Movielens Recommendation System Project

Daping Zhang

July 20, 2022

1. Introduction

Recommendation systems, or recommender systems, use ratings that users have given items to make specific recommendations. The system that seeks to predict or filter preferences according to the user's choices. Recommender systems are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general.

A movie recommendation system is to collect datasets that can be used to predict what rating a particular user will give a specific movie. Movies for which a high rating is predicted for a given user are then recommended to that user. Usually recommendation systems are based on a rating scale from 1 to 5 grades or stars, with 1 indicating lowest satisfaction and 5 is the highest satisfaction.

This report represents the buildings of the Movielens Recommendation System that consists of the preparation of data set, the exploration and analysis of data, the methods and algorithms of building models, and the evaluation and selection of optimization models.

During preliminary data exploration and modeling, several machine learning algorithm will be used and the results will be compared to get maximum possible accuracy in predictions. All machine learning models are evaluated and chosen based on the RMSE - Root Mean Squared Error.

During the developing of our algorithm we will be using the edx set and split it into a training set and a test set to design and test our algorithms; for the final algorithm test, predict movie ratings, and evaluating we will be using the validation set.

The goal of this machine learning project is to build a movie recommendation system that it can predict how many stars a user will give a specific movie and make the prediction accuracy of the model (we use RMSE) less than 0.86490. The main methods that we will use to build the recommendation system are movie effect model, movie plus user effects model, regularized movie effects model, regularized movie plus user effects model, and matrix factorization (recosystem) model.

2. Data Preparation

2.1 Importing the libraries and the Dataset

For this project, we will be creating a movie recommendation system using the MovieLens dataset. We will use the 10M version of the MovieLens dataset to make the computation a little easier. The Movielens 10M dataset will be downloaded from "https://grouplens.org/datasets/movielens/10m/".The dataset will be separated into a training dataset named edx and a validation dataset named validation.

```r
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(kableExtra)
library(ggplot2)
library(stringr)
library(tidyr)
library(recosystem)

# 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)

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 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")
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)
```

2.2 Initial exploring edx dataset and validation dataset

The edx dataset is a subset of Movielens dataset, which is to use for the building of our models. The edx dataset has 9000055 rows (observations) and 6 columns. Each row represents a rating given by one user for on movie. Columns include userId, movieID, rating, timestamp, title and genres.

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

There are 69,878 different users and 10,677 different movies in edx dataset and there are 68,534 different users and 9,809 different movies in validation dataset.

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
validation %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

Below are first 6 rows of edx dataset

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

2.3 Checking missing values

There is no missing values in the edx dataset and the validation dataset.

```
sum(is.na(edx))
```

```
## [1] 0
```

```
sum(is.na(validation))
```

```
## [1] 0
```

2.4 Cleaning the Data

Separate multiple genres into individual genres for later analysis

```
edx_genre <- edx %>% separate_rows(genres, sep = "\\|")
```

Extract year released from title for later analysis

```
edx_year_released <- edx %>% mutate(year_released = as.numeric(str_sub(title, -5, -2)))
```

Extract year rated from timestamp for later analysis

```
edx_year_rated <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
```

2.5 Splitting the edx set into training set and test set

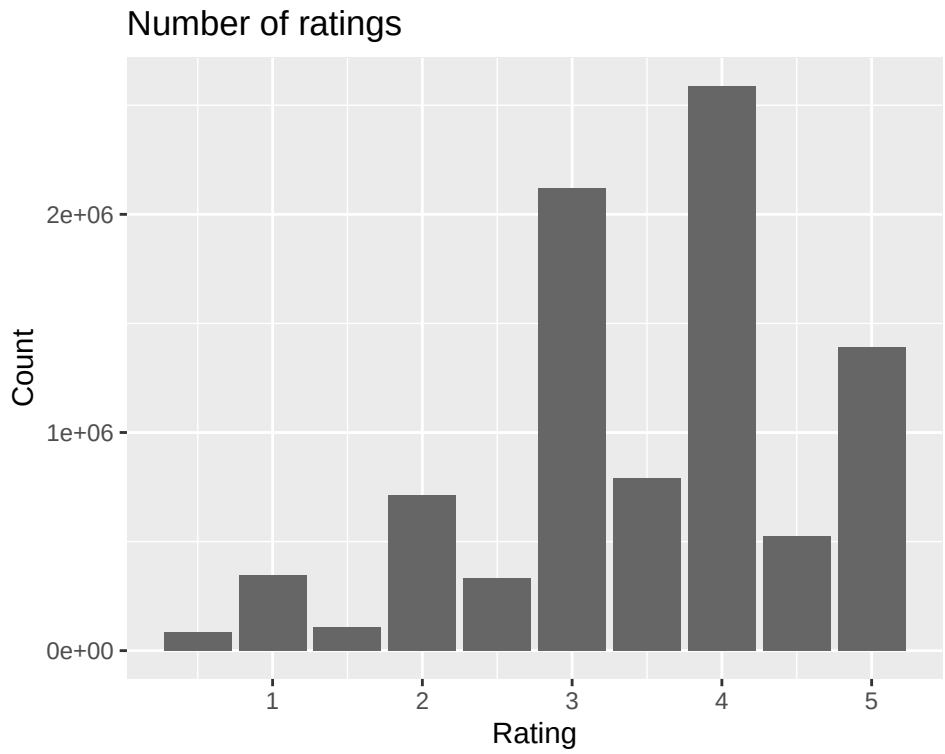
```
# Test set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

3. Data Exploration and Analysis

Before we use some methods to build some models and evaluate them based on training and test sets split from edx dataset, we need to better understand by exploring and analyzing more details of each feature and outcome of edx dataset.

3.1 Ratings

We can see that given ratings are different from 0.5 to 5.0, and half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.). But 4.0, 3.0, and 5.0 are most given ratings.



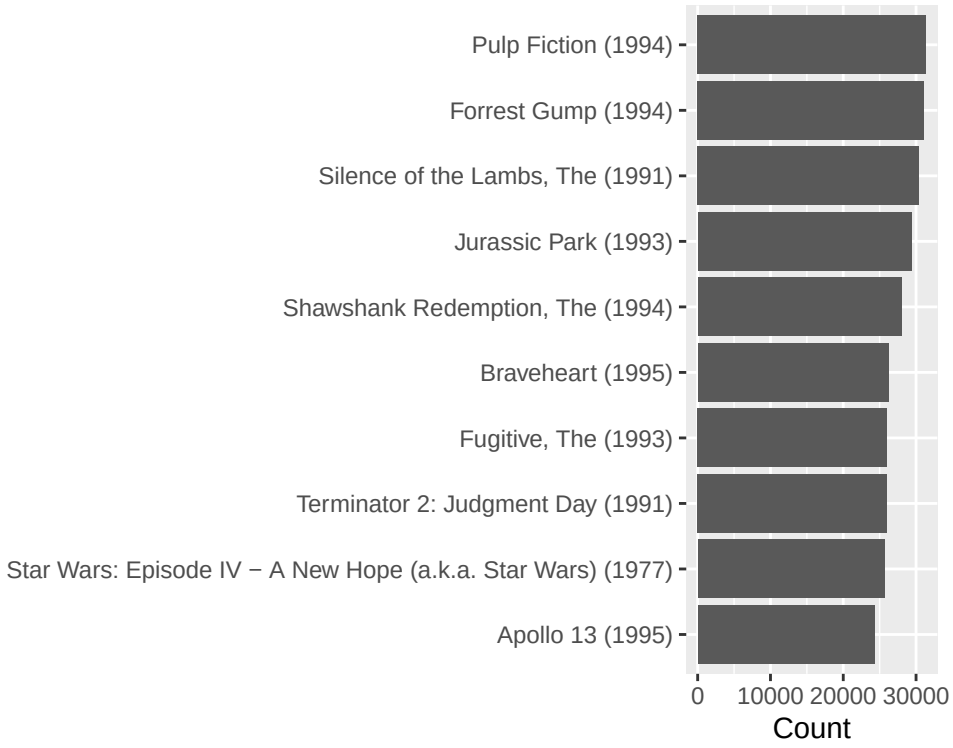
There were no zeros given for the rating in edx dataset.

```
edx %>% filter(rating == 0) %>% tally()
```

Below are top 10 movies by ratings:

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

Below are top 10 movies by titles:

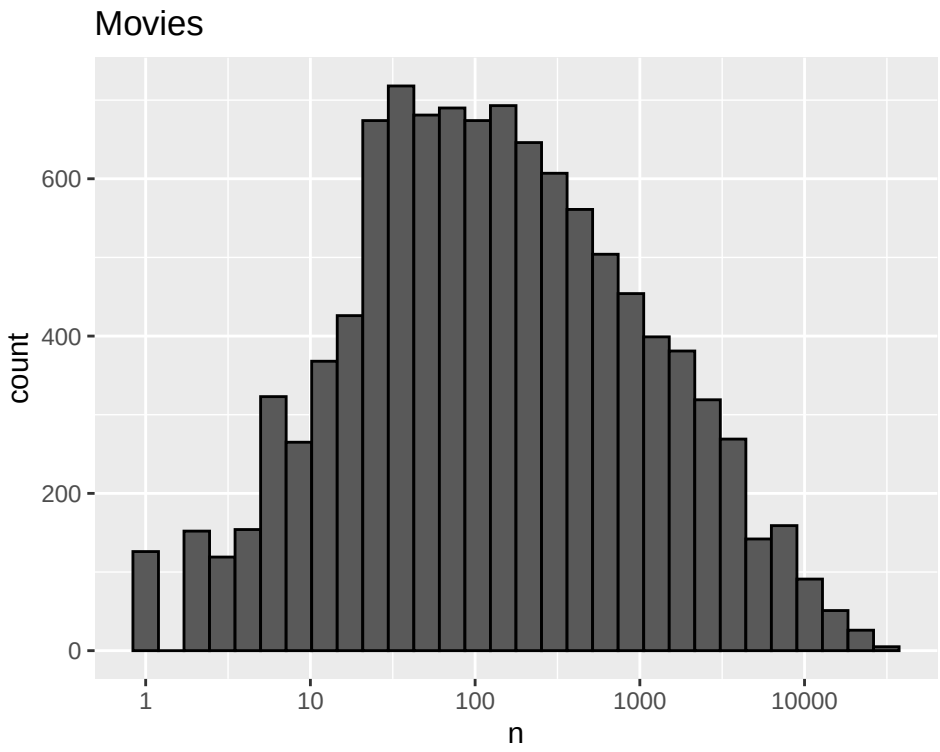


In order to build a good predictive model, we will further explore the effects of different features.

3.2 Movies

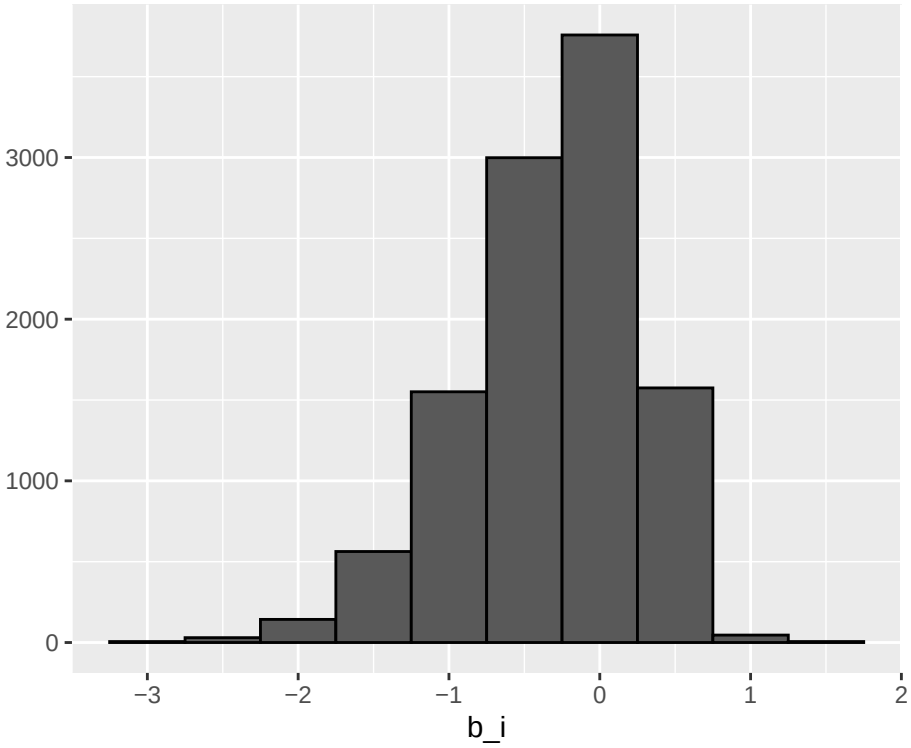
The distribution in the plot below shows that some movies are rated higher than others. That's because millions of people watched blockbusters and only a few watched artsy, independent movies. This explores the movie bias.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



We also know from experience that some movies are more popular than others and receive higher ratings. This tells us that different movies are rated differently. We can calculate the difference between ratings and the average ratings for movie i , which is term b_i, to see the distribution on plot below. We can see that the estimates vary widely.

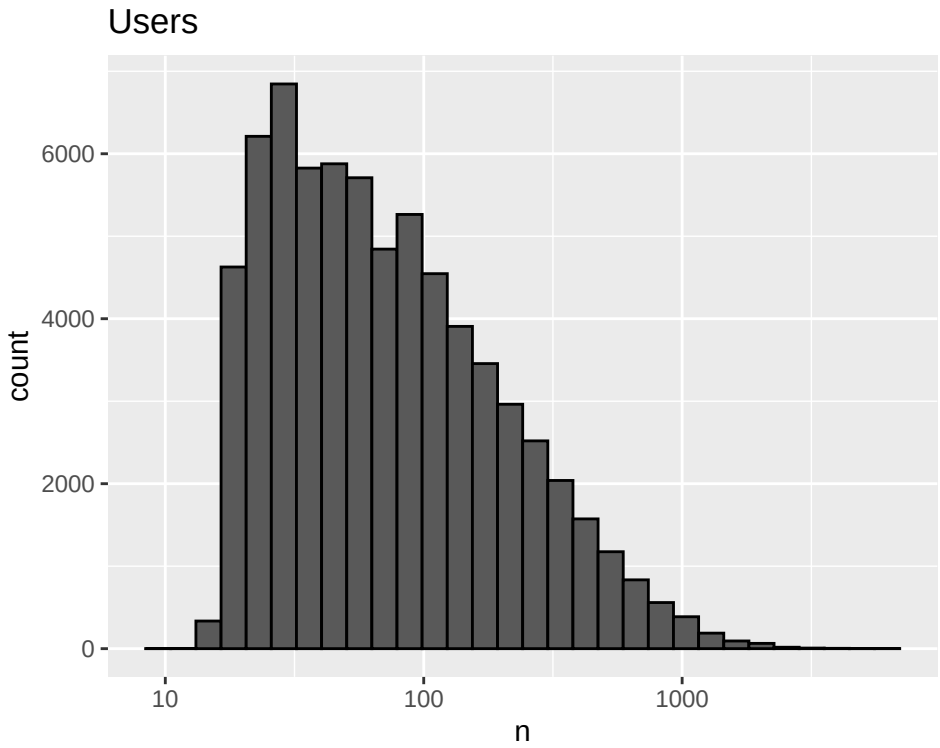
```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, data = ., bins = 10, color = I("black"))
```



3.3 Users

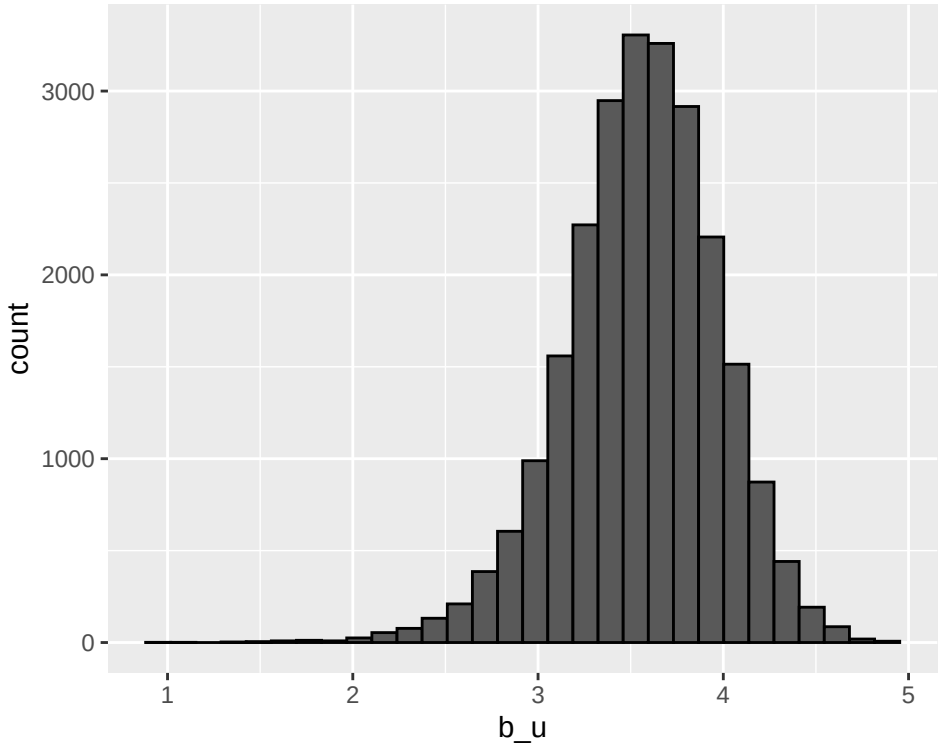
The distribution in the plot belolw shows that some users are more active than others when it comes to rating movies. This explores the user bias.

```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



Let's compute the average rating for user u for those that have rated 100 or more movies:

```
edx %>% group_by(userId) %>%
  filter(n() >= 100) %>%
  summarise(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



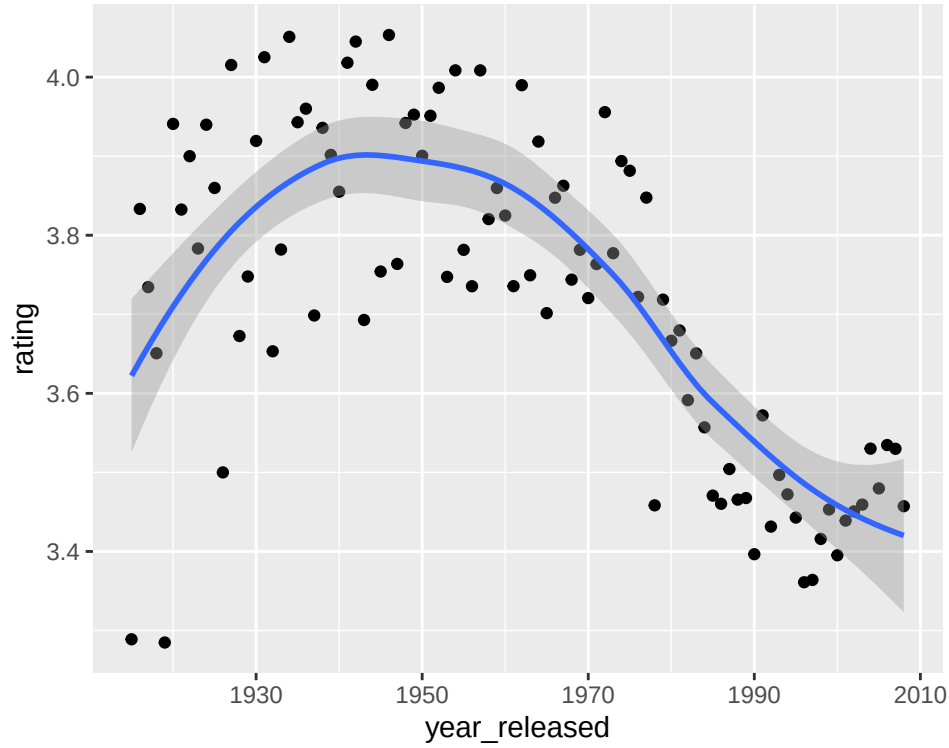
The distribution in the graph above shows that there are also large differences among users: some users are only interested in a few movies, while others like every movie. This means that considering user effects helps improve our predictive models.

3.4 Years

3.4.1 Year released

The trend in the plot below shows that users rated movies released before 1970 higher, while users rated movies released after 1970 relatively lower.

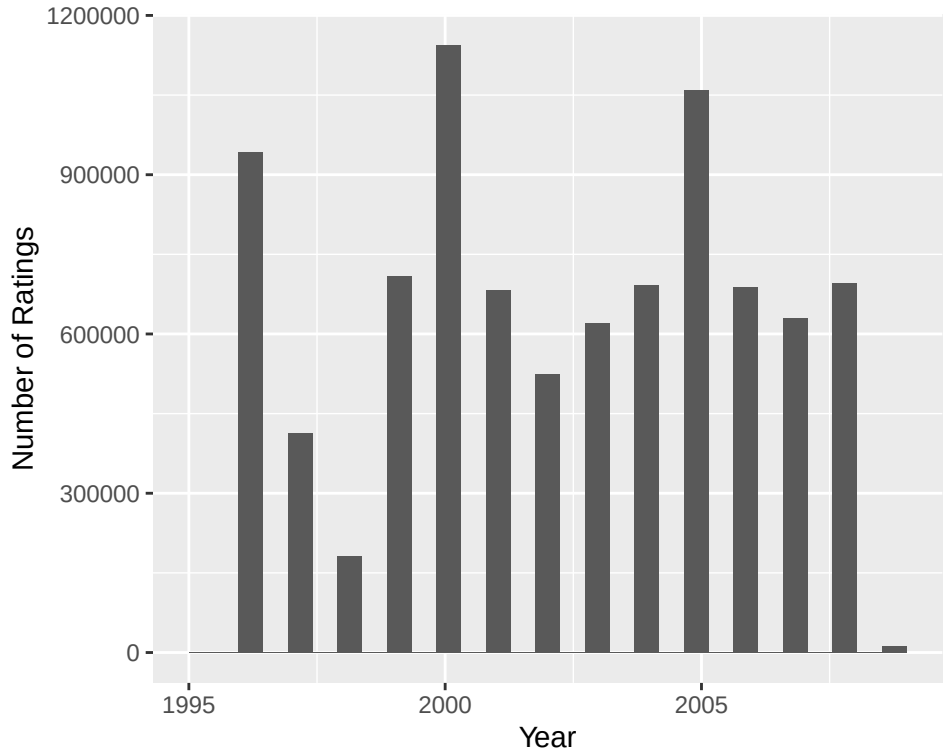
```
edx_year_released %>% group_by(year_released) %>%
  summarise(rating = mean(rating)) %>%
  ggplot(aes(year_released, rating)) +
  geom_point() +
  geom_smooth()
```



3.4.2 Year rated

Let's take a look at the distribution of the ratings given by year.

```
edx_year_rated %>% mutate(year_rated = year(as_datetime(timestamp))) %>%
    ggplot(aes(year_rated)) +
    geom_histogram() +
    labs(x = "Year", y = "Number of Ratings")
```



The distribution above shows that some years have higher ratings and some have lower ratings. The ratings by year rated are irregular. In this project, we will not consider it as a influencing factor.

3.5 Genres

From the list of genres below we see that there is 20 individual genres and the "Drama" has the highest the rating numbers.

```
edx_genre_analysis <- edx_genre %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
edx_genre_analysis
```

```
## # 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
```

For our movie recommendation system, every rating for a certain movie rated by a certain user is one record. If we separate multiple genres into indivisaul genres and it will produce more rows and it will mess up the dataset structure and even cause inconsistencies in the data relationship. In this project we do not consider genre in modeling.

4. Building and Evaluating Models

4.1 Modeling overview

We have explored the data that helps us to build a better predictive model. From the exploration and analysis, we see that the genres, year released and year rated are not the main factors affecting the rating. In the following modeling and evaluation process, we will focus on the analysis the effects of users and movies, we will build several machine learning models and compare them by measuring their RMSE values. The best model is chosen based on it having the lowest RMSE.

Models are constructed from simple to complex. The models will be baseline model, movie effective model, movie + user effective model, regularized movie effects model, regularized movie + user effects model, matrix factorization using recosystem model.

To avoid over-training caused by estimates from small sample sizes, we will use regularization to add penalties to shrink the estimates that are formed using small sample sizes.

4.2 Defining the RMSE

Accuracy will be evaluated using the residual mean squared error (RMSE) on a test set.

Below we build a function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

4.3 Developing the algorithms and building the models

Model 1: Simple Model

This model predicts the sane rating for all movies regardless of user. It asuumes the same rating for all movies and users with all the differences explained by random variation like this:

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

We calculated the average rating mu and the 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 <- data_frame(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

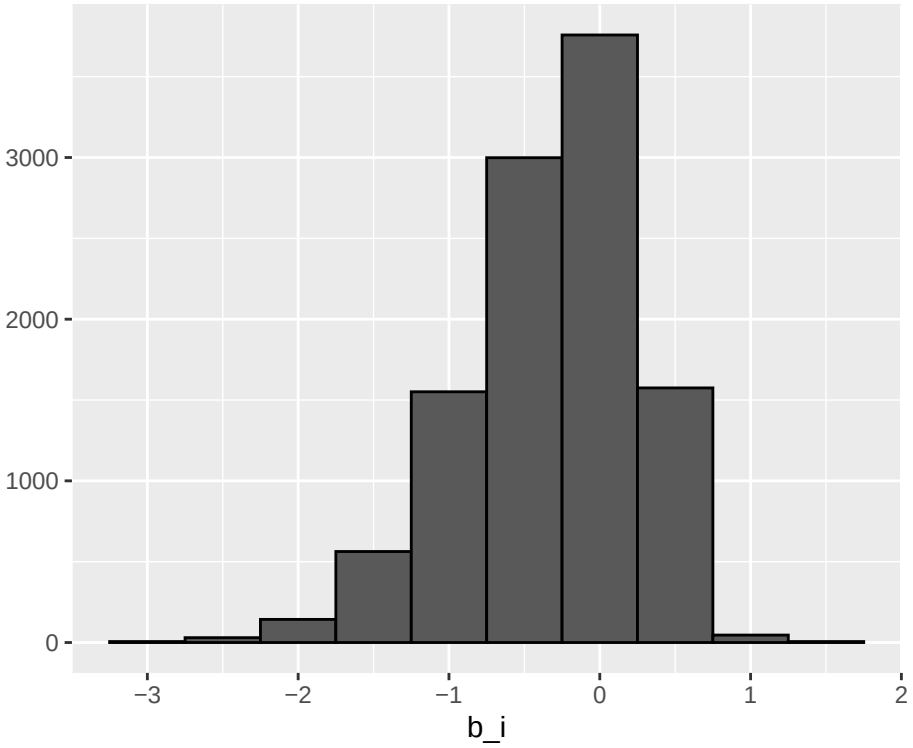
Model 2: Movie Effect Model

Consider the intrinsic character of a movie affects a movie's ratings, base on previous the exploration on the average rating for movie i, we add the term b_i into the previous model like this:

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

Plot the distribution of the bias

```
# Calculate every movie's average b_i
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, data = ., bins = 10, color = I("black"))
```



See how much our prediction improves once we used b_i for the test set:

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
```

As we going along, we will be comparing different approaches, now we start by creating a results table:

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

The results above show that by adding the movie bias, the RMSE is below 1.

Model 3: Movie + User Effects Model

Further, we also consider that the user's rating of the movie also has an effect. To be able to predict correctly we add user effect b_u into the previous model to compute average ranking for user u like this:

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

```
# Calculate every user's b_u
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)
```

```
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843

The results above show that by adding the user bias, the RMSE is lower than before.

Model 4: Regularized Movie Effect Model

Base on previous the exploration we see that some users rated every movie, but some users only rated a few movie and they may rate a movie 5 instead of 3. If these errors are large then can increse our RMSE. We will use regularization that allows us to add a penalty λ (lambda) to penalizes movies with large estimates from a small sample size. In order to optimize b_i, it necessary to use this equation:

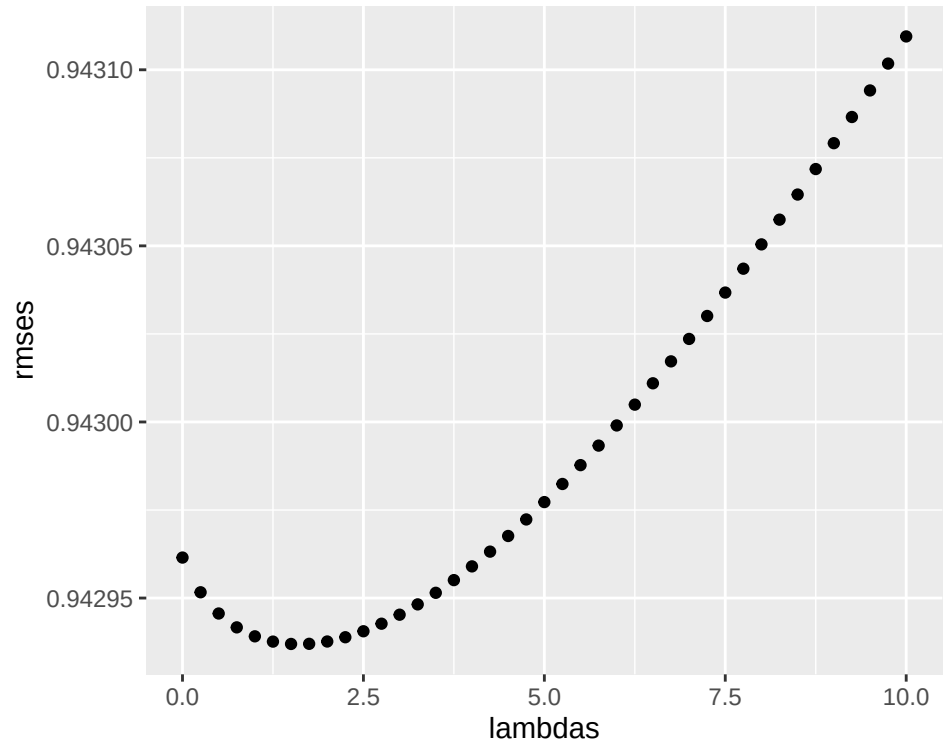
$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

To do this, a parameter λ (lambda) can be found using cross-validation and applied to our model.

Now we use cross-validation to choose a lambda:

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

mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
```

```
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```




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

```
## [1] 1.5
```

Using the optimized lambda to perform prediction and evaluate RMSE.

```
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
```

```
  pull(pred)

model_4_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Regularized Movie Effects Model",
                                     RMSE = model_4_rmse ))
rmse_results %>% knitr::kable()
```

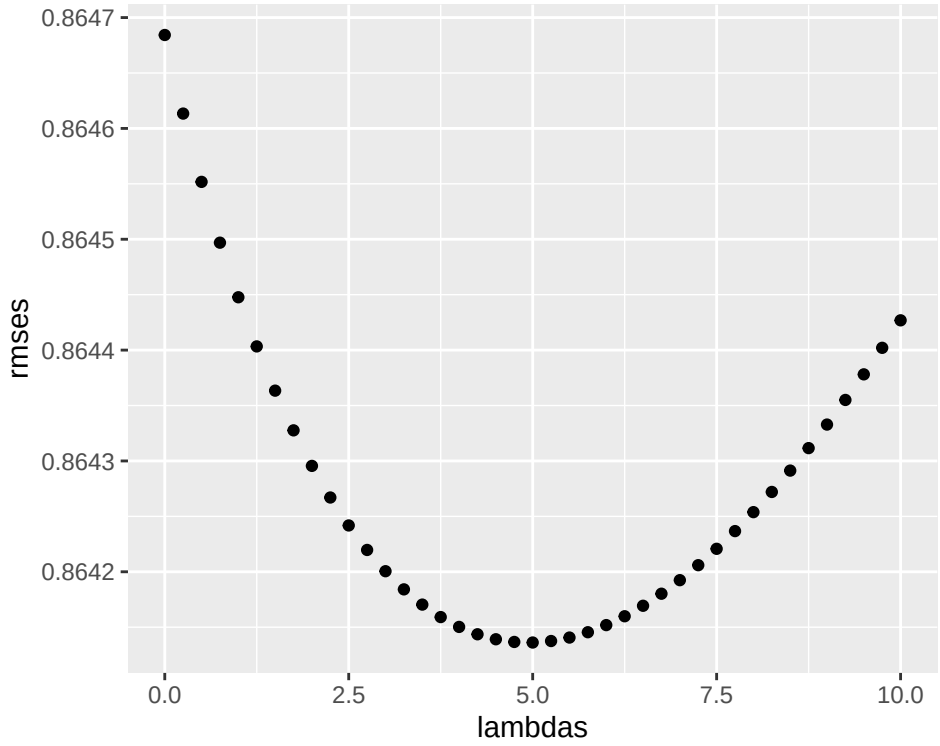
method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Regularized Movie Effects Model	0.9429370

The result above shows that using a regularized movie effect model does not improve RMSE. Next we will regularize the movie and user effects in Model 5.

Model 5: Regularized Movie + User Effects Model

In this model we also consider the effect of users, which is to optimize b_u as well,

```
# use cross-validation to choose a lambda by considering b_i and b_u effects
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))
})

qplot(lambdas, rmses)
```




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

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

Now we use the optimized lambda to perform prediction and evaluate RMSE.

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

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

```
# generate prediction on the test set
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)

# calculate the RMSE
model_5_rmse <- RMSE(predicted_ratings, test_set$rating)
```

```
# creating a results table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Regularized Movie + User Effects Model",
                                     RMSE = model_5_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362

Model 6: Matrix Factorization using recosystem Model

A popular technique to solve the recommender system problem is the matrix factorization. We will use the recosystem, Recommender System Using Parallel Matrix Factorization, to build a model.

```
# converting data into recosystem format
set.seed(123, sample.kind="Rounding")
mf_train_set <- with(train_set, data_memory(user_index = userId,
                                    item_index = movieId, rating = rating))
mf_test_set <- with(test_set, data_memory(user_index = userId,
                                    item_index = movieId, rating = rating))
```

```
# create a model object by calling Reco()
r <- Reco()

# call the $tune() method to select best tuning parameters along a set of candidate values
mf_opts <- r$tune(mf_train_set,  opts = list(dim = c(10, 20, 30),
                                             lrate = c(0.1, 0.2),
                                             costp_l1 = 0, costq_l1 = 0,
                                             nthread = 1, niter = 10))

# train the model by calling the $train() method
r$train(mf_train_set, opts = c(mf_opts$min, nthread = 1, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9816   1.1044e+07
##    1       0.8743   8.9730e+06
##    2       0.8422   8.3375e+06
##    3       0.8209   7.9594e+06
##    4       0.8050   7.6968e+06
##    5       0.7928   7.5033e+06
##    6       0.7826   7.3637e+06
##    7       0.7739   7.2459e+06
##    8       0.7665   7.1517e+06
##    9       0.7601   7.0721e+06
##   10       0.7544   7.0045e+06
##   11       0.7493   6.9460e+06
```

```
##   12       0.7448   6.8948e+06
##   13       0.7406   6.8507e+06
##   14       0.7368   6.8139e+06
##   15       0.7334   6.7792e+06
##   16       0.7302   6.7473e+06
##   17       0.7272   6.7202e+06
##   18       0.7245   6.6929e+06
##   19       0.7220   6.6710e+06
##   20       0.7197   6.6483e+06
##   21       0.7175   6.6287e+06
##   22       0.7155   6.6114e+06
##   23       0.7136   6.5972e+06
##   24       0.7118   6.5801e+06
##   25       0.7102   6.5655e+06
##   26       0.7086   6.5521e+06
##   27       0.7072   6.5411e+06
##   28       0.7058   6.5302e+06
##   29       0.7045   6.5193e+06
```

```
# use the $predict() method to compute predicted values
predicted_ratings <- r$predict(mf_test_set, out_memory())

# calculating the RMSE
model_6_rmse <- RMSE(predicted_ratings, test_set$rating)

# creating a results table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "Matrix Factorization using recosystem Model",
                                     RMSE = model_6_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362
Matrix Factorization using recosystem Model	0.7842397

4.4 The Final Evaluation

Based on the models we trained and tested, we can see that the Matrix Factorization using recosystem Model has a lower RMSE (0.78423). We now perform the final evaluation of this model, we use the edx dataset as training set and the validation dataset as test set.

```
# converting data into recosystem format
set.seed(1234, sample.kind="Rounding")
mf_edx <- with(edx, data_memory(user_index = userId, item_index = movieId,
                                rating = rating))
mf_validation <- with(validation, data_memory(user_index = userId,
```

```
                                            item_index = movieId,
                                            rating = rating))
```

```
# create a model object by calling Reco()
r <- Reco()

# call the $tune() method to select best tuning parameters along a set of candidate values
mf_opts <- r$tune(mf_edx,  opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 1, niter = 10))

# train the model by calling the $train() method
r$train(mf_edx, opts = c(mf_opts$min, nthread = 1, niter = 30))
```

```
# use the $predict() method to compute predicted values
predicted_ratings <- r$predict(mf_validation, out_memory())

# calculating the RMSE
model_62_rmse <- RMSE(predicted_ratings, validation$rating)

# creating a results table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "Final Matrix Factorization using recosystem Model",
                                     RMSE = model_62_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362
Matrix Factorization using recosystem Model	0.7842397
Final Matrix Factorization using recosystem Model	0.7848205

5. Conclusion

We have built several models and compared them. Finally we have used the validation set as a test set evaluated Matrix Factorization using recosystem Model. We recommend that the model of Matrix factorization using recosystem is the best option because it's RMSE is much lower than other models, and also by comparing these two models, the RMSE obtained using validation set and using test_set, the two RMSEs of the Matrix Factorization using recosystem Model are very close.

We recommend the Regularized Movie + User Effects Model as the next best choice. We evaluated the model using the validation set and its RMSE was less than 0.86490. The code is attached in the appendix.

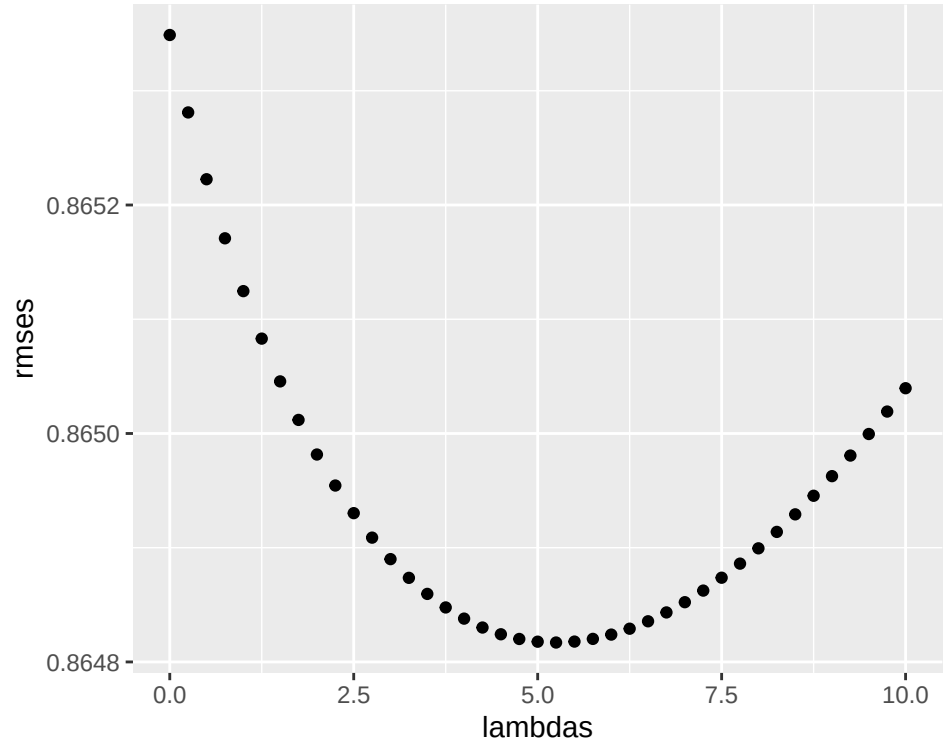
Refferences

1. Rafael A. Irizarry: Introduction to Data Science - Data Analysis and Prediction Algorithms with R https://rafalab.github.io/dsbook/
2. HarvardX PH125.8x Data Science: Machine Learning - Section 6.2 & 6.3
3. Yixuan Qiu: recosystem: Recommender System Using Parallel Matrix Factorization https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

Appendix

Use Regularized Movie + User Effects Model for final algorithm, predict movie ratings on the validation set

```
# use cross-validation to choose a lambda by considering b_i and b_u effects
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)
```




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

Using the optimized lambda to perform prediction and evaluate RMSE.

```
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

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

```
# generate prediction on 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) %>%
  pull(pred)

# calculate the RMSE
model_52_rmse <- RMSE(predicted_ratings, validation$rating)
model_52_rmse
```

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