Capstone Project: Movie Recommendation System

Dongjie Cheng

10/31/2020

Introduction

The objective of this project is to create a movie recommendation system using the MovieLens dataset and the **RMSE** should be smaller than *0.86490*. This dataset was downloaded from the GroupLens research lab website: http://files.grouplens.org. HarvardX provided the partition code to generate the training set, namely ***edx***, and the validation set, ***validation***. Although there are plenty of regression functions in **R** to solve the prediction problems, they cannot handle MovieLens **10MB** samples dataset. In this project, I used **least square** method by setting up **five** models with new method, but starting from the models learned from HarvardX **Machine Learning** course. As required, the analysis was done on the training set, whereas the validation set is only used to evaluate the models. In this paper, I will present the process in **three** major sections: data acquisition, data exploring and cleaning, and modeling methods and data analysis. Finally, I will demonstrate how a model with **RMSE** of ***0.86460*** has been achieved.

Data Acquisition

I downloaded and partitioned the MovieLens dataset using the provided code with minimum modification, e.g., the piece of code designed for R 3.6 earlier was removed. The chunk of the acquisition code is shown below.

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")


# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # 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)
```

The code has properly generated the training set and and the validation set, **edx** and **validation** respectively. Next, we look into the data sets to understand the insight.

Data Exploring and Cleaning

As a result of partition, **edx** contains 9000055 records and **validation** contains 999999 records such that **validation** holds about *10%* of the whole records while **edx** about *90%*. Meanwhile, using ***edx*** as an example shown below, the MovieLens dataset includes ***6*** features. Of them, the **rating** is our **resultant output** to be predicted and other *5* features are the potential inputs for our inverse problem.

```
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
```

Feature name	Type
ratings	Quantitative resultant output
movieId	Quantitative input
userId	Quantitative input
timestamp	Quantitative input
title	Qualitative input
genre	Qualitative input

To process the data, first, we need clean the datasets to assure appropriate results. Generally, the major issue for data analysis is the empty cells present in datasets. The following code checks ***NA's*** inside **edx** and **validation**. Nicely, there was no ***NA*** was found.

```
## check NA's
sapply(edx, function(x)sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```
sapply(validation, function(x)sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```
edx%>%group_by(genres)%>%summarise(sum=n())
```

```
## # A tibble: 797 x 2
##    genres                                              sum
##    <chr>                                             <int>
##  1 (no genres listed)                                    7
##  2 Action                                            24482
##  3 Action|Adventure                                  68688
##  4 Action|Adventure|Animation|Children|Comedy          7467
##  5 Action|Adventure|Animation|Children|Comedy|Fantasy   187
##  6 Action|Adventure|Animation|Children|Comedy|IMAX       66
##  7 Action|Adventure|Animation|Children|Comedy|Sci-Fi    600
##  8 Action|Adventure|Animation|Children|Fantasy          737
##  9 Action|Adventure|Animation|Children|Sci-Fi            50
## 10 Action|Adventure|Animation|Comedy|Drama             1902
## # ... with 787 more rows
```

```
## remove empty genre records
edx_org<-edx
edx<-filter(edx,genres!="(no genres listed)")
```

However, *7* records have no genre type in **edx** and were removed as shown above. Therefore, my actual training set contains 9000048 records. Otherwise, no abnormal records were found in both the training and validation sets. The next step is to analyze the training dataset to understand the insight.

Since our goal is to build a reasonable recommendation system, I mainly focused on the key features that will be used to build the models in the next **Method and Data Analysis** section. First, I found that the **outcome**, **ratings**, are discrete, ranging from *0.5* to *5*, with increment of *0.5*. Also, there are *10676* **movies** and *69878* **unique users** in the training set whereas *9809* and *68534* in the validation set.

```
sort((distinct(edx,rating))$rating)
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
count(distinct(edx,movieId))
```

```
##        n
## 1: 10676
```

```
count(distinct(edx,userId))
```

```
##        n
## 1: 69878
```

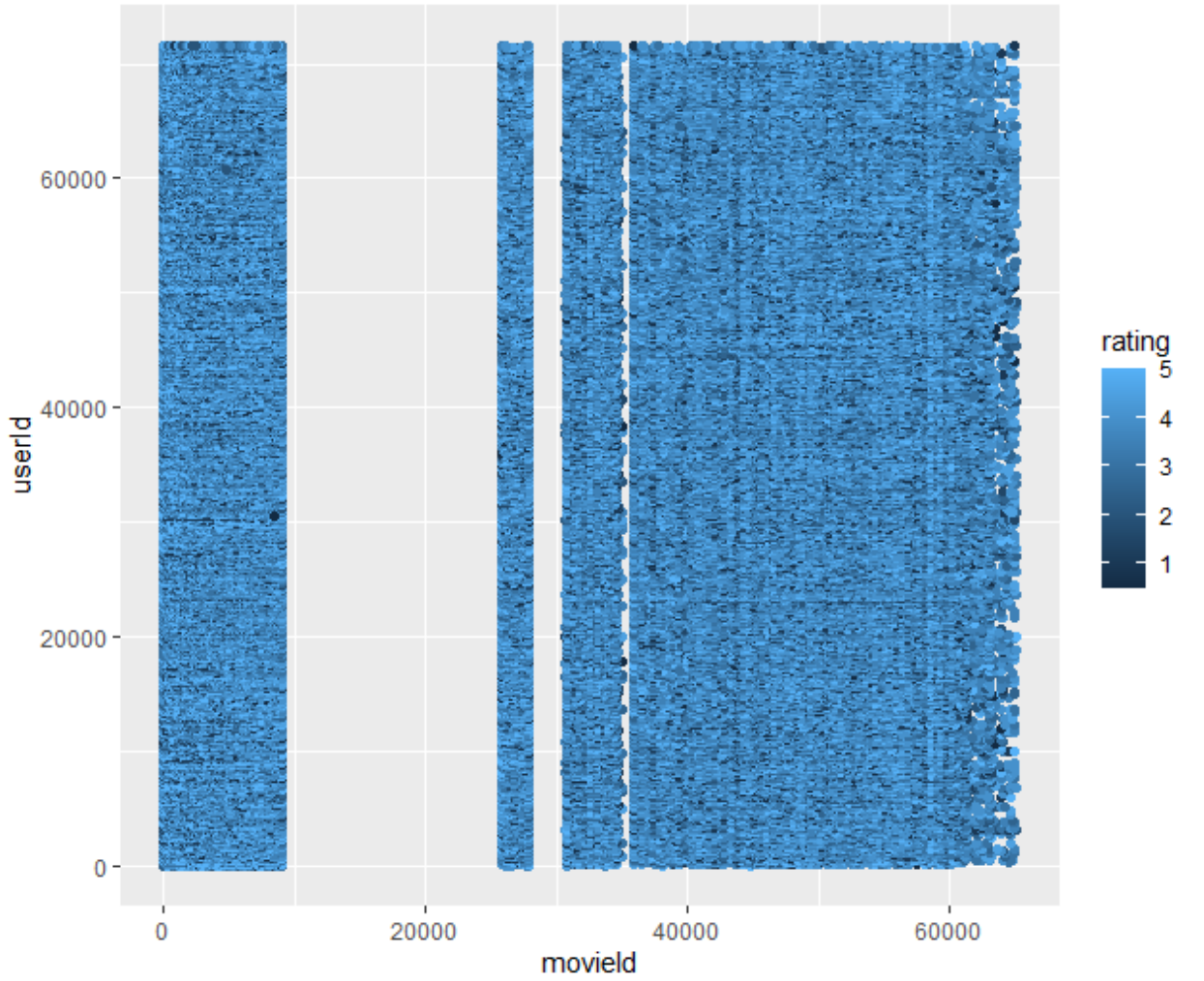
```
count(distinct(validation,movieId))
```

```
##       n
## 1: 9809
```

```
count(distinct(validation,userId))
```

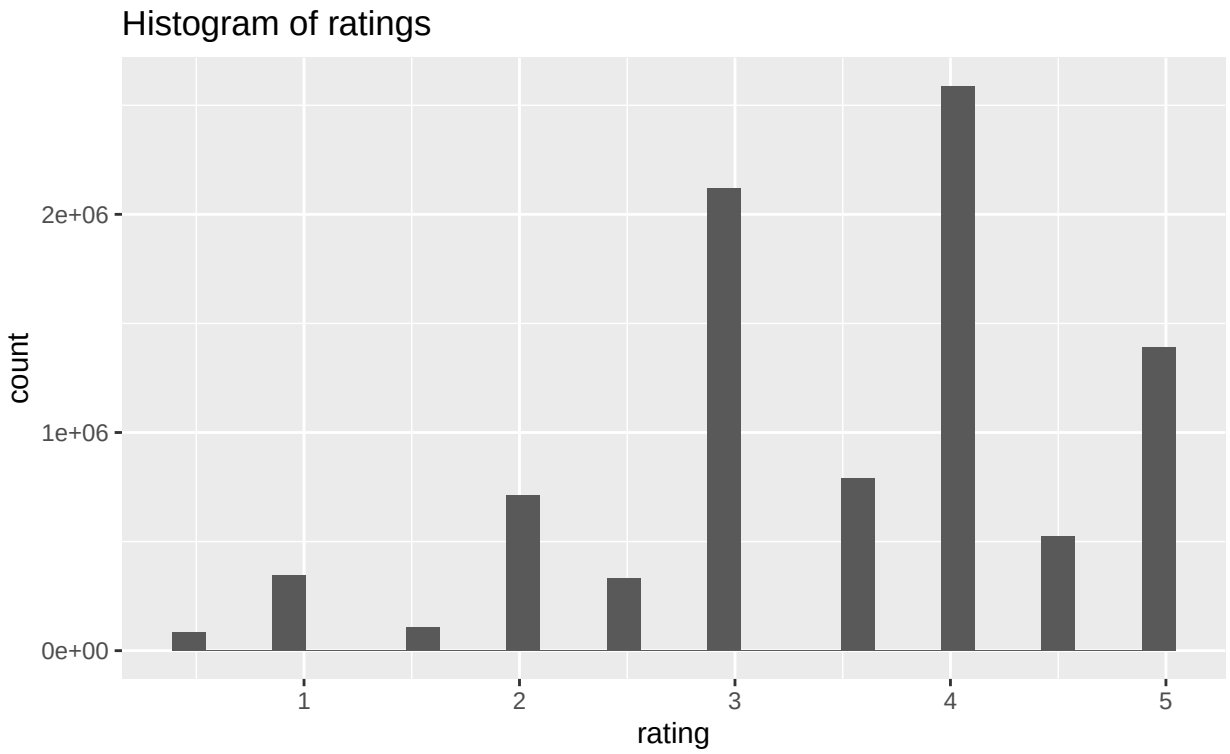
```
##        n
## 1: 68534
```

Starting graphic analysis, the figure below shows a map view of *rating* vs. *movieId* and *userId*(Because the code takes very long time to run, it is not included here). There is a large missing gap among the middle *movieId* numbers. Hence, we will have a quite sparse matrix for linear regression.

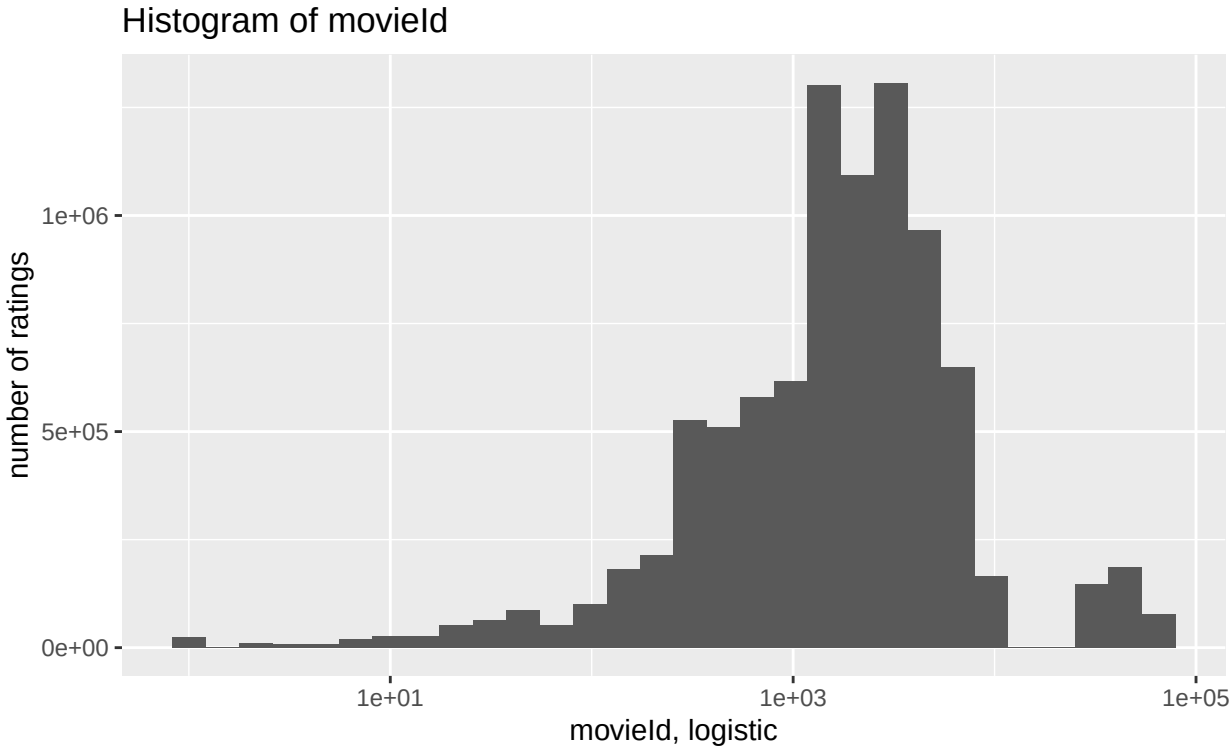


The next figure shows the histogram of the counts of *ratings*. Interestingly, the counts of the **half** and **whole** numbers have a **saw teeth** pattern. The next histogram shows the investigation of the *rating* numbers vs. *movieId* . Please notice that the *x-axis* is in $\log_{10}$ scale of **movieId**. The high number of ratings concentrates on the middle *movieIds*, however, low for both small and big *movieIds*. The third figure shows the *rating* numbers vs. *userId*. Again with the *x-axis* in $\log_{10}$ scale of the userId. Clearly, different users have different contributions to the ratings.

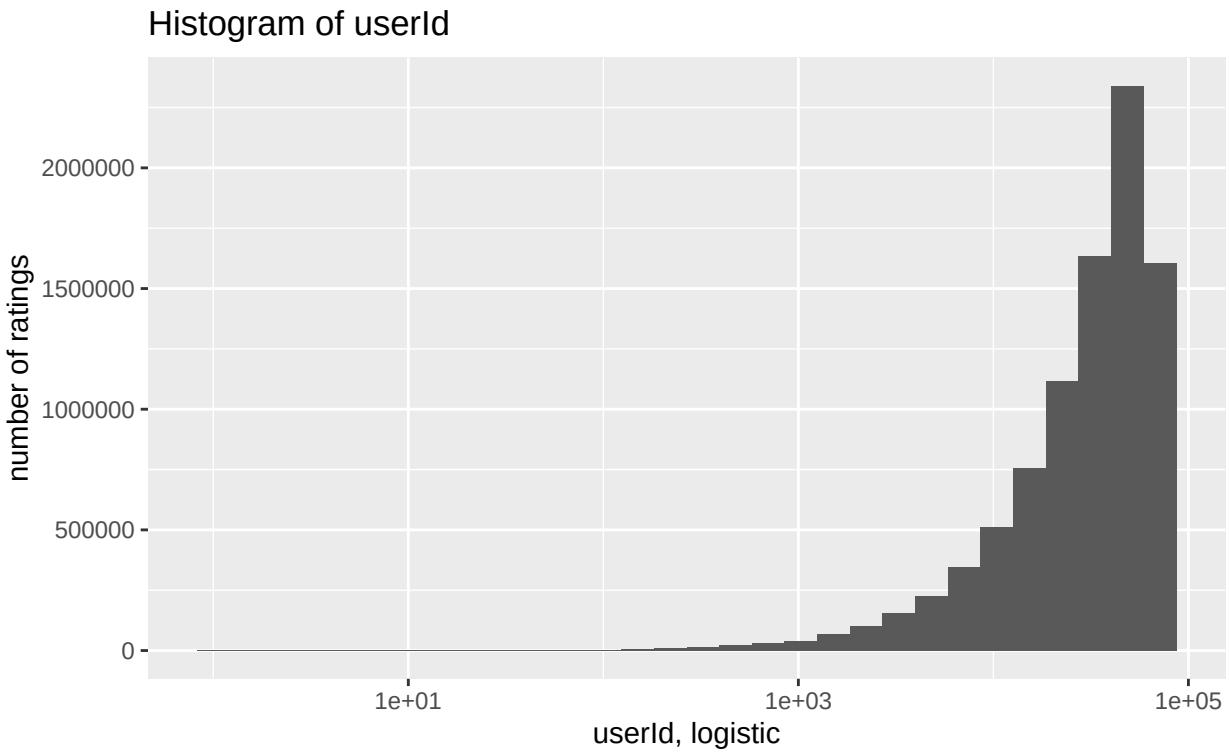
```
## rating
  edx%>%group_by(rating)%>%ggplot(aes(rating))+geom_histogram()+
    labs(title = "Histogram of ratings", y="count",x="rating" )
```




```
## rating vs movieId
  edx%>%group_by(movieId)%>%ggplot(aes(movieId))+geom_histogram()+
    scale_x_log10() +
    labs(title = "Histogram of movieId", y="number of ratings",x="movieId, logistic")
```



```
## rating vs userId
  edx%>%group_by(userId)%>%ggplot(aes(userId))+geom_histogram()+
    scale_x_log10() +
    labs(title = "Histogram of userId", y="number of ratings",x="userId, logistic")
```



In addition, I also looked into the **genre** factor, since different genres might attract different types or numbers of audiences. Consequently, the movies may receive different review coverage or different tendency of ratings. For the *796* genres in the training data set, the first figure below shows the **rating** for each *genre*. Their rating numbers range from *2* to *733296* derived from below.

```
min(edx%>%group_by(genres)%>%summarise(sum=n())%>%.$sum)
```

```
## [1] 2
```

```
max(edx%>%group_by(genres)%>%summarise(sum=n())%>%.$sum)
```

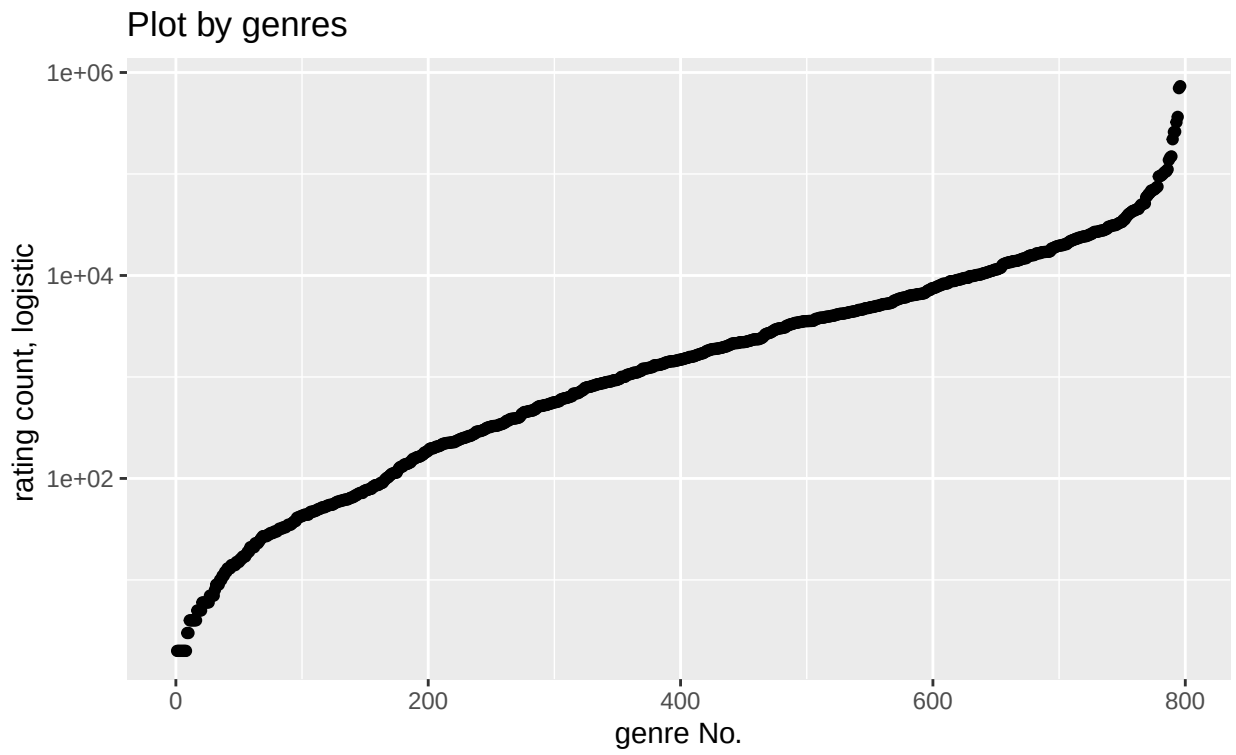
```
## [1] 733296
```

Obviously, there is a huge difference between the maximum and the minimum genres. The chunk of code below demonstrate the range of the rating numbers and two figures showing the box plots of rating numbers of the top *20* and bottom *20* genres. For the top *20*, the range of the rating numbers are from *73286* to *733296*. In contrast, the bottom *20's* only ranges from *2* to *5*. The pictures also suggest that the top *20* are more uniform distributed than the bottom *20* do. Therefore, it will be reasonable to take into account of **genre** factor in our data analysis.

```
count(distinct(edx,genres))
```

```
##      n
## 1: 796
```

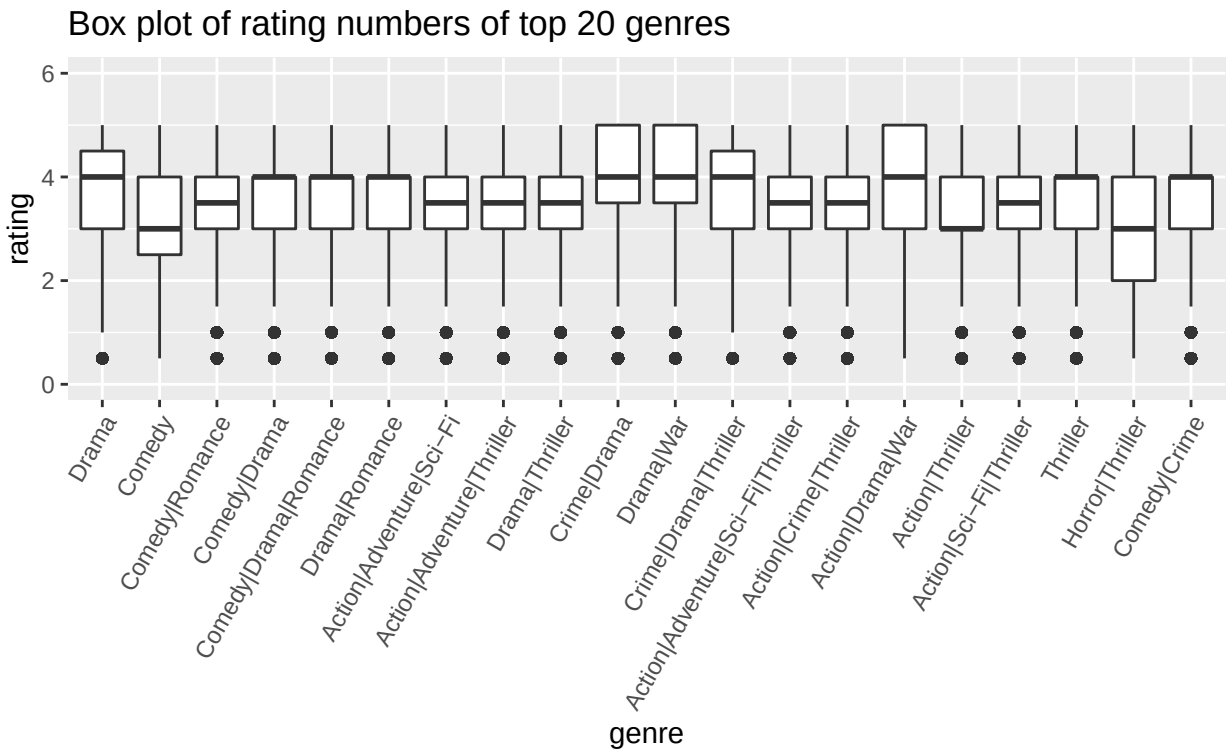
```
# rating coverage vs genre plot
  edx%>%group_by(genres)%>%summarise(sum=n())%>%arrange(sum)%>%cbind(genre_no=c(seq(1:796)))%>%
    ggplot(aes(genre_no,sum))+geom_point()+
    scale_y_log10() +labs(title = "Plot by genres", x="genre No.", y="rating count, logistic ")
```




```
## boxplot of top 20 genres with high rating numbers
genre<-edx%>%group_by(genres)%>%summarise(sum=n())%>%arrange(desc(sum))%>%top_n(20)%>%left_join(edx,by=
range(genre$sum)
```

```
## [1]  73286 733296
```

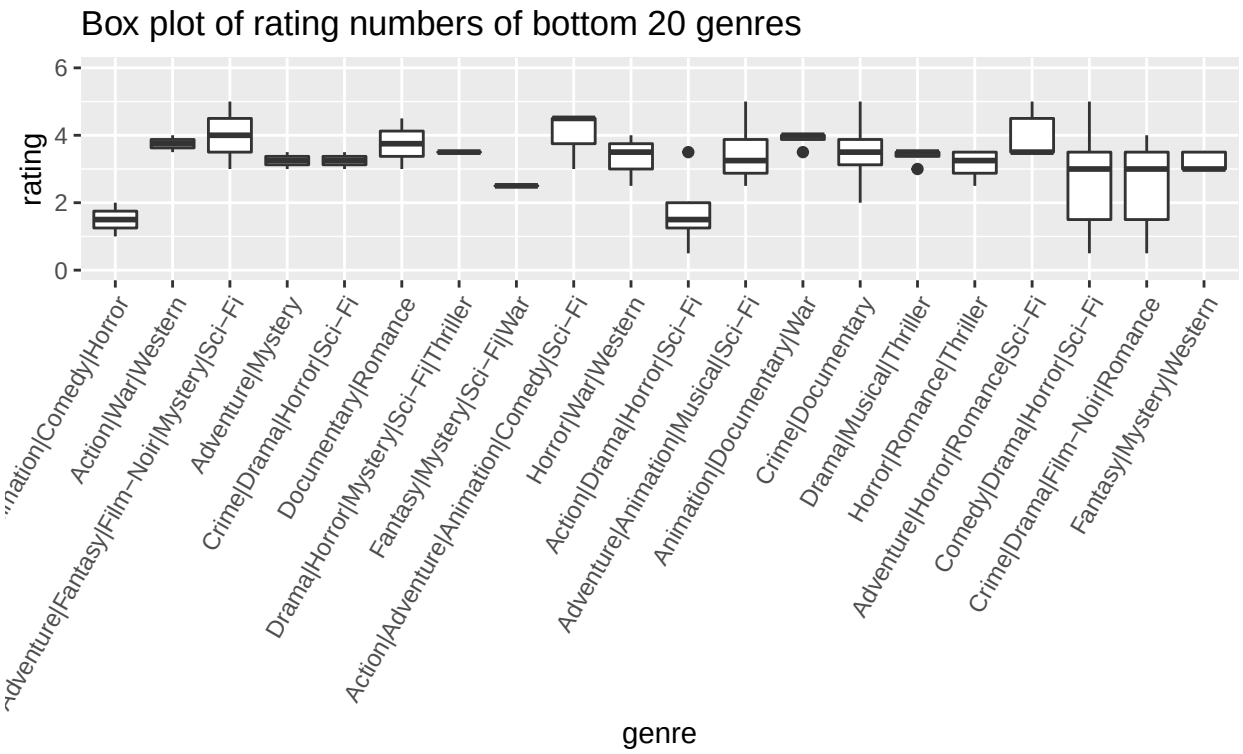
```
genre%>%ggplot(aes(x=reorder(genres,-sum),rating))+geom_boxplot()+
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  labs(title = "Box plot of rating numbers of top 20 genres", x="genre")+ ylim(0,6)
```




```
## boxplot of bottom 20 genres with high rating numbers
genre<-edx%>%group_by(genres)%>%summarise(sum=n())%>%arrange(desc(sum))%>%top_n(-20)%>%left_join(edx,by=
range(genre$sum)
```

```
## [1] 2 5
```

```
genre%>%ggplot(aes(x=reorder(genres,sum),rating))+geom_boxplot()+
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  labs(title = "Box plot of rating numbers of bottom 20 genres", x="genre")+ ylim(0,6)
```



After proper data cleaning and key features properties study, we move on to methodology study and data analysis.

Method and Data Analysis

Modeling Algorithm Methodology

From above, we know that ***edx*** has 9000048 samples. Given the current computer resources, it is impossible to render such huge amount of data using regression functions in **R**, like, ***knn***, ***glim***, etc. Therefore, my approach is ***least square*** method by following the HarvardX course, **Machine learning**. However, I add ***genres*** as a contributor in the model, making a new algorithm.

First, let us start from the initial model taught in class. Assume N is the total number of ratings, $Y_{u,i}$ is the rating value for movie i and user u. Thus, the **RMSE** expression is:

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

where $\hat{y}_{u,i}$ denotes the predicted rating.

Next, we consider the simplest model assuming same rating μ for all movies and users. Thus, we have the model equation (Model 1) as:

$$Y_{u,i} = \mu + \epsilon_{u,i} \tag{1}$$

where ϵ denotes the independent errors. The ***least square*** solution for μ is the *mean* of all *ratings* of the dataset.

Moving to the second model, we add the *movie* factor into Equation (1) while leaving μ unchanged. The new equation (Model 2) will be written as:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{2}$$

where b_i represents the bias for each *movieId* and its ***least square*** solution can be expressed as:

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

Adding *userId* factor, we obtain the third equation (Model 3) :

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \tag{3}$$

where b_u represents the bias for each *userId* and its ***least square*** solution can be expressed as:

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

Furthermore, we consider the independent *genre* factor. Let us define random variation $Y_{u,i,g}$ as the rating observation for movie i, user u and genre g. A new model can be expressed (Model 4) as:

$$Y_{u,i,g} = \mu + b_i + b_u + b_g + \epsilon_{u,i,g} \tag{4}$$

where b_g represents the bias for each *genres* and its ***least square*** solution can be expressed as:

$$\hat{b}_g = \frac{1}{N_g} \sum_{g=1}^{N_g} (Y_{u,i,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

and the **RMSE** can be defined as:

$$\sqrt{\frac{1}{N} \sum_{u,i,g} (\hat{y}_{u,i,g} - y_{u,i,g})^2}$$

Finally, let us introduce ***regularization*** on b_i and b_u as taught in the course, then we obtain **Model 5** and the **loss** function can be expressed as:

$$\frac{1}{N} \sum_{u,i,g} (y_{u,i,g} - \mu - b_i - b_u - b_g)^2 + \lambda \left(\sum b_i^2 + \sum b_u^2 \right) \tag{5}$$

The ***least square*** solutions can be expressed as:

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

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

and

$$\hat{b}_g = \frac{1}{N_g} \sum_{g=1}^{N_g} (Y_{u,i,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

Meanwhile, we have the RMSE as follows:

$$RMSE = \sqrt{\frac{1}{N} \sum_{u,i,g} (y_{u,i,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g)^2}$$

Data Analysis

With these **five** models, we analyze the MovieLens dataset to derive the optimal model.

First, Let us compute $\hat{\mu}$ starting from the simplest model (Model 1) defined in Equation (1), then we measure the **RMSE** as shown below.

```
#mu throughout the all users and movies
mu_hat<-mean(edx$rating)

# Evaluate mu model result
rmse_mu <- RMSE(mu_hat, validation$rating)
rmse_mu
```

```
[1] 1.061202
```

```
rmse_results <- data_frame(method="mu Only",RMSE = rmse_mu)
```

Second, apply Model 2 in Equation (2).

```
#b_i
bi<- edx %>%
 group_by(movieId) %>%
 summarize(b_i = mean(rating - mu_hat))


# Evaluate bi model result
predicted_ratings <- mu_hat + validation %>%
 left_join(bi, by='movieId') %>%
 .$b_i

rmse_bi <- RMSE(predicted_ratings, validation$rating)
rmse_bi
```

```
[1] 0.9439087
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = rmse_bi ))
rmse_results %>% knitr::kable()
```

method	RMSE
mu Only	1.0612018
Movie Effect Model	0.9439087

Third, Model 3 in Equation (3).

```
#b_u
b_u <- edx %>%
left_join(bi, by='movieId') %>%
group_by(userId) %>%
summarize(b_u = mean(rating - mu_hat - b_i))

predicted_ratings <- validation %>%
left_join(bi, by='movieId') %>%
left_join(b_u, by='userId') %>%
```

```
mutate(pred = mu_hat + b_i + b_u) %>%
.$pred

rmse_bu <- RMSE(predicted_ratings, validation$rating)
rmse_bu
```

```
[1] 0.8653489
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effect Model",
                                     RMSE = rmse_bu ))

rmse_results %>% knitr::kable()
```

method	RMSE
mu Only	1.0612018
Movie Effect Model	0.9439087
Movie + User Effect Model	0.8653489

Fourth, Model 4 in Equation (4).

```
#b_g genres factor

b_g <- edx %>%
  left_join(bi, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i-b_u))

predicted_ratings <- validation %>%
  left_join(bi, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u+b_g) %>%
  .$pred

rmse_bg <- RMSE(predicted_ratings, validation$rating)
rmse_bg
```

```
[1] 0.864947
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Genre Effect Model",
                                     RMSE = rmse_bg ))

rmse_results %>% knitr::kable()
```

method	RMSE
mu Only	1.0612018
Movie Effect Model	0.9439087
Movie + User Effect Model	0.8653489
Movie + User + Genre Effect Model	0.8649470

To obtain the optimal result for Model 5 in Equation (5), we run λ scan first. The test results are shown below:

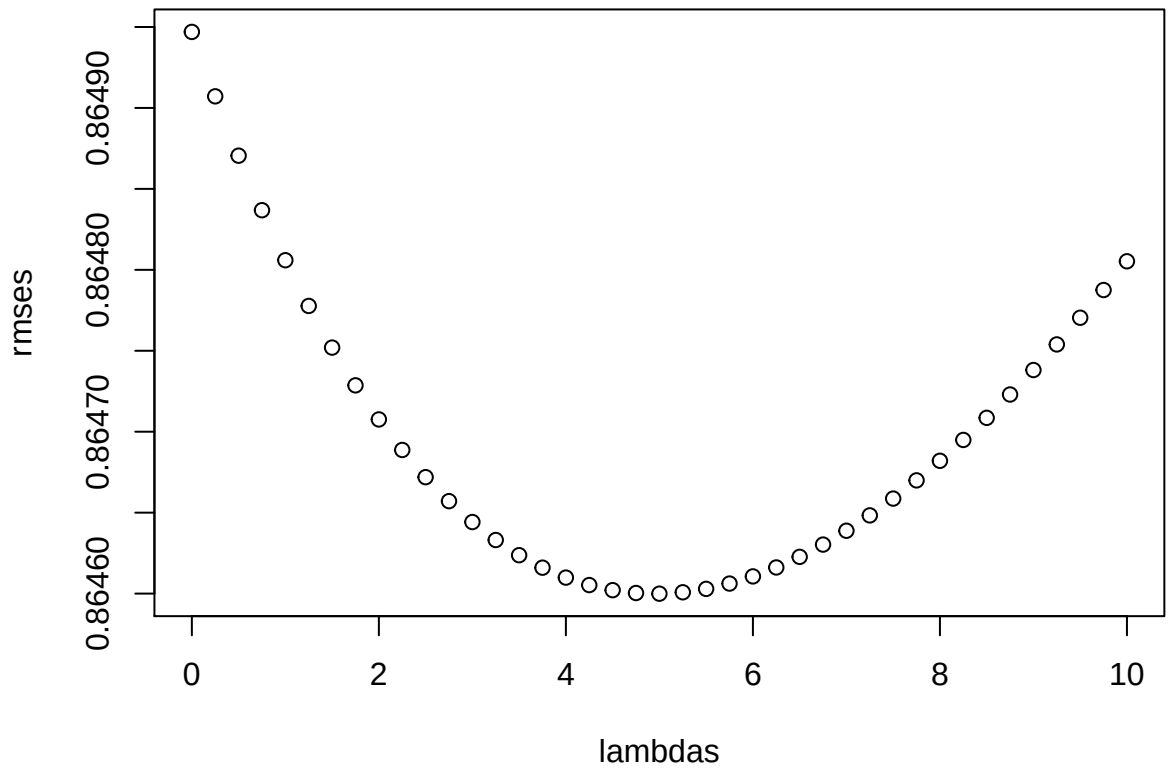
```
## lambda scan
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l) {
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat) / (n() + l))
  b_u <- edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat) / (n() + l))

  b_g <- edx %>%
    left_join(bi, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu_hat - b_i - b_u))

  predicted_ratings <- validation %>%
    left_join(bi, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu_hat + b_i + b_u + b_g) %>%
    .$pred

  return(RMSE(predicted_ratings, validation$rating))
})
plot(lambdas,rmses)
```




```
lambdas[which.min(rmses)]
```

```
[1] 5
```

The λ-**RMSE** figure clearly show the optimal λ value is ***5*** giving minimum **RMSE**. Re-run the code with fixed λ, then we have the new solution as the following:

```
############ regularized b_i, b_u + b_g
lambda<-5
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat)/(n()+lambda))

b_g <- edx %>%
  left_join(bi, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i-b_u))
```

```
predicted_ratings <- validation %>%
  left_join(bi, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u+b_g) %>%
  .$pred

rmse_bg <- RMSE(predicted_ratings, validation$rating)
rmse_bg
```

```
[1] 0.8646
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Reglarized Movie & User + Genre Effect Model",
                                     RMSE = rmse_bg ))

rmse_results %>% knitr::kable()
```

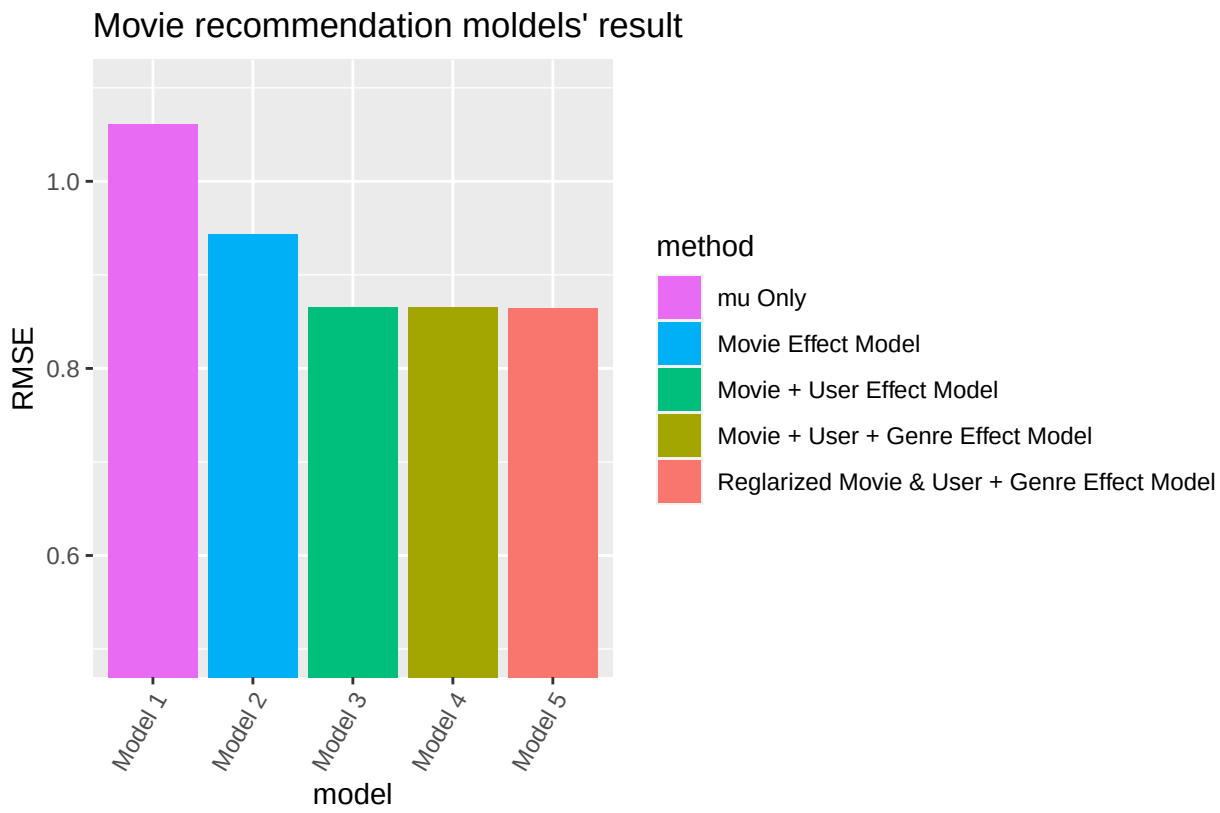
method	RMSE
mu Only	1.06120
Movie Effect Model	0.94391
Movie + User Effect Model	0.86535
Movie + User + Genre Effect Model	0.86495
Reglarized Movie & User + Genre Effect Model	0.86460

For comparison purpose, I plot the **RMSE** for all ***five*** models in a bar graph below. In the figure, all methods and corresponding models are either labeled or legended. The **RMSEs** decrease with the increase of the model number. Models 1 and 2 present big difference from other models. We can easily conclude that Model ***5*** has the minimum **RMSE**.

```
## add model numbers
rmse_results<-rmse_results%>% cbind(model=c("Model 1", "Model 2", "Model 3", "Model 4", "Model 5"))

## prepare for final plot
rmse_results$method<-reorder(rmse_results$method,rmse_results$RMSE)

## final results plot
rmse_results%>%
  ggplot(aes(model,RMSE, fill=method)) +
   geom_bar(stat='identity') +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  coord_cartesian(ylim = c(0.5, 1.1)) +
  guides(fill = guide_legend(reverse=TRUE)) +
  labs(title="Movie recommendation moldels' result" , x="model", caption = "")
```



Conclusion

The MovieLens provides us a typical large clean movie review dataset with *6* features. I have tested *5* models to build a **Movie Recommendation System**. The model features increases from small to large number models. Accordingly, the **RMSE** also decreased, when we applied the models in the same sequence. It turns out that Model 5 with regularized **movieId** and **useId**, and **genre** factor, gives us an optimal result with **RMSE** of *0.86460*. It is smaller than the required criterion of *0.86490* by the assignment. Therefore, my project has achieved its goal.

Meanwhile, I have studied **three** input factors in this project. In the future, I will study the effect of the **timestamp** factor and possible regularization on the all contributors.

Reference

- https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2020/course/
- https://grouplens.org/datasets/movielens/