MovieLens project

XuMan

2020/5/3

1 Executive summary

Recommendation systems aim to make specific recommendations to users in order to promote sales or exposure. Recommendation systems can greatly facilitate the browsing activities of users and are widely used by e-commerce websites, social platforms and content providers. They are more important these days since online platforms are providing larger varieties of products or other items, such as movies, user-generated pictures and messages.

For this project, we will focus on a movie recommendation system. Online streaming media platforms, such as Netflix, provide many movies to many users. They encourage users to provide ratings on the movies they have watched. The general idea behind the movie recommendation system is to predict the users' ratings on the movies and movies with higher predicted ratings are likely to be favored more than movies with low predicted ratings.

1.1 Goal of the project

The aim of the project is to create a machine learning algorithm that predicts the rating for a movie i by user u:

Performance of the algorithm is evaluated by residual mean squared error (RMSE) on the test set, which is defined as:

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

with N being the total number of user/movie combinations in the test set.

RMSE can be interpreted as the typical error the algorithm make when predicting a movie rating. For example, if RMSE is 1, it means the prediction of then algorithm is typically 1 star away from the real value.

1.2 Dataset description

Before starting to work with the data set, we first load the necessary packages.

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(irlba)
```

Dataset used in this report is a subset of the MovieLens dataset generated by the GroupLens research lab. The dataset is further divided into an edx set, which is used to develop the algorithm and a validation set, which is for the final test of the algorithm.

```
ratings <- fread(text = gsub("::", "\t", readLines(file("/Users/xuman/ds_projects/Capstone-MovieLens/ml-10M100
K/ratings.dat"))), col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(file("/Users/xuman/ds_projects/Capstone-MovieLens/ml-10M100K/movies.dat")),
"\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[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)
```

Next we are going to take a closer look at the edx dataset. The validation dataset is of similar structure with the edx dataset but of smaller size.

The edx dataset is in tidy form with 9000055 rows and the first 6 rows are as follows:

```
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
```

We can see that the table contains six variables:“userId”, “movieId”, “rating”, “timestamp”, “title”, and “genres”. Each row represents a rating given by one user to one movie in the table. The variables “userId” and “movieId” provide unique ID numbers for each user and each movie. The variable “rating” shows rating of the movie marked by the user. The variable “timestamp” represents the time and data in which the rating was provided with units being seconds since January 1, 1970. The variable “title” provides the movie title and also the year in which the movie was released. The variable “genres” includes every genre that applies to the movie and some movies fall under several genres.

1.3 Key methods

First we make a descriptive analysis on the data to explore the features of the data and decide on the variables that have significant influence on ratings. Tow models are built using logistic regression and the final model is evaluated by RMSE.

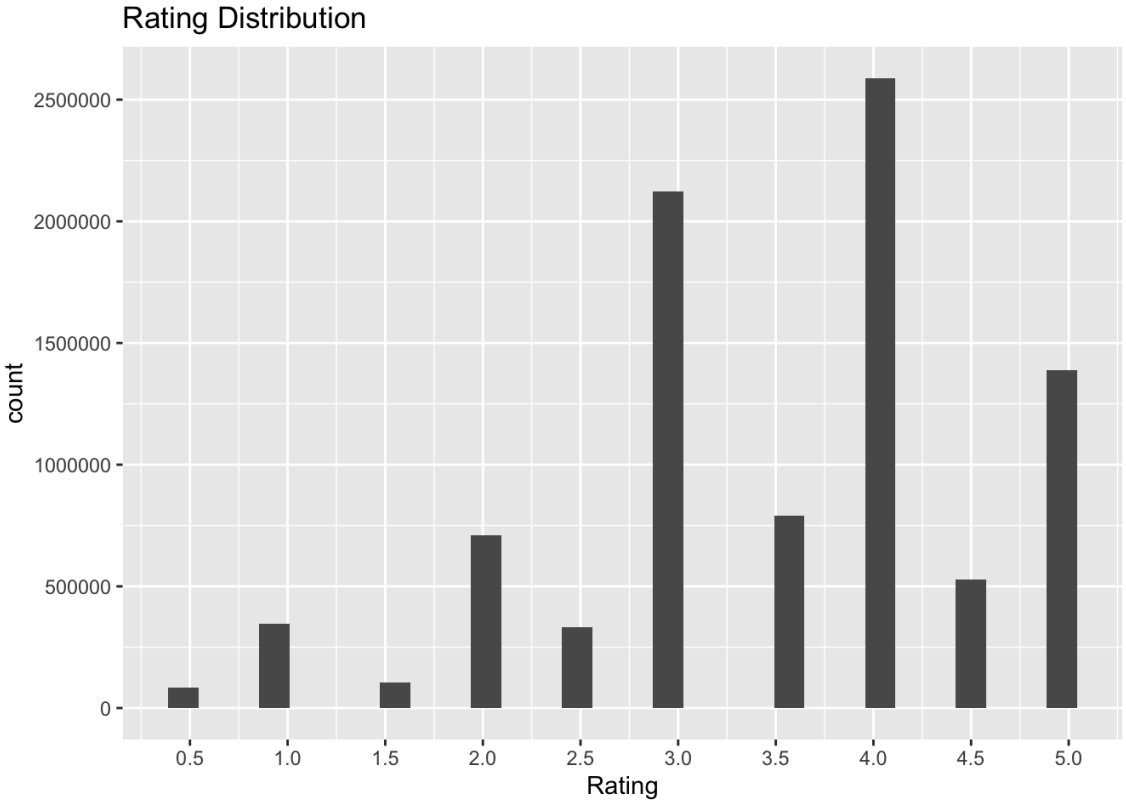
2 Exploratory data analysis

There are 10677 unique movies and 69878 unique users in the dataset. Not every user rated every movie. The average number of movies rated by an user is 129. The average number of ratings a movies received is 843

2.1 Rating distribution

If we take a look at the distribution of ratings, we can find that users tend to give higher ratings and the most frequent rating is 4, followed by 3 and 5. In general, wholestar ratings are more common than half star ratings. The average rating given by users is 3.5124652.

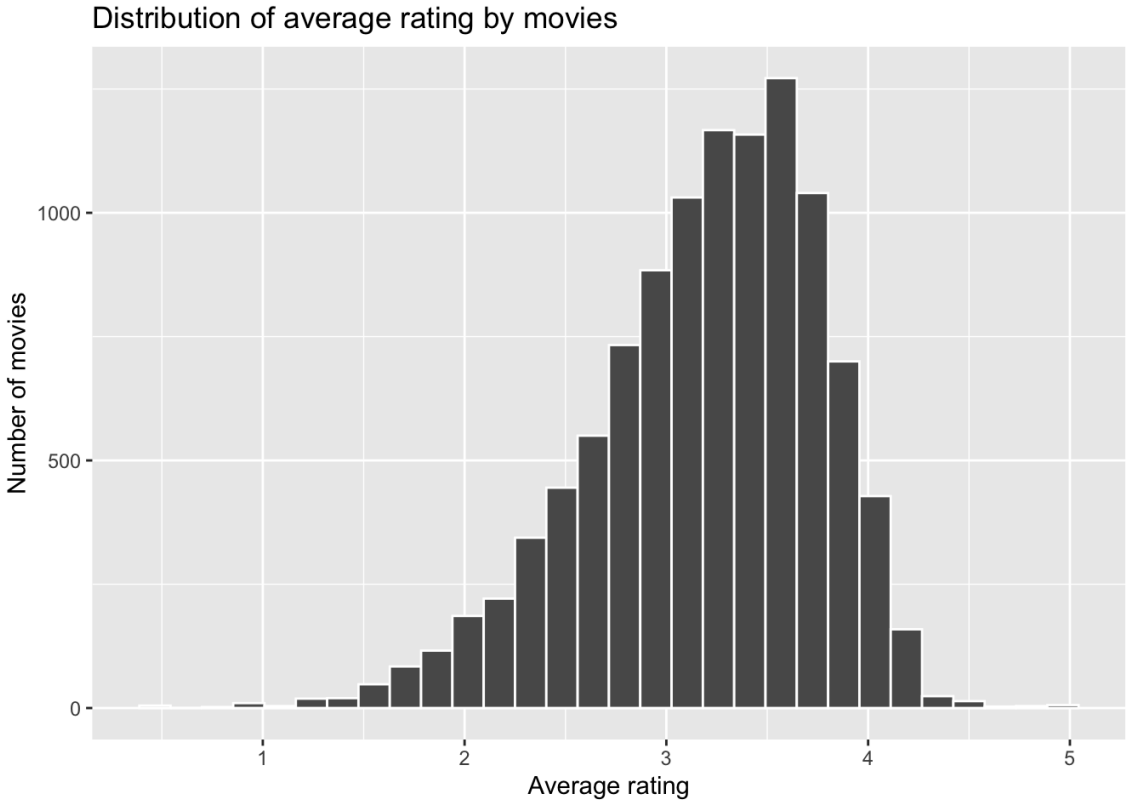
```
edx %>% ggplot(aes(rating)) +
  geom_histogram() +
  ggtitle("Rating Distribution") +
  xlab("Rating")+
  scale_x_continuous(breaks = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000)))
```



2.2 Distribution of average rating by movies

The graphic shows that movies are rated very differently. Therefore we can take into account the effect of movies in our model.

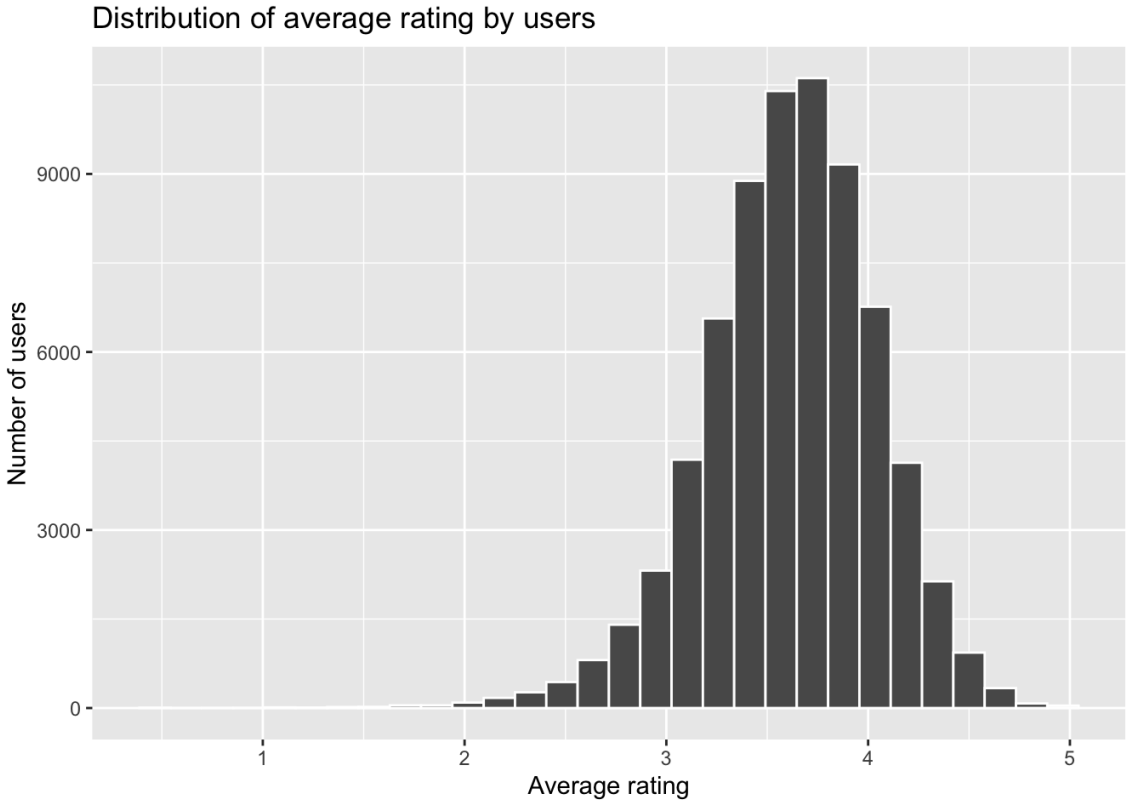
```
edx %>% group_by(movieId) %>%
  summarize(movie_avgs = mean(rating)) %>%
  ggplot(aes(movie_avgs)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of average rating by movies") +
  xlab("Average rating") +
  ylab("Number of movies")
```



2.3 Distribution of average rating by users

Users also give different average ratings so user effect needs to be included into our model as well.

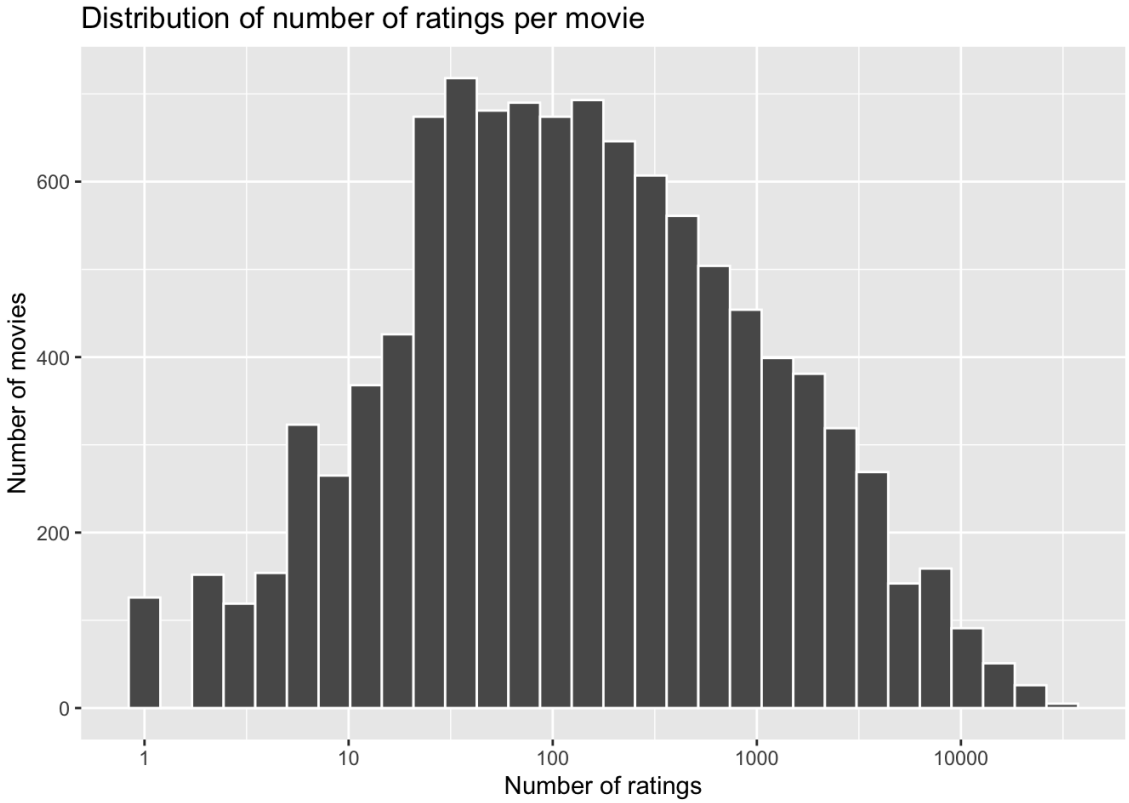
```
edx %>% group_by(userId) %>%
  summarize(user_avgs = mean(rating)) %>%
  ggplot(aes(user_avgs)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of average rating by users") +
  xlab("Average rating") +
  ylab("Number of users")
```



2.4 Distribution of number of ratings per movie

We can see that the number of ratings per movie is quite different. In fact, there are 126 movies that are only rated once. So when we try to quantify movie effect, movies with few ratings have more uncertainty because of the lack of a large enough sample size and larger estimates of movies effect are more likely, which can lead to the increase in RMSE. Therefore, we need to perform regularization and penalize large estimates that are formed using small sample sizes.

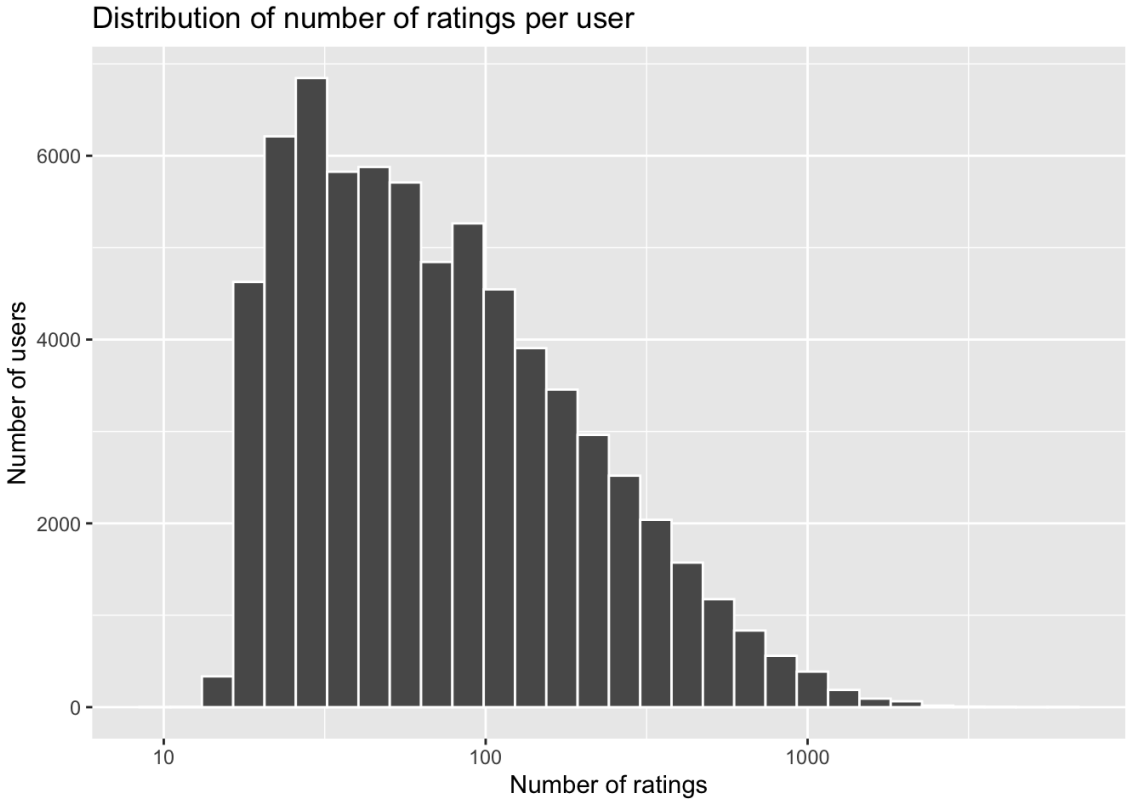
```
edx %>% group_by(movieId) %>%
  summarize(number_of_rating_per_movie = n()) %>%
  ggplot(aes(number_of_rating_per_movie)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of number of ratings per movie") +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  scale_x_log10()
```



2.5 Distribution of number of ratings per user

The number of ratings generated by users also varies greatly from person to person. There are 3470 users who are less active and gives less than 20 ratings, while there are 58 users who are much more active and gives more than 20 ratings. Penalty for small number of ratings per user will also be introduced in our model.

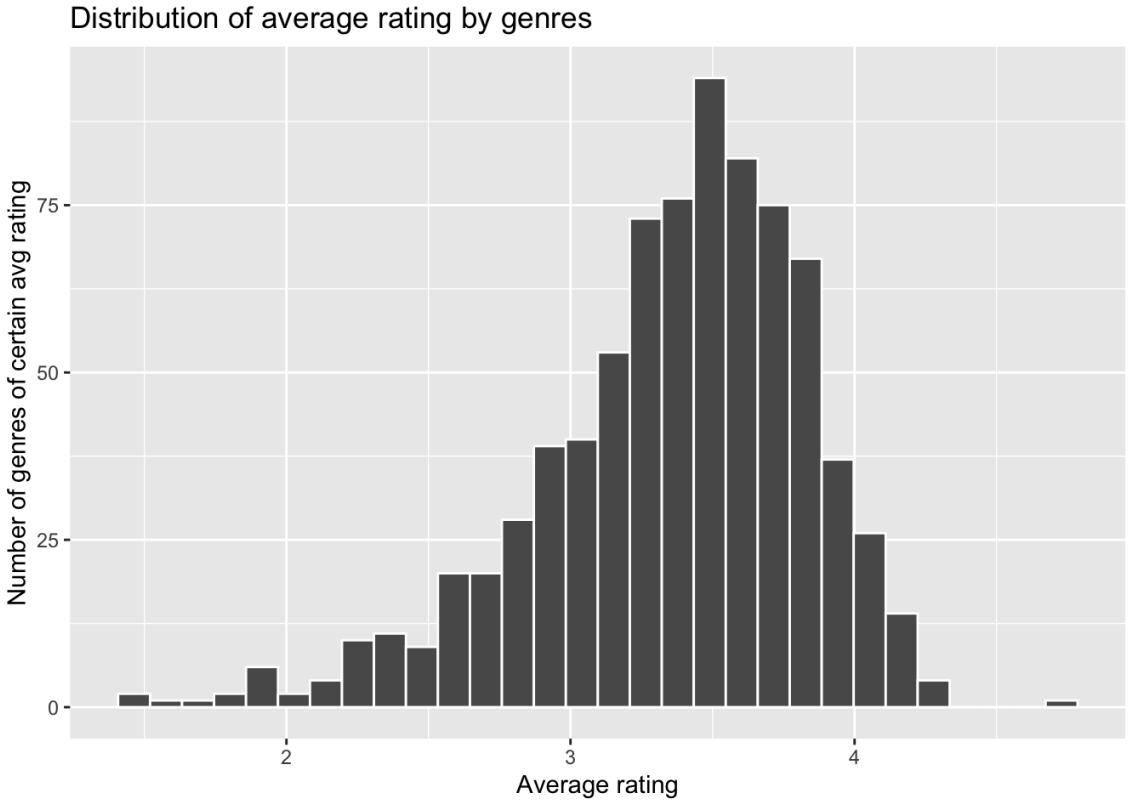
```
edx %>% group_by(userId) %>%
  summarize(number_of_rating_per_user = n()) %>%
  ggplot(aes(number_of_rating_per_user)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of number of ratings per user") +
  xlab("Number of ratings") +
  ylab("Number of users") +
  scale_x_log10()
```



2.6 Distribution of average rating by genres

We can see that genres have a significant influence on ratings, so we need to take genres into account in our model.

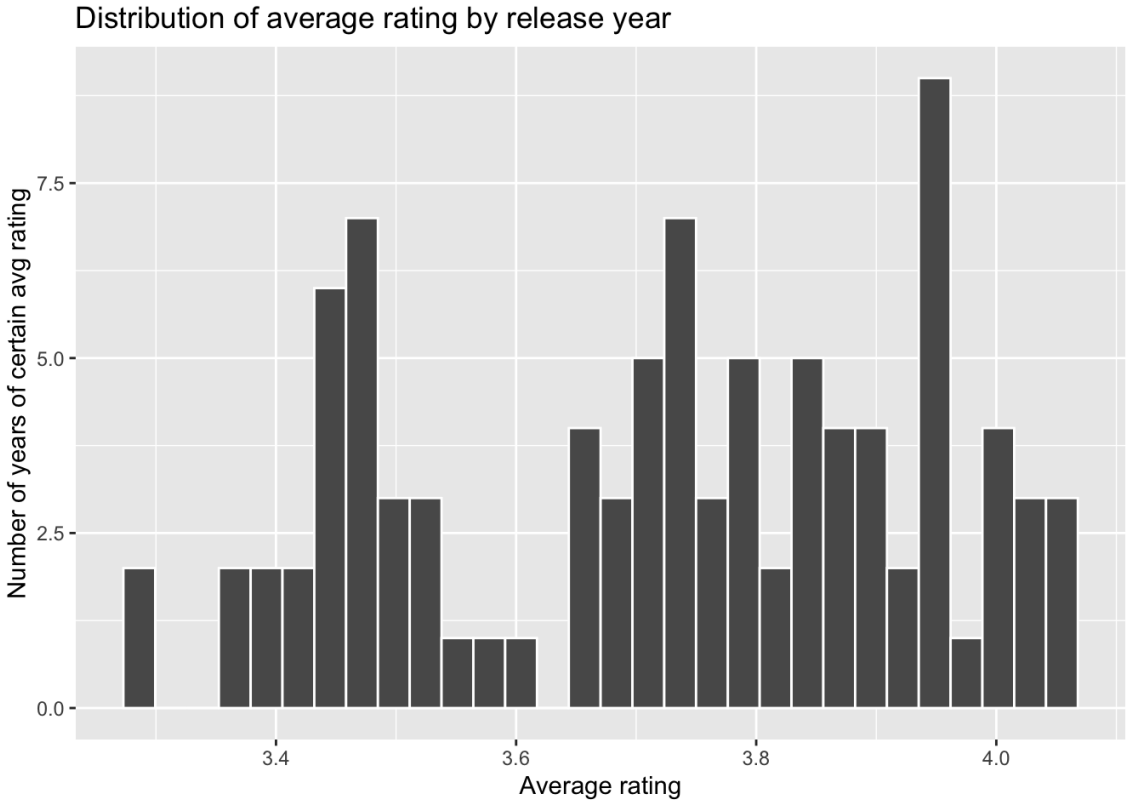
```
edx %>% group_by(genres) %>%
  summarize(genres_avgs = mean(rating)) %>%
  ggplot(aes(genres_avgs)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of average rating by genres") +
  xlab("Average rating") +
  ylab("Number of genres of certain avg rating")
```



2.7 Distribution of average rating by release year

Release year also influences the ratings.

```
edx %>% mutate(year = str_sub(title, -5, -2)) %>%
  group_by(year) %>%
  summarize(ry_avgs = mean(rating)) %>%
  ggplot(aes(ry_avgs)) +
  geom_histogram(color = "white") +
  ggtitle("Distribution of average rating by release year") +
  xlab("Average rating") +
  ylab("Number of years of certain avg rating")
```



3 Model

3.1 Create train and test set

We further divide the edx set into train set (80%) and test set (20%), so that the test set can be used to tune parameters and compare the performance of different models.

```
set.seed(2029)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3.2 Regularized movie and user effect model

Our model that takes into account movie and user effect is:

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

To constrain the total variability of the effect sizes, we introduce regularization. Instead of defining $\hat{b}_u$ as the average of $Y_{u,i} - \hat{\mu}$, we use:

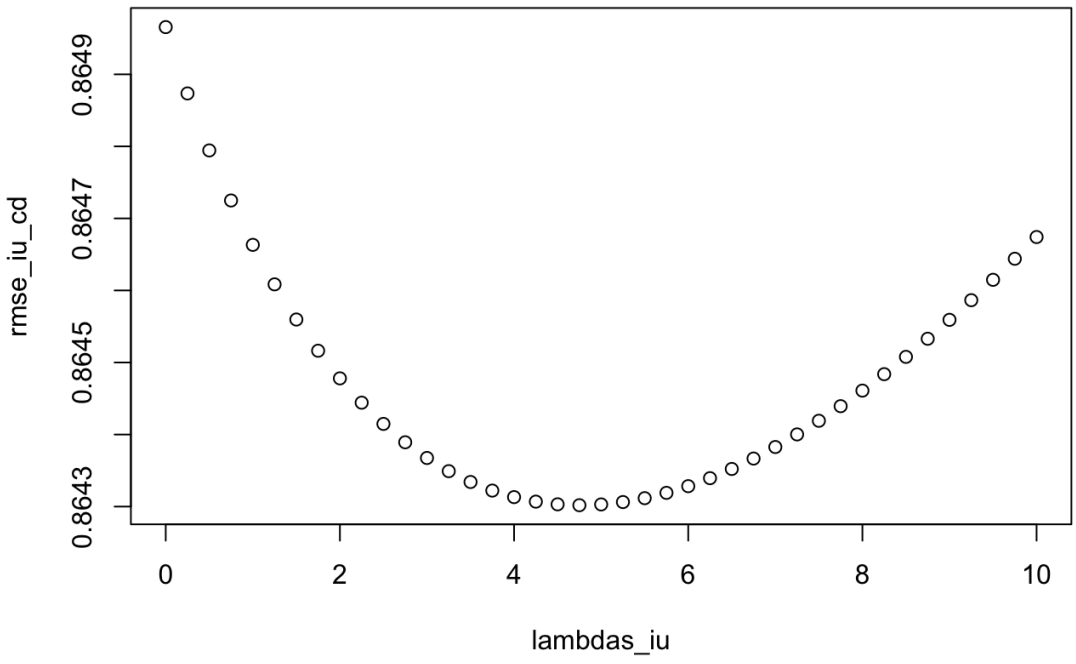
$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

Similarly, we define:

$$\hat{b}_{u,i}(\lambda) = \frac{1}{\lambda + n_u} \sum_{u=1}^{n_u} (Y_{u,i} - \hat{\mu} - \hat{b}_i)$$

By cross validation, we tested λ in the range of 1 and 10 with 0.25 intervals and pick the λ that gives the lowest RMSE for this model.

```
lambdas_iu <- seq(0, 10, 0.25)
rmse_iu_cd <- sapply(lambdas_iu, function(lambda){
  mu <- mean(train_set$rating)

  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 - mu - b_i)/(n()+lambda))

  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)

  RMSE(predicted_ratings, test_set$rating)
})

lambda_iu <- lambdas_iu[which.min(rmse_iu_cd)]

plot(lambdas_iu, rmse_iu_cd)
```



The RMSE is 0.8643019, with the optimal λ being 4.75

Let's generate prediction on the test set based on the user and movie effect model.

```
mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_iu))
b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda_iu))
predicted_ratings_iu_1 <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

In the prediction result, the minimum value is -0.7044019 and the maximum value is 5.9170918, which are beyond the limit of the rating system, so we need to confine the lower and upper boundary of the prediction result by predicting 0.5 for ratings lower than 0.5 and 5 for ratings higher than 5.

```
predicted_ratings_iu <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  mutate(pred_l = ifelse(pred>=0.5 & pred<=5, pred, ifelse(pred<0.5, 0.5, 5)))%>%
  pull(pred_l)
```

After the adjustment, RMSE on the test set goes down to 0.8641825.

3.3 Model with movie, user effect, release year and genres effect

We further take into account release year and genres effect. The model is upgraded to:

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

The release year and genres effect is calculated.

```
train_set_iu <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by = 'userId')

b_tr <- train_set_iu %>%
  mutate(year = str_sub(title, -5, -2)) %>%
  group_by(year) %>%
  summarize(b_tr = mean(rating - mu - b_i - b_u))

b_g <- train_set_iu %>%
  mutate(year = str_sub(title, -5, -2)) %>%
  left_join(b_tr, by = 'year') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_tr))

predicted_ratings_iugy <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(year = str_sub(title, -5, -2)) %>%
  left_join(b_tr, by = 'year') %>%
  left_join(b_g, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_tr + b_g) %>%
  mutate(pred_l = ifelse(pred>=0.5 & pred<=5, pred, ifelse(pred<0.5, 0.5, 5)))%>%
  pull(pred_l)

rmse <- RMSE(predicted_ratings_iugy, test_set$rating)
```

4 Results

Based on the analysis above, we are able to generate a regularized model that takes into account the effect of movie, user, genre and release year. Now let's run the model on the validation set and get the RMSE value.

```
predicted_ratings_on_validationset_uigy <- validation %>%
  mutate(year = str_sub(title, -5, -2)) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_tr, by = 'year') %>%
  left_join(b_g, by = 'genres') %>%
  mutate(b_i = ifelse(is.na(b_i), 0, b_i)) %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u)) %>%
  mutate(b_tr = ifelse(is.na(b_tr), 0, b_tr))%>%
  mutate(b_g = ifelse(is.na(b_g), 0, b_g)) %>%
  mutate(pred = mu + b_i + b_u + b_tr + b_g) %>%
  mutate(pred_l = ifelse(pred>=0.5 & pred<=5, pred, ifelse(pred<0.5, 0.5, 5))) %>%
  pull(pred_l)

rmse_v <- RMSE(predicted_ratings_on_validationset_uigy, validation$rating)
```

The RMSE on validation set is 0.8637075.

5 Conclusion

In the report, we build a movie recommendation system based on prediction of each user's rating on different movies. Two models using logistic regression are built and the final RMSE on the validation set is 0.8637075.

Surely the report can be further improved but there is the pressure of deadline. Usage of advanced techniques is greatly limited by the performance of my personal computer. In fact I tried packages such as recommenderlab, irlba and recosystem, but all of them would go beyond the memory limitation. I am current trying to solve the problem by the help of cloud service providers.