MovieLens Capstone Project

Can Zhang

Introduction

Recommendation system is one of the most popular applications of machine learning. It is widely used by companies such as Netflix, Amazon and YouTube to predict user preference in order to create highly personalized customer experience.

The MovieLens project is part of the Data Science Professional Certificate program offered by Harvard University through edX platform. The aim of the project is to create and train a recommendation algorithm to predict ratings given by certain users to specific movies. The Residual Mean Square Error (RMSE) is used to evaluate the accuracy of the algorithm. The goal is to achieve a RMSE smaller than 0.86490.

Six models were built for this project:

1. Baseline(to predict the average rating of all movies as expected rating)
2. Baseline + movie effect
3. Baseline + movie effect + user effect
4. Baseline + regularized movie effect + regularized user effect
5. Baseline + movie effect + user effect + Random Forest(ranger)
6. Matrix factorization(recosystem)

The best performing model - the matrix factorization model - is used for the final validation, which yielded a RMSE of 0.7861099(well below the target RMSE 0.86490).

This report will present an overview of the data set, the model building process, the results of the models and final validation, and the conclusion of this project.

The data set

The data set used in this project is the MovieLens 10M Dataset provided by Grouplens. Additional information of the data set can be found here.

The following code is used to generate the train set and validation set(final hold-out test set).

```
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
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)
```

Data exploration

The edx dataset contains 9000055 observations of 6 variables:

1. userId, integer. Each userId represents a unique user.
2. movieId, numeric.
3. rating, numeric. According to the "README" file on grouplens.org, ratings are made on a 5-star scale, with half-star increments.The average of all ratings is 3.512.
4. timestamp, integer. This variable originally comes from the "ratings" data file, therefore it represents the time when ratings were made. It uses Unix time, representing seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
5. title, character. It includes both the movie titles and year of release in brackets.
6. genres, character. Each movie may be associated to more than one genre, separated by a pipe.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
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
```

There are 69878 unique users and 10677 unique movies in the edx data set.

```
library(dplyr)
edx %>% summarise(n_unique_user = n_distinct(userId),
                  n_uniqe_movie = n_distinct(movieId))
```

```
##   n_unique_user n_uniqe_movie
## 1         69878         10677
```

The three most frequent ratings given by users are 4, 3,and 5.

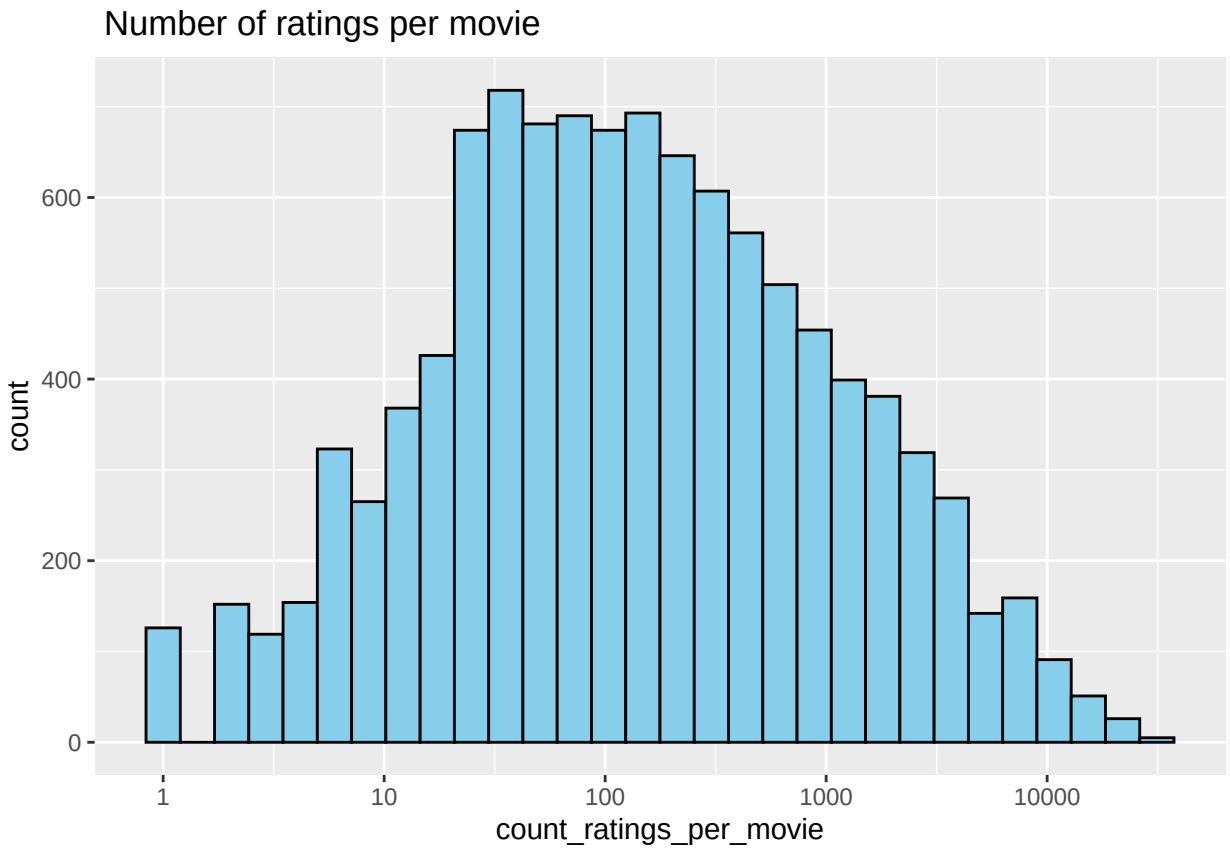
```
rating_frequency <- edx %>%
  group_by(rating) %>%
  summarise(count_rating = n()) %>%
  arrange(desc(count_rating)) %>%
    knitr::kable()
rating_frequency
```

rating	count_rating
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

Movies and ratings

There are not many movies which received more than 10000 ratings or less than 10 ratings. The distribution of number of ratings per movie is as follows:

```
ratings_per_movie <- edx %>%
  group_by(movieId, title) %>%
  summarise(count_ratings_per_movie = n()) %>%
  arrange(desc(count_ratings_per_movie))

ratings_per_movie %>% ggplot(aes(count_ratings_per_movie)) +
  geom_histogram(color = "black", fill = "sky blue", bins = 30) +
  scale_x_log10()+
  ggtitle(" Number of ratings per movie")
```



Different movies received different ratings. By calculating the average rating that each movie received, we find that some movies received extremely high or low ratings, indicating there's a "movie effect" that we should consider.

```
final_rating_of_movies <- edx %>%
  group_by(movieId, title) %>%
  summarise(final_rating = mean(rating), count_rating = n()) %>%
  arrange(desc(final_rating))
```

By checking the 10 best and 10 worst movies, we notice that most of those movies are very obscure, and most of them only received very few ratings. This is because with very few ratings, the ratings tend to be more extreme and more biased. So if we use these very small samples to make predictions, the error is likely very large. Therefore, we should try to regularize extreme estimates which are formed using small samples.

```
final_rating_of_movies[1:10,] %>% knitr::kable()
```

movieId	title	final_rating	count_rating
3226	Hellhounds on My Trail (1999)	5.00	1
33264	Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	5.00	2
42783	Shadows of Forgotten Ancestors (1964)	5.00	1
51209	Fighting Elegy (Kenka erejii) (1966)	5.00	1
53355	Sun Alley (Sonnenallee) (1999)	5.00	1
64275	Blue Light, The (Das Blaue Licht) (1932)	5.00	1

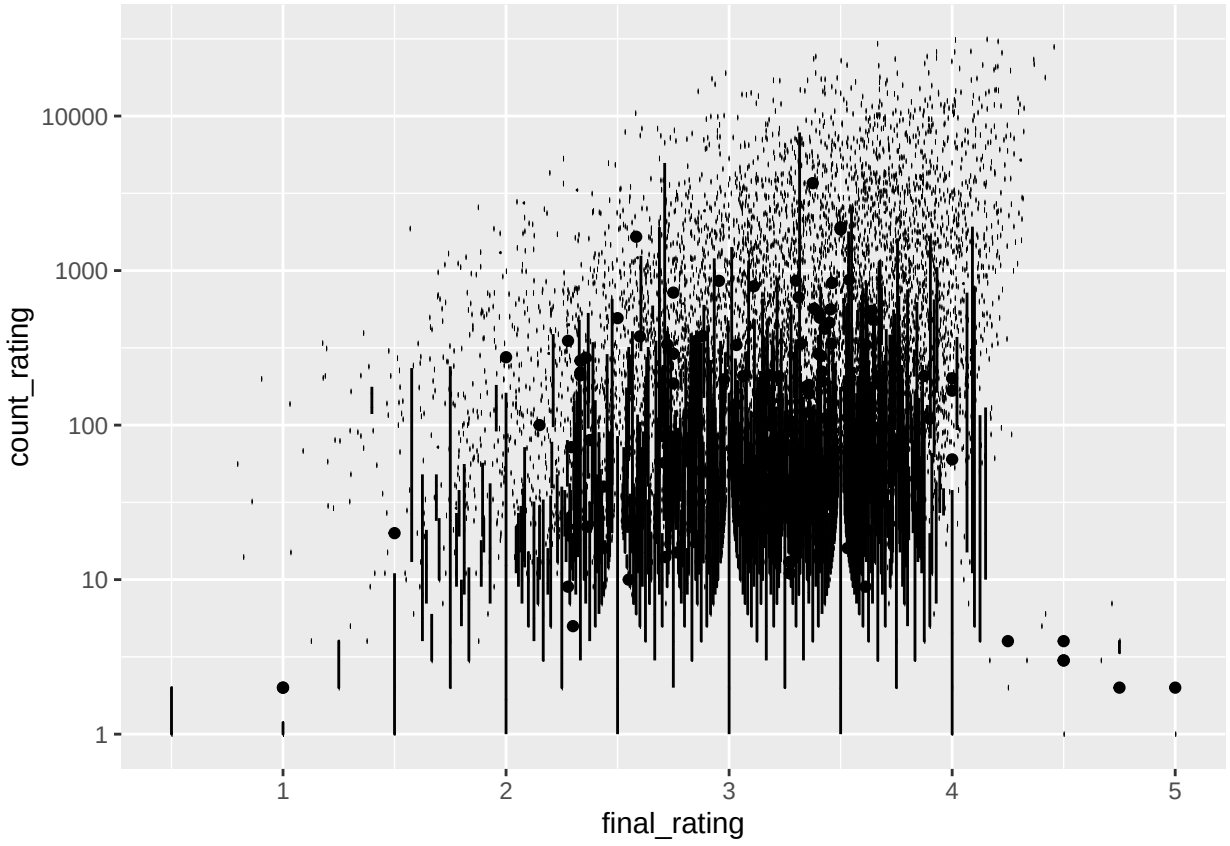
movieId	title	final_rating	count_rating
5194	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	4.75	4
26048	Human Condition II, The (Ningen no joken II) (1959)	4.75	4
26073	Human Condition III, The (Ningen no joken III) (1961)	4.75	4
65001	Constantine's Sword (2007)	4.75	2

```
final_rating_of_movies[10667:10677,]%>% knitr::kable()
```

movieId	title	final_rating	count_rating
6189	Dischord (2001)	1.0000000	1
55324	Relative Strangers (2006)	1.0000000	1
6483	From Justin to Kelly (2003)	0.9020101	199
61348	Disaster Movie (2008)	0.8593750	32
7282	Hip Hop Witch, Da (2000)	0.8214286	14
8859	SuperBabies: Baby Geniuses 2 (2004)	0.7946429	56
5805	Besotted (2001)	0.5000000	2
8394	Hi-Line, The (1999)	0.5000000	1
61768	Accused (Anklaget) (2005)	0.5000000	1
63828	Confessions of a Superhero (2007)	0.5000000	1
64999	War of the Worlds 2: The Next Wave (2008)	0.5000000	2

By plotting the rating of each movie against number of ratings they receive, we notice that the most frequently rated movies tend to have above average ratings. Therefore, the number of ratings may have an impact on the rating that a movie receives.

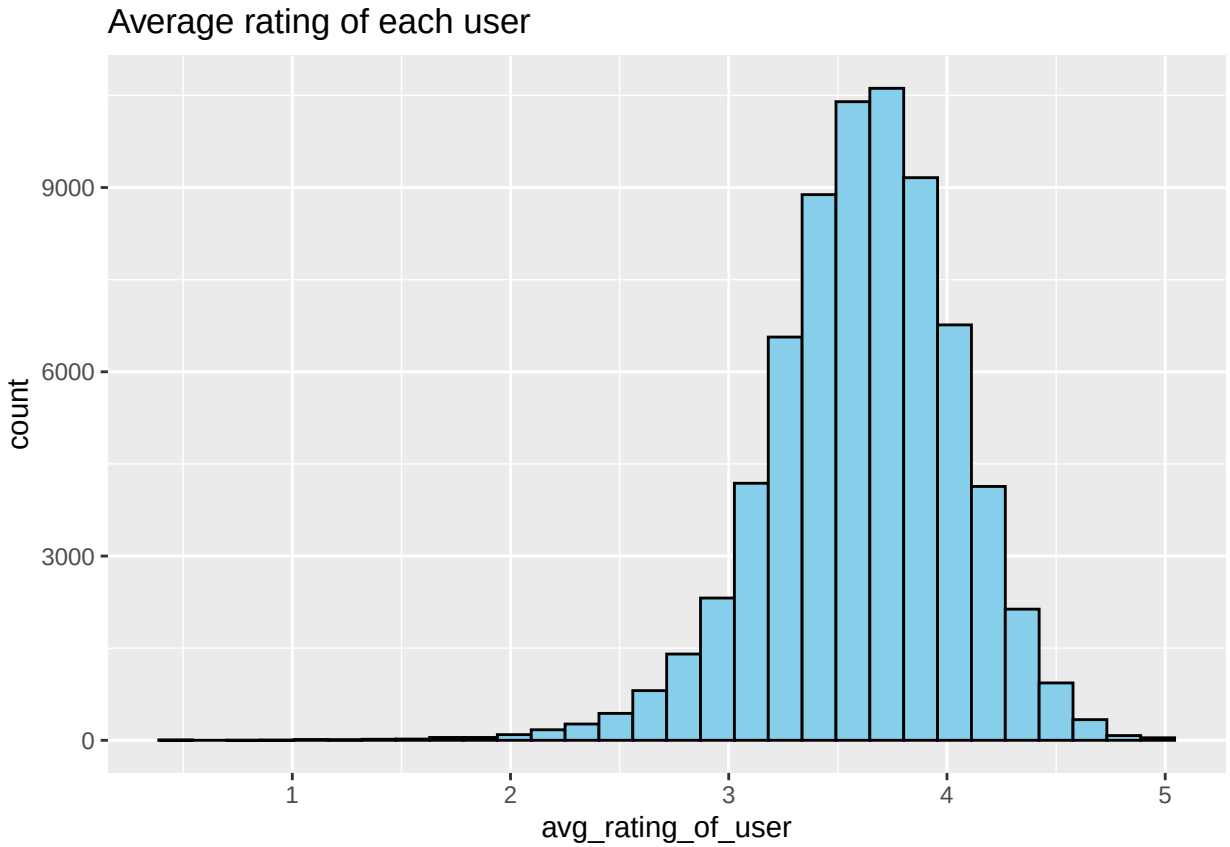
```
#plot rating of each movie against number of ratings they receive
final_rating_of_movies %>%
    ggplot(aes(final_rating, count_rating, group = final_rating)) +
    geom_boxplot(color = "black") +
    scale_y_log10()
```



Users and ratings

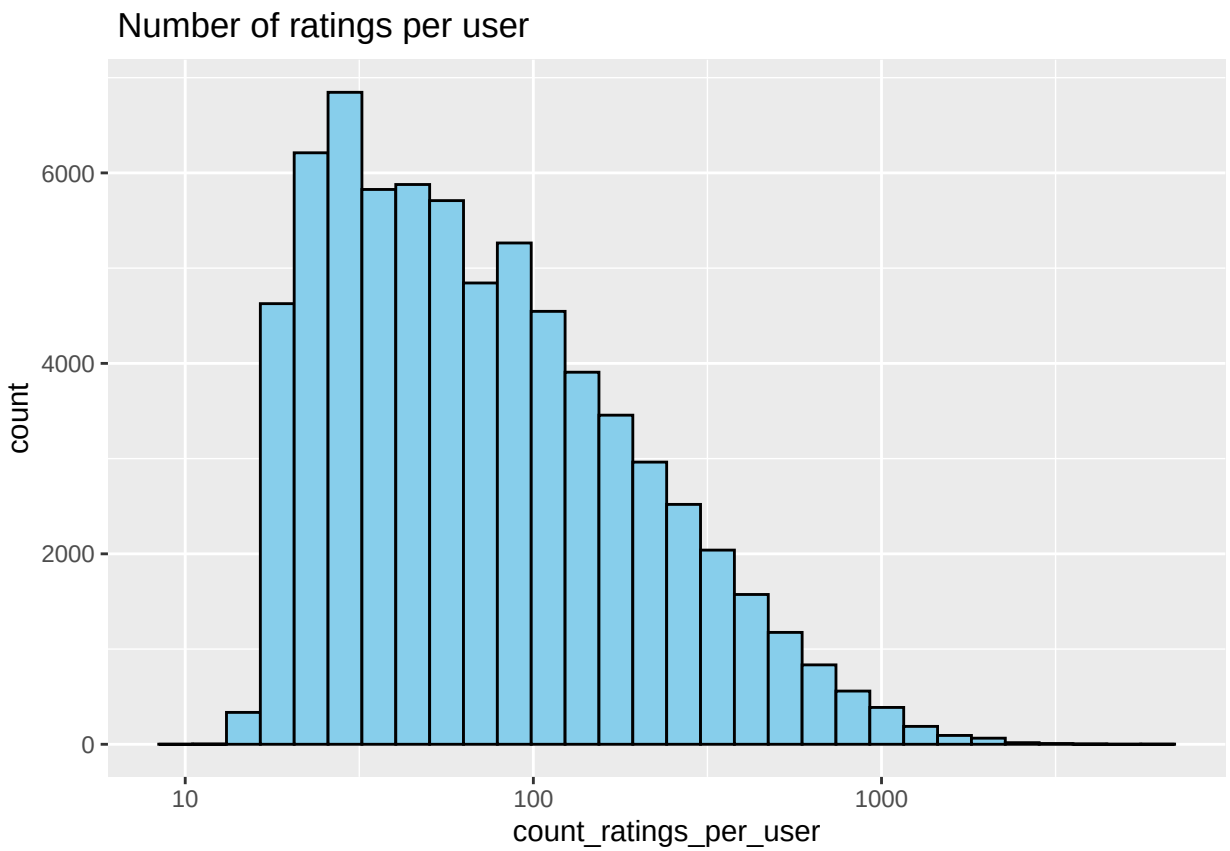
We know some people tend to be more generous than others when they rate movies. By taking a look of the average rating that each user give, we notice that some users give very high ratings to pretty much everything they watched, and some do the opposite. The variability across users implies "user effect" should be taken into consideration.

```
average_rating_of_user <- edx %>%
  group_by(userId) %>%
  summarise(avg_rating_of_user = mean(rating)) %>%
  ggplot(aes(avg_rating_of_user)) +
  geom_histogram(color = "black", fill = "sky blue", bins = 30) +
  ggtitle("Average rating of each user")

average_rating_of_user
```



Some users have rated very few movies compared to others.

```
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarise(count_ratings_per_user = n()) %>%
  arrange(desc(count_ratings_per_user)) %>%
  ggplot(aes(count_ratings_per_user)) +
  geom_histogram(color = "black", fill = "sky blue", bins = 30) +
  scale_x_log10()+
  ggtitle(" Number of ratings per user")

ratings_per_user
```



Genres and ratings

The movies in edx data set may be associated to more than one genre, separated by a pipe. With the following code, we split the "genres" variable and count how many movie ratings are in each genre.

The 3 genres that received the most ratings are Drama, Comedy and Action.

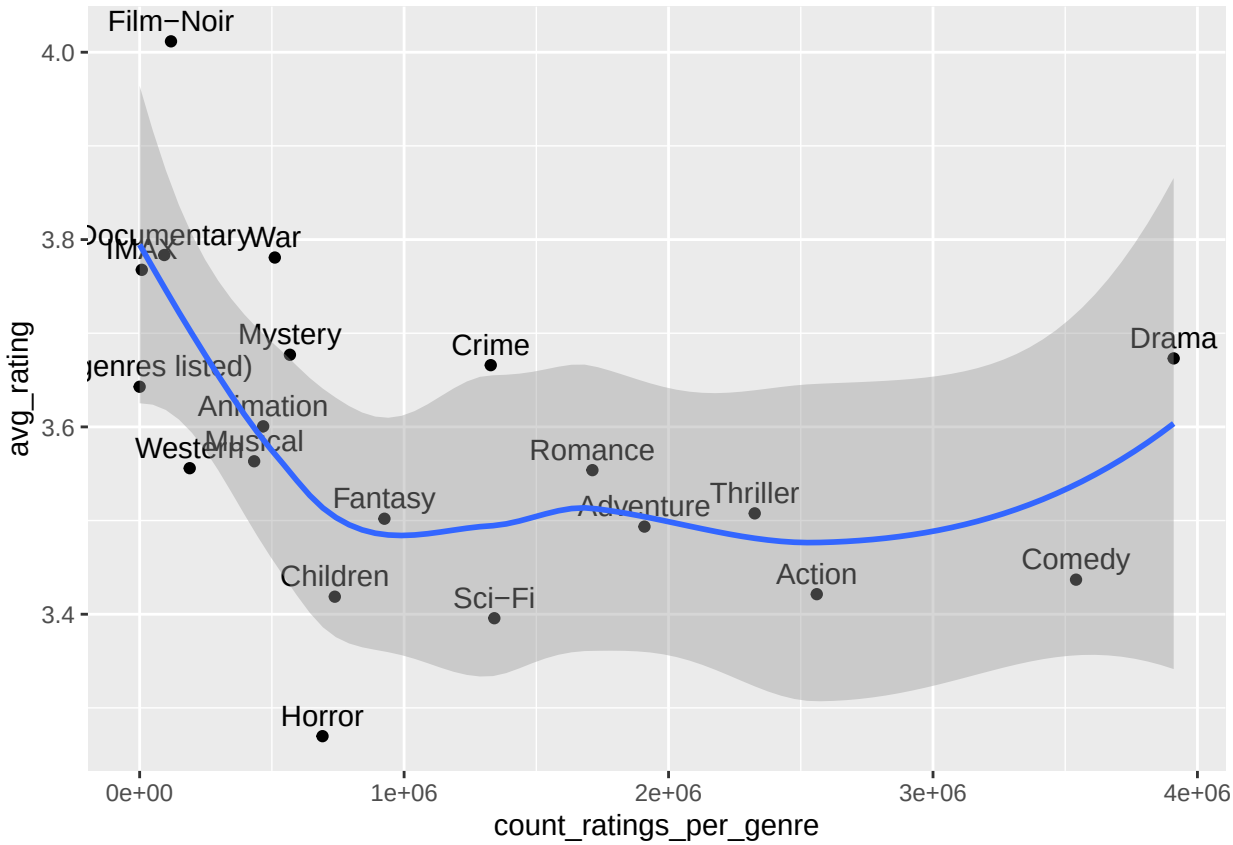
```
edx_split_genre <- edx %>%
    separate_rows(genres, sep = "\\|")

ratings_per_genre <- edx_split_genre %>%
  group_by(genres)%>%
  summarise(count_ratings_per_genre = n()) %>%
  arrange(desc(count_ratings_per_genre))
ratings_per_genre
```

```
## # A tibble: 20 x 2
##    genres             count_ratings_per_genre
##    <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
```

Now that we know some genres are more popular than others, we can explore if popular genres also receive higher ratings with the following code:

```
avg_rating_per_genre <- edx_split_genre %>%
    group_by(genres) %>%
    summarise(avg_rating = mean(rating)) %>%
    left_join(ratings_per_genre, by = "genres") %>%
    arrange(desc(avg_rating))

avg_rating_per_genre %>%
    ggplot(aes(count_ratings_per_genre, avg_rating, label = genres)) +
    geom_point() +
    geom_text(position = "nudge", vjust = -0.5) +
    geom_smooth()
```



The three most popular genres(Drama, Comedy and Action) received average ratings. The genre "Film-noir" has the highest average rating, but the number of ratings it received is quite small compared to other genres. The rating of genre "Horror" is much lower than the average.The smoothed line is around 3.5, which is the average of all ratings.

Year and ratings

To explore the connection of year and ratings, we need to create a "year" variable by splitting the year of release from "title":

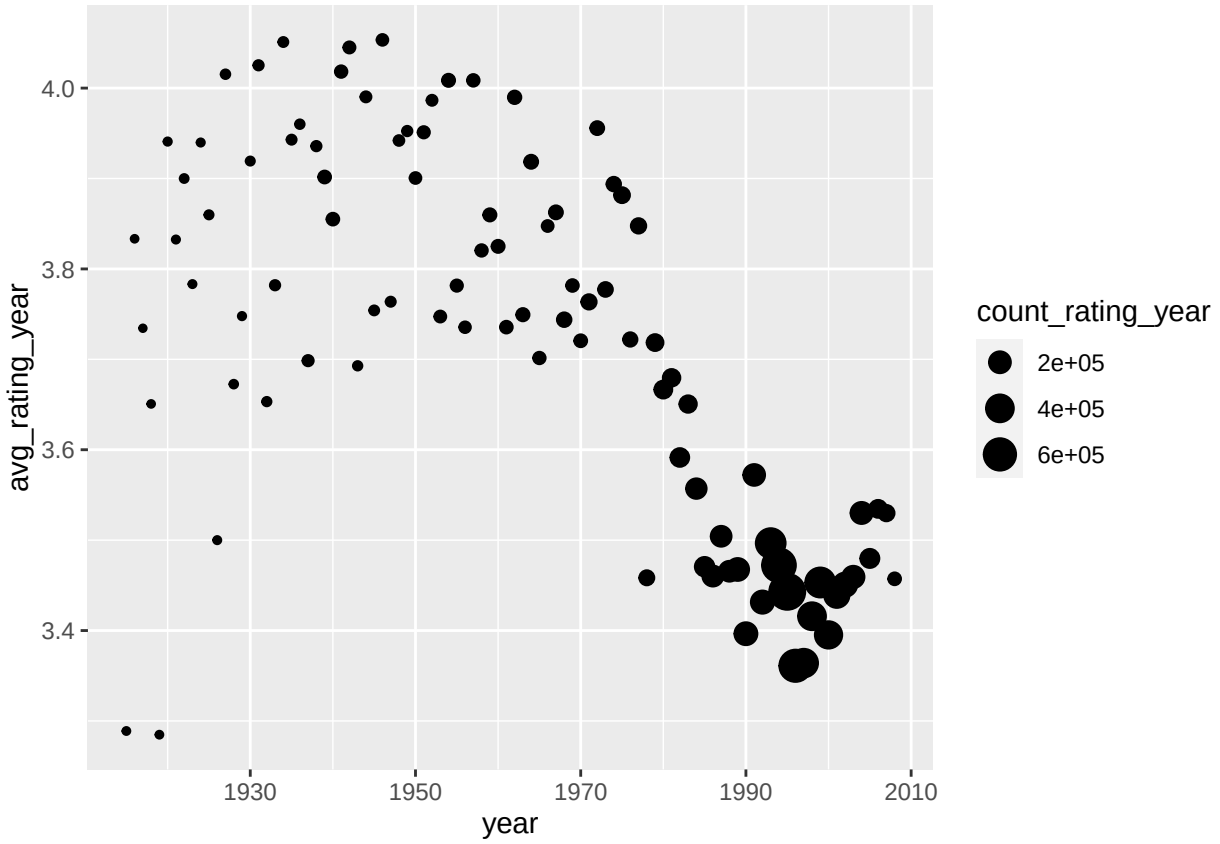
```
edx_split_year <- edx %>%
    mutate(year = as.numeric(str_sub(edx$title,-5, -2)))
```

With the following code, we calculate the number of ratings each year gets and the average rating each year gets.

```
edx_year_vs_rating <- edx_split_year %>%
    group_by(year) %>%
    mutate(count_rating_year = n(),
           avg_rating_year = mean(rating)) %>%
    select(year, count_rating_year, avg_rating_year) %>%
    unique()
```

Then we plot average rating of each year against year, and set size to represent the number of ratings of each year.

```
edx_year_vs_rating %>%
    ggplot(aes(year, avg_rating_year, size = count_rating_year)) +
    geom_point()
```



From the plot above, we can see that:

1. The movies before 1980 generally got lower number of ratings but higher average ratings.
2. Roughly between 1980 and 1995, movies got more ratings but the average of ratings were much lower than the years before the 1980.
3. Between 1995 and 2010, the number of ratings movies got were smaller than 1980-1995 but much bigger than before 1980. The average ratings showed an upward trend, but they were still lower than the years before 1980.

In short, the variable "year" might have an effect on the ratings of movie.

Model building

Generate training and test sets

The following code is used to generate the training set and test set.

```
library(caret)
set.seed(2, sample.kind = "Rounding")
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
test_set_temp <- edx[test_index, ]
train_set <- edx[-test_index, ]

# make sure userId and movieId in test set are also in train set
test_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_rows <- anti_join(test_set_temp, test_set)
train_set <- rbind(train_set, removed_rows)
```

Create the loss function

To compare different models, we use root mean squared error (RMSE) as the loss function.

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

The first model: baseline

Firstly, we try to predict the expected ratings with the average rating of all movies.

If we use "mu" to represent the true rating for all movies, then the estimate that minimizes the RMSE is the average of all ratings.

The equation of the first model can be written as follows:

y_hat = mu_hat

```
mu_hat <- mean(train_set$rating)
y_hat_1 <- mu_hat
rmse_1 <- RMSE(test_set$rating,y_hat_1)
rmse_results <- data.frame(Method = "Baseline(average of all ratings)", RMSE = rmse_1)
```

The RMSE of this model is 1.0602732.

```
##                             Method     RMSE
## 1 Baseline(average of all ratings) 1.060273
```

The second model: baseline + movie effect

While exploring data, we found that different movies are rated differently. We can improve the previous model by adding "movie effect"(b_i).

We'll minimize the following equation to obtain the least squares estimate for movie effect:

sum((true rating - mu_hat - b_i)^2)/N

Therefore, the movie effect can be calculated as:

b_i = mean(true rating - mu_hat)

The equation of the second model can be written as follows:

y_hat = mu_hat + b_i

```
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu_hat))

min(movie_bias$b_i)
```

```
## [1] -3.012444
```

```
max(movie_bias$b_i)
```

```
## [1] 1.487556
```

The highest value of b_i is 1.49 and the lowest value of b_i is -3.0, indicating that the highest rating is close to 5, and lowest is 0.5, which is in line with the data set.

With the following code, we calculate first the expected ratings of the test set, then the RMSE.

```
b_i_test_set <- test_set %>%
  left_join(movie_bias, by = "movieId") %>%
  pull(b_i)

y_hat_2 <- mu_hat + b_i_test_set

rmse_2 <- RMSE(test_set$rating, y_hat_2)
rmse_result_2 <- data.frame(Method = "Baseline + movie effect", RMSE = rmse_2)
```

The RMSE of this model is 0.9440821.

```
##                              Method      RMSE
## 1 Baseline(average of all ratings) 1.0602732
## 2          Baseline + movie effect 0.9440821
```

The third model: baseline + movie effect + user effect

Next step, we try to account for the variability across users by adding "user effect"(b_u).

User effect can be calculated as follows:

b_u = mean(true rating - mu_hat - b_i)

The equation of the third model can be written as follows:

y_hat = mu_hat + b_i + b_u

```
user_bias <- train_set %>%
  left_join(movie_bias, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu_hat - b_i))

b_u_test_set <- test_set %>%
  left_join(user_bias, by = "userId") %>%
  pull(b_u)

y_hat_3 <- mu_hat + b_i_test_set + b_u_test_set

rmse_3 <- RMSE(test_set$rating, y_hat_3)
rmse_result_3 <- data.frame(Method = "Baseline + movie effect + user effect", RMSE = rmse_3)
```

The RMSE of this model is 0.8669336. We improved a lot from the first model.

```
##                                   Method      RMSE
## 1      Baseline(average of all ratings) 1.0602732
## 2               Baseline + movie effect 0.9440821
## 3 Baseline + movie effect + user effect 0.8669336
```

The fourth model: Baseline + regularized movie effect + regularized user effect

In this model, we use regularized regression to control the total variability of the movie effect(b_i) and user effect(b_u). Specifically, instead of minimize the least squares:

sum((true rating - mu_hat - b_i - b_u)^2)/N

We add a penalty, and try to minimize the following equation:

sum((true rating - mu_hat - b_i - b_u)^2)/N + lambda*sum(b_i^2 + b_u^2)

The b_i and b_u that minimize the above equation can be calculated as:

b_i_reg = sum(true rating - mu_hat)/(lambda+N)

b_u_reg = sum(true rating - mu_hat - b_i_reg)/(lambda+N)

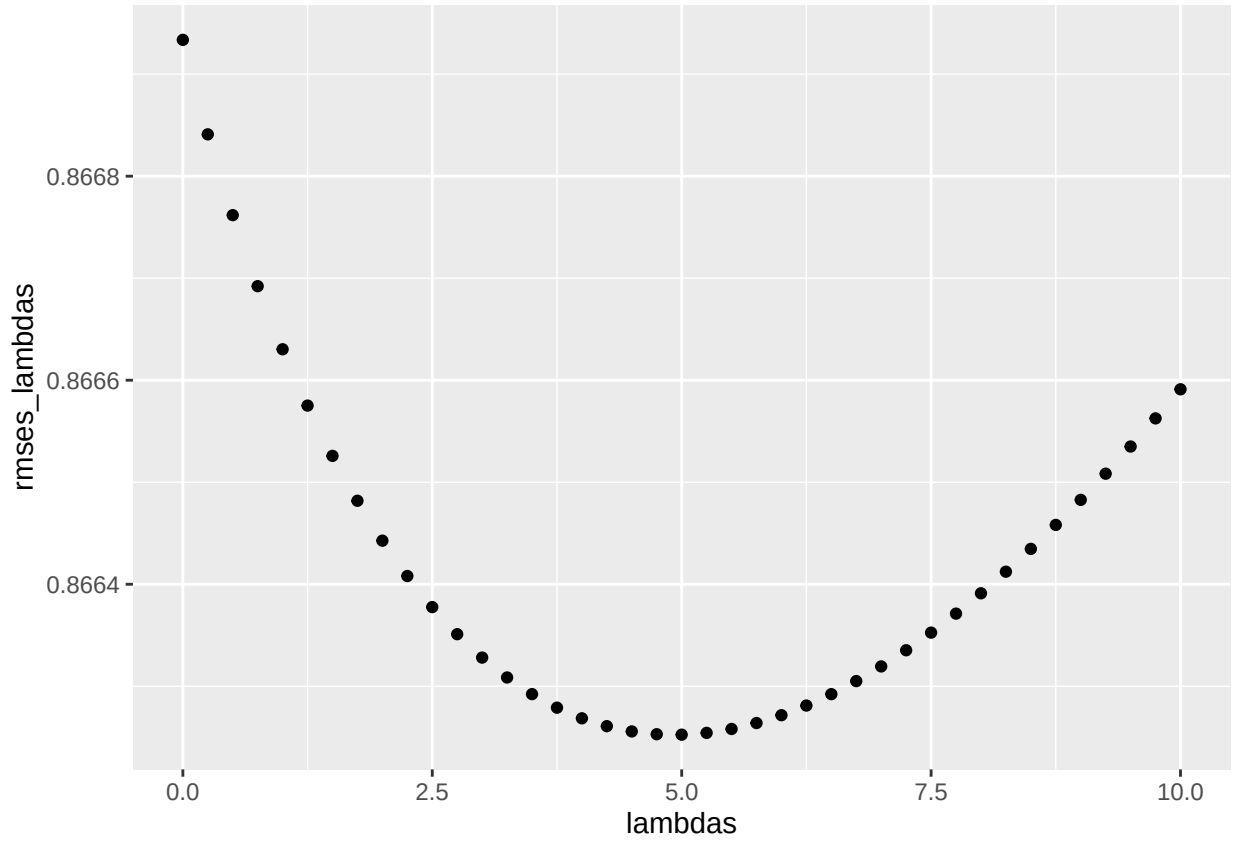
Note that N is the number of ratings for a certain movie. When N is small, lambda has a significant impact in shrinking the bias, when N is large, the lambda can be almost ignored.

The equation of the fourth model can be written as follows:

y_hat = mu_hat + b_i_reg + b_u_reg

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

rmses_lambdas <- sapply(lambdas, function(x){
    movie_bias_regularized <- train_set %>%
        group_by(movieId) %>%
        summarise(b_i_reg = sum(rating - mu_hat)/(n()+x))
    user_bias_regularized <- train_set %>%
        left_join(movie_bias_regularized, by = "movieId") %>%
        group_by(userId) %>%
        summarise(b_u_reg = sum(rating - mu_hat - b_i_reg)/(n()+x))
    y_hat_reg <- test_set %>%
```

```
        left_join(movie_bias_regularized, by = "movieId") %>%
        left_join(user_bias_regularized, by = "userId") %>%
        mutate(y_hat = mu_hat + b_i_reg + b_u_reg) %>%
        pull(y_hat)
    rmse_reg <- RMSE(test_set$rating, y_hat_reg)
    rmse_reg
})

qplot(lambdas,rmses_lambdas)
```



We use cross validation to get the best lambda, which is 5.

```
best_lambda <- lambdas[which.min(rmses_lambdas)]
best_lambda
```

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

```
rmse_4 <- rmses_lambdas[which.min(rmses_lambdas)]
rmse_result_4 <- data.frame(Method = "Baseline + regularized movie effect + regularized user effect", RM
```

The RMSE of this model is 0.8662526. We only got a tiny improvement from the previous model.

```
##                                                    Method      RMSE
## 1                          Baseline(average of all ratings) 1.0602732
```

```
## 2                                        Baseline + movie effect 0.9440821
## 3                           Baseline + movie effect + user effect 0.8669336
## 4 Baseline + regularized movie effect + regularized user effect 0.8662526
```

The fifth model: Baseline + movie effect + user effect + random forest(ranger)

In data exploration, we find that the year of release and genres of a movie may have an impact on the rating. Therefore, in this model, we'll try to use random forest to estimate the effect of "year" and "genre" on the residual of the ratings. Since random forest is too slow to run on a personal computer, we opt for a much faster implementation of random forest: ranger.

Calculate the residual of train set

First, we calculate the residual of train set by removing baseline, movie effect and user effect.

Then the residual can be calculated as follows:

residual = true rating - mu_hat - b_i - b_u

```
residual_train <- train_set %>%
    left_join(movie_bias, by = "movieId") %>%
    mutate(resi_1 = rating - b_i) %>%
    left_join(user_bias, by = "userId") %>%
    mutate(residual = resi_1 - b_u - mu_hat) %>%
    select(userId, movieId, rating, title, genres, residual)
```

Split "year" and "genre" of train set, and reshape train set from long to wide

In order to used the "ranger" package, we need to:

1. Split the year of release from "title" variable, and separate the "genres" variable into single genres.
2. Reshape the data set from long to wide.

```
residual_train_split <- residual_train %>%
  mutate(year = as.numeric(str_sub(title,-5, -2))) %>%
  separate_rows(genres, sep = "\\|") %>%
  mutate(genre_count = 1L) %>%
  pivot_wider(names_from = genres, values_from = genre_count, values_fill = 0L)
head(residual_train_split)
```

```
## # A tibble: 6 x 26
##   userId movieId rating title         residual  year Action Crime Thriller Drama
##    <int>   <dbl>  <dbl> <chr>           <dbl> <dbl>  <int> <int>    <int> <int>
## 1      1     185      5 Net, The (19~   0.171   1995      1     1        1     0
## 2      1     292      5 Outbreak (19~  -0.122   1995      1     0        1     1
## 3      1     316      5 Stargate (19~  -0.0548  1994      1     0        0     0
## 4      1     329      5 Star Trek: G~  -0.0367  1994      1     0        0     1
## 5      1     355      5 Flintstones,~   0.807   1994      0     0        0     0
## 6      1     362      5 Jungle Book,~  -0.151   1994      0     0        0     0
## # ... with 16 more variables: `Sci-Fi` <int>, Adventure <int>, Children <int>,
## #   Comedy <int>, Fantasy <int>, Romance <int>, Animation <int>, Musical <int>,
```

```
## #   War <int>, Mystery <int>, `Film-Noir` <int>, Western <int>, Horror <int>,
## #   Documentary <int>, IMAX <int>, `(no genres listed)` <int>
```

After the above operations, we make sure that the order of rows remains unchanged.

```
sum(residual_train_split$movieId != train_set$movieId)
```

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

```
sum(residual_train_split$userId != train_set$userId)
```

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

Format the train set for ranger

In this step, we prepare the data set for ranger package by:

1. removing userId, movieId, rating, title;
2. replacing minus signs in column names to underscore;
3. replacing "(no genres listed)" with "NoGenre".

We write a function with the following code:

```
format_ranger <- function(x){
    x <- x[,!(colnames(x) %in% c("userId", "movieId", "rating", "title"))]
    colnames(x) <- gsub("\\-", "", colnames(x))
    colnames(x) <- gsub("^[(]no\\sgenres\\slisted[)]$", "NoGenre", colnames(x))
    x
}
```

Then we format the "residual_train_split" data set into ranger format.

```
residual_train_split_ranger <- format_ranger(residual_train_split)
head(residual_train_split_ranger)
```

```
## # A tibble: 6 x 22
##   residual  year Action Crime Thriller Drama SciFi Adventure Children Comedy
##      <dbl> <dbl>  <int> <int>    <int> <int> <int>     <int>    <int>  <int>
## 1   0.171   1995      1     1        1     0     0         0        0      0
## 2  -0.122   1995      1     0        1     1     1         0        0      0
## 3  -0.0548  1994      1     0        0     0     1         1        0      0
## 4  -0.0367  1994      1     0        0     1     1         1        0      0
## 5   0.807   1994      0     0        0     0     0         0        1      1
## 6  -0.151   1994      0     0        0     0     0         1        1      0
## # ... with 12 more variables: Fantasy <int>, Romance <int>, Animation <int>,
## #   Musical <int>, War <int>, Mystery <int>, FilmNoir <int>, Western <int>,
## #   Horror <int>, Documentary <int>, IMAX <int>, NoGenre <int>
```

Train a model with ranger on train set

With the following code, we fit a model on the train set.

```
library(ranger)
fit_ranger <- ranger(residual ~ ., data = residual_train_split_ranger)
```

```
## Growing trees.. Progress: 0%. Estimated remaining time: 12 hours, 20 minutes, 11 seconds.
## Growing trees.. Progress: 2%. Estimated remaining time: 2 hours, 48 minutes, 12 seconds.
## Growing trees.. Progress: 3%. Estimated remaining time: 2 hours, 8 minutes, 48 seconds.
## Growing trees.. Progress: 5%. Estimated remaining time: 1 hour, 40 minutes, 29 seconds.
## Growing trees.. Progress: 5%. Estimated remaining time: 2 hours, 0 minutes, 1 seconds.
## Growing trees.. Progress: 6%. Estimated remaining time: 1 hour, 39 minutes, 56 seconds.
## Growing trees.. Progress: 7%. Estimated remaining time: 1 hour, 54 minutes, 9 seconds.
## Growing trees.. Progress: 8%. Estimated remaining time: 1 hour, 38 minutes, 42 seconds.
## Growing trees.. Progress: 8%. Estimated remaining time: 1 hour, 47 minutes, 50 seconds.
## Growing trees.. Progress: 10%. Estimated remaining time: 1 hour, 38 minutes, 52 seconds.
## Growing trees.. Progress: 10%. Estimated remaining time: 1 hour, 43 minutes, 23 seconds.
## Growing trees.. Progress: 11%. Estimated remaining time: 1 hour, 36 minutes, 12 seconds.
## Growing trees.. Progress: 11%. Estimated remaining time: 1 hour, 41 minutes, 2 seconds.
## Growing trees.. Progress: 13%. Estimated remaining time: 1 hour, 33 minutes, 53 seconds.
## Growing trees.. Progress: 13%. Estimated remaining time: 1 hour, 38 minutes, 2 seconds.
## Growing trees.. Progress: 14%. Estimated remaining time: 1 hour, 32 minutes, 2 seconds.
## Growing trees.. Progress: 15%. Estimated remaining time: 1 hour, 36 minutes, 1 seconds.
## Growing trees.. Progress: 16%. Estimated remaining time: 1 hour, 30 minutes, 18 seconds.
## Growing trees.. Progress: 16%. Estimated remaining time: 1 hour, 33 minutes, 42 seconds.
## Growing trees.. Progress: 18%. Estimated remaining time: 1 hour, 28 minutes, 47 seconds.
## Growing trees.. Progress: 18%. Estimated remaining time: 1 hour, 30 minutes, 7 seconds.
## Growing trees.. Progress: 19%. Estimated remaining time: 1 hour, 26 minutes, 40 seconds.
## Growing trees.. Progress: 19%. Estimated remaining time: 1 hour, 28 minutes, 17 seconds.
## Growing trees.. Progress: 21%. Estimated remaining time: 1 hour, 24 minutes, 31 seconds.
## Growing trees.. Progress: 21%. Estimated remaining time: 1 hour, 26 minutes, 12 seconds.
## Growing trees.. Progress: 22%. Estimated remaining time: 1 hour, 23 minutes, 1 seconds.
## Growing trees.. Progress: 23%. Estimated remaining time: 1 hour, 24 minutes, 32 seconds.
## Growing trees.. Progress: 24%. Estimated remaining time: 1 hour, 20 minutes, 45 seconds.
## Growing trees.. Progress: 24%. Estimated remaining time: 1 hour, 22 minutes, 38 seconds.
## Growing trees.. Progress: 25%. Estimated remaining time: 1 hour, 19 minutes, 3 seconds.
## Growing trees.. Progress: 26%. Estimated remaining time: 1 hour, 20 minutes, 28 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 1 hour, 17 minutes, 11 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 1 hour, 18 minutes, 36 seconds.
## Growing trees.. Progress: 28%. Estimated remaining time: 1 hour, 16 minutes, 8 seconds.
## Growing trees.. Progress: 29%. Estimated remaining time: 1 hour, 16 minutes, 32 seconds.
## Growing trees.. Progress: 30%. Estimated remaining time: 1 hour, 15 minutes, 13 seconds.
## Growing trees.. Progress: 31%. Estimated remaining time: 1 hour, 14 minutes, 50 seconds.
## Growing trees.. Progress: 31%. Estimated remaining time: 1 hour, 14 minutes, 1 seconds.
## Growing trees.. Progress: 32%. Estimated remaining time: 1 hour, 12 minutes, 29 seconds.
## Growing trees.. Progress: 32%. Estimated remaining time: 1 hour, 12 minutes, 40 seconds.
## Growing trees.. Progress: 33%. Estimated remaining time: 1 hour, 10 minutes, 57 seconds.
## Growing trees.. Progress: 34%. Estimated remaining time: 1 hour, 10 minutes, 39 seconds.
## Growing trees.. Progress: 35%. Estimated remaining time: 1 hour, 8 minutes, 57 seconds.
## Growing trees.. Progress: 35%. Estimated remaining time: 1 hour, 8 minutes, 42 seconds.
## Growing trees.. Progress: 36%. Estimated remaining time: 1 hour, 8 minutes, 16 seconds.
## Growing trees.. Progress: 37%. Estimated remaining time: 1 hour, 6 minutes, 49 seconds.
## Growing trees.. Progress: 37%. Estimated remaining time: 1 hour, 6 minutes, 37 seconds.
```

```
## Growing trees.. Progress: 38%. Estimated remaining time: 1 hour, 4 minutes, 53 seconds.
## Growing trees.. Progress: 39%. Estimated remaining time: 1 hour, 5 minutes, 11 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 1 hour, 3 minutes, 30 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 1 hour, 3 minutes, 12 seconds.
## Growing trees.. Progress: 41%. Estimated remaining time: 1 hour, 1 minute, 57 seconds.
## Growing trees.. Progress: 42%. Estimated remaining time: 1 hour, 1 minute, 21 seconds.
## Growing trees.. Progress: 42%. Estimated remaining time: 1 hour, 1 minute, 13 seconds.
## Growing trees.. Progress: 43%. Estimated remaining time: 59 minutes, 58 seconds.
## Growing trees.. Progress: 44%. Estimated remaining time: 59 minutes, 43 seconds.
## Growing trees.. Progress: 44%. Estimated remaining time: 58 minutes, 28 seconds.
## Growing trees.. Progress: 45%. Estimated remaining time: 57 minutes, 46 seconds.
## Growing trees.. Progress: 45%. Estimated remaining time: 58 minutes, 9 seconds.
## Growing trees.. Progress: 46%. Estimated remaining time: 56 minutes, 33 seconds.
## Growing trees.. Progress: 47%. Estimated remaining time: 56 minutes, 16 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 55 minutes, 10 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 54 minutes, 34 seconds.
## Growing trees.. Progress: 49%. Estimated remaining time: 53 minutes, 48 seconds.
## Growing trees.. Progress: 50%. Estimated remaining time: 52 minutes, 54 seconds.
## Growing trees.. Progress: 51%. Estimated remaining time: 51 minutes, 53 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 51 minutes, 15 seconds.
## Growing trees.. Progress: 53%. Estimated remaining time: 49 minutes, 58 seconds.
## Growing trees.. Progress: 53%. Estimated remaining time: 49 minutes, 32 seconds.
## Growing trees.. Progress: 54%. Estimated remaining time: 48 minutes, 24 seconds.
## Growing trees.. Progress: 55%. Estimated remaining time: 47 minutes, 52 seconds.
## Growing trees.. Progress: 56%. Estimated remaining time: 46 minutes, 34 seconds.
## Growing trees.. Progress: 56%. Estimated remaining time: 46 minutes, 9 seconds.
## Growing trees.. Progress: 57%. Estimated remaining time: 44 minutes, 48 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 44 minutes, 27 seconds.
## Growing trees.. Progress: 59%. Estimated remaining time: 43 minutes, 24 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 42 minutes, 45 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 41 minutes, 43 seconds.
## Growing trees.. Progress: 61%. Estimated remaining time: 41 minutes, 1 seconds.
## Growing trees.. Progress: 62%. Estimated remaining time: 39 minutes, 44 seconds.
## Growing trees.. Progress: 63%. Estimated remaining time: 39 minutes, 10 seconds.
## Growing trees.. Progress: 64%. Estimated remaining time: 38 minutes, 13 seconds.
## Growing trees.. Progress: 64%. Estimated remaining time: 37 minutes, 53 seconds.
## Growing trees.. Progress: 65%. Estimated remaining time: 37 minutes, 21 seconds.
## Growing trees.. Progress: 65%. Estimated remaining time: 36 minutes, 23 seconds.
## Growing trees.. Progress: 66%. Estimated remaining time: 35 minutes, 43 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 35 minutes, 3 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 34 minutes, 41 seconds.
## Growing trees.. Progress: 68%. Estimated remaining time: 34 minutes, 5 seconds.
## Growing trees.. Progress: 68%. Estimated remaining time: 33 minutes, 13 seconds.
## Growing trees.. Progress: 69%. Estimated remaining time: 32 minutes, 52 seconds.
## Growing trees.. Progress: 70%. Estimated remaining time: 32 minutes, 0 seconds.
## Growing trees.. Progress: 70%. Estimated remaining time: 31 minutes, 12 seconds.
## Growing trees.. Progress: 71%. Estimated remaining time: 30 minutes, 16 seconds.
## Growing trees.. Progress: 72%. Estimated remaining time: 29 minutes, 31 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 28 minutes, 37 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 27 minutes, 52 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 26 minutes, 55 seconds.
## Growing trees.. Progress: 75%. Estimated remaining time: 26 minutes, 9 seconds.
## Growing trees.. Progress: 76%. Estimated remaining time: 25 minutes, 13 seconds.
## Growing trees.. Progress: 77%. Estimated remaining time: 24 minutes, 34 seconds.
```

```
## Growing trees.. Progress: 77%. Estimated remaining time: 24 minutes, 15 seconds.
## Growing trees.. Progress: 78%. Estimated remaining time: 22 minutes, 51 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 22 minutes, 31 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 21 minutes, 52 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 21 minutes, 13 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 20 minutes, 51 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 19 minutes, 41 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 19 minutes, 33 seconds.
## Growing trees.. Progress: 82%. Estimated remaining time: 18 minutes, 55 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 18 minutes, 1 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 17 minutes, 41 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 17 minutes, 3 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 16 minutes, 9 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 15 minutes, 47 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 15 minutes, 23 seconds.
## Growing trees.. Progress: 86%. Estimated remaining time: 14 minutes, 18 seconds.
## Growing trees.. Progress: 87%. Estimated remaining time: 13 minutes, 40 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 12 minutes, 37 seconds.
## Growing trees.. Progress: 89%. Estimated remaining time: 11 minutes, 59 seconds.
## Growing trees.. Progress: 90%. Estimated remaining time: 10 minutes, 56 seconds.
## Growing trees.. Progress: 91%. Estimated remaining time: 9 minutes, 50 seconds.
## Growing trees.. Progress: 91%. Estimated remaining time: 9 minutes, 14 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 8 minutes, 36 seconds.
## Growing trees.. Progress: 93%. Estimated remaining time: 7 minutes, 33 seconds.
## Growing trees.. Progress: 93%. Estimated remaining time: 6 minutes, 55 seconds.
## Growing trees.. Progress: 94%. Estimated remaining time: 5 minutes, 51 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 5 minutes, 14 seconds.
## Growing trees.. Progress: 96%. Estimated remaining time: 4 minutes, 11 seconds.
## Growing trees.. Progress: 97%. Estimated remaining time: 3 minutes, 33 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 2 minutes, 30 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 1 minute, 53 seconds.
## Growing trees.. Progress: 99%. Estimated remaining time: 50 seconds.
## Growing trees.. Progress: 100%. Estimated remaining time: 25 seconds.
## Growing trees.. Progress: 100%. Estimated remaining time: 12 seconds.
## Computing prediction error.. Progress: 58%. Estimated remaining time: 22 seconds.
```

Format the test set

We prepare the test set into the ranger format:

```
test_split <- test_set %>%
    mutate(year = as.numeric(str_sub(title, -5, -2))) %>%
    separate_rows(genres, sep = "\\|") %>%
    mutate(genre_count = c(1)) %>%
    pivot_wider(names_from = genres, values_from = genre_count, values_fill = 0)

sum(test_split$movieId != test_set$movieId)
```

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

```
sum(test_split$userId != test_set$userId)
```

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

```
test_split_ranger <- test_split %>% format_ranger()
```

Prediction

With the fitted model, we predict the residual of test set and calculate the RMSE.

```
pred_ranger <-predict(fit_ranger, data = test_split_ranger)
y_hat_residual <- pred_ranger$predictions

y_hat_ranger <- test_split %>%
    left_join(movie_bias, by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    mutate(y_hat = mu_hat + b_i + b_u + y_hat_residual) %>%
    pull(y_hat)

rmse_ranger <- RMSE(test_set$rating, y_hat_ranger)
rmse_ranger
```

```
## [1] 0.8663254
```

```
rmse_result_ranger <- data.frame(Method = "Baseline + movie effect + user effect + random forest (range
```

The RMSE of this model is 0.8663254, which is slightly worse than the previous model, and still doesn't meet our goal(RMSE<0.86490).

```
##                                                                Method      RMSE
## 1                                 Baseline(average of all ratings) 1.0602732
## 2                                           Baseline + movie effect 0.9440821
## 3                               Baseline + movie effect + user effect 0.8669336
## 4  Baseline + regularized movie effect + regularized user effect 0.8662526
## 5 Baseline + movie effect + user effect + random forest (ranger) 0.8663254
```

The sixth model: matrix factorization(recosystem)

Our previous models lowered the RMSE to around 0.866, still higher than the goal(RMSE $<$ 0.86490). We need to find a better model for this task.

The data set of this project is quite unique: the data is very "loose" as there are a lot of missing values. However, we know that similar users might have similar tastes, and similar movies might get similar ratings. Thus the missing values can be predicted using the actual ratings in the data set. A popular method to solve this problem is the matrix factorization model.

The package we choose to use for this project is the recosystem package, a wrapper of the "libmf" library for recommender system using matrix factorization. For details of this package, please click here.

Format train set and test set

The recosystem package requires the data to be in sparse matrix triplet form. With the following code, we convert both the train set and test set into the recosystem format.

```
library(recosystem)
train_fac <- with(train_set, data_memory(user_index = userId,
                                         item_index = movieId,
                                         rating = rating))


test_fac <- with(test_set, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating = rating))
```

Create a model object and tune the parameters

```
#create a model object by calling Reco()
set.seed(345, sample.kind = "Rounding")
r <- Reco()
#tuning the parameters
opts <- r$tune(train_fac, opt = list(dim = c(10L, 20L),
                                     lrate = c(0.01, 0.1),
                                     costp_l2 = c(0.01, 0.1),
                                     costq_l2 = c(0.01, 0.1),
                                     nthread = 4,
                                     niter = 40))
```

Train the model

Train the model using the best tuning parameters from the previous step.

```
r$train(train_fac, opts = c(opts$min, nthread = 4, niter = 40))
```

```
## iter      tr_rmse          obj
##    0       0.9857   9.8960e+06
##    1       0.8791   8.0699e+06
##    2       0.8494   7.5279e+06
##    3       0.8297   7.1960e+06
##    4       0.8146   6.9598e+06
##    5       0.8032   6.7909e+06
##    6       0.7940   6.6633e+06
##    7       0.7865   6.5625e+06
##    8       0.7798   6.4774e+06
##    9       0.7740   6.4056e+06
##   10       0.7689   6.3450e+06
##   11       0.7645   6.2927e+06
##   12       0.7605   6.2482e+06
##   13       0.7571   6.2096e+06
##   14       0.7540   6.1761e+06
##   15       0.7512   6.1452e+06
##   16       0.7486   6.1177e+06
##   17       0.7463   6.0939e+06
##   18       0.7443   6.0732e+06
##   19       0.7423   6.0527e+06
```

```
##   20       0.7405   6.0350e+06
##   21       0.7389   6.0176e+06
##   22       0.7374   6.0033e+06
##   23       0.7360   5.9891e+06
##   24       0.7347   5.9776e+06
##   25       0.7334   5.9654e+06
##   26       0.7324   5.9555e+06
##   27       0.7313   5.9444e+06
##   28       0.7303   5.9349e+06
##   29       0.7294   5.9269e+06
##   30       0.7285   5.9183e+06
##   31       0.7277   5.9109e+06
##   32       0.7269   5.9045e+06
##   33       0.7261   5.8961e+06
##   34       0.7255   5.8906e+06
##   35       0.7248   5.8848e+06
##   36       0.7241   5.8799e+06
##   37       0.7235   5.8731e+06
##   38       0.7230   5.8708e+06
##   39       0.7224   5.8653e+06
```

Prediction

We compute the predicted values and store the prediction to a R vector, then calculate the RMSE.

```
y_hat_fac <- r$predict(test_fac, out_memory())
head(y_hat_fac)
```

```
## [1] 4.335582 5.424050 5.033813 4.720483 4.478789 2.816613
```

```
rmse_fac <- RMSE(test_set$rating, y_hat_fac)
rmse_result_fac <- data.frame(Method = "Matrix factorization (recosystem)", RMSE = rmse_fac)
```

```
##                                                      Method      RMSE
## 1                              Baseline(average of all ratings) 1.0602732
## 2                                       Baseline + movie effect 0.9440821
## 3                            Baseline + movie effect + user effect 0.8669336
## 4  Baseline + regularized movie effect + regularized user effect 0.8662526
## 5 Baseline + movie effect + user effect + random forest (ranger) 0.8663254
## 6                                Matrix factorization (recosystem) 0.7937016
```

The RMSE of this model is 0.7937016. We made significant improvement from all previous models.

Results and final validation

The following six models are tested for this project:

```
##                                                      Method      RMSE
## 1                              Baseline(average of all ratings) 1.0602732
## 2                                       Baseline + movie effect 0.9440821
```

```
## 3                           Baseline + movie effect + user effect 0.8669336
## 4  Baseline + regularized movie effect + regularized user effect 0.8662526
## 5 Baseline + movie effect + user effect + random forest (ranger) 0.8663254
## 6                              Matrix factorization (recosystem) 0.7937016
```

The best performing model is the last one: Matrix factorization using recosystem package. We'll use this model for our final validation.

Final Validation

First, we'll train it with edx data set:

```
##transform the edx and validation data sets to sparse matrix triplet form.
edx_fac <- with(edx, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))
validation_fac <- with(validation, data_memory(user_index = userId,
                                               item_index = movieId,
                                               rating = rating))

##create a model object by calling Reco()
set.seed(3456, sample.kind = "Rounding")
r_edx <- Reco()

##tune the parameters
opts_edx <- r$tune(edx_fac, opt = list(dim = c(10L, 20L),
                                        lrate = c(0.01, 0.1),
                                        costp_l2 = c(0.01, 0.1),
                                        costq_l2 = c(0.01, 0.1),
                                        nthread = 6,
                                        niter = 40))

## train the model by calling the $train() method
r_edx$train(edx_fac, opts = c(opts_edx$min, nthread = 6, niter = 40))
```

```
## iter      tr_rmse          obj
##    0       0.9648   1.1839e+07
##    1       0.8726   9.8718e+06
##    2       0.8398   9.1828e+06
##    3       0.8189   8.7736e+06
##    4       0.8052   8.5092e+06
##    5       0.7954   8.3286e+06
##    6       0.7878   8.1938e+06
##    7       0.7814   8.0893e+06
##    8       0.7758   8.0005e+06
##    9       0.7712   7.9309e+06
##   10       0.7669   7.8677e+06
##   11       0.7633   7.8164e+06
##   12       0.7601   7.7709e+06
##   13       0.7574   7.7335e+06
##   14       0.7548   7.6992e+06
##   15       0.7525   7.6693e+06
```

```
##   16       0.7504   7.6414e+06
##   17       0.7486   7.6200e+06
##   18       0.7468   7.5965e+06
##   19       0.7453   7.5795e+06
##   20       0.7438   7.5596e+06
##   21       0.7424   7.5433e+06
##   22       0.7411   7.5271e+06
##   23       0.7400   7.5156e+06
##   24       0.7390   7.5040e+06
##   25       0.7380   7.4923e+06
##   26       0.7371   7.4836e+06
##   27       0.7362   7.4726e+06
##   28       0.7354   7.4626e+06
##   29       0.7346   7.4537e+06
##   30       0.7339   7.4466e+06
##   31       0.7333   7.4394e+06
##   32       0.7326   7.4335e+06
##   33       0.7320   7.4263e+06
##   34       0.7314   7.4197e+06
##   35       0.7308   7.4139e+06
##   36       0.7303   7.4092e+06
##   37       0.7298   7.4033e+06
##   38       0.7293   7.3987e+06
##   39       0.7289   7.3942e+06
```

Then, we calculate the final RMSE on the validation data set.

```
##use the $predict() method to compute predicted values
##and generate prediction to a R vector
y_hat_edx_fac <- r_edx$predict(validation_fac, out_memory())

##calculate the final RMSE
rmse_final <- RMSE(validation$rating, y_hat_edx_fac)
rmse_result_final <- data.frame(Method = "Matrix Factorization (Final Validation)", RMSE = rmse_final)
```

The final RMSE is 0.7861099.

```
##                                     Method      RMSE
## 1 Matrix Factorization (Final Validation) 0.7861099
```

Conclusion

For this project, we created six models to build a recommendation system. The best model (matrix factorization) achieved a RMSE of 0.7861099, well below the target RMSE 0.86490.

These models we used have their limitations. For instance, after we accounted for the movie and user bias, we used ranger to make a prediction on the "residual" based on "year of release" and "genre". This only gave us a tiny improvement on the prediction. Due to limited time and resources(an old personal laptop), we didn't continue testing the effect of other variables on the residual, for example, the "timestamp" that indicates the time when ratings were made.

With more time and better hardware, a few more approaches could be tested out:

- Text analysis.
  - We could extract words from the "title" variable, run sentiment analysis of them and look for correlation between certain sentiments and the ratings.
  - We could check if there's a connection between the length of the titles and the ratings.
- Tuning parameters. Especially with the ranger and recosystem models, we could spend more time tuning the parameters to find the best options that lowers the RMSE even more.