to predict better and to avoid missing movies/users during prediction.

METHODS/ANALYSIS:

The different columns present in the edx and validation data frame are userId, movieId, rating, timestamp, title and genres. The initial step was to find out the user, movie and year bias of the rating. Then we do the prediction by adding the user, movie and year bias to the average rating of a particular movie. The RMSE obtained using this approach is not enough. So, regularization was done to improve the RMSE. As part of regularization, we need to give more importance to the users, movies and year with more number of ratings and give less importance to users, movies and years with less rating. So, we introduce a new parameter lambda as a penalizing value for users, movies and years with less number of ratings. In order to get better RMSE, we need to find out the optimal lambda. So, we plot lambda values against RMSE values for differnt values of lambda and pick the lambda which produces the least(better) RMSE. We then use the optimal lambda obtained from the previous step to do the prediction on the test set and we find that RMSE has improved. The code for doing all of these is mentioned below.

```
##################################
# Data setup begins
##################################


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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

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

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

```
## Loading required package: caret
```

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

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

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

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

```
## Loading required package: 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(tidyverse)
library(caret)
library(data.table)

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")


movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

```r
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
```

```r
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)
```

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

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

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

#################################
# Data setup complete
#################################


#############################################################################################
#Add year column to edx and validation data set by extracting year from the title.
#############################################################################################
edx <- edx %>% mutate(year=str_sub(str_extract(title , "^*\\(\\d{4}\\)$"), 2,5))
validation <- validation %>% mutate(year=str_sub(str_extract(title , "^*\\(\\d{4}\\)$"), 2,5))

mu <- mean(edx$rating)
#Normal movie bias
movie_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_m = mean(rating - mu), .groups ="keep")

#Normal user bias
user_avgs <- edx %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_m), .groups ="keep")

#Year  bias
year_avgs <- edx %>%
     left_join(movie_avgs, by='movieId') %>%
```

```
     left_join(user_avgs, by='userId') %>%
     group_by(year) %>%
     summarize(b_y = mean(rating - mu - b_m - b_u), .groups ="keep")



############################################################
# Prediction with movie, user and year effect
############################################################
predicted_ratings <- validation %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     left_join(year_avgs, by='year') %>%
     mutate(pred = mu + b_m + b_u + b_y) %>%
     .$pred

#Calculate RMSE
RMSE(validation$rating, predicted_ratings)
```

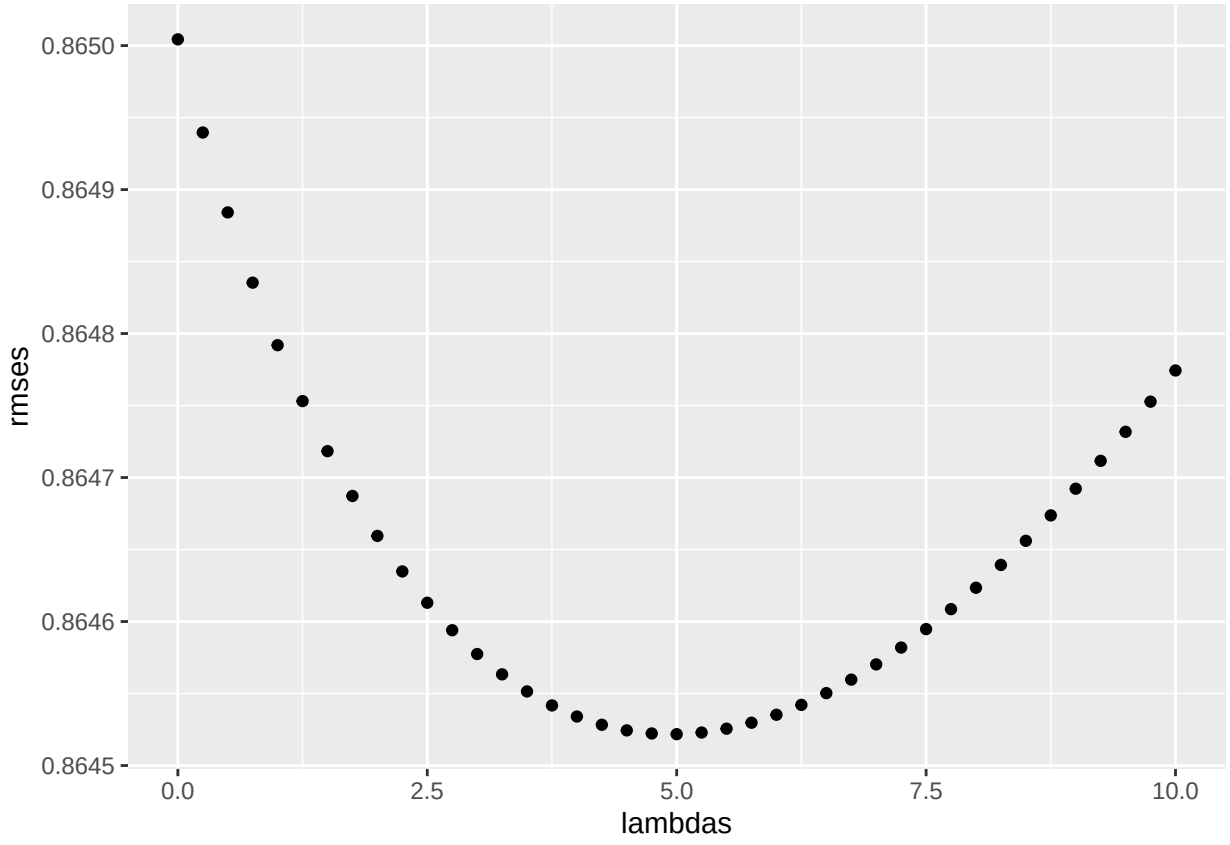
```
## [1] 0.8650043
```

```
####################################
#Code to find out optimal lambda
####################################
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
     mu <- mean(edx$rating)
     #Regularized movie bias
     b_m <- edx %>%
          group_by(movieId) %>%
          summarize(b_m = sum(rating - mu)/(n()+l), .groups ="keep")
     #Regularized user bias
     b_u <- edx %>%
          left_join(b_m, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_m - mu)/(n()+l), .groups ="keep")

     #Regularized year bias
     b_y <- edx %>%
          left_join(b_m, by="movieId") %>%
          left_join(b_u, by='userId') %>%
          group_by(year) %>%
          summarize(b_y = sum(rating - b_m - b_u - mu)/(n()+l), .groups ="keep")

     predicted_ratings <-
          validation %>%
          left_join(b_m, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          left_join(b_y, by = "year") %>%
          mutate(pred = mu + b_m + b_u + b_y) %>%
          .$pred
     return(RMSE(predicted_ratings, validation$rating))
})
```

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




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

##########################
#Print the Optimal lambda
##########################
lambda
```

```
## [1] 5
```

```
############################################################
# Prediction with regularized movie and user effect
############################################################

       mu <- mean(edx$rating)
     #Calculate regularized movie bias
     b_m <- edx %>%
          group_by(movieId) %>%
          summarize(b_m = sum(rating - mu)/(n()+lambda), .groups ="keep")

     #Calculate regularized user bias
     b_u <- edx %>%
          left_join(b_m, by="movieId") %>%
```

```
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_m - mu)/(n()+lambda), .groups ="keep")

#Calculate regularized year bias
b_y <- edx %>%
      left_join(b_m, by="movieId") %>%
      left_join(b_u, by="userId") %>%
      group_by(year) %>%
      summarize(b_y = sum(rating - b_m - b_u - mu)/(n()+lambda), .groups ="keep")

#Predict by considering regularized user, movie and year bias
predicted_ratings <-
      validation %>%
      left_join(b_m, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_y, by = "year") %>%
      mutate(pred = mu + b_m + b_u + b_y) %>%
      .$pred
#Calculate RMSE
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8645218
```

```
FINAL_RMSE <-    RMSE(predicted_ratings, validation$rating)


###################################
#Print the final RMSE (< 0.86490)
###################################
FINAL_RMSE
```

```
## [1] 0.8645218
```

RESULTS:

With just user, movie and year bias included, we were able to get an RMSE of 0.8650 only. But when we did regularization, the RMSE improved to 0.86452(which is better than the required RMSE of 0.86490).

CONCLUSION:

We were able to produce the desired RMSE by using regularization of the model. The RMSE can still be improved using other methods like matrix factorization, singular value decomposition(SVD) and principal components analysis(PCA). But we are not doing that here since we have achieved the required RMSE. The limiation of this model is that it won't be able to correctly(with reasonable RMSE) predict the rating of new movies or new users unless the user has rated reasonable number movies or the movie has got reasonable number of ratings respectively. The accuracy of the prediction increases with the number of ratings.