MovieLens Report

Masayoshi Sato

2021/5/3

Introduction

Nowadays, recommendation systems have become common to us. Many products and services are rated by consumers using stars and points, and ratings are utilized by companies and organizations. These ratings are precious; they are not only helpful for customers to choose goods or services, but are also valuable for makers and service providers in terms of predicting consumer behaviors. If you explore a user ratings in detail, you will find his/her preference, and be able to recommend things which may be most likely to be bought.

One of the field in which this system is used is movie industries. In this paper, I will take a movie recommendation dataset, and build a model which will predict rating using other factors as accurately as possible.

Dataset The data used in this report, MovieLens 10M Dataset, is available at the Grouplens website. According to the site, this data set has 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

MovieLens 10M dataset https://grouplens.org/datasets/movielens/10m/

Goal The goal is to train a machine learning algorithm to predict movie ratings based on factors in the provided dataset.

As GroupLens web site explains, it has several factors, such as user ID, movie title, genres, rating as well. The assumption is that they are inter-correlated to some extent and helpful to forecast ratings. The challenge of a recommendation system, however, is that the data is sparse and each factor in the dataset might affect others. Namely, some users rate movies more than others, and some movies are rated more. Genres, released year may also affect ratings.

Therefore, a linear regression will be used to find weights that minimize the residual mean squared error, the RMSE. Predictors are user ID, movie ID, genres. and released year. In addition to this, regulation is used to penalize large estimates that are produced using small sample sizes.

Exploratory analysis

The downloaded data is split into two, edx dataset (90%) and validation dataset (10%).

The validation dataset has 999999 rows and 6 columns. And the edx dataset has 9000055 rows and 6 columns.The validation dataset is left to final evaluation.

```
dim(edx)
```

```
## [1] 9000055       6
```

```
dim(validation)
```

```
## [1] 999999      6
```

The first six rows of the edx dataset is as follows;

```
##    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
```

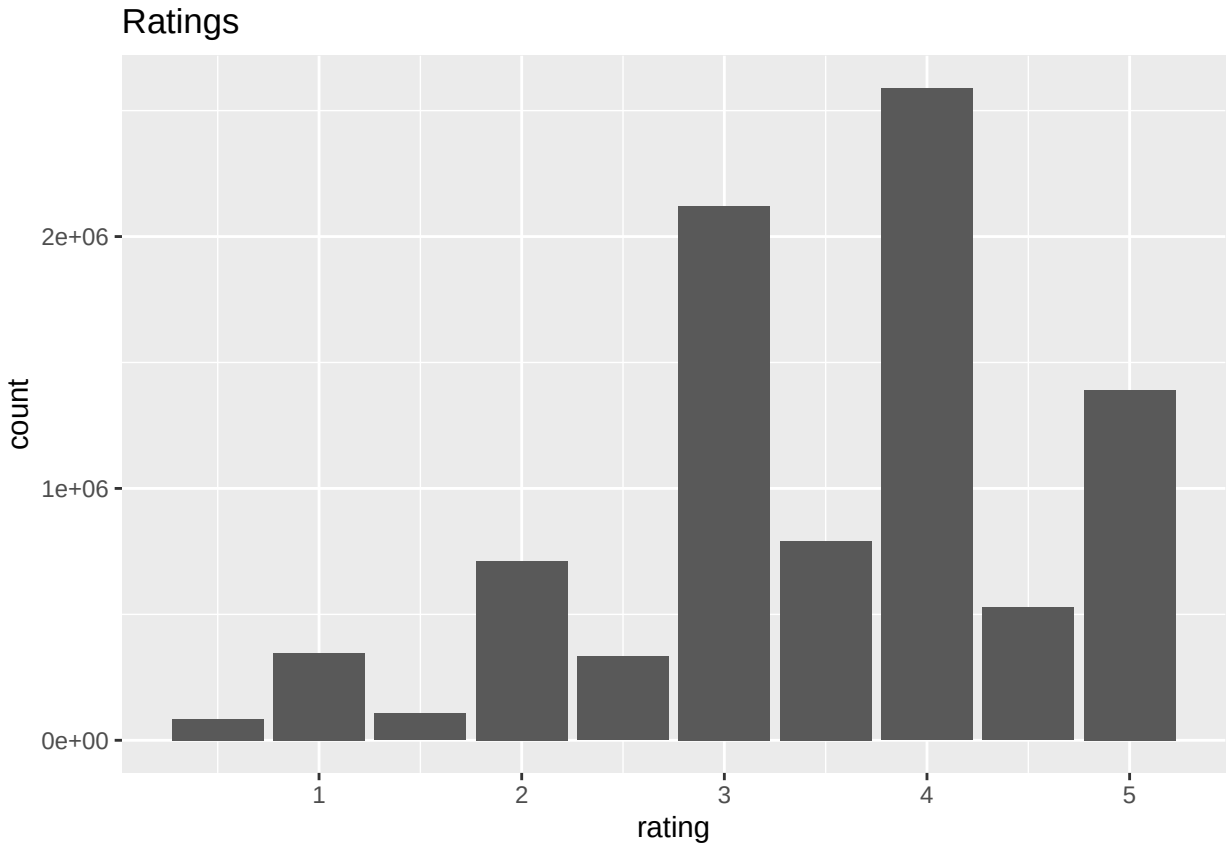
"userId" indicates a person who rate a movie("rating"). Users may rate multiple movies. A movie corresponds "movieId" and "title". "title" also has a movie's release year in a parenthesis.Each movie is categorized into "genres", which consists of several words, such as "Comedy", "Crime", or "Sci-Fi" separated by "|". Then, we will investigate each column to find out its features, especially its distribution.

Rating To begin with, we look into "rating" column. It ranges from 0.5 to 5.0 by 0.5.

```
unique(edx$rating)%>% sort()
```

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

Plot its distribution.



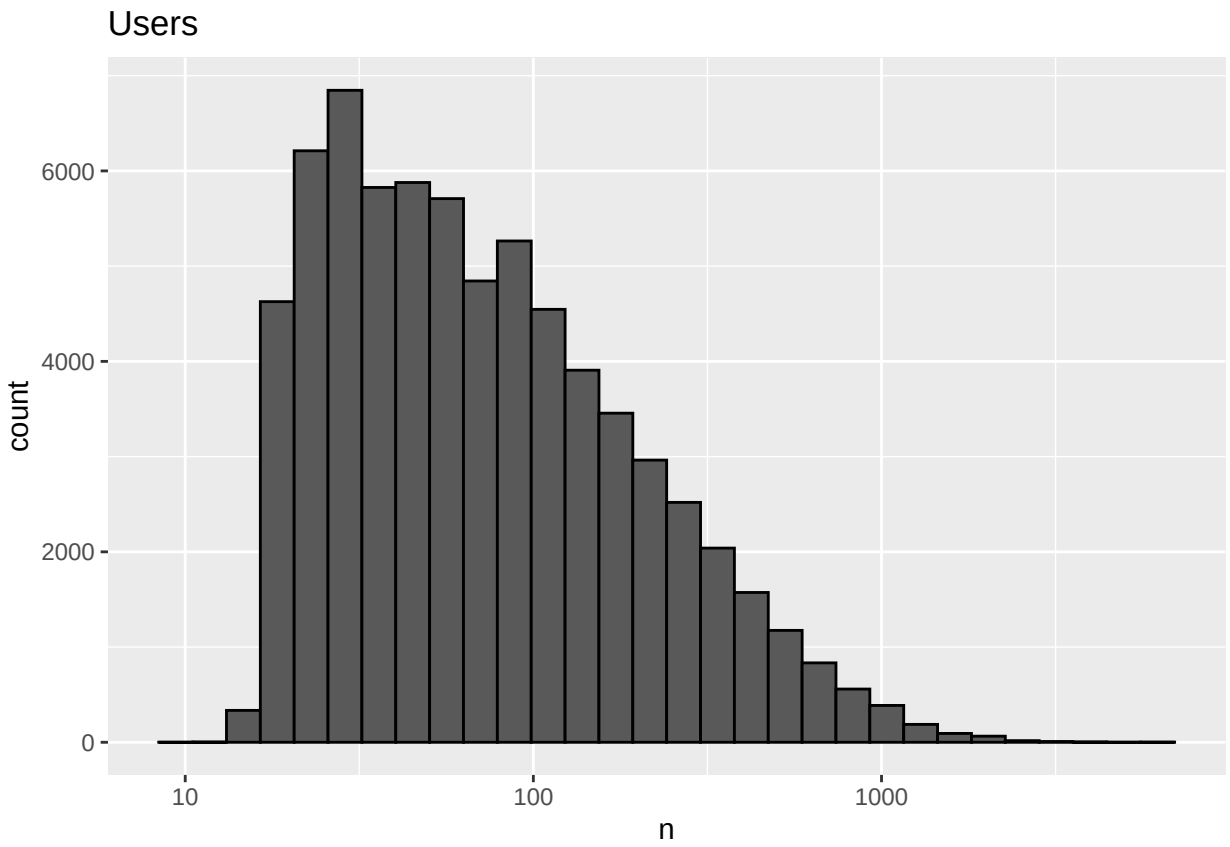
The distribution is right-skewed. There is a tendency to rate 3.0 and 4.0. Ratings which have .5 are rare compared to integer ratings.

Users Then, we look into "userId". As is seen in the first six rows of the edx, users rate multiple movies. Unique number of users is 69878.

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

Then, plot a histogram of "userId" (x-axis are using logarithmic scale).



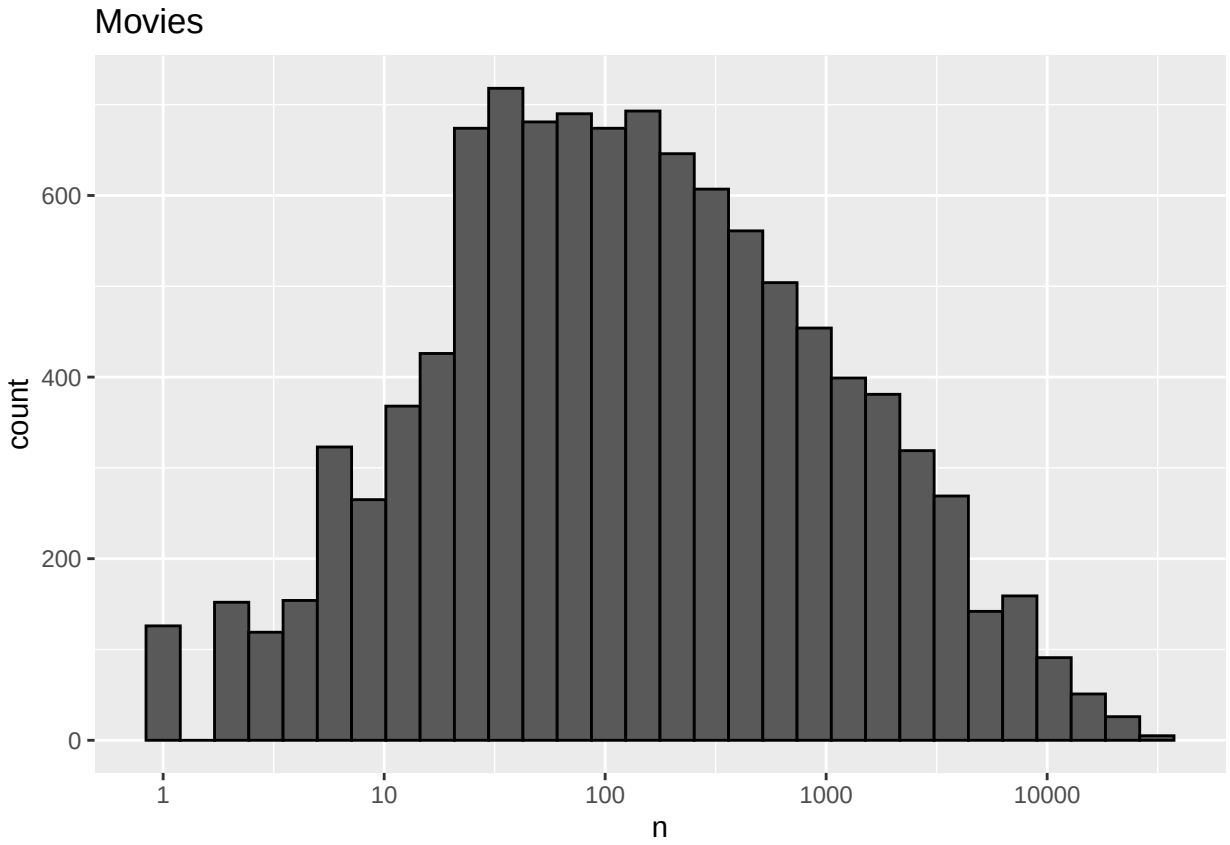
It's distribution is left-skewed. Some users are very active in rating movies. We clearly see that there is a user bias.

Movies Likewise, movies are rated by multiple users. In the edx, 10677 movies are rated.

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

Plot its distribution (x-axis are using logarithmic scale).



This histogram tells us some movies get very small amount of ratings. It surely leads to biases.

Timestamp "timestamp" is integer. It is the exact date of rating, shown in integer counting days from the day "1970.01.01".To deal with, these figures need to covert to POSIXct data.

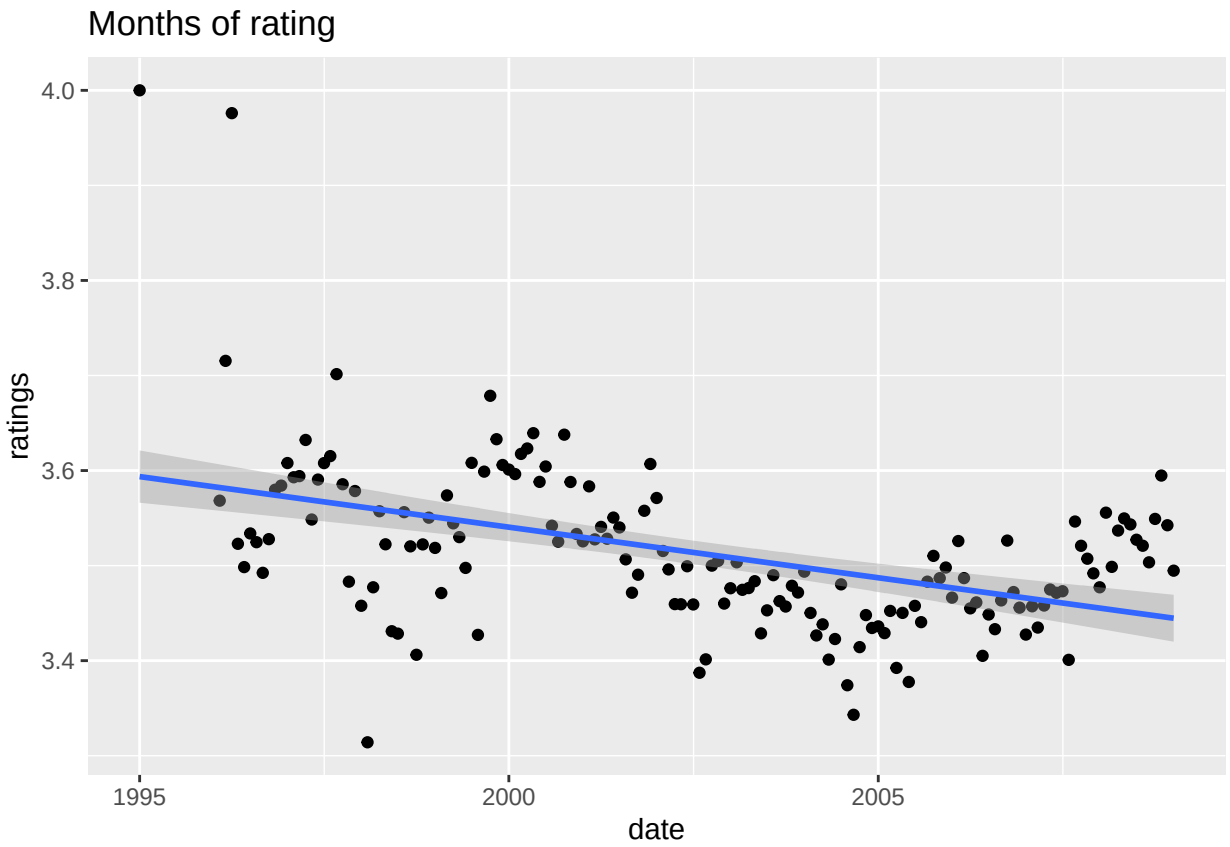
```
edx$timestamp <- as.POSIXct(edx$timestamp, origin= "1970-01-01",tz="GMT")
```

Find out the correlation between the date of rate and rating, creating a new column "date". The column have a month and year of the date when being rated.

```
edx <- edx %>% mutate(date = round_date(timestamp, unit = "month"))
```

Plot rating and date showing a regression line.

```
## `geom_smooth()` using formula 'y ~ x'
```



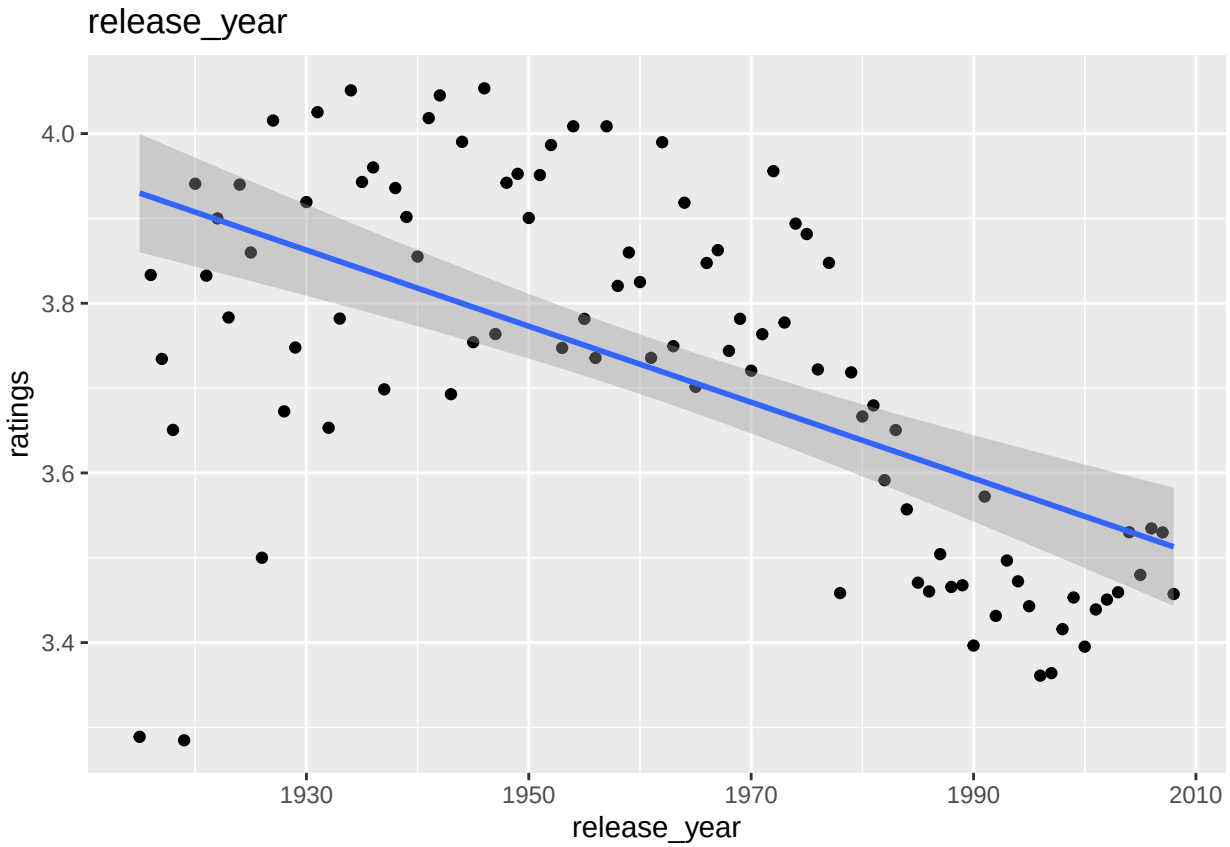
Points are mainly gathered around 3.5 rating, and the regression line is relatively flat. We assume the release dates do not have any significance.

Title Titles are difficult to deal with. But as we mentioned before, they have release years .Extract release years from titles. We mutate new columns,"release year".

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

Plot rating and release year showing a regression line.

```
## `geom_smooth()` using formula 'y ~ x'
```



There seems to be a tendency that recent movies are rated low and old movies are rated high. The release year works as a bias to predict rating.

Genres Some like comedy, but do not like thriller. Some like Sci-fi but do not like animation. It is natural to think such preferences may affect movies' rating. The edx has 797 genres.

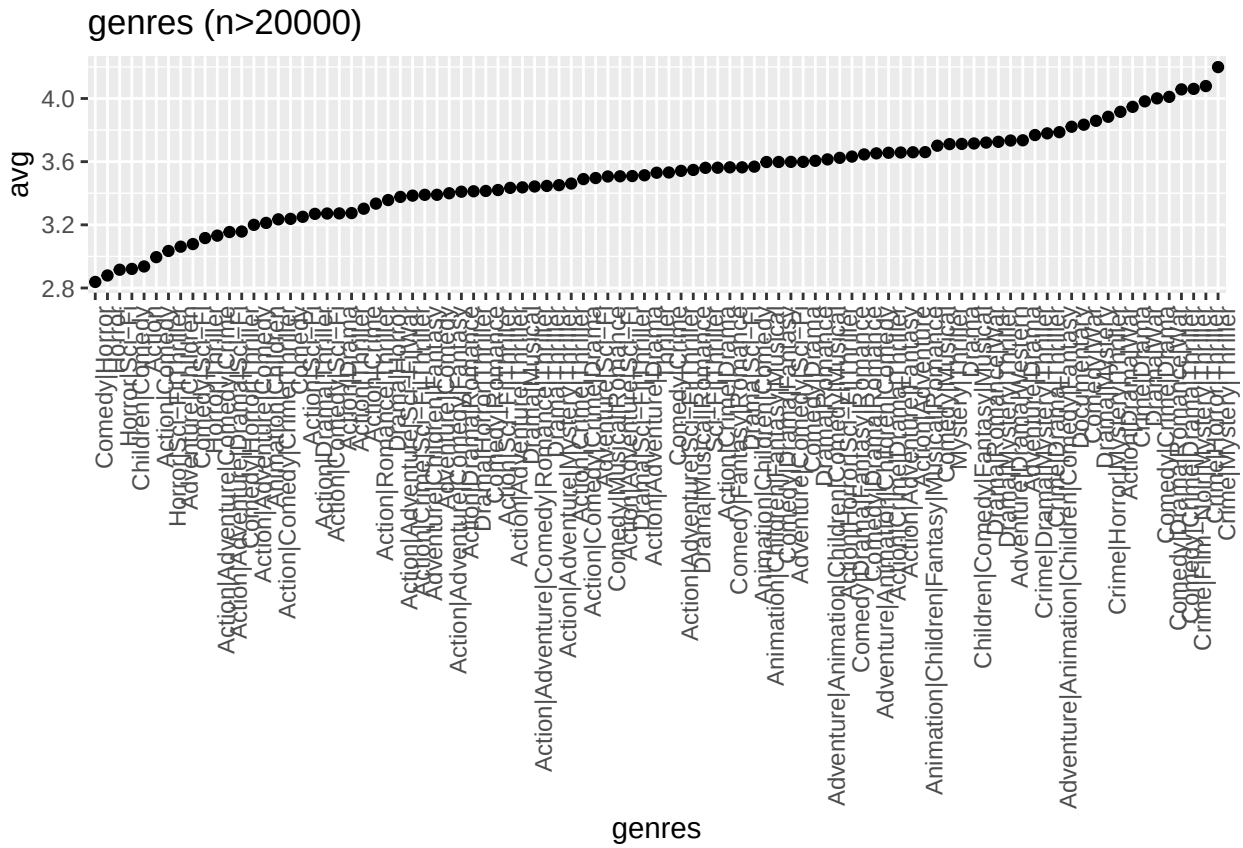
```
edx$genres %>% n_distinct()
```

```
## [1] 797
```

Here are top 10 genres in terms of high rating.

```
## # A tibble: 10 x 2
##    genres                                      ratings
##    <chr>                                         <dbl>
##  1 Animation|IMAX|Sci-Fi                          4.71
##  2 Drama|Film-Noir|Romance                        4.30
##  3 Action|Crime|Drama|IMAX                        4.30
##  4 Animation|Children|Comedy|Crime                4.28
##  5 Film-Noir|Mystery                              4.24
##  6 Crime|Film-Noir|Mystery                        4.22
##  7 Film-Noir|Romance|Thriller                     4.22
##  8 Crime|Film-Noir|Thriller                       4.21
##  9 Crime|Mystery|Thriller                         4.20
## 10 Action|Adventure|Comedy|Fantasy|Romance        4.20
```

Each genre consists of several words, such as “Action”, “Drama”, or“Crime”. Instead of splitting them, we try to deal with them as one combination, and plot relationship between rating and genre. To make it simple, genres which have more than 20000 ratings are plotted.



As in seen in the graph, some genres are rated very low (minimum approx. 2.8) and some are rated very high (maximum approx. more than 4.2). From this, we understand that genres (even though they are combined) affect ratings considerably.

Exploration summary From these studies, we found that “movieId”, “userId”, “genres”, and “release year (extracted from”title") are important factors deciding rating. In order to make models, we split the edx dataset into training set and test set.

To make sure we don’t include users and movies in the test set that do not appear in the training set, we remove these entries using the semi_join function

To assess models, we use the RMSE. It is denoted as ;

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

$y_{u,i} : the\ rating\ for\ movie\ i\ by\ user\ u \hat{y}_{u,i}\ : the\ predictionN : the\ numberof\ user/movie\ combinations$

Models, which will be described in the next chapter, are evaluated by the RMSE. We create R function for this purpose.

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

The model which minimize the RMSE will be applied to the validation dataset, and extract its RMSE .

Model building

```
mu <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

Naive model assuming the same rating for all movies

```
## [1] 1.060054
```

```
#[1] 1.060054
```

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

movie_avg <- train_set %>%
  group_by(movieId)%>%
  summarize(b_i =mean(rating -mu))

movie_effect_pred <- mu + test_set %>%
  left_join(movie_avg, by="movieId") %>%
  pull(b_i)

movie_effect_rmse <- RMSE(test_set$rating, movie_effect_pred)
movie_effect_rmse
```

2 movie effects

```
## [1] 0.9429615
```

```
#[1] 0.9429615
```

```
user_avg <- train_set %>%
  left_join(movie_avg, by= "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating- mu -b_i))

movie_user_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  mutate(pred = mu + b_i + b_u)%>%
  pull(pred)

movie_user_effect_rmse <- RMSE(test_set$rating, movie_user_effect_pred)
movie_user_effect_rmse
```

3 movie and user effects

```
## [1] 0.8646844
```

```
#[1] 0.8646844
```

```
g_avg <- train_set %>%
  left_join(movie_avg, by= "movieId")%>%
  left_join(user_avg, by="userId") %>%
  group_by(genres) %>%
  summarize(g = mean(rating- mu -b_i -b_u))

movie_user_genre_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  mutate(pred = mu + b_i + b_u + g)%>%
  pull(pred)

movie_user_genre_effect_rmse <- RMSE(test_set$rating,
                                     movie_user_genre_effect_pred)
movie_user_genre_effect_rmse
```

4 genre effects

```
## [1] 0.8643242
```

```
#[1] 0.8643242
```

```
y_avg <- train_set %>%
  left_join(movie_avg, by= "movieId")%>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  group_by(release_year) %>%
  summarize(y = mean(rating- mu -b_i -b_u -g))

movie_user_genre_year_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  left_join(y_avg, by="release_year")%>%
  mutate(pred = mu + b_i + b_u + g +y)%>%
  pull(pred)

movie_user_genre_year_effect_rmse <- RMSE(test_set$rating,
                                          movie_user_genre_year_effect_pred)
movie_user_genre_year_effect_rmse
```

5 release year effects

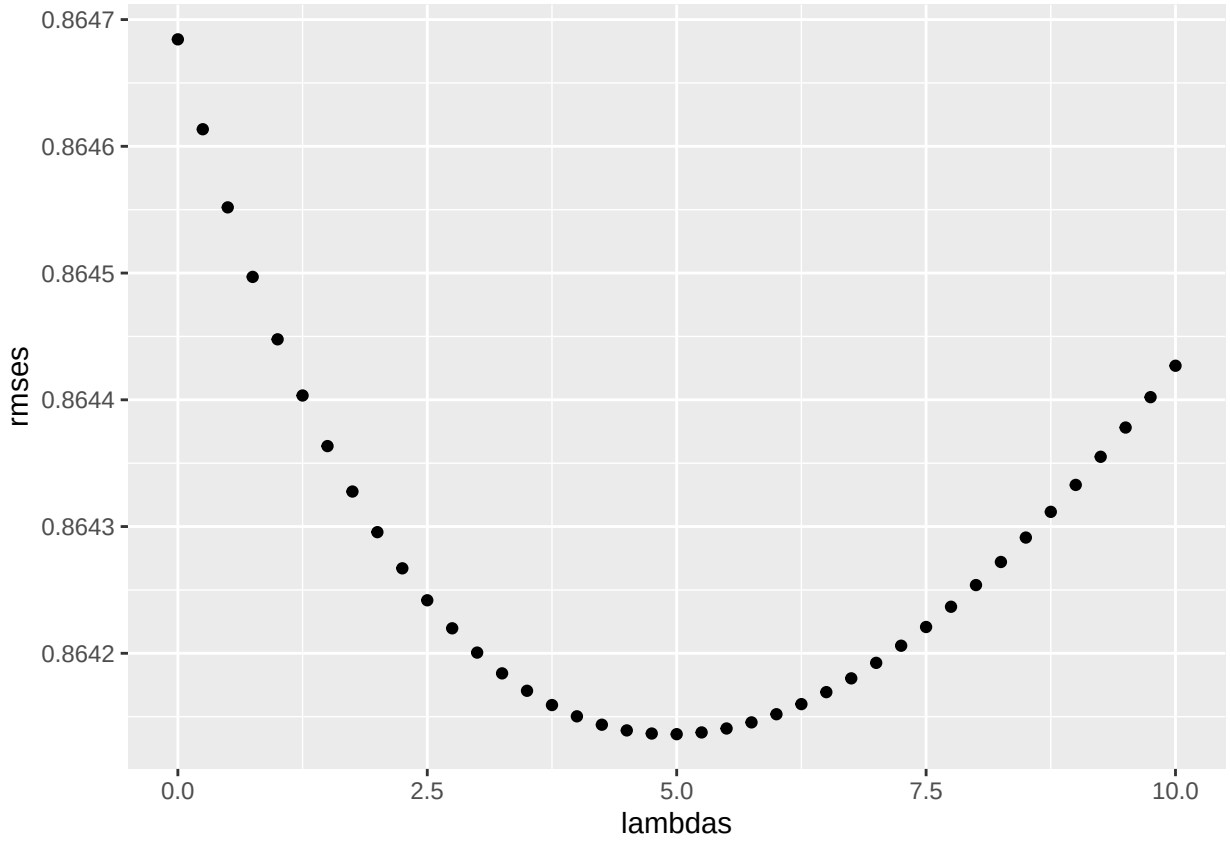
```
## [1] 0.8641262
```

```
#[1] 0.8641262
```

6 regularization (movie, user) introducing lambda

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

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  reg_movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  reg_user_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <- test_set %>%
    left_join(reg_movie_avg, by = "movieId") %>%
    left_join(reg_user_avg ,by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))

})
```

```
qplot(lambdas,rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda
```

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

```
#[1] 5
```

```
reg_movie_user_rmse <- rmses
reg_movie_user_rmse[5]
```

```
## [1] 0.8644477
```

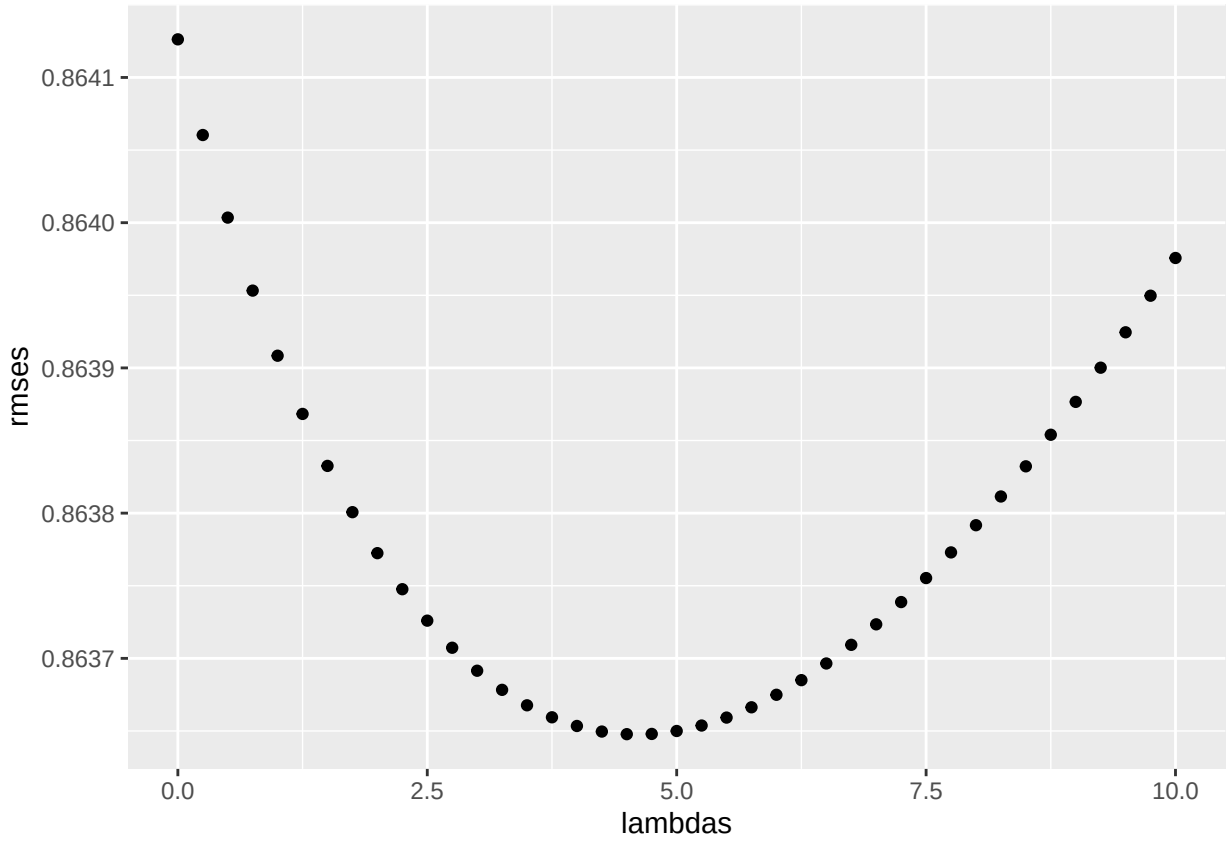
```
#[1] 0.8644477
```

7 regularization (movie, user,genres, release_year) calculating lambda

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

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  reg_movie_avg <- train_set %>%
```

```
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  reg_user_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  g_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    group_by(genres) %>%
    summarize(g = sum(rating - b_i -b_u - mu)/(n()+l))

    y_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    left_join(g_avg, by="genres") %>%
    group_by(release_year) %>%
    summarize(y = sum(rating - b_i -b_u -g - mu)/(n()+l))

  predicted_ratings <-test_set %>%
    left_join(reg_movie_avg, by = "movieId") %>%
    left_join(reg_user_avg, by = "userId") %>%
    left_join(g_avg, by="genres") %>%
    left_join(y_avg, by="release_year")%>%
    mutate(pred = mu + b_i + b_u + g + y) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
```

```
qplot(lambdas,rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.5
```

```
#[1] 4.5

min(rmses)
```

```
## [1] 0.8636478
```

```
#[1] 0.8636478
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method=
                                 "Reg Movie User Genre Release Year Effects Model",
                                 MSE =rmses[4.5]))%>% as.data.frame()
```

8 model summary

method	RMSE
Rating Average	1.0600537
Movie Effects Model	0.9429615
Movie User Effects Model	0.8646844
Movie User Genre Effects Model	0.8643242
Movie User Genre Release Year Effects Model	0.8641262
Reg Movie User Effects Model	0.8644477
Reg Movie User Genre Release Year Effects Model	0.8639532

final model "Reg Movie User Genre Release Year Effects Model" is proved to extract the least RMSE Value.

Evaluation

validation calculating lambda add and cut columns in the process of training

```
validation <- validation %>% mutate(release_year =
                                      as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% select(-timestamp)
```

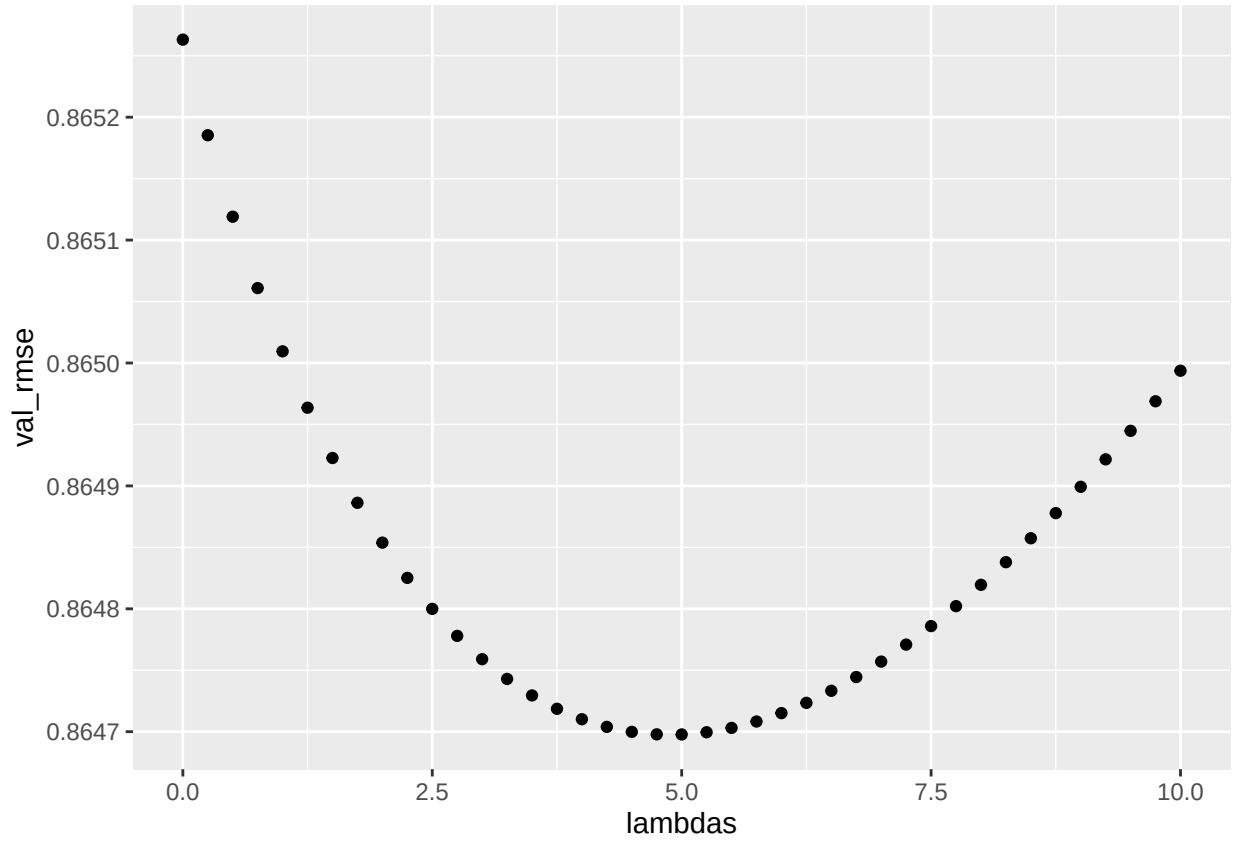
define lambdas as a variable

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

#calculate rmse using "Reg Movie User Genre Release Year Effects Model"

val_rmse <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  reg_movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  reg_user_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  g_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    group_by(genres) %>%
    summarize(g = sum(rating - b_i -b_u - mu)/(n()+l))

  y_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    left_join(g_avg, by="genres") %>%
    group_by(release_year) %>%
    summarize(y = sum(rating - b_i -b_u -g - mu)/(n()+l))
```

```
  val_pred <-validation %>%
    left_join(reg_movie_avg, by = "movieId") %>%
    left_join(reg_user_avg, by = "userId") %>%
    left_join(g_avg, by="genres") %>%
    left_join(y_avg, by="release_year")%>%
    mutate(pred = mu + b_i + b_u + g + y) %>%
    pull(pred)

    #replace NA with mu(the average rating in the training)
    val_pred <-replace(val_pred, is.na(val_pred),mu)

    return(RMSE(val_pred,validation$rating))
})
```

plot lambdas and rmses

```
qplot(lambdas,val_rmse)
```




```
#find lambda and rmse which indicate the least value
lambda <- lambdas[which.min(val_rmse)]
lambda
```

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

```
#[1] 5
```

```
min(val_rmse)
```

```
## [1] 0.8646978
```

```
##[1] 0.8646978
```

Conclusion