# Movielens Project Report

Yuanhang Zhang (Charles)

2024-08-10

## Introduction

Now we are in the time of streaming media. Different from the traditional way to see a movie, where we decide what we want to see and then head directly to it, streaming platform now pushes personalized contents to users to catch their eyes. To do that, the platform must build an algorithm to predict how an individual would think about new content.

That question leads us here, the Movielens Project. In this project, we aim to predict the rate users would give to movies (stored in `final_holdout_test` data set) based on the existing rating records on the platform (stored in `edx` data set). Once we are able to do that, in the real life, we can then recommend the movies to the persons who will rate them high.

In this report, we will follow the following steps to build a viable predicting algorithm:

1. Download and review the data we will use
2. Generate training set and test set from the known data
3. Design multiple models (here, we apply linear models)
4. Compare the performance of different models on the test set and select the optimal one
5. Apply the optimal model to the `final_holdout_test` and get the result of accuracy (measured by RMSE)

In the following parts, we will go through all these steps to generate and test our algorithm. Along with this report, there will be an R Script file named `Algorithm.R` to extract only our algorithm for prediction without detailed explanation and discover process.

*Here is the packages needed to run the code in this report. This chunk will automatically download and install the packages needed if not existent in the environment:*

## Data

### Downloading data

The project is built on the data set from the website of edx. Here is the code to get the necessary data `edx` and `final_holdout_test`:

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# Note: this process could take a couple of minutes

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

library(tidyverse)
```

```r
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
```

```
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

## Descriptive statistics

Now we get 2 data sets: `edx` and `final_holdout_test`. Here, we will build the model on the given data set: `edx` and test out the final algorithm on the `final_holdout_test`.

The size of these two data frames can be derived as follows:

```
sapply(list(edx, final_holdout_test), function(df){
  nrow <- nrow(df)
  ncol <- ncol(df)
  tibble("nrow" = nrow, "ncol" = ncol)
})
```

```
##      [,1]    [,2]
## nrow 9000055 999999
## ncol 6       6
```

The training set has 9000055 rows and the test set has 999999 rows, with 6 columns in both sets.

And this is the information stored in the data sets:

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

We can know the classes of the indicators using this code:

```
sapply(edx, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "integer"   "numeric"   "integer" "character" "character"
```

From above, we know that the `userId`, `movieId` are integers, `rating` is numeric, `timestamp` is date-time data in the format of integer, and the `title` and `genres` are characters.

The summary of the indicators is listed here:

```
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
##
##
```

##

Combined with the introduction of the data, we can summarize these indicators:

- `userId`: Integer. Used to identify each user on the platform.
- `movieId`: Integer. Used to identify each movie on the platform. A `userId` with a `movieId` together compose a unique rating record.
- `rating`: Numeric. Rating given by a user from 0.5 to 5.
- `timestamp`: Integer. Stores the time when the rate was given.
- `title`: Character. Stores the title of the movies.
- `genres`: Character. Stores the genres a movie belongs to.

We will just build our model based on the information stored in these columns.

### Test set and training set

To further train our data and tune the parameters, we also need to partition our data into traning set and test set. We can use this code to extract **10%** of the data set as the test set:

```
set.seed(1) # To ensure the code and results are reproducible
test_indices <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)

test <- edx[test_indices,]
train <- edx[-test_indices,]
```

Here is the dimension of the generated data:

```
sapply(list(train, test), function(df){
  nrow <- nrow(df)
  ncol <- ncol(df)
  tibble("nrow" = nrow, "ncol" = ncol)
})
```

```
##      [,1]    [,2]
## nrow 8100048 900007
## ncol 6       6
```

## Methods

Since the data set for the Movielens project is too large, most machine learning algorithms and direct linear regression methods may consume too large amount of time, exceeding the limit of the PC processor. So here we manually construct linear regression models to capture the characteristics of the data.

### Average rating

We first simply assume that the all ratings are the same across the data set. Mathematically we can interpret this simple model in this way:

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

for each movie i and each user u.

We name the average of all ratings `mu`, and calculate that using the training set:

```r
mu <- train %>% summarise(mu = mean(rating)) %>% #Calculating the mean ratings
  select(mu) #extract the mu column

mu
```

```
##         mu
## 1 3.512457
```

For convenience, we turn `mu` into a numeric value:

```r
 mu <- mu[1,1]
 mu
```

```
## [1] 3.512457
```

Next, we will add the `mu` column in to our training set and test set. Please kindly note that any operation we make to the test set (including `test` and the set used for final prediction `final_holdout_test` ) will **not cause data leakage**, because what we will be doing is just put the parameters based on the `train` set into the test set for further prediction. No pre-processing and modeling step uses the data from the test set.

Also, to make sure the original data set is not polluted, we here use the `train_n` , `test_n` and `final_holdout_test_n` to stand for the data sets that's been processed for prediction.

```r
train_n <- train %>% mutate(mu = mu)
test_n <- test %>% mutate(mu = mu)
final_holdout_test_n <- final_holdout_test %>% mutate(mu = mu)
```

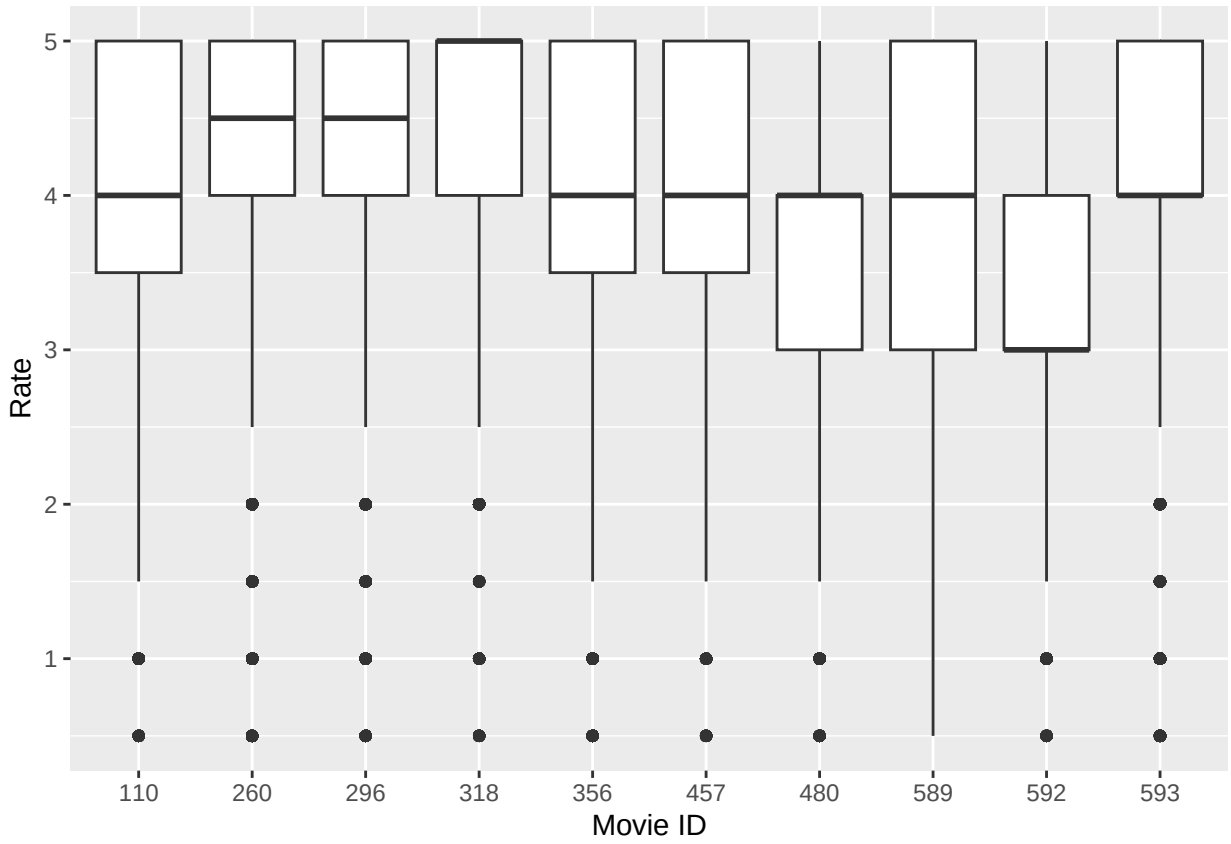
## Movie effect

We know that the ratings for different movies should be different. We can confirm this using the box plot. Here we select the 10 most rated movies to derive the plot:

```r
# Select the top 10 most rated movies as example
top_10_mv_nrat <- train %>% group_by(movieId) %>%
  summarise(ave_rat_mv = mean(rating), n_rat_mv = n()) %>%
  arrange(desc(n_rat_mv)) %>%
  top_n(10) %>%
  .$movieId
```

```
## Selecting by n_rat_mv
```

```r
# Draw the box plot of the selected movies to see the different distribution of the movies.
train %>% filter(movieId %in% top_10_mv_nrat) %>%
  ggplot() +
  geom_boxplot(aes(x = as.character(movieId), y = rating, group = movieId)) +
  labs(
    x = "Movie ID",
    y = "Rate"
  )
```




So, we include the movie effect into the model, which turns our formula into:

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

$b_i$ stands for the deviation of each movie from the average rating level, representing the movie effect. From the formula above, we can derive that $\hat{b_i} = Y_{u,i} - \hat{\mu}$. To calculate $b_i$, we must first group our training data by `movieId` and calculate the mean of this deviation.

Here we first calculate the `b_i`:

```
b_i <- train %>% mutate(mu = mu) %>%
  mutate(b_i = rating - mu) %>%
  group_by(movieId) %>% #Calculating b_i at the aggregation of movie level
  summarise(b_i = mean(b_i))
```

We can then merge the information into the training set and test set:

```
train_n <- train_n %>%
  left_join(b_i, by = "movieId")
```

Since we calculated our `b_i` based on the data in the training set, it is probable that some `movieId` in the test set is not included in the training set. So after merging, there might be `NA` in the column. We fill the `NA`s with 0.

```
test_n <- test_n %>%
  left_join(b_i, by = "movieId") %>% #merge the data set to put b_i in the predictors
  mutate(b_i = ifelse(is.na(b_i), 0, b_i)) #fill the NA with 0

final_holdout_test_n <- final_holdout_test_n %>%
```

```
  left_join(b_i, by = "movieId") %>%
  mutate(b_i = ifelse(is.na(b_i), 0, b_i)) #fill the NA with 0
```

### User effect

We also expect the users would rate the movies they saw differently. The following box plot of top 10 users who rate most can confirm this finding:

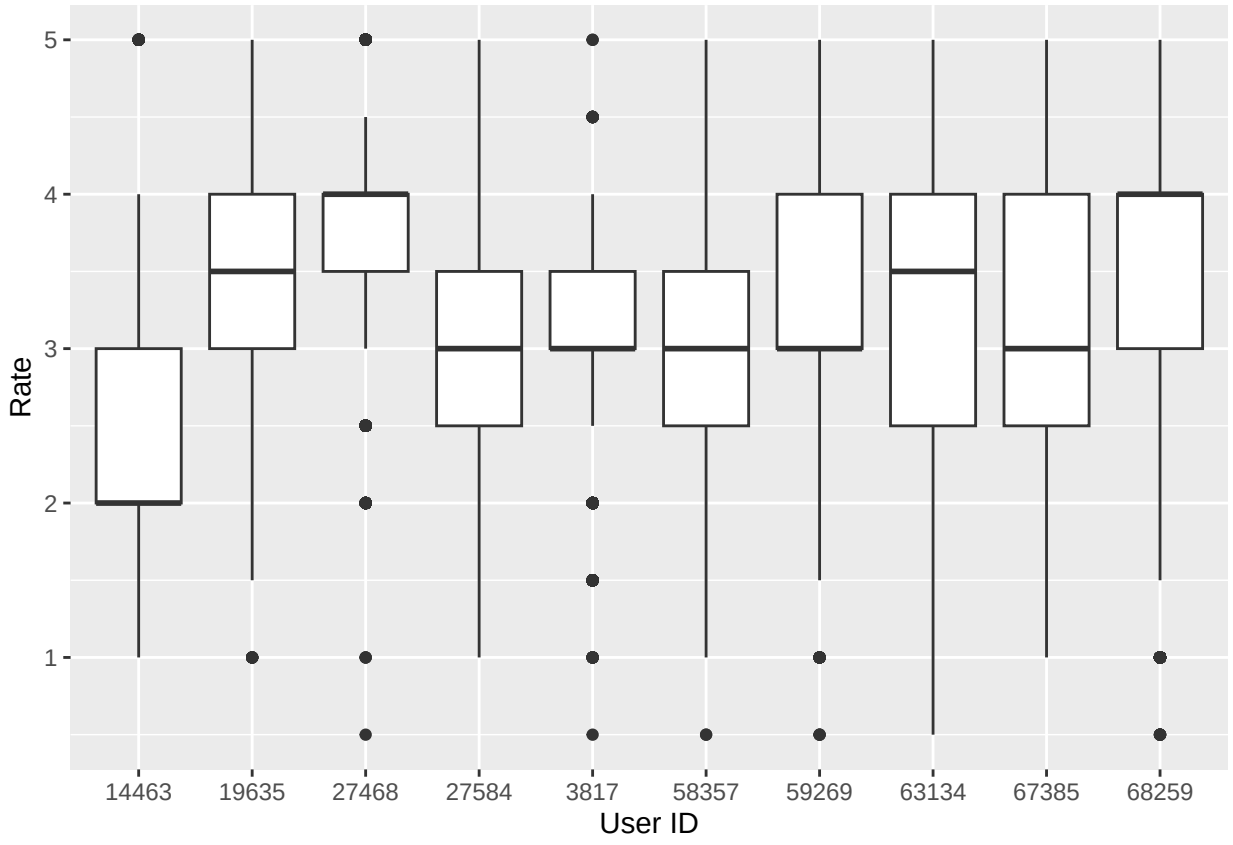
```
# Select the top 10 users who rate the most as example

top_10_usr_nrat <- train %>% group_by(userId) %>%
  summarise(ave_rat_mv = mean(rating), n_rat_mv = n()) %>%
  arrange(desc(n_rat_mv)) %>%
  top_n(10) %>%
  .$userId
```

```
## Selecting by n_rat_mv
```

```
# Draw the box plot of the selected users to see the different distribution of the movies

train %>% filter(userId %in% top_10_usr_nrat) %>%
  ggplot(aes(x = as.character(userId), y = rating, group = userId)) +
  geom_boxplot() +
  labs(
    x = "User ID",
    y = "Rate"
  )
```



Like what we found in the movie effect, the rating distributions of these reviewers differ. So it will be necessary to include this effect into the model.

The formula would turn into:

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

Now, we can calculate the `b_u` . Note that $\hat{b_i} = Y_{u,i} - \hat{\mu} - \hat{b_u}$, so the `b_u` is calculated as the average of `rating` minus `b_i` for every user. Here is the code to calculate it:

```
b_u <- train_n %>%
  mutate(b_u = rating - mu - b_i) %>%
  group_by(userId) %>% #Calclulating b_u at the aggregation of movie level
  summarise(b_u = mean(b_u))
```

Then, `b_u` is ready to be merged in the training set and test set:

```
train_n <- train_n %>%
  left_join(b_u, by = "userId")
```

Just like movie effect, there also might be NA in the test set when we merge our `b_u` . So additional NA-filling is necessary.

```
# Merge user effect into the test sets

test_n <- test_n %>%
  left_join(b_u, by = "userId") %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u))

final_holdout_test_n <- final_holdout_test_n %>%
  left_join(b_u, by = "userId") %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u))
```

## Age effect

Besides the information mentioned above, we notice that there is release year information in the `title` column. That makes us think what can the birth year of a film mean. We can tell whether a movie is old or late from its birth year, and that, its age, is always an important criteria when we think of a movie.

Start from here, we can tentatively calculate the age of a movie and see whether it has impact on its viewer's opinion. Note we define `age` as the distance from a movie's birth to the viewer's rating. So we first need to extract the year from the movie's title:

```
# Extract birth year from the title

mv_time <- train %>% group_by(movieId, title) %>% # group by movie to extract birth year
  summarise(ave_rat = mean(rating)) %>% # make sure the grouping is executed
  mutate(rls_yr = str_extract(title, "\\((\\d{4})\\)$", group = 1)) %>%
  # extract the continuous 4 numerical value (standing for the year) from the title
  mutate(rls_yr = as.numeric(rls_yr)) # transform the year into numeric
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
mv_time
```

```
## # A tibble: 10,661 x 4
## # Groups:   movieId [10,661]
```

```
##    movieId title                                ave_rat rls_yr
##      <int> <chr>                                  <dbl>  <dbl>
##  1       1 Toy Story (1995)                        3.93   1995
##  2       2 Jumanji (1995)                          3.21   1995
##  3       3 Grumpier Old Men (1995)                 3.15   1995
##  4       4 Waiting to Exhale (1995)                2.88   1995
##  5       5 Father of the Bride Part II (1995)      3.07   1995
##  6       6 Heat (1995)                             3.81   1995
##  7       7 Sabrina (1995)                          3.36   1995
##  8       8 Tom and Huck (1995)                     3.12   1995
##  9       9 Sudden Death (1995)                     3.00   1995
## 10      10 GoldenEye (1995)                        3.43   1995
## # i 10,651 more rows
```

The birth year data is stored in `mv_time`. Before we continue, we also need to transform `timestamp` into datetime format to calculate movies' ages, which is originally a integer value.

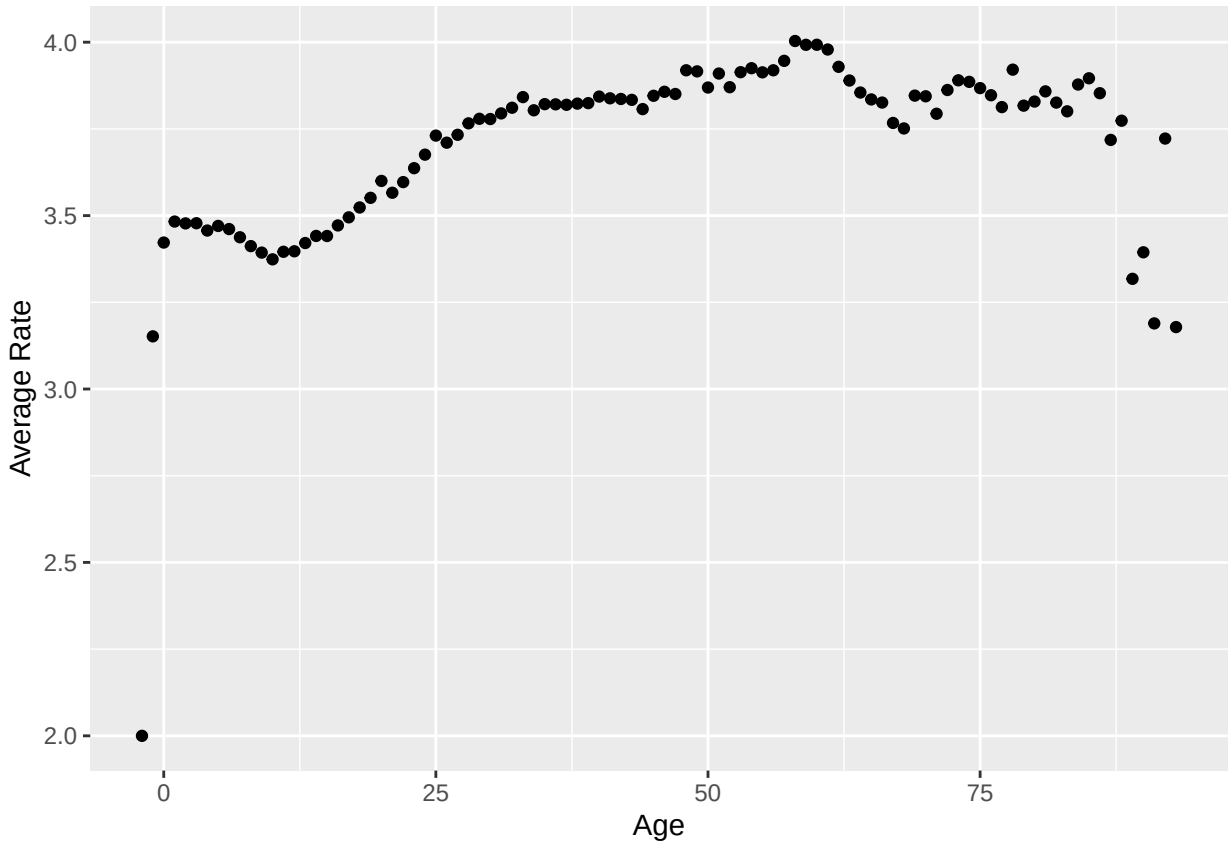
```
# Transform the timestamp

train_n <- train_n %>% mutate(timestamp = as_datetime(timestamp))
test_n <- test_n %>% mutate(timestamp = as_datetime(timestamp))
final_holdout_test_n <- final_holdout_test_n %>% mutate(timestamp = as_datetime(timestamp))
```

Then, we can draw a box plot to identify the existence of an age effect:

```
# Calculate age and then drow a plot

train_n %>% left_join(mv_time, by = "movieId") %>% # merge the birth year data
  mutate(age = year(timestamp) - rls_yr) %>% # compute age
  group_by(age) %>% # group the data by different ages
  summarise(ave_rat_age = mean(rating)) %>% # calculate average rating
  ggplot(aes(x = age, y = ave_rat_age)) +
  geom_point() +
  labs(
    x = "Age",
    y = "Average Rate"
  )
```




This dot plot shows us that we tend to give different rates for movies at different age, indicating it's helpful to include the age effect in our model:

$$Y_{u,i} = \mu + b_i + b_u + f(Age_{u,i}) + \epsilon_{u,i}$$

$Age_{u,i}$ stands for the age of the movie $i$ when the rating is made, and here we use the average rating for each age to calculate $f(Age_{u,i})$.

```
# Calculating age effect

age_effect <- train_n %>%
  mutate(age_effect = rating - mu - b_u - b_i) %>% # calculate residual
  left_join(mv_time, by = "movieId") %>% # merge the birth year data
  mutate(age = year(timestamp) - rls_yr) %>% # compute age
  group_by(age) %>% # group the data by different ages
  summarise(age_effect = mean(age_effect)) # calculate average rating
```

After that, we merge the `age_effect` into the data sets:

```
# Merge age effect into data sets
train_n <- train_n %>%
  left_join(mv_time, by = "movieId") %>% # First merge the birth year
  mutate(age = year(timestamp) - rls_yr) %>%
  left_join(age_effect, by = "age") %>% # Then merge the age effect
  mutate(age_effect = ifelse(is.na(age_effect), 0, age_effect)) # Fill the NA

test_n <- test_n %>%
  left_join(mv_time, by = "movieId") %>% # First merge the birth year
```

```
  mutate(age = year(timestamp) - rls_yr) %>%
  left_join(age_effect, by = "age") %>% # Then merge the age effect
  mutate(age_effect = ifelse(is.na(age_effect), 0, age_effect)) # Fill the NA

final_holdout_test_n <- final_holdout_test_n %>%
  left_join(mv_time, by = "movieId") %>% # First merge the birth year
  mutate(age = year(timestamp) - rls_yr) %>%
  left_join(age_effect, by = "age") %>% # Then merge the age effect
  mutate(age_effect = ifelse(is.na(age_effect), 0, age_effect)) # Fill the NA
```

### Genre effect

In the data set, the `genre` column indicated the genres a movie belongs to. From the training set, we can see that the `genre` is settled for every movie:

```
# Check the genres of one movie

train %>% filter(movieId == 1) %>% # take movieId 1 for example
  select(movieId, title, genres) %>%
  slice(1:10)
```

```
##      movieId            title                                        genres
## 114        1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 1567       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 1772       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 2465       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 2509       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 2684       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 2831       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 3496       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 3628       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
## 4681       1 Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
```

We expect that users has different preference to genres. To validate this assumption, we first need to process `genre` column in our data sets. The given format of a genre is usually "A|B|...|Z". We want to turn that into a tidy format.

First, let's count all the genres out there.

```
# Count all the genres:

movie_genres <- train %>% group_by(movieId, title, genres) %>%
  summarise(ave_rat = mean(rating)) # First group the training set into the movie-level data
```

```
## `summarise()` has grouped output by 'movieId', 'title'. You can override using
## the `.groups` argument.
```

```
all_genres <- unlist(str_split(movie_genres$genres, "\\|"))
# Use str_split() to split the genre string into some individual words.

movie_genres <- as.data.frame(table(all_genres)) %>%
  # Count the frequency of the genres and transform it into the data frame
  rename(genres = all_genres) # Rename a column

movie_genres
```

```
##                genres Freq
```

```
## 1  (no genres listed)    1
## 2              Action 1473
## 3           Adventure 1025
## 4           Animation  285
## 5            Children  527
## 6              Comedy 3702
## 7               Crime 1116
## 8         Documentary  480
## 9               Drama 5323
## 10            Fantasy  543
## 11          Film-Noir  148
## 12             Horror 1012
## 13               IMAX   29
## 14            Musical  435
## 15            Mystery  509
## 16            Romance 1683
## 17             Sci-Fi  754
## 18           Thriller 1704
## 19                War  510
## 20            Western  275
```

*Note: the code chunk above (line 6 and line 8, about the frequency calculation technique) is composed with the help of this Stack Overflow post. (Daniel)*

```
movie_genres <- movie_genres[-1,] #exclude the first row "no genres listed"

movie_genres %>% nrow()
```

```
## [1] 19
```

Except for the first row ("no genres listed"), there are 19 genres in total. In order to catch the genre effect, we can transform these genres into dummy variables. For example, if the `genre` column includes a genre A, then the variable `A` will be 1, and 0 otherwise.

To reach that goal, we can design a for loop:

```
# Generate dummy variables for each genre

for (i in movie_genres$genres){ #for all the generes

  genre_str <- as.character(i) # turn the genre into a string
  genre_sym <- sym(genre_str) # generate the symbol of this genre

  train_n <- train_n %>% mutate(!!genre_sym := ifelse(str_detect(genres, genre_str), 1, 0))
  #if the genre column include the specific pattern, we make the identifyer 1

}
```

*Note: the code chunk above (line 4 and line 6, about the usage of sym() ) is composed with the help of this Stack Overflow post.*

After the generation, we can see the training set now has one column for every genre:

```
head(train_n)
```

```
##   userId movieId rating           timestamp                    title.x
## 1      1     122      5 1996-08-02 11:24:06          Boomerang (1992)
## 2      1     292      5 1996-08-02 10:57:01           Outbreak (1995)
```

```
## 3      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 4      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
## 5      1     355      5 1996-08-02 11:14:34       Flintstones, The (1994)
## 6      1     356      5 1996-08-02 11:00:53           Forrest Gump (1994)
##                        genres       mu         b_i      b_u
## 1               Comedy|Romance 3.512457 -0.65708071 1.668561
## 2  Action|Drama|Sci-Fi|Thriller 3.512457 -0.09554569 1.668561
## 3       Action|Adventure|Sci-Fi 3.512457 -0.16142827 1.668561
## 4 Action|Adventure|Drama|Sci-Fi 3.512457 -0.17156155 1.668561
## 5       Children|Comedy|Fantasy 3.512457 -1.02442731 1.668561
## 6      Comedy|Drama|Romance|War 3.512457  0.50087084 1.668561
##                         title.y  ave_rat rls_yr age  age_effect Action
## 1              Boomerang (1992) 2.855376   1992   4 -0.01377347      0
## 2               Outbreak (1995) 3.416911   1995   1  0.02711236      1
## 3               Stargate (1994) 3.351029   1994   2 -0.01019223      1
## 4 Star Trek: Generations (1994) 3.340895   1994   2 -0.01019223      1
## 5       Flintstones, The (1994) 2.488029   1994   2 -0.01019223      0
## 6           Forrest Gump (1994) 4.013328   1994   2 -0.01019223      0
##   Adventure Animation Children Comedy Crime Documentary Drama Fantasy Film-Noir
## 1         0         0        0      1     0           0     0       0         0
## 2         0         0        0      0     0           0     1       0         0
## 3         1         0        0      0     0           0     0       0         0
## 4         1         0        0      0     0           0     1       0         0
## 5         0         0        1      1     0           0     0       1         0
## 6         0         0        0      1     0           0     1       0         0
##   Horror IMAX Musical Mystery Romance Sci-Fi Thriller War Western
## 1      0    0       0       0       1      0        0   0       0
## 2      0    0       0       0       0      1        1   0       0
## 3      0    0       0       0       0      1        0   0       0
## 4      0    0       0       0       0      1        0   0       0
## 5      0    0       0       0       0      0        0   0       0
## 6      0    0       0       0       1      0        0   1       0
```

Then, we can calculate the average rate for every genre to see if there exists a genre effect. To derive the plot, we first extract the table of the information we need:

```
# Generate average rate table:

genre_ave <- function(genre){

  genre_str <- as.character(genre) # turn the genre into a string
  genre_sym <- sym(genre_str) # generate the symbol of this genre

  ave_rat_val <- train_n %>% filter(!!genre_sym == 1) %>%
    summarise(ave_rat = mean(rating), max_rat = max(rating), min_rat = min(rating)) %>%
    as.numeric() #extract average rate

  c(genre_str, ave_rat_val) # return a sequence with the genre and the rate
}

genre_ave_tab <- sapply(movie_genres$genres, genre_ave) %>% # apply the function to all genres
  t() %>% # transpose the table
  as.data.frame() %>%  # turn into a dataframe
  rename(genre = V1, ave_rat = V2) %>%  # rename the columns
  mutate(ave_rat = as.numeric(ave_rat)) # store the average rate as numeric value
```

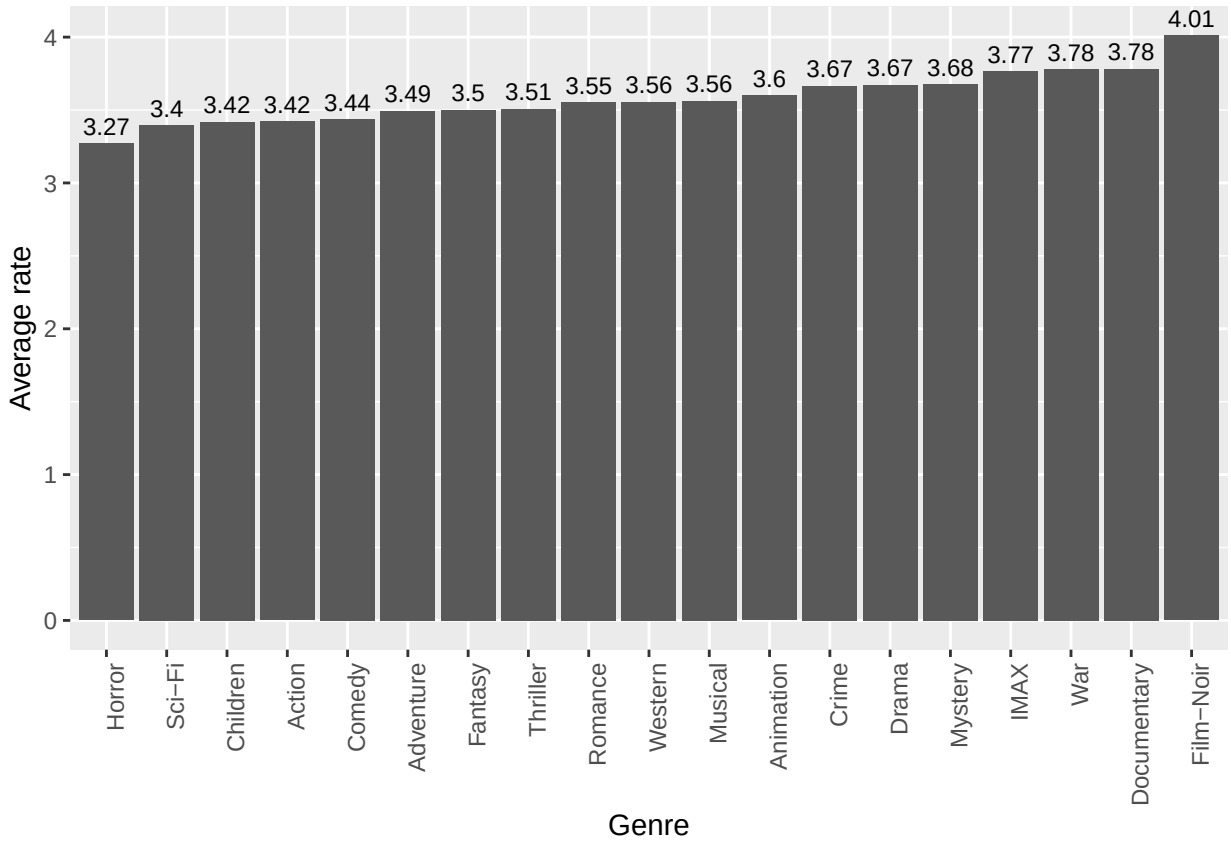
```
genre_ave_tab
```

```
##          genre  ave_rat V3  V4
## 1       Action 3.421218  5 0.5
## 2    Adventure 3.493315  5 0.5
## 3    Animation 3.599827  5 0.5
## 4     Children 3.418223  5 0.5
## 5       Comedy 3.437090  5 0.5
## 6        Crime 3.665727  5 0.5
## 7  Documentary 3.781935  5 0.5
## 8        Drama 3.673083  5 0.5
## 9      Fantasy 3.501964  5 0.5
## 10   Film-Noir 4.011087  5 0.5
## 11      Horror 3.270305  5 0.5
## 12        IMAX 3.765086  5 0.5
## 13     Musical 3.563537  5 0.5
## 14     Mystery 3.677420  5 0.5
## 15     Romance 3.553850  5 0.5
## 16      Sci-Fi 3.395690  5 0.5
## 17    Thriller 3.508034  5 0.5
## 18         War 3.780249  5 0.5
## 19     Western 3.555620  5 0.5
```

The plot is derived here:

```
# Generate the average rate plot

genre_ave_tab %>% mutate(genre = reorder(genre, ave_rat)) %>% # Order the genres by its rate
  ggplot(aes(x = genre, y = ave_rat)) +
  geom_col() +
  geom_text(aes(label = round(ave_rat, 2)), size = 3, vjust = -0.5) + # addint labels on the bar
  labs(
    x = "Genre",
    y = "Average rate"
  ) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




Given that genres are rated differently, we can now add the genre effect into our model:

$$Y_{u,i} = \mu + b_i + b_u + f(Age_{u,i}) + \sum_{k=1}^{K} x_i^k \cdot \beta_k + \epsilon_{u,i}$$

In the formula above, $x_i^k$ stands for the genre identifier of movie $i$. If the movie $i$ belongs to the genre $k$, then $x_i^k = 1$. And $\beta_k$ is the parameter we calculate for every genre based on the residual from other effects.

Here is the code we compute the genre effect:

```
# Calculate the genre effect

train_n <- train_n %>% mutate(genre_effect = 0) # generate the genre effect column

for (i in movie_genres$genres){

  genre_str <- as.character(i) # turn the genre into a string
  genre_sym <- sym(genre_str) # generate the symbol of this genre

  genre_effect_val <- train_n %>% filter(!!genre_sym == 1) %>%
    mutate(genre_effect_tmp = rating - mu - b_i - b_u - age_effect - genre_effect) %>%
    # calculate the residual
    summarise(mean(genre_effect_tmp))  %>%
    as.numeric() # extract the residual for this genre


  train_n <- train_n %>%
```

```
    mutate(!!genre_sym := ifelse(!!genre_sym == 1, genre_effect_val, 0))  %>%
    # put the genre effect value in the appropriate column
    mutate(genre_effect = genre_effect + !!genre_sym)
  # put the genre effect we calculated added in the genre_effect column
}
```

Now in the `train_n` set, we have mutated all 19 genre columns into the corresponding effect value ( $\beta_k$ in the model ). And `genre_effect` is the sum of all individual genre effects for every movie ( $\sum_{k=1}^{K} x_i^k \cdot \beta_k$ in the model).

```
head(train_n)
```

```
##   userId movieId rating           timestamp                       title.x
## 1      1     122      5 1996-08-02 11:24:06                Boomerang (1992)
## 2      1     292      5 1996-08-02 10:57:01                 Outbreak (1995)
## 3      1     316      5 1996-08-02 10:56:32                 Stargate (1994)
## 4      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
## 5      1     355      5 1996-08-02 11:14:34       Flintstones, The (1994)
## 6      1     356      5 1996-08-02 11:00:53           Forrest Gump (1994)
##                          genres       mu         b_i      b_u
## 1                Comedy|Romance 3.512457 -0.65708071 1.668561
## 2  Action|Drama|Sci-Fi|Thriller 3.512457 -0.09554569 1.668561
## 3       Action|Adventure|Sci-Fi 3.512457 -0.16142827 1.668561
## 4 Action|Adventure|Drama|Sci-Fi 3.512457 -0.17156155 1.668561
## 5       Children|Comedy|Fantasy 3.512457 -1.02442731 1.668561
## 6      Comedy|Drama|Romance|War 3.512457  0.50087084 1.668561
##                          title.y  ave_rat rls_yr age  age_effect      Action
## 1              Boomerang (1992) 2.855376   1992   4 -0.01377347  0.00000000
## 2               Outbreak (1995) 3.416911   1995   1  0.02711236 -0.01250859
## 3               Stargate (1994) 3.351029   1994   2 -0.01019223 -0.01250859
## 4 Star Trek: Generations (1994) 3.340895   1994   2 -0.01019223 -0.01250859
## 5       Flintstones, The (1994) 2.488029   1994   2 -0.01019223  0.00000000
## 6           Forrest Gump (1994) 4.013328   1994   2 -0.01019223  0.00000000
##     Adventure Animation    Children      Comedy Crime Documentary      Drama
## 1  0.00000000         0  0.00000000 0.004511318     0           0 0.00000000
## 2  0.00000000         0  0.00000000 0.000000000     0           0 0.01084147
## 3 -0.00963348         0  0.00000000 0.000000000     0           0 0.00000000
## 4 -0.00963348         0  0.00000000 0.000000000     0           0 0.01084147
## 5  0.00000000         0 -0.01516583 0.004511318     0           0 0.00000000
## 6  0.00000000         0  0.00000000 0.004511318     0           0 0.01084147
##       Fantasy Film-Noir Horror IMAX Musical Mystery      Romance      Sci-Fi
## 1 0.000000000         0      0    0       0       0 -0.007299976  0.00000000
## 2 0.000000000         0      0    0       0       0  0.000000000 -0.00484944
## 3 0.000000000         0      0    0       0       0  0.000000000 -0.00484944
## 4 0.000000000         0      0    0       0       0  0.000000000 -0.00484944
## 5 0.004485398         0      0    0       0       0  0.000000000  0.00000000
## 6 0.000000000         0      0    0       0       0 -0.007299976  0.00000000
##       Thriller           War Western genre_effect
## 1  0.000000000  0.0000000000       0 -0.002788658
## 2 -0.005988547  0.0000000000       0 -0.012505112
## 3  0.000000000  0.0000000000       0 -0.026991512
## 4  0.000000000  0.0000000000       0 -0.016150045
## 5  0.000000000  0.0000000000       0 -0.006169111
## 6  0.000000000 -0.0009227436       0  0.007130065
```

We are now able to extract that genre effect and merge it into the test sets:

```
# Extract genre effect and merge that into the test set
genre_effect <- train_n %>% group_by(movieId, genre_effect) %>%
  summarise(mean(rating)) %>%
  select(movieId, genre_effect)
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
test_n <- test_n %>% left_join(genre_effect, by = "movieId") %>%
  mutate(genre_effect = replace_na(genre_effect, 0))

final_holdout_test_n <- final_holdout_test_n %>% left_join(genre_effect, by = "movieId") %>%
  mutate(genre_effect = replace_na(genre_effect, 0))
```

# Results

## Choosing the model

According to our analysis above, we have the following models to choose:

$$\text{Model 1: } Y_{u,i} = \mu + \epsilon_{u,i}$$

$$\text{Model 2: } Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

$$\text{Model 3: } Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

$$\text{Model 4: } Y_{u,i} = \mu + b_i + b_u + f(Age_{u,i}) + \epsilon_{u,i}$$

$$\text{Model 5: } Y_{u,i} = \mu + b_i + b_u + f(Age_{u,i}) + \sum_{k=1}^{K} x_i^k \cdot \beta_k + \epsilon_{u,i}$$

Now, we will test these models on the `test` set to choose our final model:

```
# Calculate the RMSE on the test set:
# Model 1
y_hat_m1 <- test_n$mu
rmse_m1 <- RMSE(y_hat_m1, test_n$rating)

# Model 2
y_hat_m2 <- test_n$mu + test_n$b_i
rmse_m2 <- RMSE(y_hat_m2, test_n$rating)

# Model 3
y_hat_m3 <- test_n$mu + test_n$b_i + test_n$b_u
rmse_m3 <- RMSE(y_hat_m3, test_n$rating)

# Model 4
y_hat_m4 <- test_n %>%
  mutate(y_hat = mu + b_i + b_u + age_effect) %>%
```

```
  .$y_hat
rmse_m4 <- RMSE(y_hat_m4, test_n$rating)

# Model 5
y_hat_m5 <- test_n %>%
  mutate(y_hat = mu + b_i + b_u + age_effect + genre_effect) %>%
  .$y_hat
rmse_m5 <- RMSE(y_hat_m5, test_n$rating)
```

Here is the result:

```
# List RMSE results

data.frame(Model = c("Model 1",     "Model 2",  "Model 3",  "Model 4",  "Model 5" ),
             RMSE = c(rmse_m1,  rmse_m2,    rmse_m3,    rmse_m4,    rmse_m5)
             )
```

```
##     Model      RMSE
## 1 Model 1 1.0600561
## 2 Model 2 0.9429666
## 3 Model 3 0.8646859
## 4 Model 4 0.8642613
## 5 Model 5 0.8641010
```

Here, the model 5 has the least RMSE on the `test` set (0.8641010), which is the model we will apply on `final_holdout_test`.

### Performance on the `final_holdout_test`

From the result above, here we would apply the 5th model to generate the rate prediction on the `final_holdout_test`. Since we have connected relevant effect indicators to the `final_holdout_test_n` set, we are able to directly derive our result:

```
#Generate final prediction

y_hat_final <- final_holdout_test_n %>%
  mutate(y_hat = mu + b_i + b_u + age_effect + genre_effect) %>%
  .$y_hat
```

```
#Calculate RMSE of the final prediction:

RMSE(y_hat_final, final_holdout_test_n$rating)
```

```
## [1] 0.8652576
```

Utilizing our model, we finally reached the **RMSE of 0.86526**, generating relatively accurate rate prediction.

## Conclusion

The goal of this report is to predict how users would rate new movies given the data of existing rating records. To address it, we manually built a linear model composed of average rating, movie effect, user effect, age effect, and genre effect. After deciding the final model given the results on the test set, we finally applied our model to the `final_holdout_test` and gained the RMSE of 0.86526.

The limitation of this model is mainly lack of accuracy. Apart from the effects caught, there may be some other significant information that's not used in this data set. Also, the way this model deals with the known effects are relatively simple – simply applying the average of the subset to represent. For the future work, we

will keep dig in the existent data to capture more effects and design more accurate and complete way (like k-means or random forest or other continuous function etc.) to express the effects found.

# References

Daniel, "R: Count the frequency of every unique character in a column" *Stack Overflow, 27 Apr. 2019,* https://stackoverflow.com/questions/55885090/r-count-the-frequency-of-every-unique-character-in-a-column