HarvardX: PH125.9x Data Science
MovieLens Rating Prediction Project

Sakshi Gupta

16/11/2020

Contents

1 Introduction

This is the report on the movie reccomendation system which recommends the movie based on the rating scale. This is the capstone report of the HarvardX: Data Science- Capstone course

2 Objective of the project

Objective of this project is to develop a machine learning model that could predict the user ratings (from 0.5 to 5) using the edx as the train set and validation data set as the test set. We will calculate the RMSE (Root Mean Squared Error), which is a frequently used measure of the differences between values predicted by a model or an estimator and the values observed. The less the value of the RMSE, the more good is the model.

Formula of RMSE is

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

3 Data

The original data was obtained from GroupLens research and can be found at "https://grouplens.org/datasets/movielens/10m/" This data was provided by the course instrutor in the video

3.1 Loading of Data

The data can be loaded by the below code provided in the course intructions. We will split the complete data into edx set and validation set. Validation set will be 10% of the total data. we will train our model on the edx data set and test it on the Validation data set Validation set will be used at the final stage to validate the model perfromance.

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

# Note: this process could take a couple of minutes

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

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

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

```
## 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 ------------------------------------------------------------------------ tidyverse_confli
## 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

#loading libraries

library(tidyverse)
library(caret)
library(data.table)
library(dplyr)

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

#downloading the dataset
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 <- 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)
```

Is is important to check for any NA value present in both edx and Validation data

```
anyNA(edx)
```

```
## [1] TRUE
```

```
anyNA(validation)
```

```
## [1] TRUE
```

4 Analysis

4.1 Data Exploration Analysis

Before proceeding ahead, we will analyse the data set structure.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
```

```
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  NA NA NA NA ...
##  $ genres   : chr  NA NA NA NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  NA NA NA NA ...
##  $ genres   : chr  NA NA NA NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

using $Summary()$, we get the complete insights of the dataset, including mean,mediam, min and max values

```
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
##
##
##
```

Count for movies and users in the dataset

```
edx %>% summarise(userscount = n_distinct(userId), moviescount = n_distinct(movieId))
```

```
##   userscount moviescount
## 1      69878       10677
```

The total number of unique users and movies are 69878 and 10677 respectively.

The maximum movies received the rating 4 followed by rating 3 and 5.

```
edx %>% group_by(rating) %>%
  summarise(count = n()) %>%
  top_n(5) %>% arrange(desc(count))
```

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

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

The top 10 most rated movies are-

```
edx  %>% group_by(movieId, title) %>% summarise(count = n()) %>% arrange(desc(count)) %>% top_n(5)
```

```
## `summarise()` regrouping output by 'movieId' (override with `.groups` argument)
```

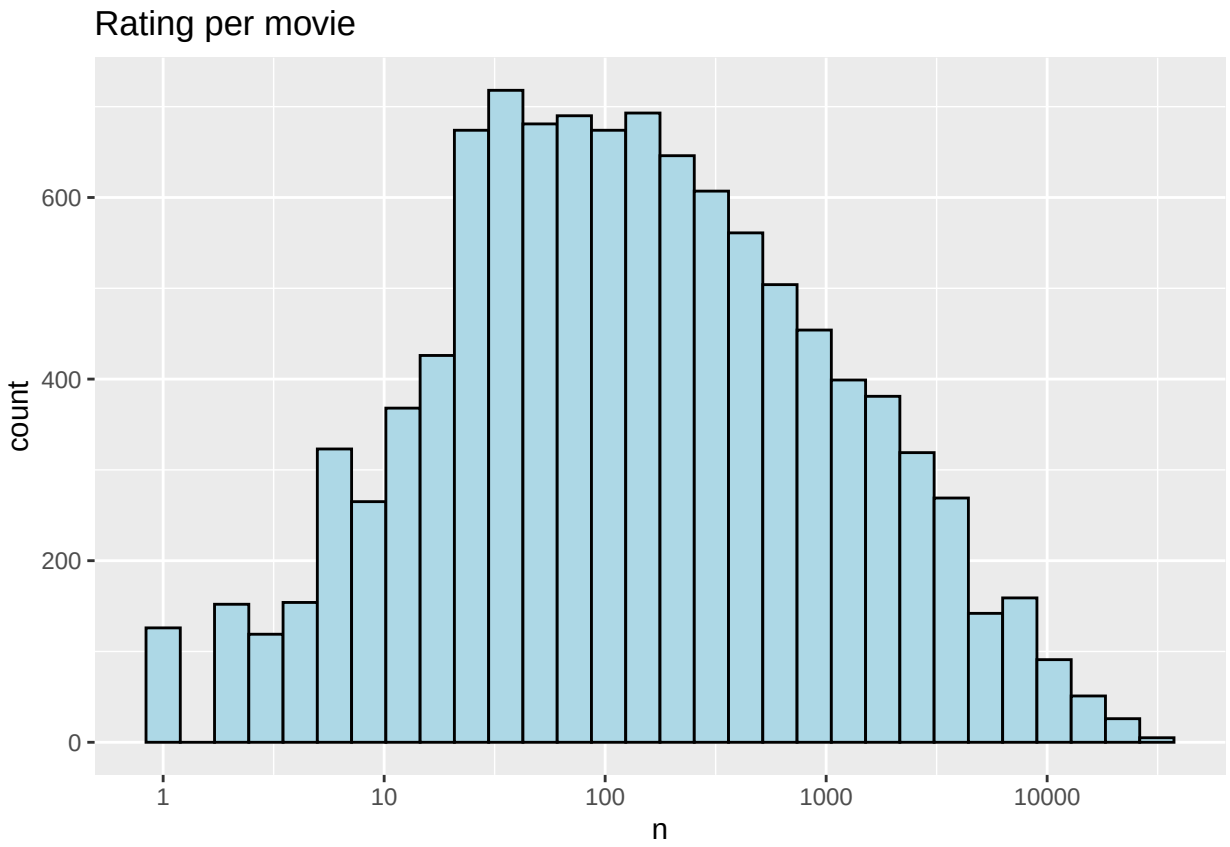
```
## Selecting by count
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title count
##      <dbl> <chr> <int>
##  1     296 <NA>  31362
##  2     356 <NA>  31079
##  3     593 <NA>  30382
##  4     480 <NA>  29360
##  5     318 <NA>  28015
##  6     110 <NA>  26212
##  7     457 <NA>  25998
##  8     589 <NA>  25984
##  9     260 <NA>  25672
## 10     150 <NA>  24284
## # ... with 10,667 more rows
```

We plot the number of ratings per movie

```
edx %>% count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(color = "black" , fill= "light blue") + scale_x_log10() +
  ggtitle("Rating per movie") + theme_gray()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The users who rated the most of the movies-

```
edx %>% group_by(userId) %>%
  summarise(count = n()) %>% arrange(desc(count)) %>% top_n(10)
```

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

## Selecting by count

## # A tibble: 10 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
```

We plot the number of ratings per user

```
edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(color = "black", fill = "light blue") +
  scale_x_log10()+ ggtitle("Ratings per user") + theme_gray()
```

As observed many movies are categorized as more than 1 genre. Before calculating the highly rated genres, we have to seperate th genres on '|'

The most rated genres are

```
edx %>% separate_rows(genres , sep = "\\|") %>%
  group_by(genres) %>% summarise(count = n()) %>%
  arrange(desc(count)) %>% top_n(10)
```

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

## Selecting by count

## # A tibble: 1 x 2
##   genres   count
##   <chr>    <int>
## 1 <NA>   9000055
```

We can see that Drama , Comedy and Action are the most rated genres

5 Model Building

5.1 RMSE Calcultion

we will now write the loss-function that will compute the RMSE.

RMSE is the measure of our model accuracy. It is the error we make when predicting the movie rating. The lesser the RMSE , the lesser the error. In this case, if the RMSE is more than 1, the error in predicting the rating of movie is of 1 rating.

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

RMSE function is

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((predicted_ratings - true_ratings)^2, na.rm = TRUE))
}
```

5.2 First RMSE - Average movie rating model

The first basic model is where we compute the mean rating for the complete data set. This predict the same rating for all the movies. The expected rating of the underlying dataset is between 3 and 4.

We start building the model by the simplest possible reccomendation system where the we predict the same rating for all the movies, regardless of other factors like users who gave it or which movie received more ratings.

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

```
## [1] 3.512465
```

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

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

Lets save this RMSE in a table and keep on adding more RMSE to this table to analyse the best model

```
rmse_results <- data_frame(method = "Average model approach", RMSE= naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

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

method	RMSE
Average model approach	1.061202

Now we have got our baseline RMSE to compare with the other model results

In order to do better than simply predicting the average rating, we will consider some of the insights we gained during the exploratory data analysis.

5.3 Second Model - Movie effect model

To improve the accuracy of the prediction, we will focus on the fact that some movies receive more ratings than others.

Higher ratings are mostly linked to popular movies among users. Here we are taking into account the effect of b_i, where we will be subtracting the mean of rating received by a particular movie from the actual rating of that movie

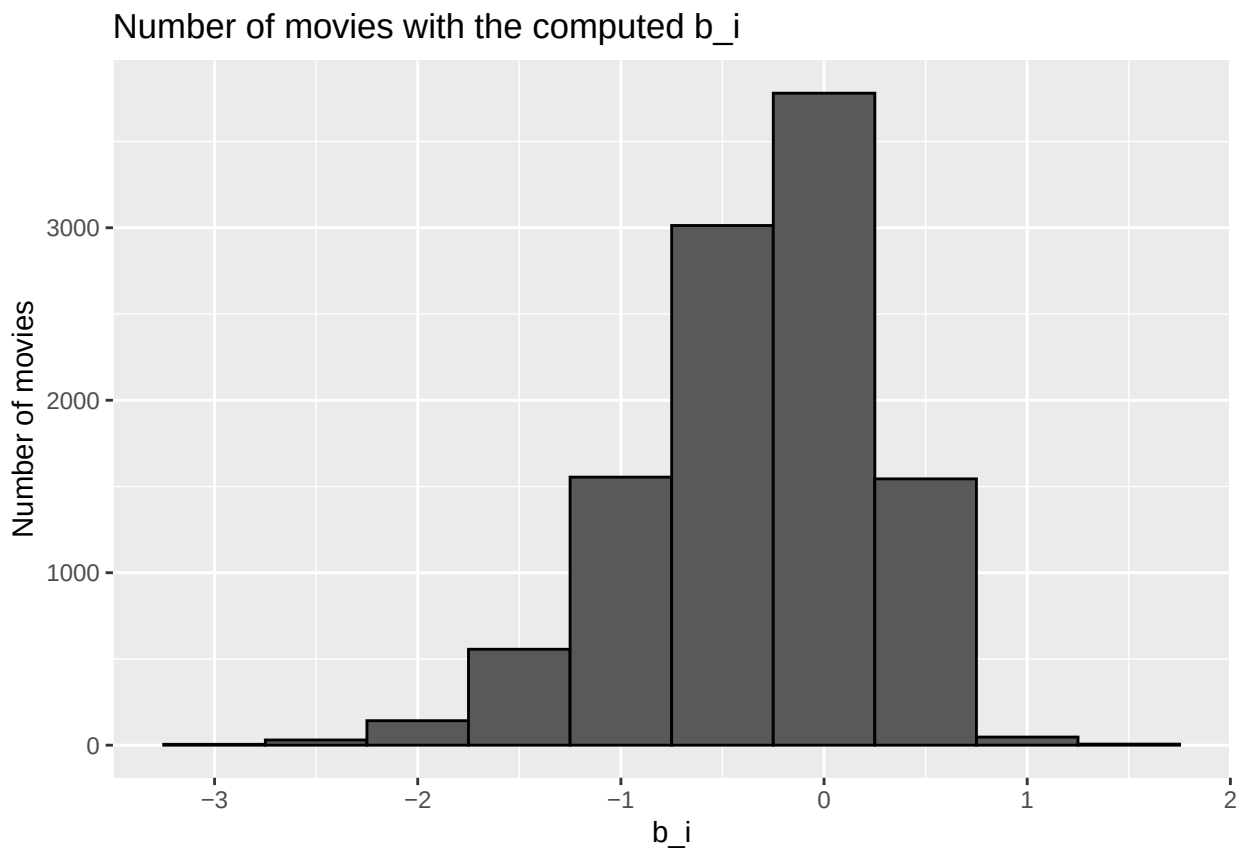
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

```
movie_avgs <- edx %>%
  group_by(movieId) %>% summarise(b_i = mean(rating - mu_hat))
```

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

We plot this and analyse

```
movie_avgs %>%
   ggplot(aes(b_i, color = I("black") ) ) +
  geom_histogram(bins= 10) +
  labs(title = "Number of movies with the computed b_i") +
  ylab("Number of movies")
```



Now lets see how the prediction improves after altering the equation $Y\ u, i\ = ?? + b\ i$

```
predicted_ratings <- mu_hat + validation %>% left_join(movie_avgs , by = 'movieId') %>%
  pull(b_i)
```

The RMSE of second model (movie effect model) is

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

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

Adding this RMSE to the results table

```
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie effect model" , RMSE = rmse_with_bi))
rmse_results %>% knitr::kable()
```

method	RMSE
Average model approach	1.0612018
Movie effect model	0.9439087

5.4 Third Model - Movie and User effect model

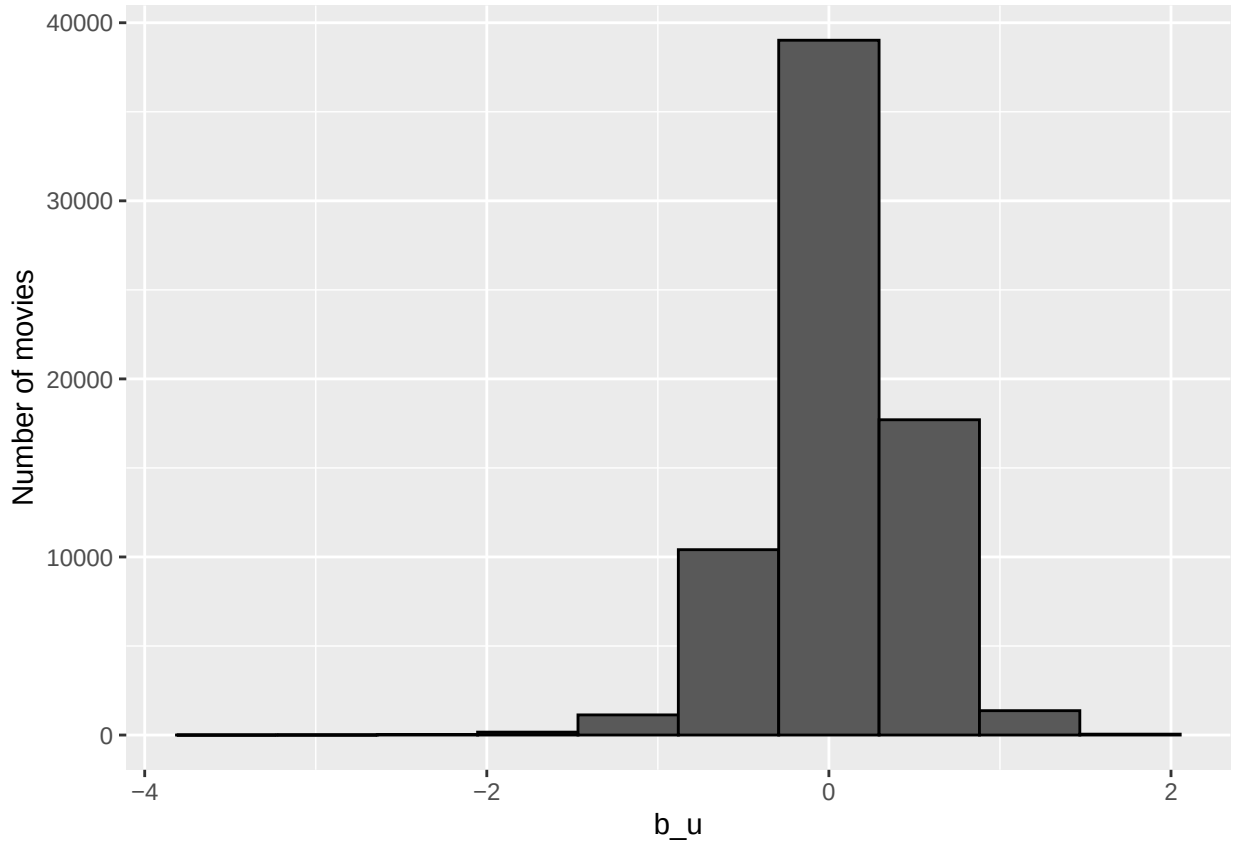
Here we are taking into account the effect of user also along with the movie

```
user_avgs <- edx %>% left_join(movie_avgs, by= 'movieId') %>%
  group_by(userId) %>% summarise(b_u = mean(rating - mu_hat - b_i))
```

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

Now lets plot this observation

```
user_avgs %>% ggplot(aes(b_u, color= I("black"))) + geom_histogram(bins = 10) + labs("Number of movies w
  ylab("Number of movies")
```



Now lets see how the prediction improves after altering the equation $Y\ u, i\ = ?? + b\ i + b\ u$

```
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>% mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

The RMSE of third model (movie + user effect model) is

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

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

Adding this third model RMSE to the results table

```
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie and User effect model", RMSE = rmse_
rmse_results %>% knitr::kable()
```

method	RMSE
Average model approach	1.0612018
Movie effect model	0.9439087
Movie and User effect model	0.8653488

Here we can see that considering the movie and user effects helps in giving better prediction. Let us now apply this model to the validation set to validate our model and it's accuracy

```
validation_pred_rating <- validation %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(user_avgs, by= 'userId') %>%
  mutate(pred = mu_hat + b_i+ b_u) %>%
  pull(pred)
```

RMSE of the validation set

```
rmse_validation <- RMSE(validation_pred_rating, validation$rating)
```

Save this rmse to the results table

```
rmse_results <- bind_rows(rmse_results, data_frame(method = "RMSE of Validation set", RMSE = rmse_valida
rmse_results %>% knitr::kable()
```

method	RMSE
Average model approach	1.0612018
Movie effect model	0.9439087
Movie and User effect model	0.8653488
RMSE of Validation set	0.8653488

5.5 Regularization

Regularizing the movie and user effect. We will use lambda as the tunning parameter, and cross-validation to choose the best tuning parameter.

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

Now we will find bias value- b_i and b_u for every lambda value

```r
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))
})
```

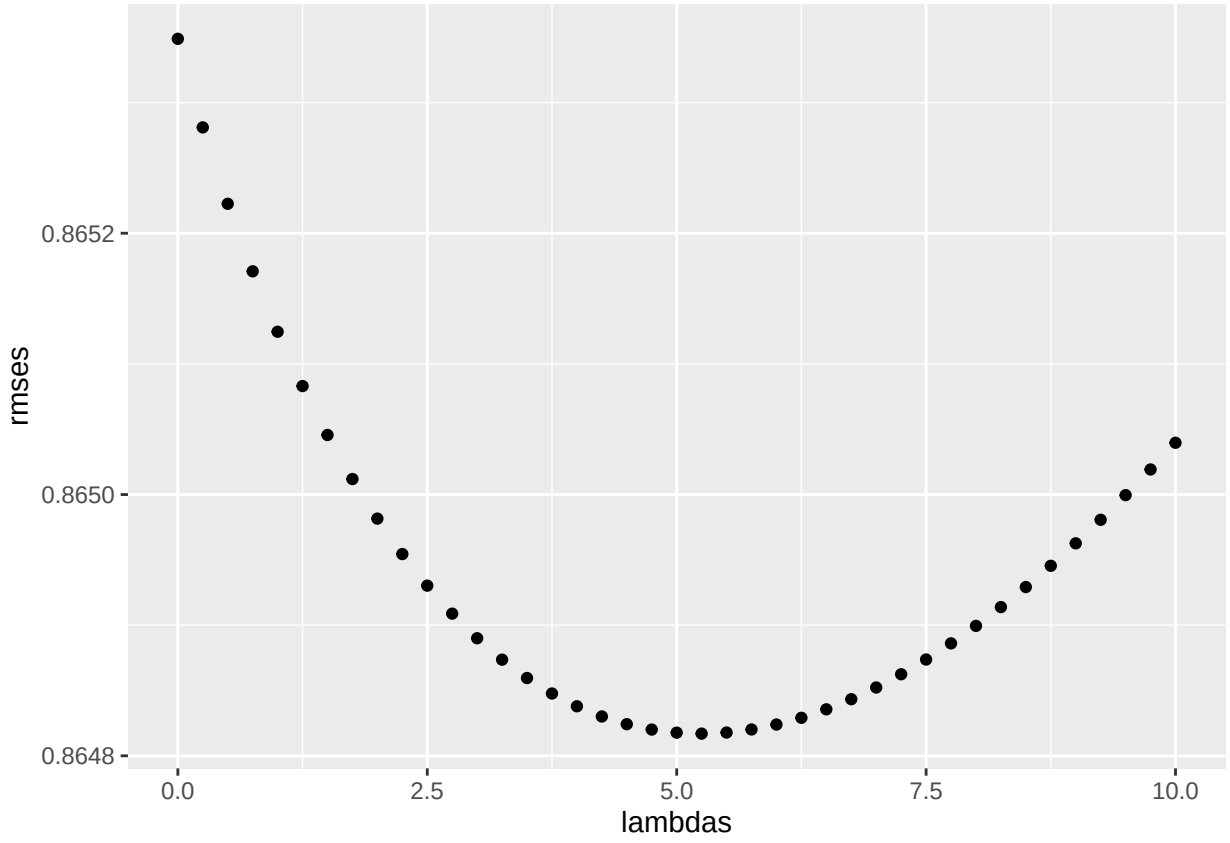
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()`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

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

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

Plot rmses vs lambda to select the optimal value of lambda

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



The optimal value of lambda is

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

```
## [1] 5.25
```

Save this result in the rmse_results table

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = min(rmses)))
```

6 Results

The RMSE values of all the represented models are the following:

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

method	RMSE
Average model approach	1.0612018
Movie effect model	0.9439087
Movie and User effect model	0.8653488
RMSE of Validation set	0.8653488
Regularized movie and user effect model	0.8648170

We therefore observe that the lowet value of the RMSE is 0.8648170

7 Conclusion

So here in this project we build a machine learning model to predict the movie rating using the dataset provided by Movielens

So our final model for the movie reccomendation system is :

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

Here we observed that the regularized model including the user and movie effect gave us the lowest RMSE value and is the optimal model i.e: 0.8648170

We can further improve the RMSE by adding other effects like genre, year etc.

This model works well if the average user does not rate any popular movie with large positive b_i by disliking a popular movie.