HarvardX Data Science Professional Certificate: Capstone Project

Delong Meng

12/4/2019

Introduction

Recommendation system represents a classical application of machine learning technology. For example, for a movie recommendation system, the goal is to predict how a given user will rate a specific movie based on how the user rate other movies and how the movie is rated by other users. In this project, I will combine several machine learning strategies to construct a movie recommendation system based on the "MovieLens" dataset. The full MovieLens dataset, which can be found here: https://grouplens.org/datasets/movielens/latest/, is rather large. To make the computation easier, in this project I use the "10M" version of MovieLens dataset instead (https://grouplens.org/datasets/movielens/10m/). This 10M MovieLens dataset has around 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

Methods

I first downloaded the dataset and organize it into a format that is ready to explore. There are several features of this dataset, including movie ID, user ID, movie title (including the year the movie was released), genres, rating, and the timestamp of the rating. I extracted the release year from the title, and the rating year, and then calculated the age of the movie at the time of the rating. For the genres information, because it is a combination of different classifications, I split one single combination into individual records for further exploration.

To train the machine learning algorithm, I first randomly divided the 10M MovieLens dataset into a training set called "edx" (90%), and a test set called "validation" (10%). The "validation" set does not contain any user or movie that are absent in the "edx" set.

To evaluate the performance of the algorithm, I use Root Mean Square Error (RMSE) as an indicator. Only the "edx" set is used to develop the algorithm, and the "validation" set is used to get the RMSE value. I will first build a baseline prediction model including age effect, movie effect, user effect, and regularization of movie effect and user effect. I will evaluate those model and choose the best one to continue with using matrix factorization technique. More details can be found in the Results section.

1. Installing essential packages

I first set up the working envrionment by installing essential packages:

```
install.packages("tidyverse")
install.packages("caret")
install.packages("data.table")
install.packages("lubridate")
install.packages("recosystem")
install.packages("kableExtra")
```

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

2. Data downloading and preparation

Information of the MovieLens 10M dataset can be found: https://grouplens.org/datasets/movielens/10m/

The dataset can be downloaded here: http://files.grouplens.org/datasets/movielens/ml-10m.zip

I use the following code to download the dataset and combine the information of the ratings and the movies into *movielens*:

```r
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Now we split the MovieLens dataset into Training (*edx*) and Validation (*validation*) sets. The Validation set will be 10% of MovieLens data.

To successfully perform validation, we need to make sure all userIds and movieIds in the *validation* set are also in the *edx* set.

```r
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,]
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

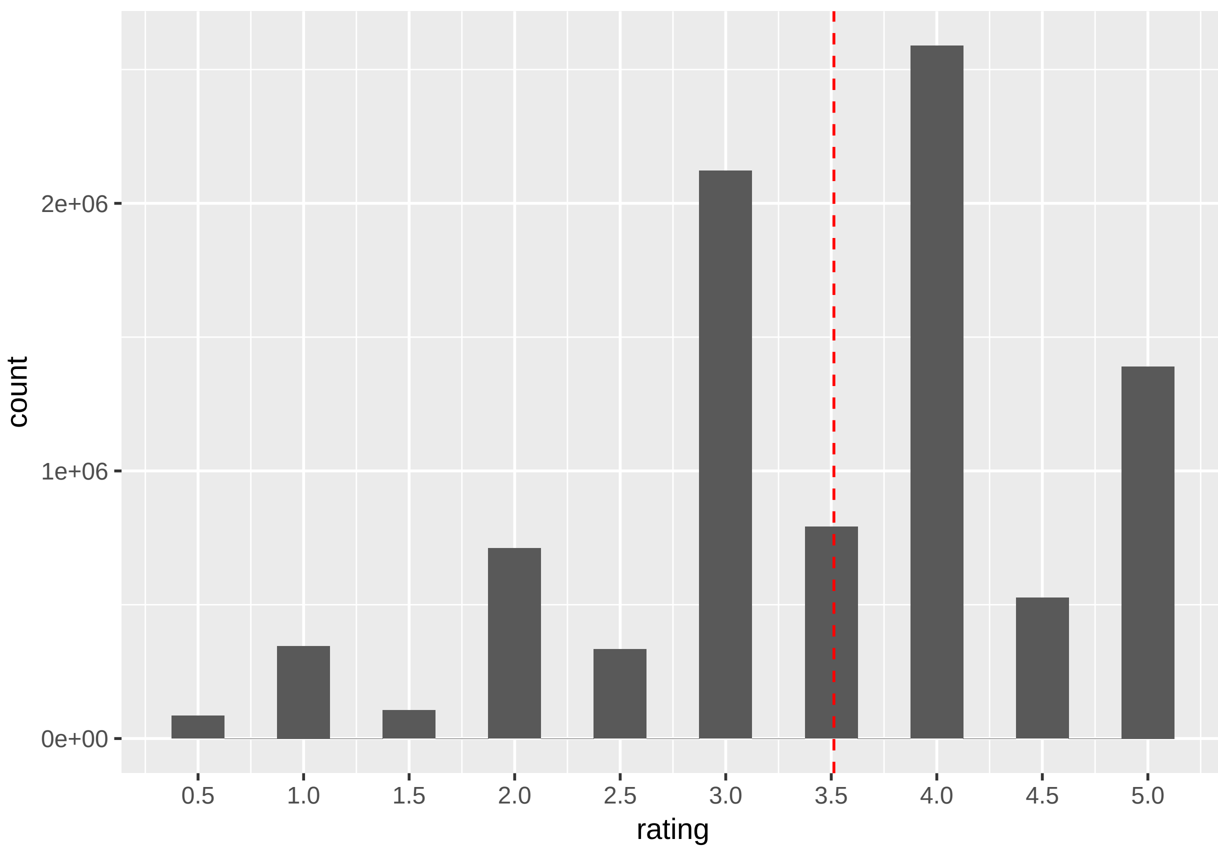
3. Data exploration

3.1 Overall profile of the dataset

Let's first have a general overview of the dataset:

```r
head(edx)
```

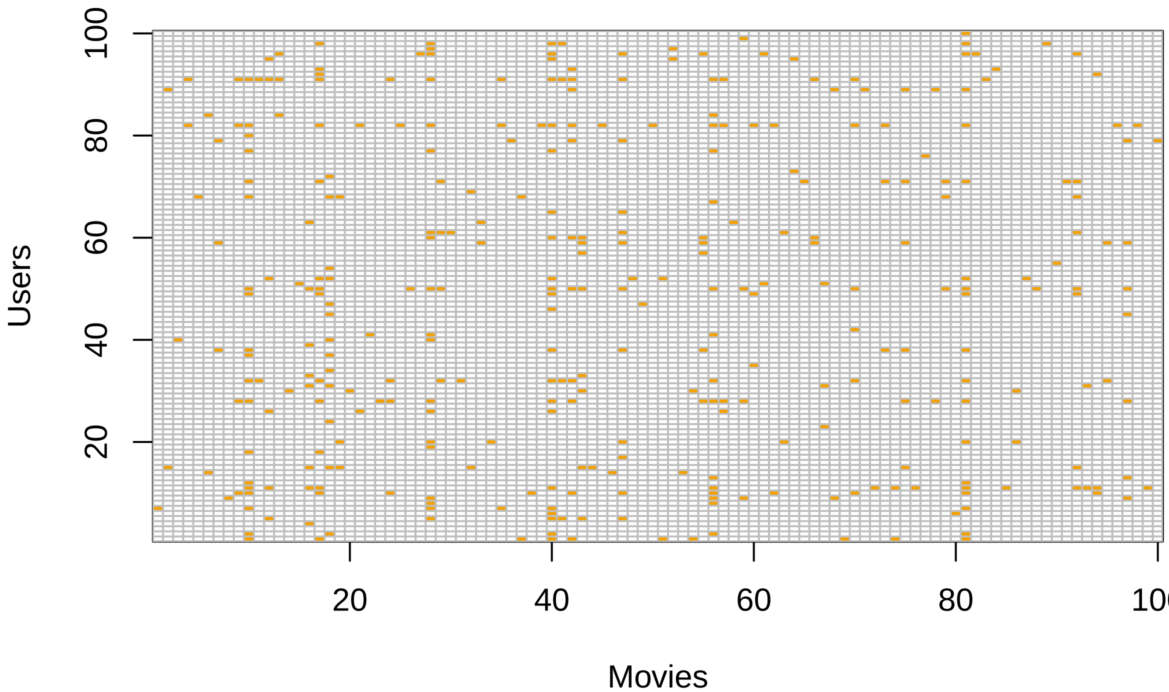
```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4       Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6       Children|Comedy|Fantasy
```



We can see the overall distribution of all of the ratings. It is screwed to the right. All half stars are less frenquient than full stars. A red dased line of the overall average rating is also plotted here as a reference.

```r
dim(edx) # 9000055       6
n_distinct(edx$movieId) # 10677
n_distinct(edx$title) # 10676: there might be movies of different IDs with the same title
n_distinct(edx$userId) # 69878
n_distinct(edx$movieId)*n_distinct(edx$userId) # 746087406
n_distinct(edx$movieId)*n_distinct(edx$userId)/dim(edx)[1] # 83
```

As shown above, this edx dataset has 10677 distinct movies and 69878 distinct users. If every user rated on every movie, we would have 10677 x 69878 = 746087406 ratings. However, we only have 9000055 ratings, which is only 1/83 of the number of all possible ratings. We can visualize 100 random samples of users and 100 random samples of movies to see how sparse this dataset is, which gives us an idea why this recommendation system is such a chanllenging task:



3.2 Extracting age of movies at rating

Every movie was released in a certain year, which is provided in the *title* of the movie. Every user rated a movie in a certain year, which is included in the *timestamp* information. I define the difference between these two years, i.e., how old the movie was when it was watched/rated by a user, as the **age of movies at rating**. From the original dataset, I first exacted the rating year (*year_rated*) from *timestamp*, and then exacted the release year (*year_released*) of the movie from the *title*. *age_at_rating* was later calculated.

```
# convert timestamp to year
edx_1 <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
# extract the release year of the movie
# edx_1 has year_rated, year_released, age_at_rating, and titles without year information
edx_1 <- edx_1 %>% mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp)
edx_1 <- edx_1 %>% mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))
head(edx_1)
```

```
##   userId movieId rating                  title year_released
## 1      1     122      5              Boomerang          1992
## 2      1     185      5               Net, The          1995
## 3      1     292      5               Outbreak          1995
## 4      1     316      5               Stargate          1994
## 5      1     329      5 Star Trek: Generations          1994
## 6      1     355      5       Flintstones, The          1994
##                          genres year_rated age_at_rating
## 1                Comedy|Romance       1996             4
## 2         Action|Crime|Thriller       1996             1
## 3  Action|Drama|Sci-Fi|Thriller       1996             1
## 4       Action|Adventure|Sci-Fi       1996             2
## 5 Action|Adventure|Drama|Sci-Fi       1996             2
## 6       Children|Comedy|Fantasy       1996             2
```

3.3 Extracting the genres information

The genres information was provided in the original dataset as a combination of different classifications. For example (see above output), the movie "Boomerang" (movieId 122) was assigned "Comedy|Romance", and "Flintstones, The" (movieId 355) is "Children|Comedy|Fantasy". Both are combinations of different ones, while they actually share one genre (Comedy). It'll make more sense if we first split these combinations into single ones:

```
# edx_2: the mixture of genres is split into different rows
edx_2 <- edx_1 %>% separate_rows(genres,sep = "\\|") %>% mutate(value=1)
n_distinct(edx_2$genres)  # 20: there are 20 differnt types of genres
genres_rating <- edx_2 %>% group_by(genres) %>% summarize(n=n())
genres_rating
```

```
## [1] 20
```

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

```
## 20 Western             189394
```

We get the information of 20 different types of genres, and numbers of movie ratings in each type. Note that the first type is "(no genres listed)", which is not really a genre, but just reflects the fact that for 7 ratings (of one singel movie), genres info was not provided.

We can figure out the only one movie with no genres infomation is movieId 8606 titled "Pull My Daisy", released in 1958:

```
## # A tibble: 1 x 5
## # Groups:   movieId, title, year_released [1]
##   movieId title         year_released genres                 n
##     <int> <chr>         <chr>         <fct>              <int>
## 1    8606 Pull My Daisy 1958          (no genres listed)     7
```

Spliting the genres information into multiple row can facilitate the exploration of genres. However, one thing to keep in mind is, if we consider one row as one record (here for our movie recommendation system every rating for a certain movie rated by a certain user should be one record), the above transformation of the dataset actually duplicated each record into multiple ones, depending on the combination of the genres for each movie.

To avoid this problem and make more sense if we want to utilize the genres information in building the prediction model, genres of each movie (also each record) should be split into multiple columns to indicate different combinations of the 19 basic genres. We can achieve this goal by spreading genres to the "wide" format:

```r
# edx_3 is the final version for exploration of the effects of movie year, age, rating year, and genres
edx_3 <- edx_2 %>% spread(genres, value, fill=0) %>% select(-"(no genres listed)")
dim(edx_3) #  9000055      26
```

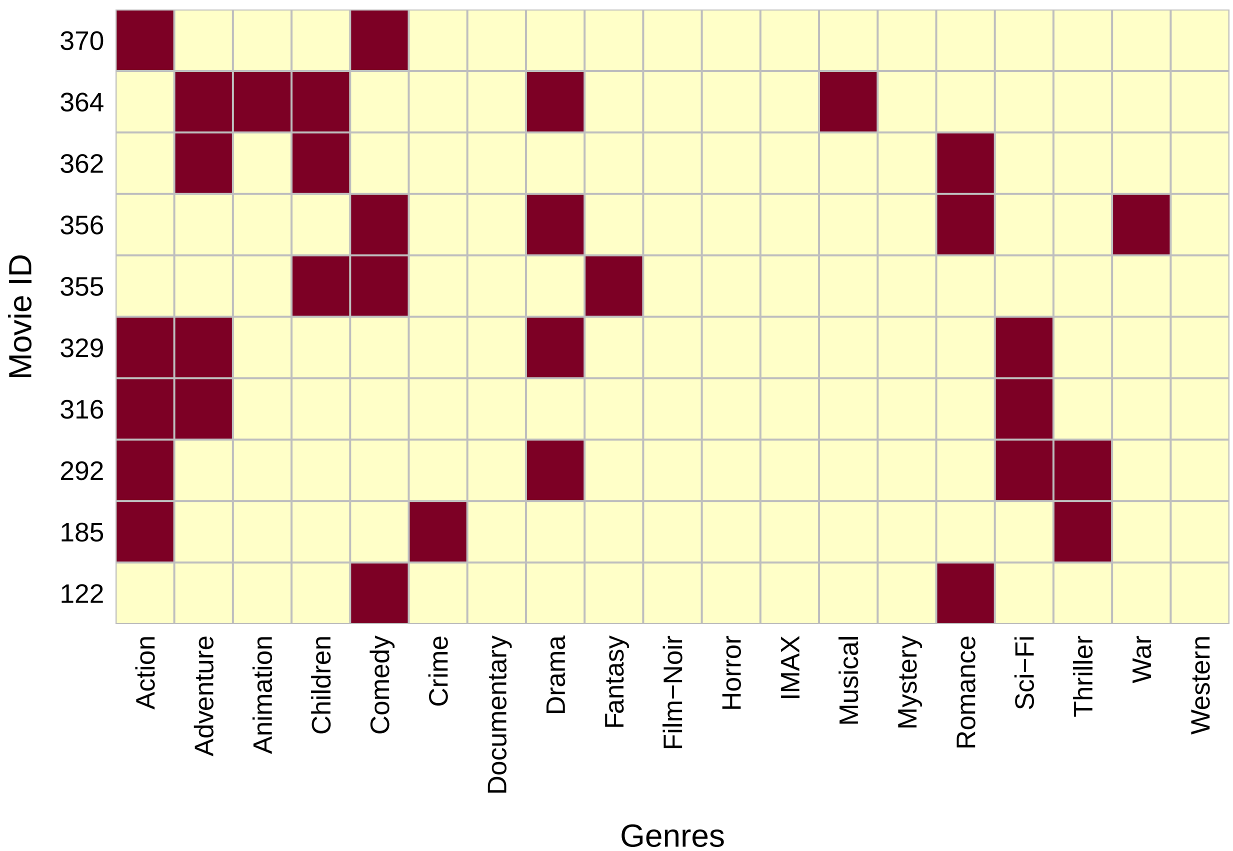
```
## [1] 9000055      26
```

```r
head(edx_3)
```

```
##   userId movieId rating                  title year_released year_rated
## 1      1     122      5              Boomerang          1992       1996
## 2      1     185      5               Net, The          1995       1996
## 3      1     292      5               Outbreak          1995       1996
## 4      1     316      5               Stargate          1994       1996
## 5      1     329      5 Star Trek: Generations          1994       1996
## 6      1     355      5       Flintstones, The          1994       1996
##   age_at_rating Action Adventure Animation Children Comedy Crime
## 1             4      0         0         0        0      1     0
## 2             1      1         0         0        0      0     1
## 3             1      1         0         0        0      0     0
## 4             2      1         1         0        0      0     0
## 5             2      1         1         0        0      0     0
## 6             2      0         0         0        1      1     0
##   Documentary Drama Fantasy Film-Noir Horror IMAX Musical Mystery Romance
## 1           0     0       0         0      0    0       0       0       1
## 2           0     0       0         0      0    0       0       0       0
## 3           0     1       0         0      0    0       0       0       0
## 4           0     0       0         0      0    0       0       0       0
```

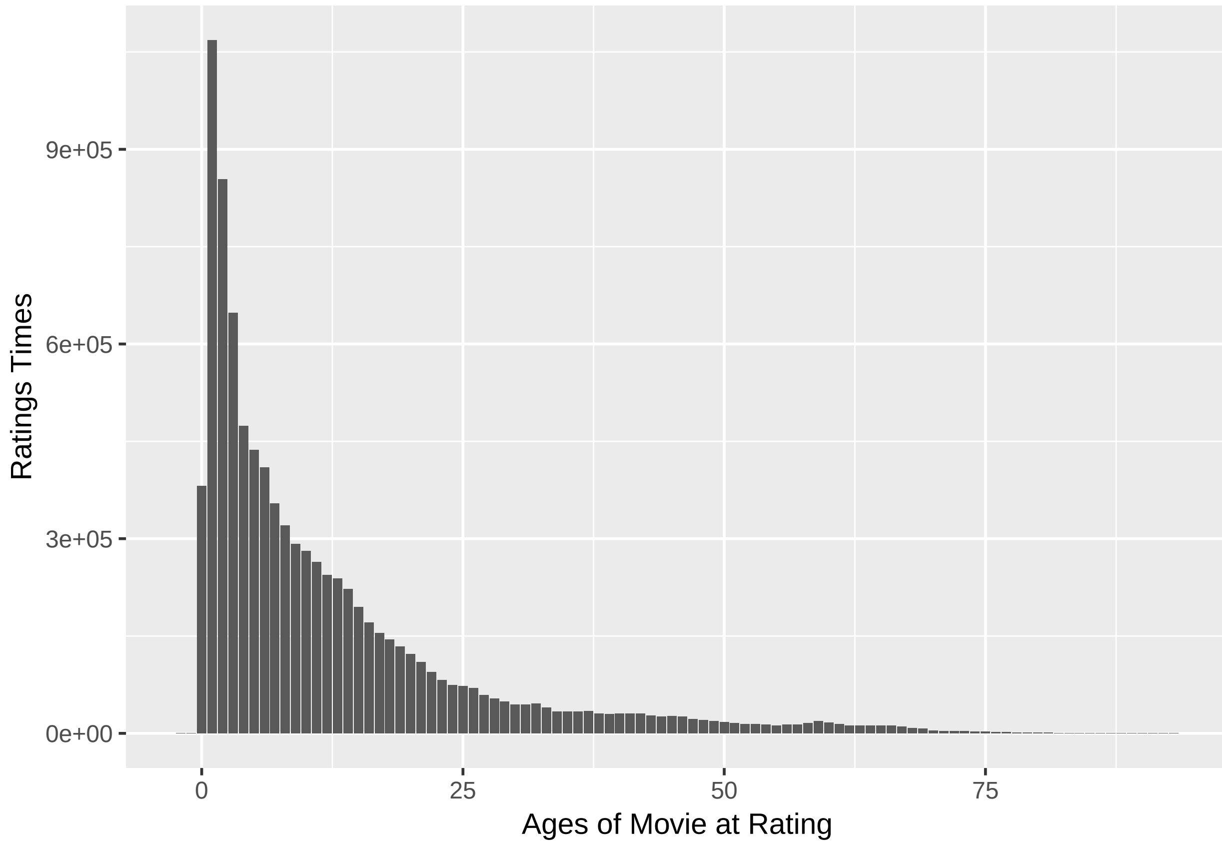
```
## 5          0      1        0         0       0    0       0       0       0
## 6          0      0        1         0       0    0       0       0       0
##   Sci-Fi Thriller War Western
## 1      0        0   0       0
## 2      0        1   0       0
## 3      1        1   0       0
## 4      1        0   0       0
## 5      1        0   0       0
## 6      0        0   0       0
```

By doing this, we can actually visualize the genres of each movie. For example, each row represents a movie, and each column respresents a genre. Each movie can have a unique combination of different genres. Here we look at the first 10 movies:

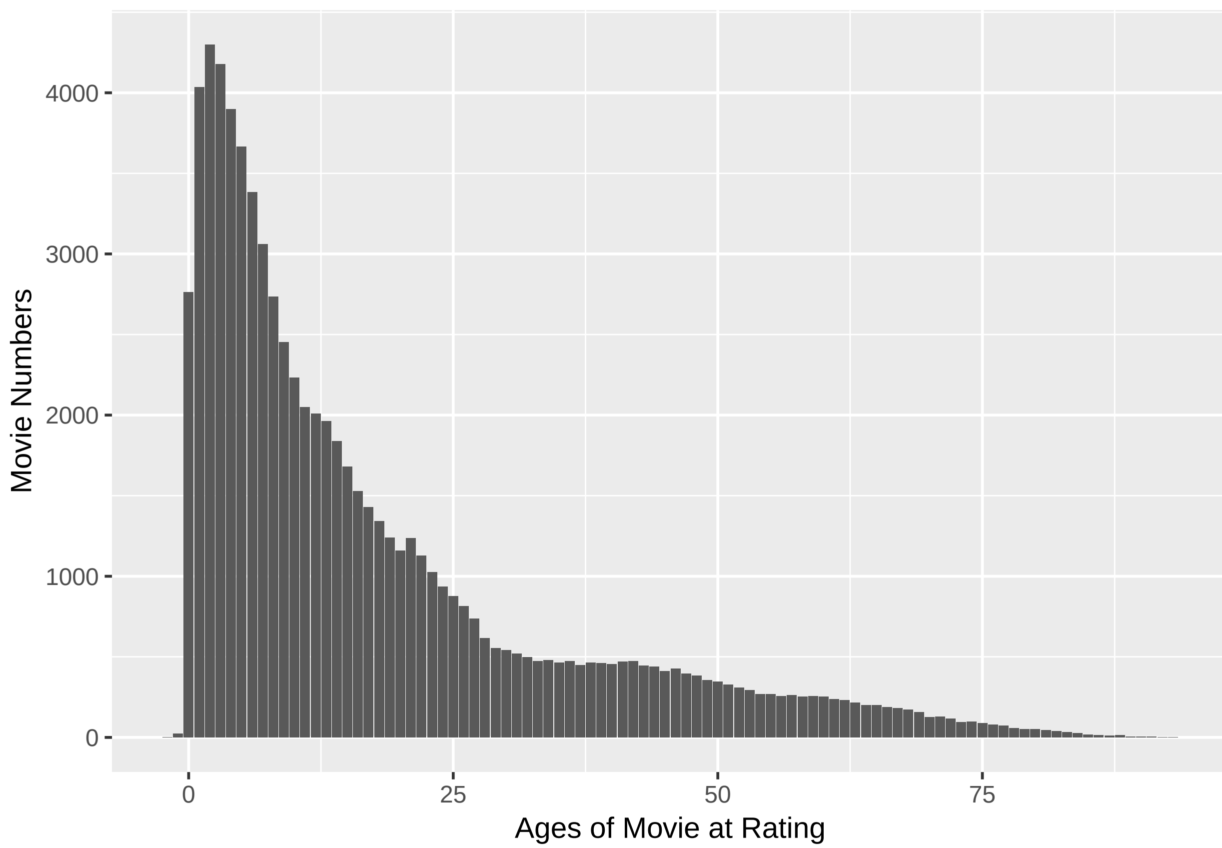


3.4 Exploration of age effect

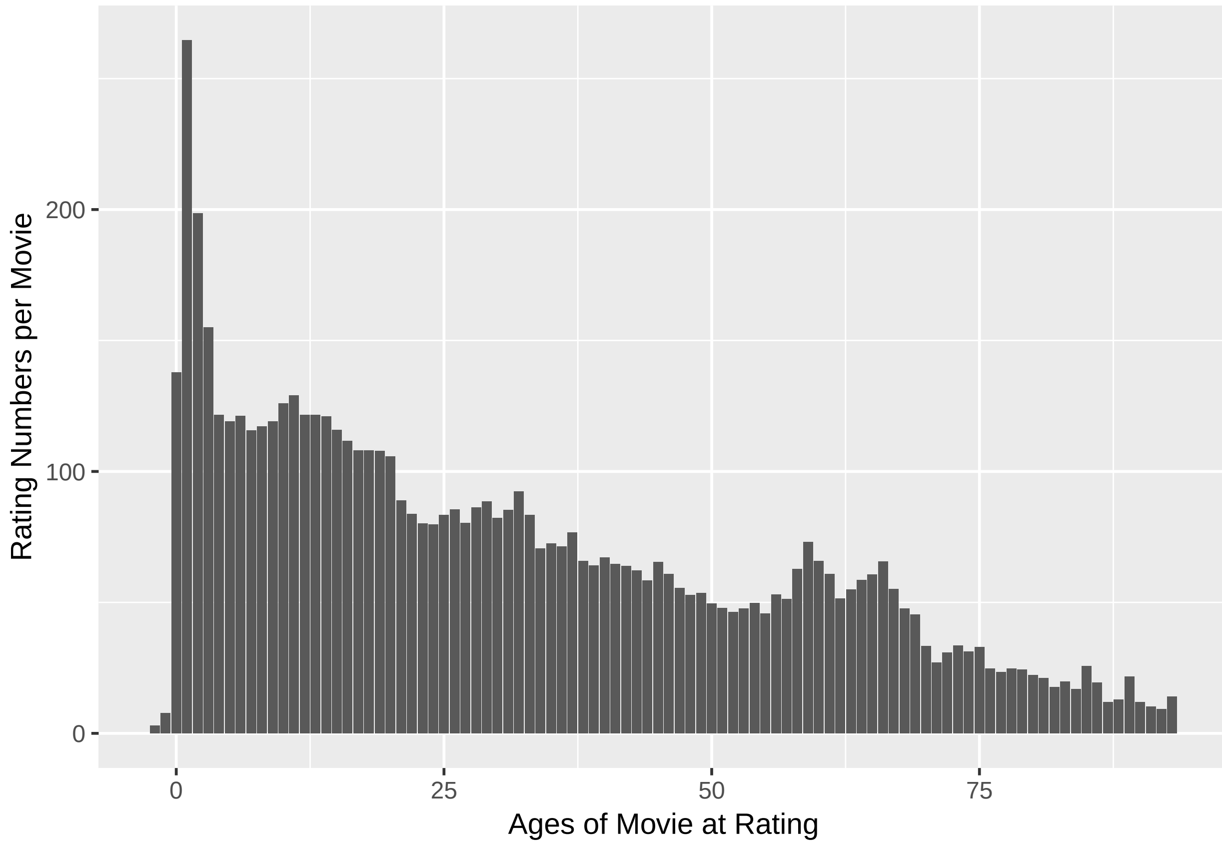
Could how old/new a movie was when watched/rated by a user affect the rating it got? First, let's take a look at the distribution of rating numbers according to the age of movies at the time of rating.



It appears that mosting ratings were given for newer movies (within 20 years old at the time of rating). However, is it simply because there are more newer movies in the whole dataset that have been rated by users? Let's take a look at the distribution of movie numbers:

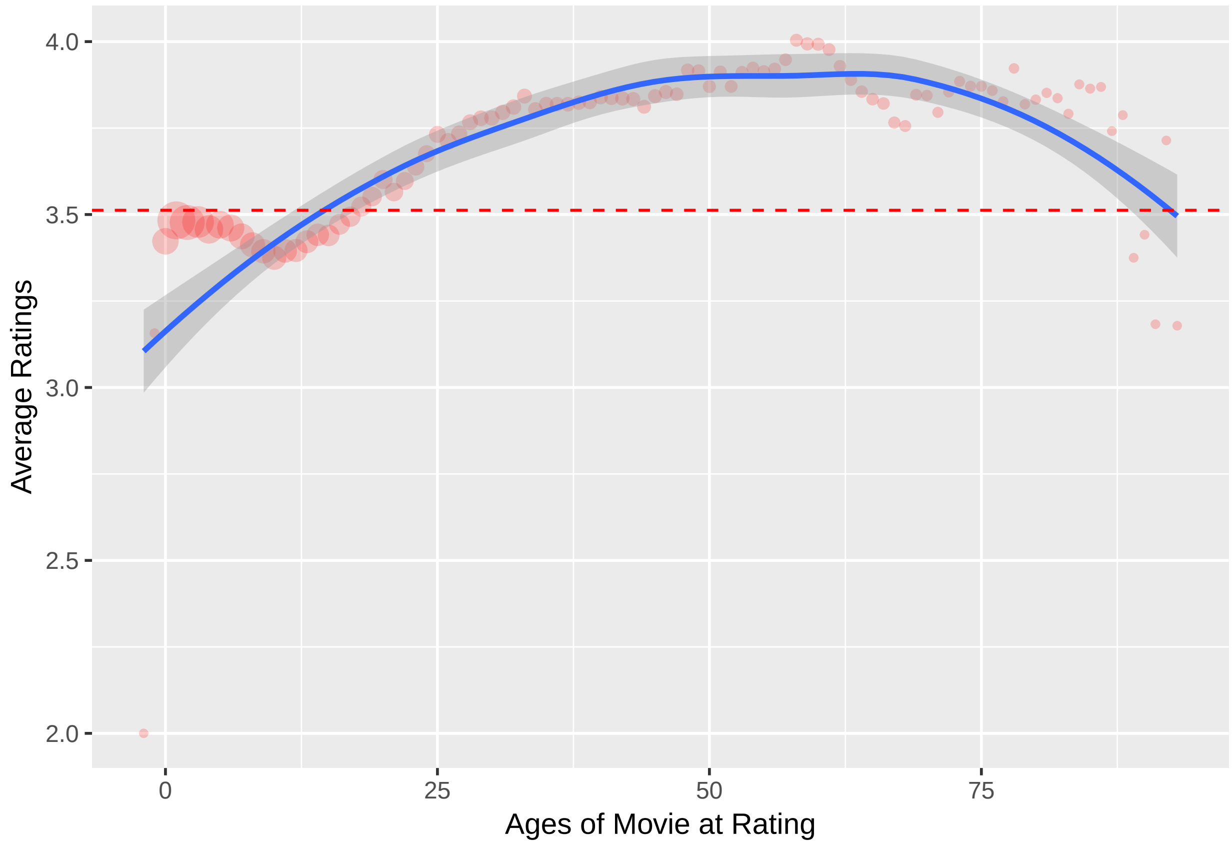


We see that actually as you can imagine, we have a similar trend that most movies in the dataset are within around 25 years old when being rated. To have a better idea of if newer movies get rated more frequently than older movies, we'd better look at rating times per movie for a given age group:

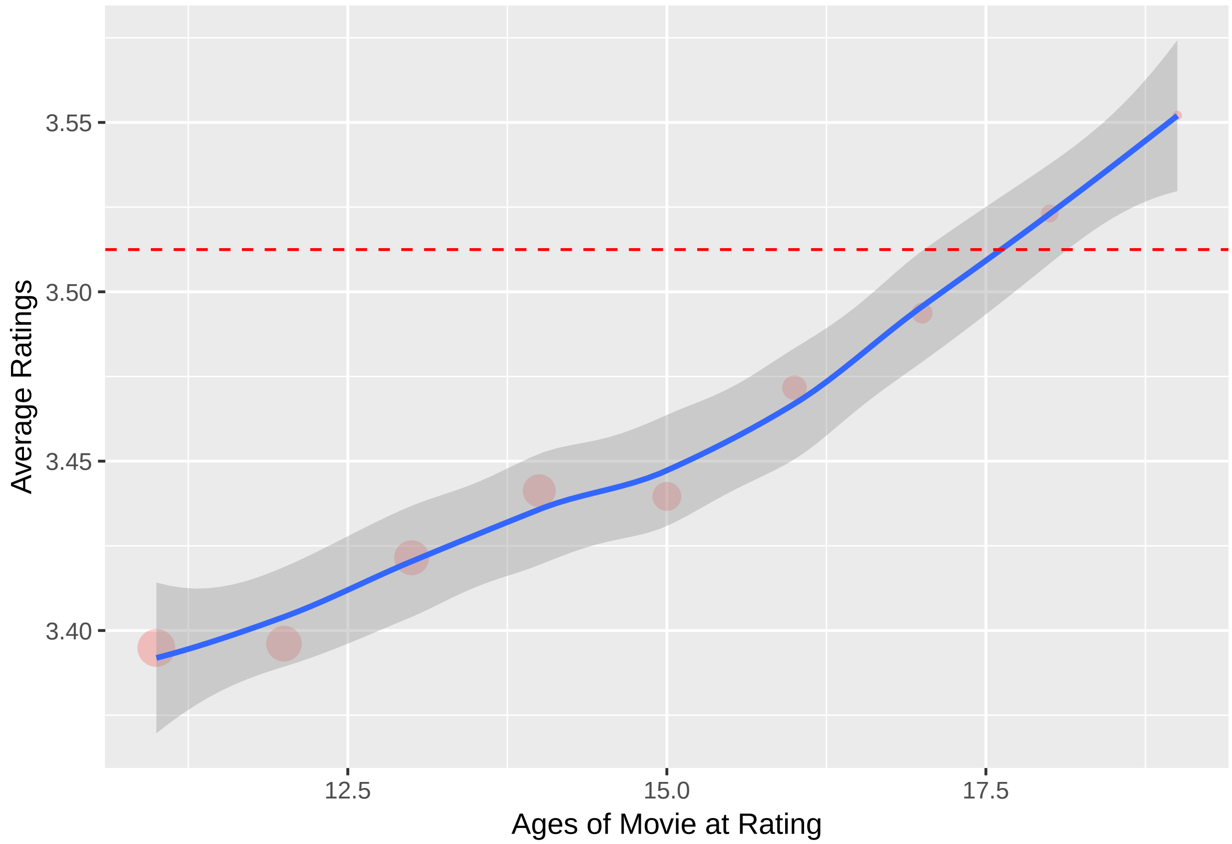


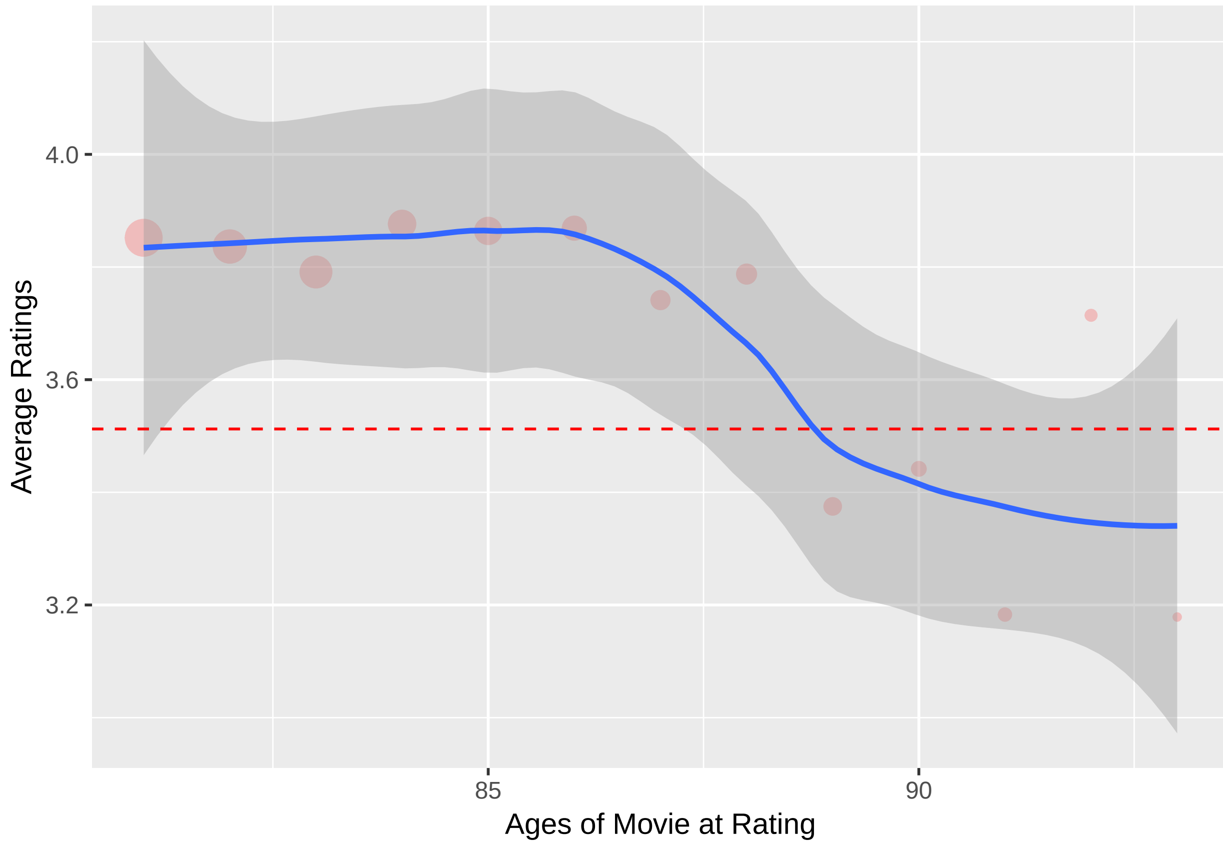
Although the trend is still that older movies get less frequently rated. It's not as dramatic as the overall rating times when we normalize by the movie numbers now.

The next question we try to address here is whether the age of movie at the time of rating affects rating? We can plot the average ratings for movies with a certain age at the time of ratings against the age. The red circles represent the average ratings while the size of a circle corresponses to the numbers of ratings for the movies with the same age at the time of rating. The blue line shows a smoothened overall trend. I also add a red dashed line here to represent the overall average rating. I further zoom in the critical area when the ratings get close to the average.



Average Ratings
4.0
3.5
3.0
2.5
2.0
0
25
50
75
Ages of Movie at Rating


3.55
3.50
3.45
3.40
Average Ratings
12.5
15.0
17.5
Ages of Movie at Rating

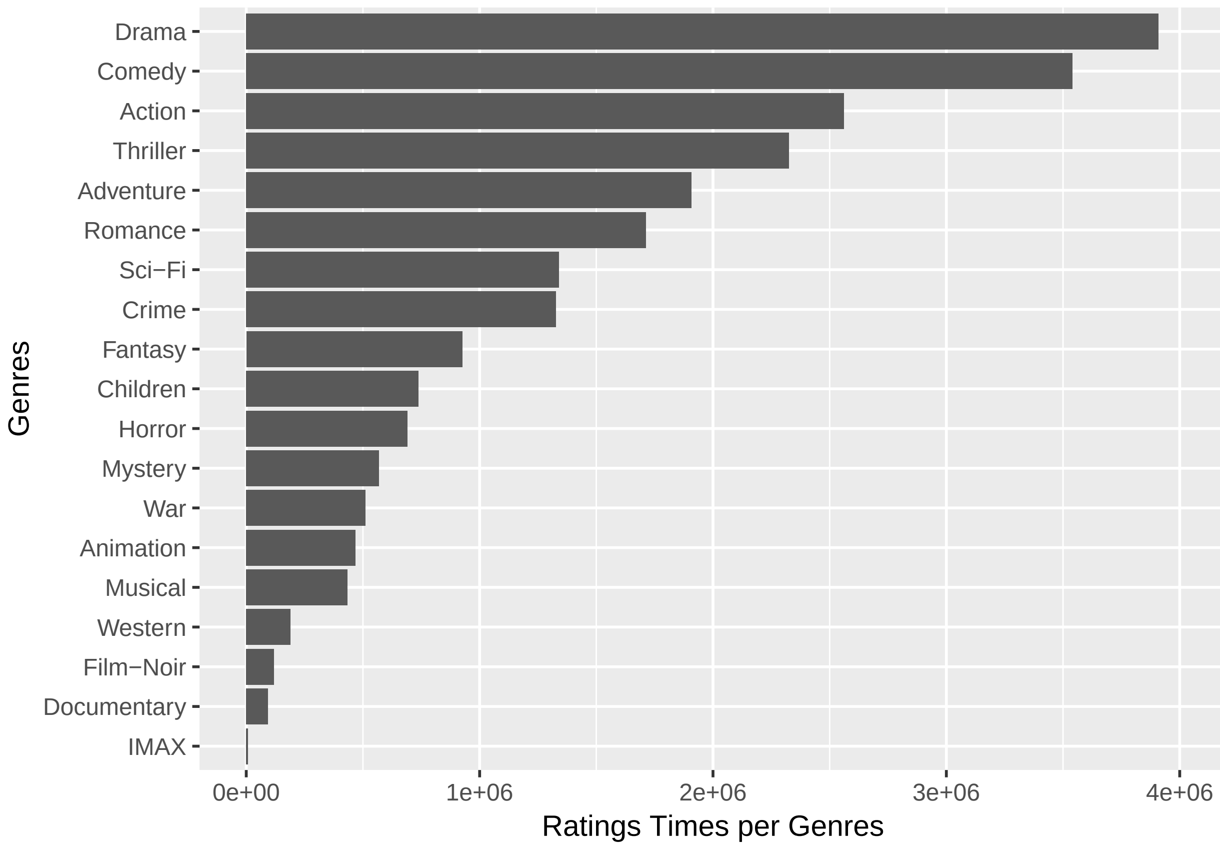


Interestingly, movies with less than 18 years tend to have around average or slightly lower than average ratings, while older movies tend to have much greater than average ratings (with only handful exceptions).

This could be because old movies that are still widely watched are usually those with good reputation and are widely recommended. In another word, they have been already selected. On the other hand, newer movies have not been judged enough by customers to have a selection effect.

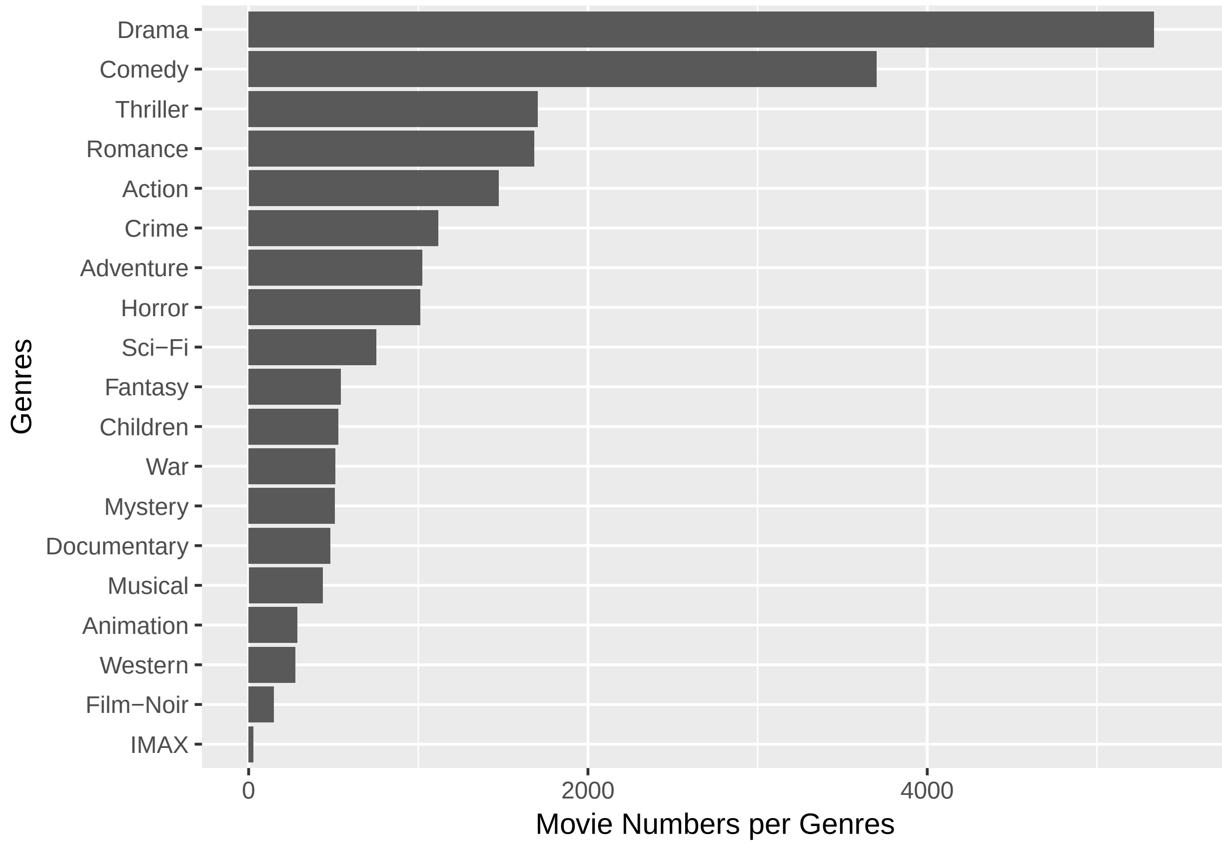
3.5 Exploration of genres

Next let's explore the effect of genres on movie ratings. First, are movies of different genres rated in different frequencies?



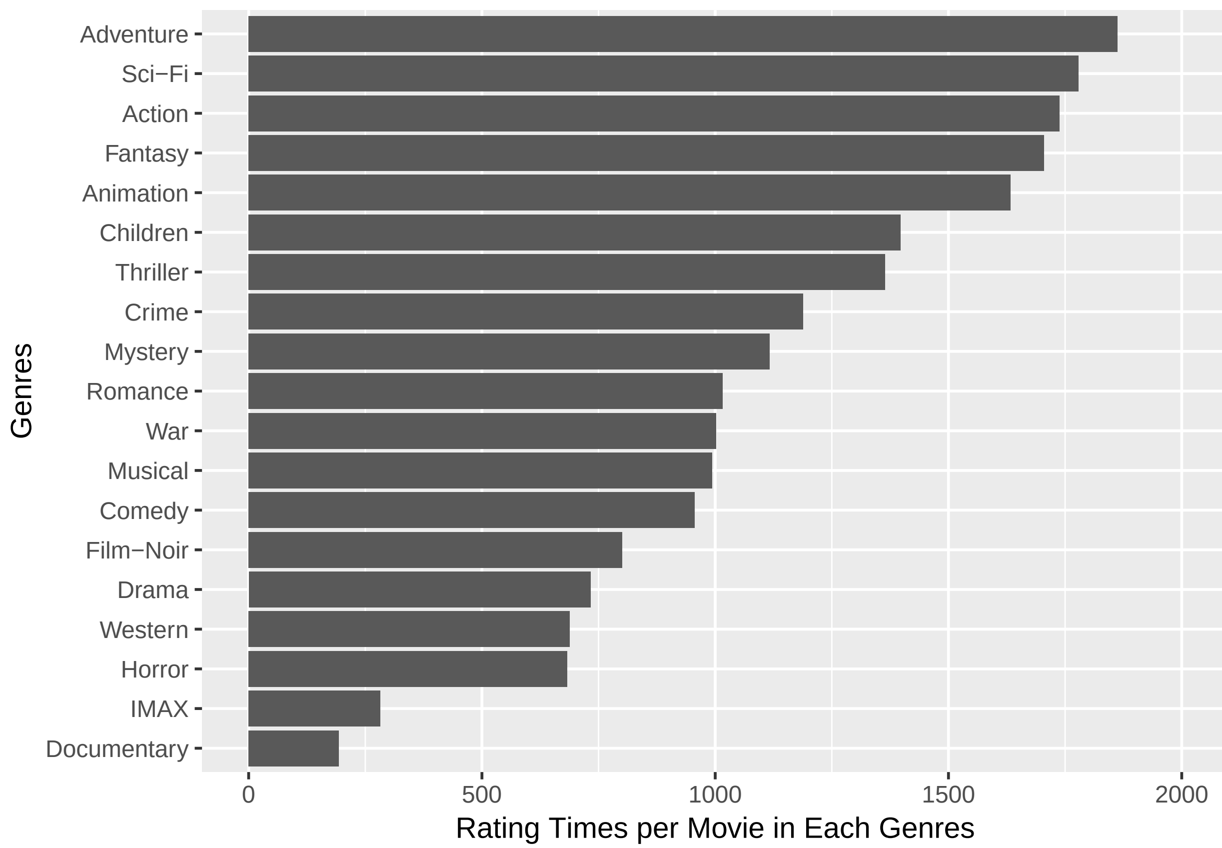
This figure shows the number of movies in each genres. Note that one single movie can actually belong to different genres. The Drama genres is rated the most while the IMAX genres is the least rated. However, this doesn't necessarily mean people prefer to rate the Drama movies over other types, because this could simply reflects there are more movies in the Drama genres.

Let's check the numbers of movies of each genres:



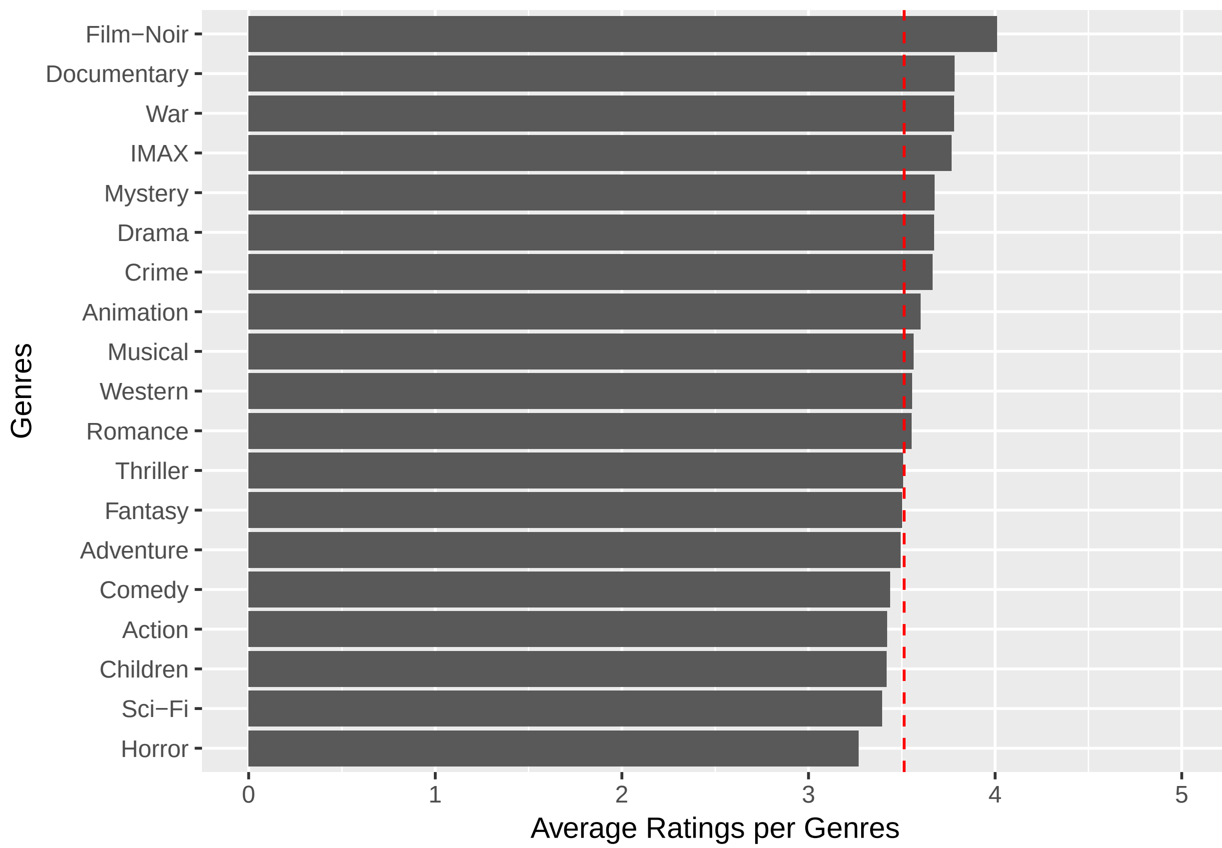
As expected, we see the same trend as we saw in total rating numbers.

So, to evaluate really how genres affect rating frequencies of movies, let's check the average numbers of ratings per movie in each genres:



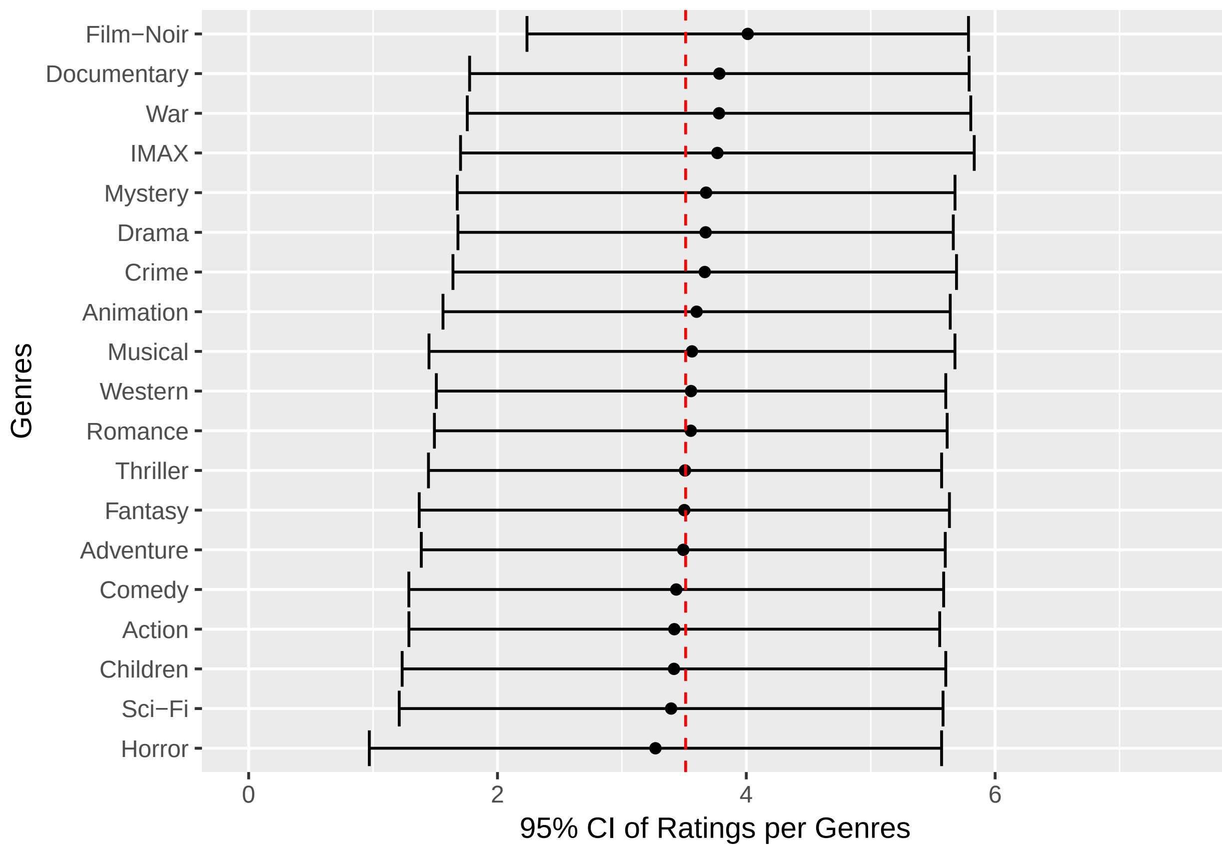
Now we see a different trend. Interestingly, Adventure and Sci-Fi are more tented to be rated although they don't include as many movies compared to Drama.

Next question is: what about the value of ratings themselves (which we actually care about)?



Now we find that some genres tend to have higher ratings than the average (such as Film-Noir) and some tend to have lower ratings (such as Horror). However, overall, the genres effect seems to be rather minor.

Another way to visulize the data is: instead of only looking at average, we can also see the distributioin. So we calculate both mean and sd of each genres and generate the 95% CI:



We can also see that the genres only slightly affect movie ratings.

4. Modeling strategy

Based on the exploration of the data described above, age of movie at rating seems to affect the rating, while genres does not add much information. Also, the effect of genres could also be included in the movie effect itself. Therefore, in the next section, to first build a baseline prediction model, I will consider the effects of movie age at rating, movie effect, and user effect. Regularization of movie effect and user effect will also be used to build a more robust model. I will evaluate these models and choose the best to go with. Residuals will be calculated and used as the input of matrix factorization technique.

The baseline model was generated following the instructions described in the "Recommendation Systems" of the text book (https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems). For example, the movie and user effect model describes the rating Y_u_i = mu + b_i + b_u + error, while mu is the average of all ratings, b_i and b_u are item (i.e., movie) bias and user bias, respectively. To avoid over-training caused by estimates from small sample sizes, I use regularization to add penalties to shrink the estimates from small samples sizes.

As mentioned in the text book (https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems), matrix factorization is a very useful techinique used in recommendation systems. It can identify the similar patterns of movies as well as users in terms of being rated or rating a movie. I first calculated the residual (r_u_i = y_u_i - mu - b_i - b_u) of the predictions based on the baseline model, and used the matrix factorization strategy to model the residual. Matrix factorization tries to decompose the rating matrix into user matrix p_u and item (movie) matirx q_i, so that r_u_i can be explained by p_u*q_i. To achieve this goal, I used the R package "recosystem" (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html), which can conveiniently apply matrix factorization to my dataset using the parallel stochastic gradient descent algorithm.

This "recosystem" R package is a wrapper of an open source C++ libraray LIMBF. A good thing about "recosystem" is that it can significantly reduce memory use by storing input, model and output information in the hard disk instead of using memory.

Results

Define RMSE: residual mean squared error

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

Model 1

First model: use average ratings for all movies regardless of user

In the first model, just based on the ratings itself, to minimize the RMSE, the best prediction of ratings for each movie will be the overall average of all ratings. The average rating is mu = 3.51247, and the naive RMSE is 1.0612.

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

```
## [1] 3.51247
```

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

```
## [1] 1.0612
```

```
rmse_results <- data_frame(Model = "Just the average", RMSE = naive_rmse)
rmse_results
```

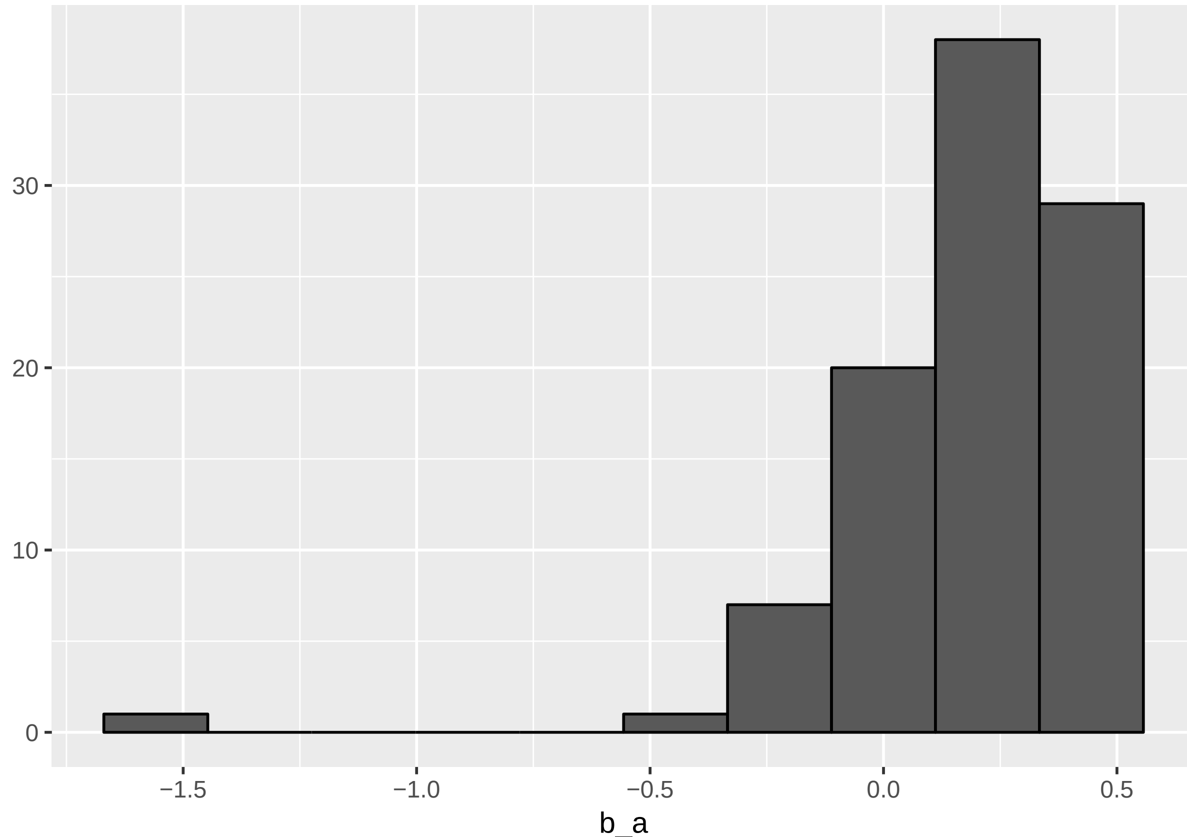
```
## # A tibble: 1 x 2
##   Model               RMSE
##   <chr>              <dbl>
## 1 Just the average 1.06120
```

Model 2

Modeling Age Effects: adding b_a to represent ratings on movies with certain age

Because earlier we saw that the age of movies at the time of rating seems to affect ratings, I try to see if add a bias of age (b_a) to the model could better predict the ratings. First let's calculate the age bias and take a look at its distribution. Then we will make predictions and evaluate the RMSE using the *validation* set.

```r
age_effect<- edx_1 %>%
  group_by(age_at_rating) %>%
  summarize(b_a = mean(rating)-mu)
age_effect %>% qplot(b_a, geom ="histogram", bins = 10, data = ., color = I("black"))
```




```r
validation_1 <- validation %>%
  mutate(year_rated = year(as_datetime(timestamp)))%>%
  mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp) %>%
  mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))

predicted_ratings_2 <- mu + validation_1 %>%
  left_join(age_effect, by='age_at_rating') %>%
  pull(b_a)
model_2_rmse <- RMSE(validation$rating,predicted_ratings_2) # 1.05239
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Age Effect Model",
                                     RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   Model                 RMSE
##   <chr>                <dbl>
## 1 Just the average 1.06120
```

```
## 2 Age Effect Model 1.05239
```

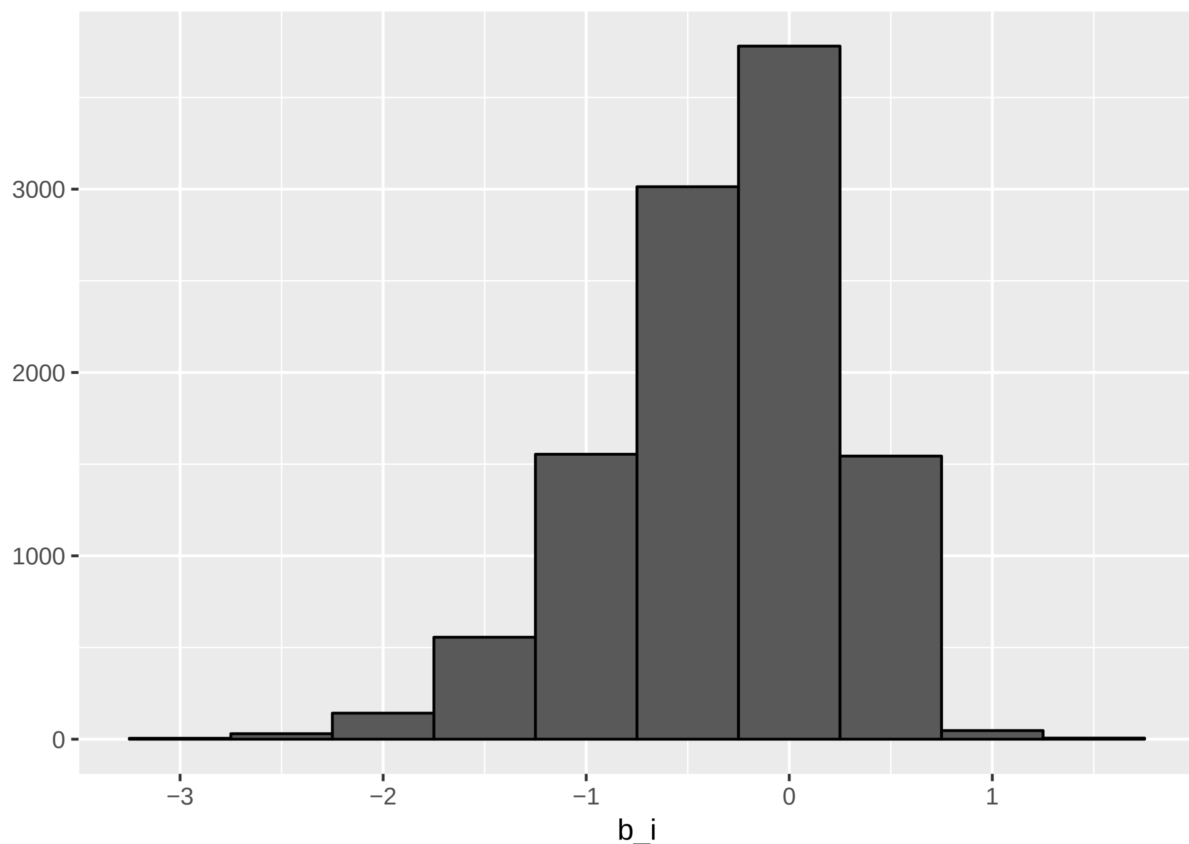
We can see that Age Effect Model did not improve the RMSE much. For this reason we will stop using the ages of movies as a predictor.

Model 3

Modeling movie effects: adding b_i to represent average ranking for movie_i

Since the intrinsic features of a movie could obviously affect the ratings of a movie, we add the bias of movie/item (b_i) to the model, i.e., for each movie, the average of the ratings on that specific movie will have a difference from the overall average rating of all movies. We can plot the distribution of the bias and calculate the RMSE of this model.

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




```
predicted_ratings_3 <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_3_rmse <- RMSE(validation$rating,predicted_ratings_3)
rmse_results <- bind_rows(rmse_results,
```

```r
                          data_frame(Model="Movie Effect Model",
                                     RMSE = model_3_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Just the average   1.06120
## 2 Age Effect Model   1.05239
## 3 Movie Effect Model 0.943909
```

Adding the movie bias successfully brought the RMSE to lower than 1.

Model 4

User effects: adding b__u to represent average ranking for user__u

Similar to the movie effect, intrinsic features of a given user could also affect the ratings of a movie. For example, a stricter user could give lower scores for all movies he/she watched than rated by other users. We now further add the bias of user (b__u) to the movie effect model.

```r
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings_4 <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_4_rmse <- RMSE(validation$rating,predicted_ratings_4)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie + User Effects Model",
                                     RMSE = model_4_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   Model                          RMSE
##   <chr>                         <dbl>
## 1 Just the average           1.06120
## 2 Age Effect Model           1.05239
## 3 Movie Effect Model         0.943909
## 4 Movie + User Effects Model 0.865349
```

Adding the user effect dramatically increased the RMSE to lower than 0.9.

Model 5

Regularization of movie effect: control the total variability of the movie effects

A machine learning model could be over-trained if some estimates were from a very small sample size. Regularization technique should be used to take into account the number of ratings made for a specific movie, by adding a larger penalty to estimates from smaller samples. To do this, a parameter lambda will be used. Cross validation within the test set can be performed to optimize this parameter before being applied to the validation set.

1. Perform cross validation to determine the parameter lambda

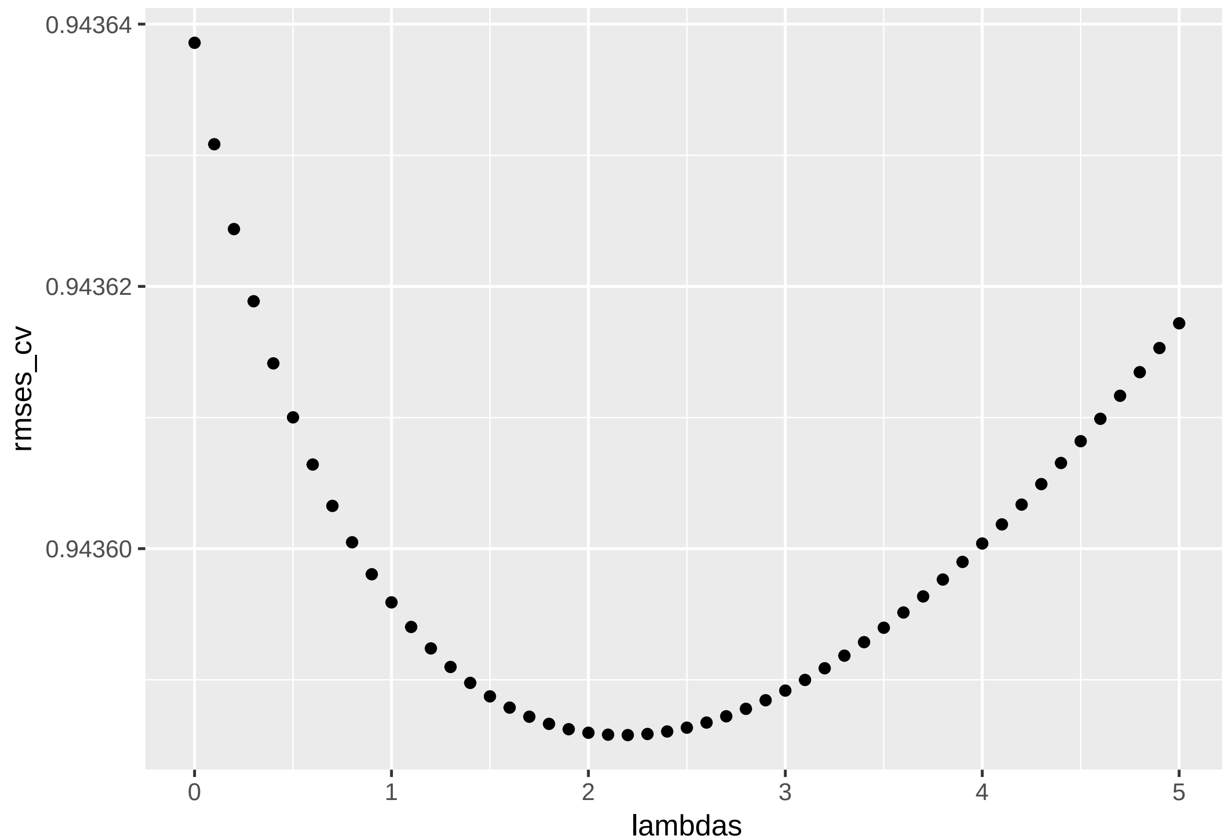
To train the parameter lambda, I use 10-fold cross validation here within only the *edx* set, because the *validation* set should not be used to train any parameter.

Specifically, I first randomly split the training set (*edx*) into 10 parts. Each time, I combine 9 parts as a *train_set*, and use the 10th part as a *test_set*. I will make sure all userIds and movieIds in test set are also in the train set. For a given range of different values of lambda, I will build the model using the *train_set* and evaluate the performance using the *test_set*. By doing this, I will get a set of RMSEs corresponding to these different lambdas.

In total, there will be 10 possible combinations of the 10 subsets of *edx*, so I will have 10 sets of *train_set* and *test_set*. Thus, I will do 10 times of training and get 10 sets of RMSEs. For each lambda value I try, I will have 10 RMSEs and take the average. Then I will determine the minimal RMSE and use the corresponding lambda as the optimized lambda for model building and performance evaluation in the *validation* set.

```r
# use 10-fold cross validation to pick a lambda for movie effects regularization
# split the data into 10 parts
set.seed(2019, sample.kind = "Rounding")
cv_splits <- createFolds(edx$rating, k=10, returnTrain =TRUE)

# define a matrix to store the results of cross validation
rmses <- matrix(nrow=10,ncol=51)
lambdas <- seq(0, 5, 0.1)

# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

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

  rmses[k,] <- sapply(lambdas, function(l){
    predicted_ratings <- test_final %>%
      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_final$rating))
  })
}

rmses_cv <- colMeans(rmses)
qplot(lambdas,rmses_cv)
lambda <- lambdas[which.min(rmses_cv)   #2.2
```



From the 10-fold cross validation, we get an optimized value of lambda: 2.2.

2. Model generation and prediction

Regularized Movie Effect Model

Using the optimized lambda, we can now perform prediction and evaluate the RMSE in the *validation* set.

```
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
predicted_ratings_5 <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
model_5_rmse <- RMSE(predicted_ratings_5, validation$rating)   # 0.943852 not too much improved
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie Effect Model",
                                     RMSE = model_5_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   Model                              RMSE
##   <chr>                             <dbl>
## 1 Just the average               1.06120
## 2 Age Effect Model               1.05239
## 3 Movie Effect Model             0.943909
## 4 Movie + User Effects Model     0.865349
## 5 Regularized Movie Effect Model 0.943852
```

The application of regularization does not improve the RMSE too much.

Model 6

Regularization of both movie and user effects (use the same lambda for both movie and user effects)

1. Perform cross validation to determine the parameter lambda

Similar to the movie effect, now we perform regularization on both movie and user effects. Still using 10-fold cross validation as described above, we will train one single lambda value for both movie and user effects.

```r
# define a matrix to store the results of cross validation
lambdas <- seq(0, 8, 0.1)
rmses_2 <- matrix(nrow=10,ncol=length(lambdas))
# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_2[k,] <- sapply(lambdas, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
```

```
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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_final$rating))
  })
}

rmses_2
rmses_2_cv <- colMeans(rmses_2)
rmses_2_cv
qplot(lambdas,rmses_2_cv)
lambda <- lambdas[which.min(rmses_2_cv)]   #4.9
```

From the 10-fold cross validation, we get an optimized value of lambda: 4.9.

2. Model generation and prediction

Regularized Movie Effect and User Effect Model

Now we use this parameter lambda to predict the validation dataset and evaluate the RMSE.

```
mu <- mean(edx$rating)
b_i_reg <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_6 <-
    validation %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
model_6_rmse <- RMSE(predicted_ratings_6, validation$rating)   # 0.864818
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie + User Effect Model",
                                     RMSE = model_6_rmse))
rmse_results
```

```
## # A tibble: 6 x 2
##   Model                                      RMSE
##   <chr>                                     <dbl>
## 1 Just the average                        1.06120
## 2 Age Effect Model                        1.05239
## 3 Movie Effect Model                      0.943909
## 4 Movie + User Effects Model              0.865349
```

```
## 5 Regularized Movie Effect Model        0.943852
## 6 Regularized Movie + User Effect Model 0.864818
```

Regularization slightly improved the prediction performance of the model.

Model 7

Regularization of movie and user effects: use dfferent lambdas

Optimizing lambda_u (user effect) with fixed lambda_i (movie effect)

1. Perform cross validation to determine the parameter lambda_u for a given lambda_i

Instead of optimizing the same lambda for both user and movie effect, here I tried to fix the lambda for movie using the value we got in model 5 (lambda_i=2.2), and optimize the lambda for user (lambda_u).

```
# define a matrix to store the results of cross validation
lambda_i <- 2.2
lambdas_u <- seq(0, 8, 0.1)
rmses_3 <- matrix(nrow=10,ncol=length(lambdas_u))

# perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

  # Make sure userId and movieId in test set are also in the train set
  test_final <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_3[k,] <- sapply(lambdas_u, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+lambda_i))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      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_final$rating))
  })
```

```r
}
rmses_3
rmses_3_cv <- colMeans(rmses_3)
rmses_3_cv
qplot(lambdas_u,rmses_3_cv)
lambda_u <-lambdas_u[which.min(rmses_3_cv)]   #5
```

For a given lambda_i of 2.2, we get an optimized lambda_u of 5.

2. Model generation and prediction

Regularized Movie and User Effect Model with fixed lambda for Movie Effect

Using the lambda_i and lambda_u we determined, I generated the prediction model and evaluated the RMSE in the validation set.

```r
lambda_i <- 2.2
lambda_u <- 5
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_i))
b_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_u))
predicted_ratings_7 <-
  validation %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_7_rmse <- RMSE(predicted_ratings_7, validation$rating)   # 0.86485
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie + User Effect Model Version 2",
                                     RMSE = model_7_rmse))
rmse_results
```

```
## # A tibble: 7 x 2
##   Model                                               RMSE
##   <chr>                                              <dbl>
## 1 Just the average                                 1.06120
## 2 Age Effect Model                                 1.05239
## 3 Movie Effect Model                               0.943909
## 4 Movie + User Effects Model                       0.865349
## 5 Regularized Movie Effect Model                   0.943852
## 6 Regularized Movie + User Effect Model            0.864818
## 7 Regularized Movie + User Effect Model Version 2 0.864850
```

While regularization using different parameters for user and item slightly improve RMSE (comparing Model 7 and Model 4), it did not improve the last regularization model (Model 6).

Model 8

Regularization of movie and user effects: use dfferent lambdas

Optimizing lambda_i (movie effect) with fixed lambda_u (user effect)

1. Perform cross validation to determine the parameter lambda_i for a given lambda_u

Here I want to see if anything changes when I slightly change the strategy to fix lambda_u and then choose lambda_i. For the fixed lambda_u = 5 (based on Model 7), I optimized lambda_i using 10-fold cross validation and got a lambda_i = 4.6.

2. Model generation and prediction

Regularized Movie and User Effect Model with fixed lambda for User Effect

A new model was generated similarly and RMSE determined using the *validation* set.

```
## # A tibble: 8 x 2
##   Model                                              RMSE
##   <chr>                                             <dbl>
## 1 Just the average                                 1.06120
## 2 Age Effect Model                                 1.05239
## 3 Movie Effect Model                               0.943909
## 4 Movie + User Effects Model                       0.865349
## 5 Regularized Movie Effect Model                   0.943852
## 6 Regularized Movie + User Effect Model            0.864818
## 7 Regularized Movie + User Effect Model Version 2 0.864850
## 8 Regularized Movie + User Effect Model Version 3 0.864819
```

The RMSE is slightly better than Model 7 and now comparable to Model 6, in which the same lambda was used for both user and item effects.

Model 9

Matrix Factorization based on the residuals of the baseline model

1. Best baseline model

Models 1-8 are all baseline models mainly based on movie effect and user effect. I compared the RMSEs and determined to go with "Regularized Movie + User Effect Model" (model 6) prior to further exploration because it gives the least RMSE.

2. Calculating the residuals

Next, we need to calculate the residuals. Although earlier we evaluated the models using the *validation* set, to calculate the residual we cann't use the *validation* set here, because the *validation* set can only be used at the end when evaluating the performance of the final model. Instead, we need to still use the training set *edx*. So how is the RMSE based on the *edx* set (training set)? As expected, it is slightly better than calculated using the *validation* set:

```r
lambda <- 4.9
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_6_edx <-
  edx %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_6_rmse_edx <- RMSE(predicted_ratings_6_edx, edx$rating)
model_6_rmse_edx
```

```
## [1] 0.857032
```

All right, let's get the residuals of the prediction and then perform matrix factorization on the residuals.

```r
edx_residual <- edx %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(residual = rating - mu - b_i - b_u) %>%
  select(userId, movieId, residual)
head(edx_residual)
```

```
##   userId movieId residual
## 1      1     122 0.805170
## 2      1     185 0.535750
## 3      1     292 0.247181
## 4      1     316 0.315499
## 5      1     329 0.327707
## 6      1     355 1.176398
```

3. Use the recosystem library to perform the matrix factorization

Now let's use the recosystem library to perform the matrix factorization on the residuals. Both training and validation sets need to be organized to 3 columns: user, item (movies), value (ratings or residuals). Then they need to be transformed into matrix format. Next we write these datasets into hard disk, which will later be assigned to *train_set* and *valid_set* to build the "recosystem". A recommender object r will be built using Reco() in the *recosystem* package and parameters trained using the *train_set*.

Next the parameters will be used to build the prediction model. Because here we are modeling the residuals after Model 6, we add up the base prediction of Model 6 and the residuals predicted here to get the final prediction for the *validation* set. RMSE can be evaluated and compared with previous models.

```r
# as matrix
edx_for_mf <- as.matrix(edx_residual)
validation_for_mf <- validation %>%
```

```r
  select(userId, movieId, rating)
validation_for_mf <- as.matrix(validation_for_mf)

# write edx_for_mf and validation_for_mf tables on disk
write.table(edx_for_mf , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(validation_for_mf, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE

# use data_file() to specify a data set from a file in the hard disk.
set.seed(2019)
train_set <- data_file("trainset.txt")
valid_set <- data_file("validset.txt")

# build a recommender object
r <-Reco()

# tuning training set
opts <- r$tune(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))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.793797
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.807220
## 2   20        0     0.01        0     0.01   0.1 0.811662
## 3   30        0     0.01        0     0.01   0.1 0.820999
## 4   10        0     0.10        0     0.01   0.1 0.804869
## 5   20        0     0.10        0     0.01   0.1 0.802334
## 6   30        0     0.10        0     0.01   0.1 0.803364
## 7   10        0     0.01        0     0.10   0.1 0.803780
```

```
## 8   20         0     0.01        0      0.10    0.1 0.795706
## 9   30         0     0.01        0      0.10    0.1 0.793797
## 10  10         0     0.10        0      0.10    0.1 0.824731
## 11  20         0     0.10        0      0.10    0.1 0.823860
## 12  30         0     0.10        0      0.10    0.1 0.823186
## 13  10         0     0.01        0      0.01    0.2 0.809948
## 14  20         0     0.01        0      0.01    0.2 0.822562
## 15  30         0     0.01        0      0.01    0.2 0.837361
## 16  10         0     0.10        0      0.01    0.2 0.805661
## 17  20         0     0.10        0      0.01    0.2 0.804932
## 18  30         0     0.10        0      0.01    0.2 0.807245
## 19  10         0     0.01        0      0.10    0.2 0.802191
## 20  20         0     0.01        0      0.10    0.2 0.799993
## 21  30         0     0.01        0      0.10    0.2 0.799319
## 22  10         0     0.10        0      0.10    0.2 0.823480
## 23  20         0     0.10        0      0.10    0.2 0.822147
## 24  30         0     0.10        0      0.10    0.2 0.820740
```

```
# training the recommender model
r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))

# Making prediction on validation set and calculating RMSE:
pred_file <- tempfile()
r$predict(valid_set, out_file(pred_file))
predicted_residuals_mf <- scan(pred_file)
predicted_ratings_mf <- predicted_ratings_6 + predicted_residuals_mf
rmse_mf <- RMSE(predicted_ratings_mf,validation$rating) # 0.786256
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Matrix Factorization",
                                     RMSE = rmse_mf))
rmse_results
```

```
## # A tibble: 9 x 2
##   Model                                               RMSE
##   <chr>                                              <dbl>
## 1 Just the average                                  1.06120
## 2 Age Effect Model                                  1.05239
## 3 Movie Effect Model                                0.943909
## 4 Movie + User Effects Model                        0.865349
## 5 Regularized Movie Effect Model                    0.943852
## 6 Regularized Movie + User Effect Model             0.864818
## 7 Regularized Movie + User Effect Model Version 2 0.864850
## 8 Regularized Movie + User Effect Model Version 3 0.864819
## 9 Matrix Factorization                              0.786256
```

Conclusion

From the summarized RMSEs of different models, we can see that matrix factorization largely improved the accuracy of the prediction.

Model	RMSE
Just the average	**1.061202**
Age Effect Model	**1.052393**
Movie Effect Model	**0.943909**
Movie + User Effects Model	**0.865349**
Regularized Movie Effect Model	**0.943852**
Regularized Movie + User Effect Model	**0.864818**
Regularized Movie + User Effect Model Version 2	**0.864850**
Regularized Movie + User Effect Model Version 3	**0.864819**
Matrix Factorization	**0.786256**

MovieLens is a classical dataset for recommendation system and represents a challenge for development of better machine learning algorithm. In this project, the "Just the average" model only gives a RMSE of 1.0612, and the best baseline model (Model 6: Regularized Movie + User Effect Model) could largely improved it to 0.8648. Furthermore, matrix factorization greatly brought it down to 0.7863. In conclusion, matrix factorization appears to be a very powerful technique for recommendation system, which usually contains large and sparse dataset making it hard to make prediction using other machine learning strategies. The effects of age and genres could be further explored to improve the performance of the model. The Ensemble method should also be considered in the future to apply on the MovieLens dataset, in order to combine the advantages of various models and enhance the overall performance of prediction.