Project - MovieLens Recommender System

Edwin Leonardi Liong

February 9, 2019

Introduction

A recommender system is a filtering system that attempts to predict users' ratings based on their preferences. Using this prediction model, the system can suggest a personalised recommendations to the audience. In the recent years, we are seeing more and more adoption of the recommendation system in various areas of industries.

In this project, we will look into a movie recommender system to predict how a user will give movie ratings based on the user's rating behaviour. The challenge lies in the complexity of the model. Each individual user has own preferences and dislikes.

We are going to use and analyze a MovieLens dataset of 100,000 ratings. Later on, we will be working on a training set and apply a couple of experiments to arrive on an algorithm with the least residual mean squared error (RMSE). The RMSE will be used as valuation of our prediction ratings against the true ratings in the validation set.

Dataset - Exploration and Preparation

MovieLens data is publicly available for download. The code to generate the dataset has been provided in the Capstone Project. For consistency to the grading in the project, we refrain from making any amendments on this part of dataset preparation process and decided to follow the steps that is outlined in the supplied code. This includes the separation of the full data set into training set and the validation set (90% to 10% ratio). It is also worth noting that the userId and movieId in the validation set are ensured to have matching values with the respective attributes in the training set edx.

Having said that, we will still conduct some walk-through of the dataset to study the structure, the attributes and its completeness. However, we are going to only analyze the edx training set as we treat this dataset as the only data we can work on and simulate before assessing the model with the validation set

The dataset preparation process is given as follows:

```
############################################################
# Create edx set, validation set, and submission file
############################################################
# 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")
```

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

library(tidyverse)
library(caret)
library(Hmisc)
library(cvTools)

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

#unzipped the downloaded file and read ratings.dat and store in ratings variable
ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),col.names = c("userId", "movieId", "rating", "timestamp"))

#unzipped the downloaded file and read movies.dat and store in movies variable
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

#assign appropriate data type to the attributes
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],title = as.character(title),genres = as.character(genres))

#join both data frames ratings and movies by movieId
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
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)
```

We run following script and find that the edx dataset contains 9000055 movie rating details along with the 6 variables

```
#observe the structure of the data and its content
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```
str(edx)
```

```
## '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 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)"
"Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller"
"Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
```

```
describe(edx)
```

```
## edx
##
##  6  Variables      9000055  Observations
## ---------------------------------------------------------------------------
-
## userId
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  9000055        0    69878        1    35870    23769     3810     7521
##      .25      .50      .75      .90      .95
##    18124    35738    53607    64479    68093
##
## lowest :     1     2     3     4     5, highest: 71563 71564 71565 71566
```

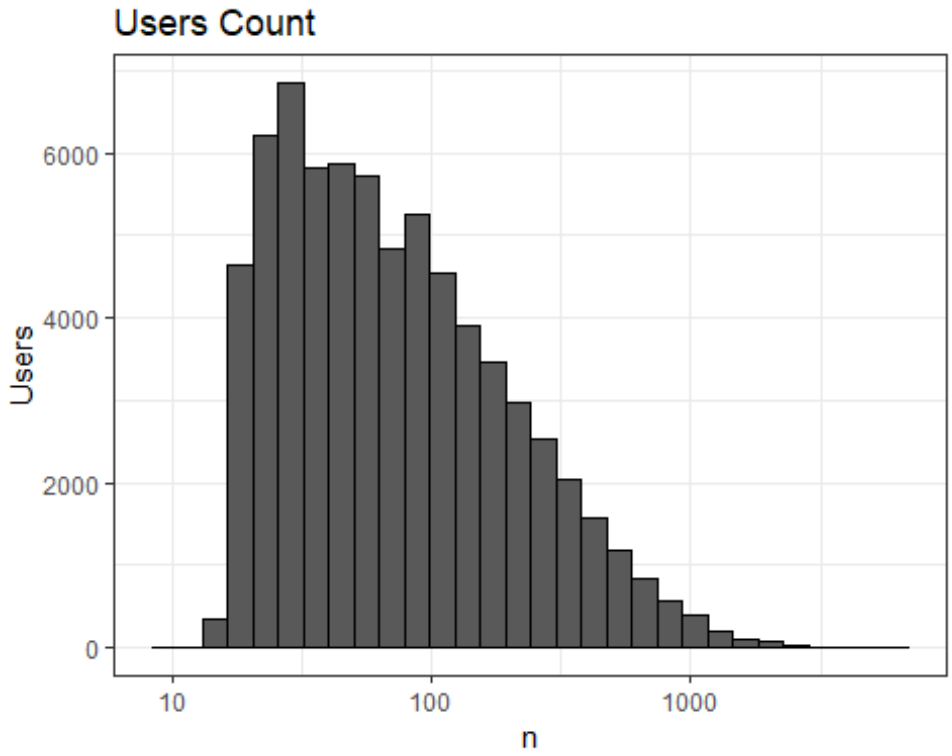
```
71567
## ------------------------------------------------------------------------
-
## movieId
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  9000055        0    10677        1     4122     5535      110      260
##      .25      .50      .75      .90      .95
##      648     1834     3626     6502     8917
##
## lowest :     1     2     3     4     5, highest: 65088 65091 65126 65130
65133
## ------------------------------------------------------------------------
-
## rating
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  9000055        0       10    0.958    3.512    1.166      1.5      2.0
##      .25      .50      .75      .90      .95
##      3.0      4.0      4.0      5.0      5.0
##
## Value          0.5     1.0     1.5     2.0     2.5     3.0     3.5     4.0
## Frequency    85374  345679  106426  711422  333010 2121240  791624 2588430
## Proportion   0.009   0.038   0.012   0.079   0.037   0.236   0.088   0.288
##
## Value          4.5     5.0
## Frequency   526736 1390114
## Proportion   0.059   0.154
## ------------------------------------------------------------------------
-
## timestamp
##         n   missing  distinct      Info      Mean       Gmd       .05
##   9000055         0   6519590         1 1.033e+09 133321774 8.395e+08
##       .10       .25       .50       .75       .90       .95
## 8.506e+08 9.468e+08 1.035e+09 1.127e+09 1.188e+09 1.213e+09
##
## lowest :  789652009  822873600  823185196  823185197  823185198
## highest: 1231131132 1231131137 1231131142 1231131303 1231131736
## ------------------------------------------------------------------------
-
## title
##        n  missing distinct
##  9000055        0    10676
##
## lowest : 'burbs, The (1989)                     'night Mother (1986)
'Round Midnight (1986)                'Til There Was You (1997)
"Great Performances" Cats (1998)
## highest: Zoot Suit (1981)                       Zorba the Greek (Alexis
Zorbas) (1964) Zorro, the Gay Blade (1981)           Zulu (1964)
Zus & Zo (2001)
## ------------------------------------------------------------------------
-
```

```
## genres
##        n  missing distinct
##  9000055        0      797
##
## lowest : (no genres listed)                                    Action
Action|Adventure
Action|Adventure|Animation|Children|Comedy
Action|Adventure|Animation|Children|Comedy|Fantasy
## highest: Thriller|War
Thriller|Western                                        War
War|Western                                             Western
## ------------------------------------------------------------------------
-
```

The variables are analyzed as below

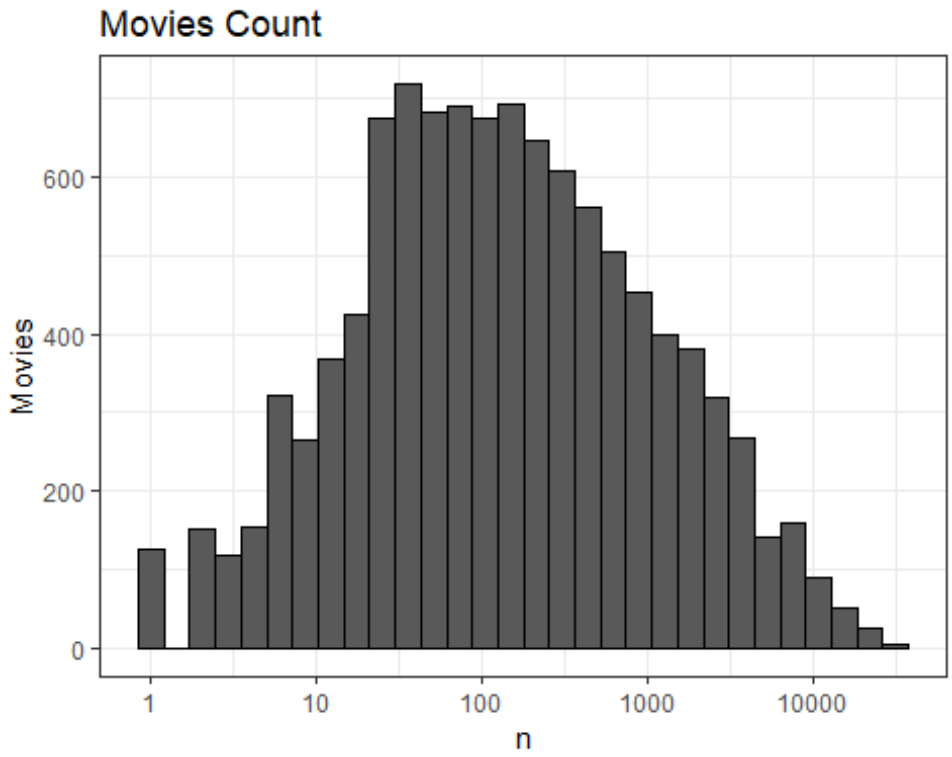
userId attribute is a unique identifier of a user. There are 69878 distinct users in the dataset and no missing values detected. Here is the histogram for the users count showing the rating activities by the users. Some users rated movies more than the others

```
edx %>% count(userId) %>% ggplot(aes(n))+ geom_histogram(bins = 30, colour =
"black", position = "dodge",alpha=1)+ theme_bw()+ggtitle("Users Count")
+xlab("n")+ylab("Users")+scale_x_log10()
```



movieId attribute is a unique identifier of a movie. There are 10677 distinct movies and no missing values detected. Here is the histogram for the movie count showing the distribution of the movies rated by the users. Some movies are rated more than the others

```
edx %>% count(movieId) %>% ggplot(aes(n))+ geom_histogram(bins = 30, colour =
"black", position = "dodge",alpha=1)+ theme_bw()+ggtitle("Movies Count")
+xlab("n")+ylab("Movies")+scale_x_log10()
```

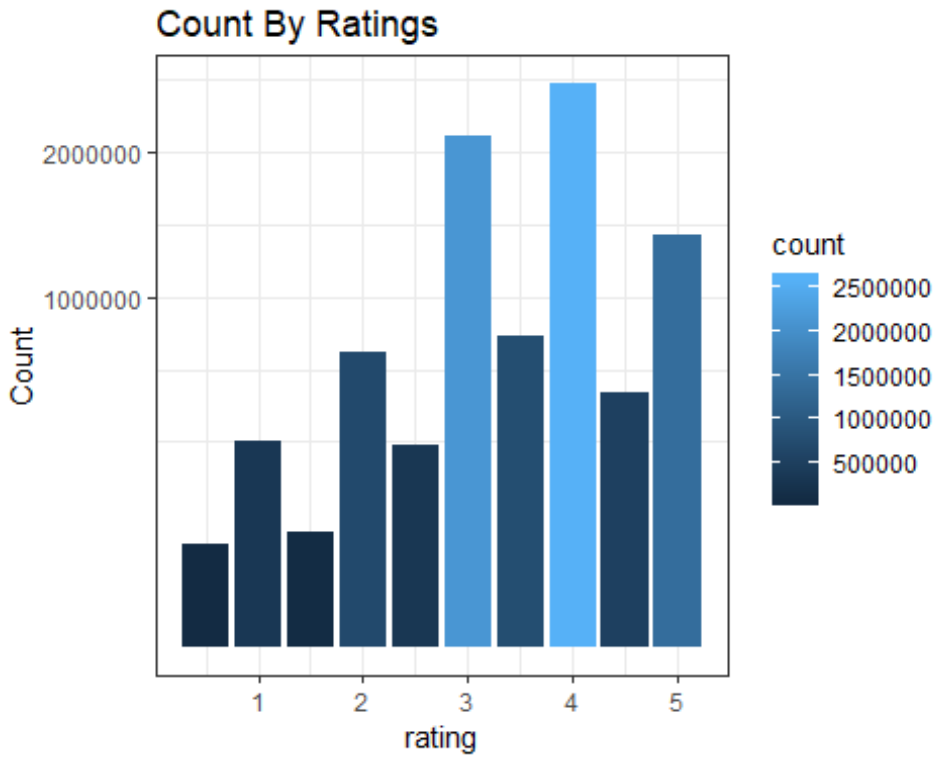


In addition, we are also checking to make sure that the combination between userid and movieid should be distinct and no duplication should exist.

```
#check no duplicate of user and movie id combination in the dataset
edx %>% group_by(userId,movieId) %>% summarise(count=n()) %>% filter(count>1)
%>% nrow

## [1] 0
```

rating attribute is a value given by a user for a movie. There are 10 distinct ratings spanning from 0 to 5 with a 0.5 unit of variation. No missing values detected. Here is the histogram for the rating value distribution. We can also see that the five most given ratings in order from most to least are 4, 3, 5, 3.5, 2 and that half star ratings are less common than whole star ratings

```
#Plot Count By Ratings
options(scipen=10000)
edx %>% group_by(rating) %>% summarise(count=n()) %>% ggplot(aes(y=count,
x=rating, fill=count))+geom_bar(stat="identity") +  ylab("Count") +
theme_bw()+ggtitle("Count By Ratings")+scale_y_sqrt()
```



timestamp attribute captures the datetime the rating was logged into the dataset. At this stage, we will not consider this variable in our model. Thus, we will not perform datetime conversion of the timestamp numeric

title attribute is the name of the movie appended with the year of release. This attribute will be useful later on when we map to the movieid, which is a much better reference in the analysis rather than using an id number. Interestingly, we discover one occurrence where two different movieIds have the same name of movie and year, namely 'War of The Worlds (2005'. We leave it as it is for now as the nature of the relationship still remains as a one-to-one mapping of movieId to title

```
#Same Title and Year But Different MovieId
edx %>% group_by(movieId,title) %>% summarise(count=n()) %>% group_by(title)
%>% summarise(count=n()) %>% filter(count>1)

## # A tibble: 1 x 2
##   title                     count
##   <chr>                     <int>
## 1 War of the Worlds (2005)     2

edx %>% filter(title=='War of the Worlds (2005)') %>% distinct(movieId)

##   movieId
## 1   34048
## 2   64997
```

genres attribute specifies a number of genres that categorize the movie theme. A movie can belong to multiple genres and this is formatted with '|' separated. This is the list of genre with drama and comedy being the two top most rated. For now, we will not be using genres in the model prediction.

```
#break down the genres into rows
edx %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>%
summarise(count = n()) %>% arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama               3910127
##  2 Comedy              3540930
##  3 Action              2560545
##  4 Thriller            2325899
##  5 Adventure           1908892
##  6 Romance             1712100
##  7 Sci-Fi              1341183
##  8 Crime               1327715
##  9 Fantasy              925637
## 10 Children             737994
## 11 Horror               691485
## 12 Mystery              568332
## 13 War                  511147
## 14 Animation            467168
## 15 Musical              433080
## 16 Western              189394
## 17 Film-Noir            118541
## 18 Documentary           93066
## 19 IMAX                   8181
## 20 (no genres listed)        7
```

Modeling Approach and Result

In this section, we are going to create and discuss about prediction modelling based on linear regression. First, we will implement a very simple average algorithm and then we explore if we can enhance the model by introducing effects and regularization

Residual Mean Squared Error (RMSE) is used as a measurement of performance for each of the model. RMSE basically will be the standard deviation of prediction errors (comparing between predicted value and true value in the validation set). Ideally, the lower the RMSE, the better prediction ability of the model. That said, a value greater than 1 will denote an error greater than one star predicted.

1. Simple Average Algorithm

Our very initial approach is to set a baseline by calculating the average rating value of the data in the training set edx. The model is simply expressed as

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

with sampling error ϵ and μ being single average value of all ratings. The average is then evaluated againts the rating in the validation set. The evaluation will be our first RMSE.

```
#Calculate the average of rating in train set
mu <- mean(edx$rating)

#Calculate RMSE
rmse <- RMSE(validation$rating, mu)
rmse

## [1] 1.061202

#Store the results
rmse_results <- data_frame(Model = "The Simple Average", RMSE = rmse)
rmse_results %>% knitr::kable()
```

Model	RMSE
The Simple Average	1.061202

2. Movie Effect

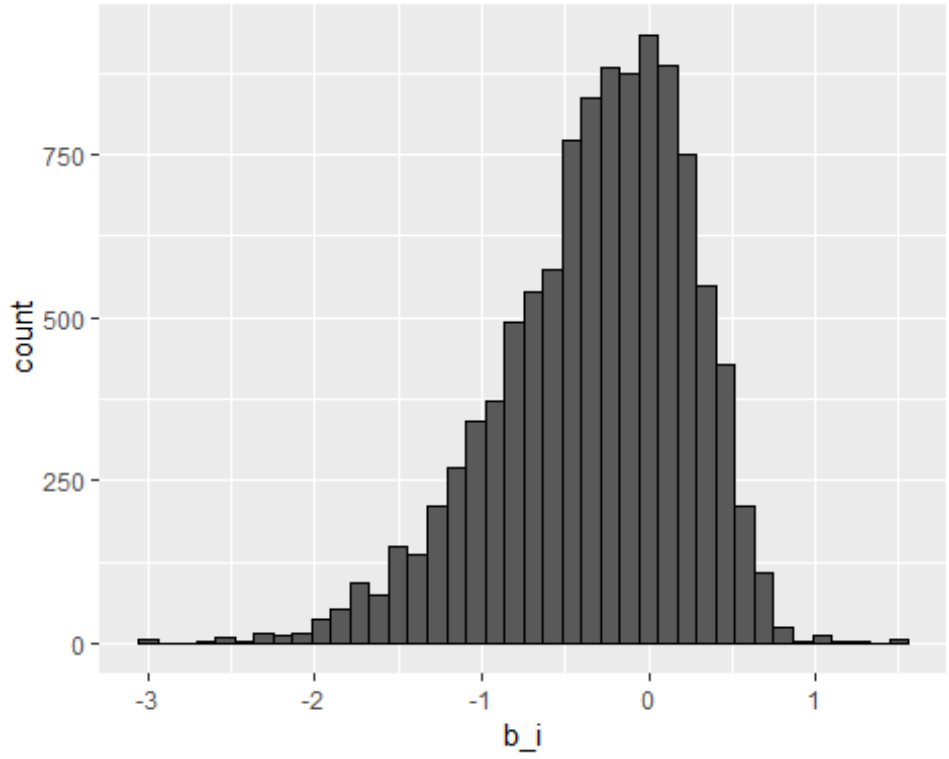
In this experiment, we are going to make some sense by stating that there is a bias factor in the movies being rated. That is understandable since not all movies are equally good. Most people can rave a particular movie and give rating of at least 4 star or maximum 5 star. Each movie will have its own bias effect and it can be different from the others. The model can then be augmented to include the bias b_i

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

To obtain the movie bias, we first calculate the difference value between rating and average rating for each n movie, then average out these differences and group by the movies. The same bias value will be used in the prediction model for the respective movie that exists in the validation set

$$b_i = \frac{1}{n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \mu)$$

```
#Calculate movie bias in train set, grouping by movieId and get the average
delta value of rating and average rating
movie_eff <- edx %>% group_by(movieId) %>% summarise(b_i = mean(rating - mu))

#Plotting the distribution of b_i and observe the variability
movie_eff %>% ggplot(aes(b_i)) + geom_histogram(bins=40,colour = "black",
position = "dodge",alpha=1)
```




```
#Assign the movie bias for validation set
b_i <-  validation %>% left_join(movie_eff, by='movieId') %>% .$b_i

#Enhanced model
predicted_ratings <- mu + b_i

#Calculate RMSE
rmse <- RMSE(predicted_ratings, validation$rating)

#Append the results to rmse_results
rmse_results <- bind_rows(rmse_results, data_frame(Model="Movie Effect
Model", RMSE = rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
The Simple Average	1.0612018
Movie Effect Model	0.9439087

We can see that our movie effect model has now improved with a lower RMSE

3. Regularised Movie Effect

Despite the Movie Effect model generating a lower RMSE than the 'Simple Average' algorithm, it can still prone to overfitting problem, which is often the case when the model is too simple and very likely to capture the noise of the data.

When we run the following script, we will notice that movies with lowest or highest b_i are mostly unknown. This is shown by the value of number of ratings made for a movie (n), which is a low figure. Just because only few number of users rated the movies, it does not mean that the ratings can be generally trusted.

```
#Top 10 best movies from the Movie Effect model
edx %>% left_join(movie_eff, by="movieId") %>% group_by(title,movieId, b_i)
%>% summarise(count=n()) %>% arrange(desc(b_i))  %>% head(.,10) %>%
knitr::kable()
```

title	movieId	b_i	count
Blue Light, The (Das Blaue Licht) (1932)	64275	1.487535	1
Fighting Elegy (Kenka erejii) (1966)	51209	1.487535	1
Hellhounds on My Trail (1999)	3226	1.487535	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	33264	1.487535	2
Shadows of Forgotten Ancestors (1964)	42783	1.487535	1
Sun Alley (Sonnenallee) (1999)	53355	1.487535	1
Constantine's Sword (2007)	65001	1.237535	2
Human Condition II, The (Ningen no joken II) (1959)	26048	1.237535	4
Human Condition III, The (Ningen no joken III) (1961)	26073	1.237535	4
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	5194	1.237535	4

```
#Top 10 worst movies from the Movie Effect model
edx %>% left_join(movie_eff, by="movieId") %>% group_by(title,movieId, b_i)
%>% summarise(count=n()) %>% arrange(b_i)  %>% head(.,10) %>% knitr::kable()
```

title	movieId	b_i	count
Accused (Anklaget) (2005)	61768	-3.012465	1
Besotted (2001)	5805	-3.012465	2
Confessions of a Superhero (2007)	63828	-3.012465	1
Hi-Line, The (1999)	8394	-3.012465	1
War of the Worlds 2: The Next Wave (2008)	64999	-3.012465	2
SuperBabies: Baby Geniuses 2 (2004)	8859	-2.717822	56
Hip Hop Witch, Da (2000)	7282	-2.691037	14
Disaster Movie (2008)	61348	-2.653090	32
From Justin to Kelly (2003)	6483	-2.610455	199
Criminals (1996)	604	-2.512465	2

Regularization technique can be applied to mitigate this by adding a penalty term lambda λ. Essentially, our objective is to penalize/minimize the equation with the lambda. The

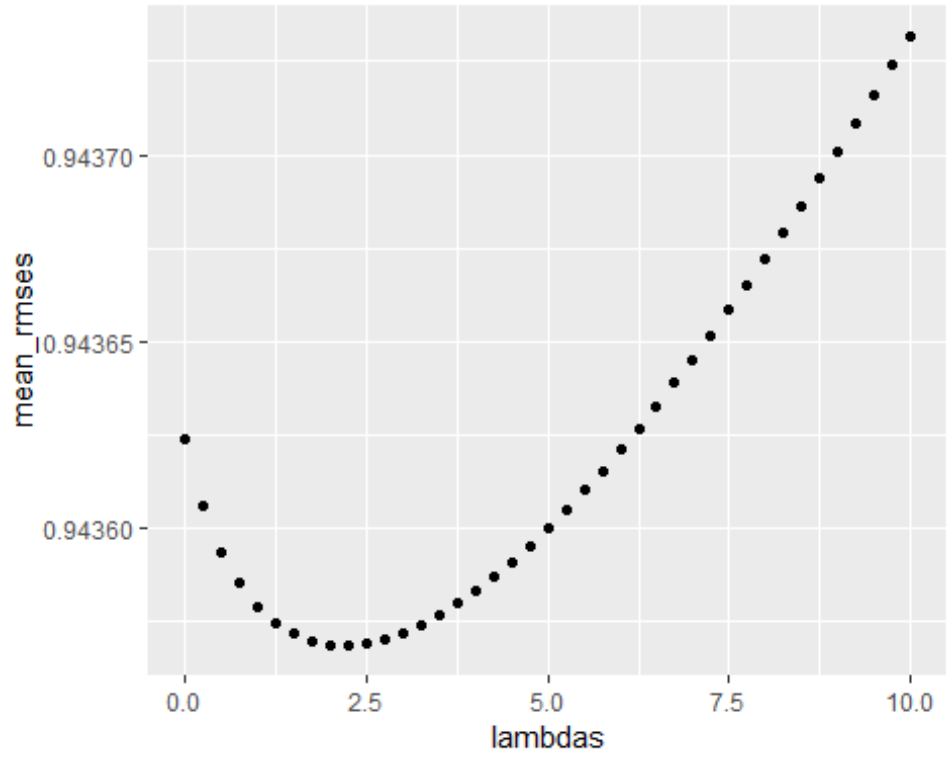
greater value of the lambda, the more the bias value will shrink. Consequently, a large n will make the lambda value trivial, hence giving more consistency of the model estimate.

$$b_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \mu)$$

The next question is how can we tune in the lambda. For this, we will perform 10-fold cross validations. For each fold, we apply a set of sequence of lambda values from 0 to 5 with an increment of 0.25. Also for each fold, we use one training set and the remaining folds will be the test sets and we repeat this procedure until all folds have been used. From this exercise of tuning, the lambda with the minimum average result of loss function RMSE will be chosen as the penalty term

```
#create an empty list
rmse_cv <- list()
set.seed(1)

#set a sequence of lambdas values
lambdas <- seq(0, 10, 0.25)

#set the number of fold
k<-10

#perform the cross validation
folds <- cvFolds(NROW(edx), K=k)
for(i in 1:k)
{
    #split to train and test set
    train_cv <- edx[folds$subsets[folds$which != i], ] #Set the training set
    temp_cv <- edx[folds$subsets[folds$which == i], ] #Set the test set

    test_cv <- temp_cv %>% semi_join(train_cv , by = "movieId") %>%
semi_join(train_cv , by = "userId")

    # Add rows removed from validation set back into train set
    removed_cv <- anti_join(temp_cv, test_cv)
    train_cv <- rbind(train_cv, removed_cv)

    #get average rating value from train set
    mu <- mean(train_cv$rating)

    #summing bi_i and count n by movieId
    summ<- train_cv %>% group_by(movieId) %>% summarise(s = sum(rating - mu),
n_i = n())

    #calculate rmses for each lambda sequence
    rmses <- sapply(lambdas, function(l){
        predicted_ratings <- test_cv %>% left_join(summ, by='movieId') %>%
```

```
        mutate(b_i = s/(n_i+l)) %>% mutate(pred = mu + b_i) %>% .$pred
      return(RMSE(predicted_ratings, test_cv$rating))
    })

    #store RMSE value into a list
    rmse_cv[[i]]<-rmses
}

#convert list to dataframe
rmse_cv_df <- as.data.frame(do.call(rbind, rmse_cv))

#calculate average RMSE
mean_rmses <- colMeans(rmse_cv_df)

#plot lambda and rmse
qplot(lambdas, mean_rmses)
```




```
#find lambda value with the lowest average RMSE
lambda_cv <- lambdas[which.min(mean_rmses)]
lambda_cv

## [1] 2.25
```

Once we obtain the optimal lambda, we then apply it to the calculation of b_i in the existing Movie Effect Model

```
#Regularised Movie Effect Model
lambda <- lambda_cv

#Calculate b_i with lambda penalty term
reg_movie_eff <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating -
mu)/(n()+lambda), n_i = n())
```

Now take a look at the top 10 best and worst movies after we implement the regularization. The list shows the result of a more reliable estimate of b_i

```
#Top 10 best movies from the Movie Effect model after regularized
edx %>% left_join(reg_movie_eff , by="movieId") %>% group_by(title,movieId,
b_i) %>% summarise(count=n()) %>% arrange(desc(b_i))  %>% head(.,10) %>%
knitr::kable()
```

title	movieId	b_i	count
Shawshank Redemption, The (1994)	318	0.9426756	28015
More (1998)	4454	0.9095504	7
Godfather, The (1972)	858	0.9028717	17747
Usual Suspects, The (1995)	50	0.8533851	21648
Schindler's List (1993)	527	0.8510309	23193
Casablanca (1942)	912	0.8078821	11232
Rear Window (1954)	904	0.8060431	7935
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	922	0.8028815	2922
Third Man, The (1949)	1212	0.7984404	2967
Double Indemnity (1944)	3435	0.7976041	2154

```
#Top 10 worst movies from the Movie Effect model after regularized
edx %>% left_join(reg_movie_eff , by="movieId") %>% group_by(title,movieId,
b_i) %>% summarise(count=n()) %>% arrange(b_i)  %>% head(.,10) %>%
knitr::kable()
```

title	movieId	b_i	count
SuperBabies: Baby Geniuses 2 (2004)	8859	-2.612760	56
From Justin to Kelly (2003)	6483	-2.581186	199
Disaster Movie (2008)	61348	-2.478720	32
PokÃ©mon Heroes (2003)	6371	-2.443059	137
Carnosaur 3: Primal Species (1996)	3574	-2.346503	68
Glitter (2001)	4775	-2.321456	339
Hip Hop Witch, Da (2000)	7282	-2.318358	14
Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)	5672	-2.308449	202
Gigli (2003)	6587	-2.302537	313

Barney's Great Adventure (1998) 1826 -2.300000 208

Let's see our new prediction with the enhanced model

```
#Enhanced Model
predicted_ratings <- validation %>% left_join(reg_movie_eff, by='movieId')
%>%
 mutate(pred = mu + b_i) %>% .$pred

#Calculate RMSE
rmse <- RMSE(predicted_ratings, validation$rating)

#Append the results to rmse_results
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie Effect Model",
                                     RMSE = rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
The Simple Average	1.0612018
Movie Effect Model	0.9439087
Regularized Movie Effect Model	0.9438521

The result is a slight improvement to the non-regularized version of Movie Effect model

4. Movie + User Effect

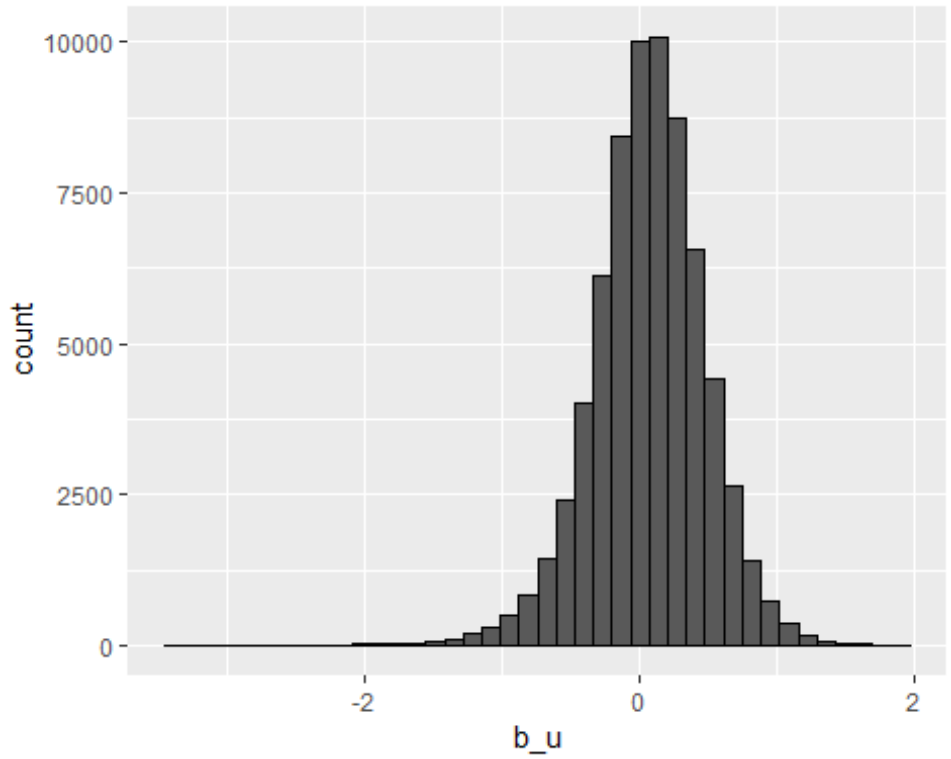
Next, let's see if we can explore further another factor to enhance the model. It is fair to say that every individual user is different when rating for the same movie. Some people can be quite critical and do not rave movies easily, whereas some set lower expectation and are often pleased with decent movies.

This bring us to the idea of introducing a combination of movie bias (b_i) and user bias (b_u) into the model.

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

We have discussed about how we come out with the value of the movie bias (b_i) for each movie. This time, we perform a similar fashion to determine the user bias (b_u) for each user.

```
#Calculate user bias in train set, grouping by userId and get the average
delta value of (rating-b_i) and average rating
user_eff <- edx %>% left_join(movie_eff, by='movieId') %>% group_by(userId)
%>% summarise(b_u = mean(rating - mu - b_i))

#Plotting the distribution of b_u and observe the variability
user_eff %>% ggplot(aes(b_u)) + geom_histogram(bins=40,colour = "black",
position = "dodge",alpha=1)
```




```
#Assign the user bias for validation set
b_u <-  validation %>% left_join(movie_eff, by='movieId') %>%
left_join(user_eff, by='userId') %>% .$b_u

#Enhanced model
predicted_ratings <- mu + b_i + b_u

#Calculate RMSE
rmse <- RMSE(predicted_ratings, validation$rating)

#Append the results to the table
rmse_results <- bind_rows(rmse_results,data_frame(Model="Movie + User Effect
Model",RMSE = rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
The Simple Average	1.0612018
Movie Effect Model	0.9439087
Regularized Movie Effect Model	0.9438521
Movie + User Effect Model	0.8653488

By using both movie and user bias, our intention is that both effects are able to balance out the weight in the model and come up with a better prediction rating. We have achieved much more lower RMSE compared with the first 3 experiments. However, we are yet to apply regularization into this Movie+User Effect model.

5. Regularized Movie + User Effect

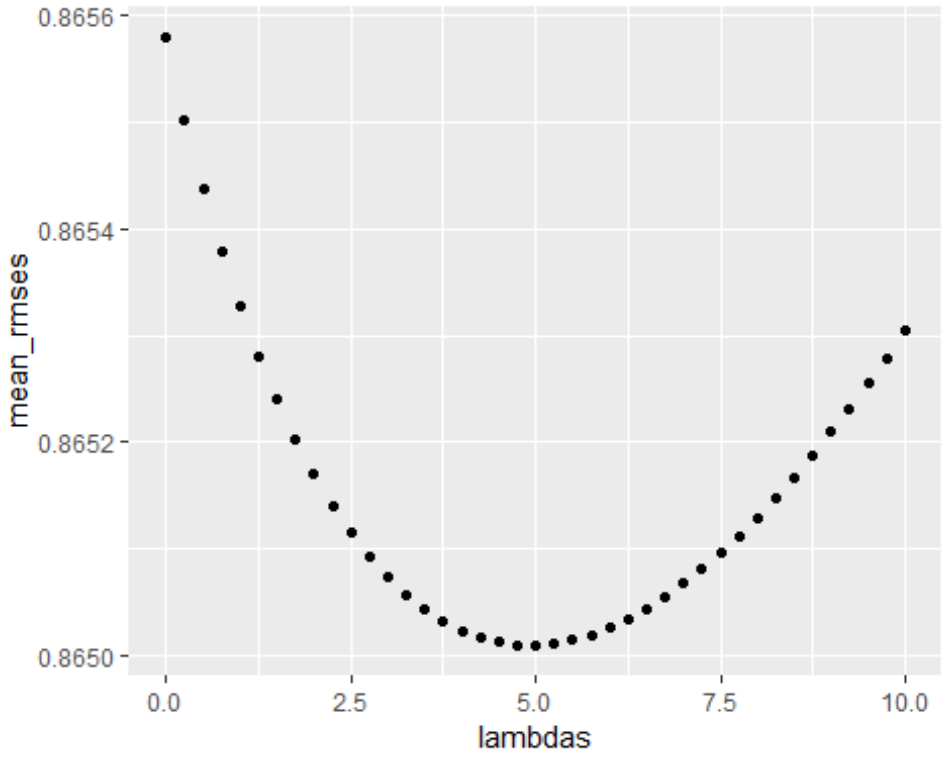
Now we are going to apply the same regularisation method that we applied earlier on the regularized movie effect model, but this time we will need to recalculate the user bias b_u with a lambda λ value. Similarly, 10-fold cross validations used.

```
#create an empty list
rmse_cv <- list()
set.seed(1)

#set a sequence of lambdas values
lambdas <- seq(0, 10, 0.25)

#set the number of fold
k<-10

#perform the cross validation
folds <- cvFolds(NROW(edx), K=k)

for(i in 1:k)
{
    #split to train and test set
    train_cv <- edx[folds$subsets[folds$which != i], ] #Set the training set
    temp_cv <- edx[folds$subsets[folds$which == i], ] #Set the test set

    test_cv <- temp_cv %>% semi_join(train_cv , by = "movieId") %>%
semi_join(train_cv , by = "userId")

    # Add rows removed from validation set back into edx set
    removed_cv <- anti_join(temp_cv, test_cv)
    train_cv <- rbind(train_cv, removed_cv)

    #calculate rmses for each lambda sequence
    rmses <- sapply(lambdas, function(l){

    #get average rating value from train set
    mu <- mean(train_cv$rating)

    #calculate b_i value with lambda
    b_i <- train_cv %>% group_by(movieId) %>% summarise(b_i = sum(rating -
mu)/(n()+l))

    #calculate b_u value with lambda
    b_u <- train_cv %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
summarise(b_u = sum(rating - b_i - mu)/(n()+l))

    #predict rating with test set
    predicted_ratings <- test_cv%>% left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>% .$pred
```

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

    #store RMSE value into a list
    rmse_cv[[i]]<-rmses
}
#convert list to dataframe
rmse_cv_df <- as.data.frame(do.call(rbind, rmse_cv))
rmse_cv_df
```

```
##           V1        V2        V3        V4        V5        V6        V7
## 1  0.8652175 0.8651459 0.8650834 0.8650279 0.8649782 0.8649338 0.8648940
## 2  0.8650855 0.8650035 0.8649346 0.8648747 0.8648219 0.8647749 0.8647331
## 3  0.8659216 0.8658424 0.8657754 0.8657167 0.8656647 0.8656184 0.8655769
## 4  0.8658186 0.8657398 0.8656715 0.8656110 0.8655569 0.8655082 0.8654644
## 5  0.8648786 0.8648069 0.8647470 0.8646953 0.8646500 0.8646102 0.8645751
## 6  0.8661947 0.8661182 0.8660514 0.8659920 0.8659389 0.8658912 0.8658483
## 7  0.8649098 0.8648416 0.8647827 0.8647307 0.8646845 0.8646433 0.8646066
## 8  0.8670044 0.8669170 0.8668432 0.8667786 0.8667212 0.8666698 0.8666237
## 9  0.8651815 0.8651069 0.8650420 0.8649844 0.8649329 0.8648866 0.8648451
## 10 0.8655902 0.8655070 0.8654363 0.8653745 0.8653196 0.8652705 0.8652265
##           V8        V9       V10       V11       V12       V13       V14
## 1  0.8648584 0.8648267 0.8647986 0.8647740 0.8647525 0.8647340 0.8647183
## 2  0.8646959 0.8646628 0.8646336 0.8646079 0.8645854 0.8645661 0.8645496
## 3  0.8655400 0.8655071 0.8654780 0.8654525 0.8654301 0.8654109 0.8653946
## 4  0.8654251 0.8653898 0.8653582 0.8653302 0.8653054 0.8652837 0.8652650
## 5  0.8645443 0.8645174 0.8644942 0.8644742 0.8644574 0.8644436 0.8644324
## 6  0.8658099 0.8657755 0.8657448 0.8657176 0.8656936 0.8656728 0.8656548
## 7  0.8645741 0.8645454 0.8645201 0.8644982 0.8644794 0.8644634 0.8644502
## 8  0.8665822 0.8665451 0.8665118 0.8664822 0.8664560 0.8664329 0.8664128
## 9  0.8648078 0.8647746 0.8647450 0.8647189 0.8646960 0.8646761 0.8646592
## 10 0.8651871 0.8651518 0.8651203 0.8650924 0.8650678 0.8650464 0.8650278
##          V15       V16       V17       V18       V19       V20       V21
## 1  0.8647053 0.8646947 0.8646866 0.8646807 0.8646770 0.8646753 0.8646756
## 2  0.8645358 0.8645246 0.8645158 0.8645093 0.8645050 0.8645028 0.8645025
## 3  0.8653809 0.8653699 0.8653613 0.8653550 0.8653509 0.8653489 0.8653489
## 4  0.8652489 0.8652355 0.8652244 0.8652158 0.8652093 0.8652049 0.8652026
## 5  0.8644239 0.8644178 0.8644140 0.8644124 0.8644129 0.8644153 0.8644197
## 6  0.8656395 0.8656268 0.8656165 0.8656085 0.8656027 0.8655990 0.8655973
## 7  0.8644395 0.8644313 0.8644255 0.8644218 0.8644202 0.8644206 0.8644229
## 8  0.8663956 0.8663809 0.8663688 0.8663591 0.8663516 0.8663462 0.8663429
## 9  0.8646449 0.8646332 0.8646239 0.8646169 0.8646122 0.8646095 0.8646087
## 10 0.8650120 0.8649988 0.8649881 0.8649797 0.8649735 0.8649695 0.8649674
##          V22       V23       V24       V25       V26       V27       V28
## 1  0.8646777 0.8646816 0.8646871 0.8646943 0.8647030 0.8647132 0.8647248
## 2  0.8645041 0.8645074 0.8645125 0.8645192 0.8645274 0.8645372 0.8645483
## 3  0.8653508 0.8653545 0.8653599 0.8653670 0.8653757 0.8653858 0.8653974
## 4  0.8652021 0.8652034 0.8652065 0.8652113 0.8652176 0.8652254 0.8652347
## 5  0.8644258 0.8644336 0.8644431 0.8644541 0.8644666 0.8644805 0.8644957
```

```
## 6  0.8655975 0.8655994 0.8656031 0.8656085 0.8656153 0.8656237 0.8656336
## 7  0.8644270 0.8644328 0.8644403 0.8644493 0.8644599 0.8644718 0.8644852
## 8  0.8663415 0.8663420 0.8663442 0.8663481 0.8663536 0.8663606 0.8663691
## 9  0.8646099 0.8646129 0.8646176 0.8646239 0.8646319 0.8646413 0.8646521
## 10 0.8649673 0.8649690 0.8649724 0.8649775 0.8649841 0.8649923 0.8650020
##          V29       V30       V31       V32       V33       V34       V35
## 1  0.8647377 0.8647519 0.8647673 0.8647840 0.8648017 0.8648206 0.8648405
## 2  0.8645608 0.8645746 0.8645896 0.8646058 0.8646232 0.8646416 0.8646611
## 3  0.8654103 0.8654246 0.8654401 0.8654569 0.8654748 0.8654938 0.8655139
## 4  0.8652454 0.8652574 0.8652707 0.8652852 0.8653008 0.8653176 0.8653355
## 5  0.8645123 0.8645300 0.8645490 0.8645691 0.8645903 0.8646125 0.8646357
## 6  0.8656448 0.8656573 0.8656710 0.8656860 0.8657021 0.8657194 0.8657377
## 7  0.8644998 0.8645158 0.8645329 0.8645511 0.8645705 0.8645910 0.8646124
## 8  0.8663791 0.8663903 0.8664029 0.8664167 0.8664316 0.8664477 0.8664649
## 9  0.8646643 0.8646779 0.8646927 0.8647087 0.8647259 0.8647442 0.8647635
## 10 0.8650130 0.8650254 0.8650391 0.8650539 0.8650700 0.8650872 0.8651054
##          V36       V37       V38       V39       V40       V41
## 1  0.8648613 0.8648832 0.8649060 0.8649296 0.8649541 0.8649795
## 2  0.8646816 0.8647031 0.8647255 0.8647488 0.8647730 0.8647979
## 3  0.8655350 0.8655570 0.8655801 0.8656040 0.8656288 0.8656545
## 4  0.8653544 0.8653743 0.8653951 0.8654169 0.8654396 0.8654631
## 5  0.8646598 0.8646849 0.8647109 0.8647377 0.8647653 0.8647937
## 6  0.8657570 0.8657773 0.8657986 0.8658207 0.8658438 0.8658677
## 7  0.8646349 0.8646582 0.8646825 0.8647077 0.8647337 0.8647605
## 8  0.8664832 0.8665024 0.8665226 0.8665438 0.8665658 0.8665887
## 9  0.8647839 0.8648053 0.8648276 0.8648508 0.8648749 0.8648999
## 10 0.8651248 0.8651451 0.8651663 0.8651885 0.8652116 0.8652356

#calculate average RMSE
mean_rmses <- colMeans(rmse_cv_df)

#plot lambda and rmse
qplot(lambdas, mean_rmses)
```




```
#find lambda value with the lowest average RMSE
lambda_cv <- lambdas[which.min(mean_rmses)]
lambda_cv

## [1] 5

#Regularised Movie+User Effect Model
lambda <- lambda_cv

#Calculate b_i with lambda penalty term
b_i <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating -
mu)/(n()+lambda))

#Calculate b_u with lambda penalty term
b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))

#predict ratings with the validation set
predicted_ratings <- validation%>% left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>% .$pred

#Calculate RMSE
rmse <- RMSE(predicted_ratings, validation$rating)

#Append the results to rmse_results
rmse_results <- bind_rows(rmse_results,data_frame(Model="Regularized Movie +
```

```
User Effect Model",RMSE = rmse ))
rmse_results %>% knitr::kable()
```

Model	RMSE
The Simple Average	1.0612018
Movie Effect Model	0.9439087
Regularized Movie Effect Model	0.9438521
Movie + User Effect Model	0.8653488
Regularized Movie + User Effect Model	0.8648178

Finally, we can produce the lowest RMSE among all other efforts that we had earlier by taking into account the regularisation into the movie+user effect model.

Conclusion

We began with the exploration on MovieLens data set where we looked at the content and structure of the movie ratings data. We went through the attributes and provided some visualization for better grasp on how each feature related to rating. As been provided by the code in the project, the dataset was split into training and validation set.

We were able to produce models for predicting the movie ratings using regression model that incorporate movie and user bias as well as implement the regularization technique. All the models, except the Simple Average resulted in RMSE below 1, with Regularized Movie+User effect model had the lowest loss function.

Despite this, we have not yet achieved much lower RMSE, leaving the room for improvement. For a later stage, we can possibly look into bringing the movie genre into the model and also conduct a couple experiments with other algorithms.