title: MovieLens Project Submission

by JING JING SUN

Aug 2, 2021

I.Introduction

This report is all learners the capstone project of the EdX course 'HarvardX: PH125.9x Data Science: Capstone'. The MovieLen data set we uses for our project comes from NetflixTM sponsored a competition, but EdX course create a smaller one—-10M version of the MovieLens dataset for our project.
EdX course have provide code let student generate 2 data set from 10M version of the MovieLens dataset: the edx set is training set, we will use it to develop our algorithm; the validation set, we will use it for the final test of our final algorithm.

The goal of this project is let student creating our own recommendation system using the knowledge we have learned from R course series and help us get the ability to solve real world problem.

Running the code course provided, we got 2 data set—edx and validation. I created new columns of year_rating, premier and s_geners which split each gener into separate row. Then separate edx data set into train_set and test_set. I created models based on Matrix factorization with predictor movie, user, rating year and genres. And found if I created model based on all these predictors together, I got better prediction. Then I introduced regularization to model building to is to constrain the total variability of the effect sizes. I do get improvement of residual mean square error (rmse) with a value of 0.8569315. I use this model as the final one. Then I used edx as a training set and validation as the test set to test the final model, I got a rmse= 0.8622863.

II. Method/Analysis

Input data set and clean the data with the code provided by course

Since the code is not written by individual student, I will not show the running result here.

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)
```

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")
```

create data frame:

```r
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

```r
set.seed(1, sample.kind="Rounding")
```

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

```r
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

```r
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

Add rows removed from validation set back into edx set

```r
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

know our data

```r
library(tidyverse)
library(dslabs)
library(matrixStats)
```

```
##
## Attaching package: 'matrixStats'
```

```
## The following object is masked from 'package:dplyr':
##
##     count
```

```
library(caret)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(dplyr)
```

```
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
```

```
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|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
dim(edx)
```

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

```
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
##
##
##
```

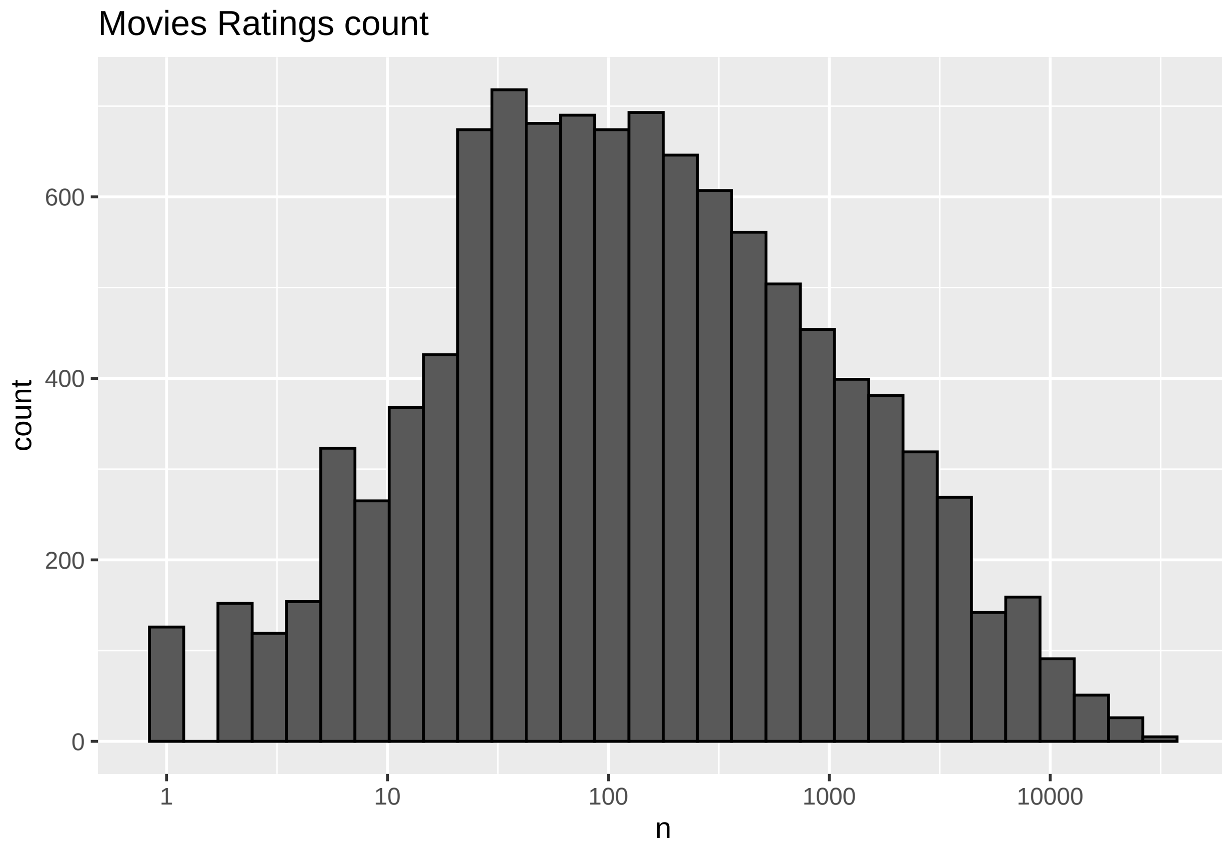
how many user and how many movies in this dataset

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

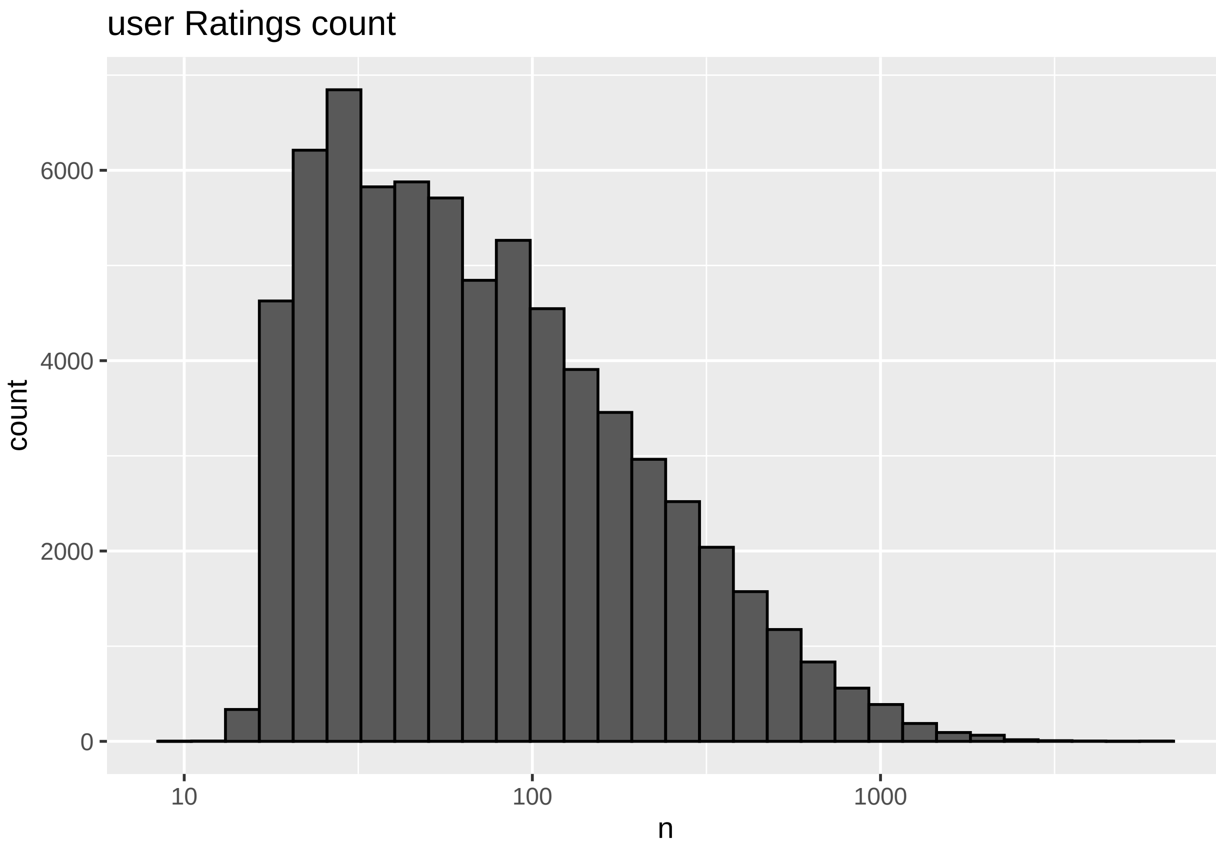
Distribution of Movie Ratings

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies Ratings count")
```



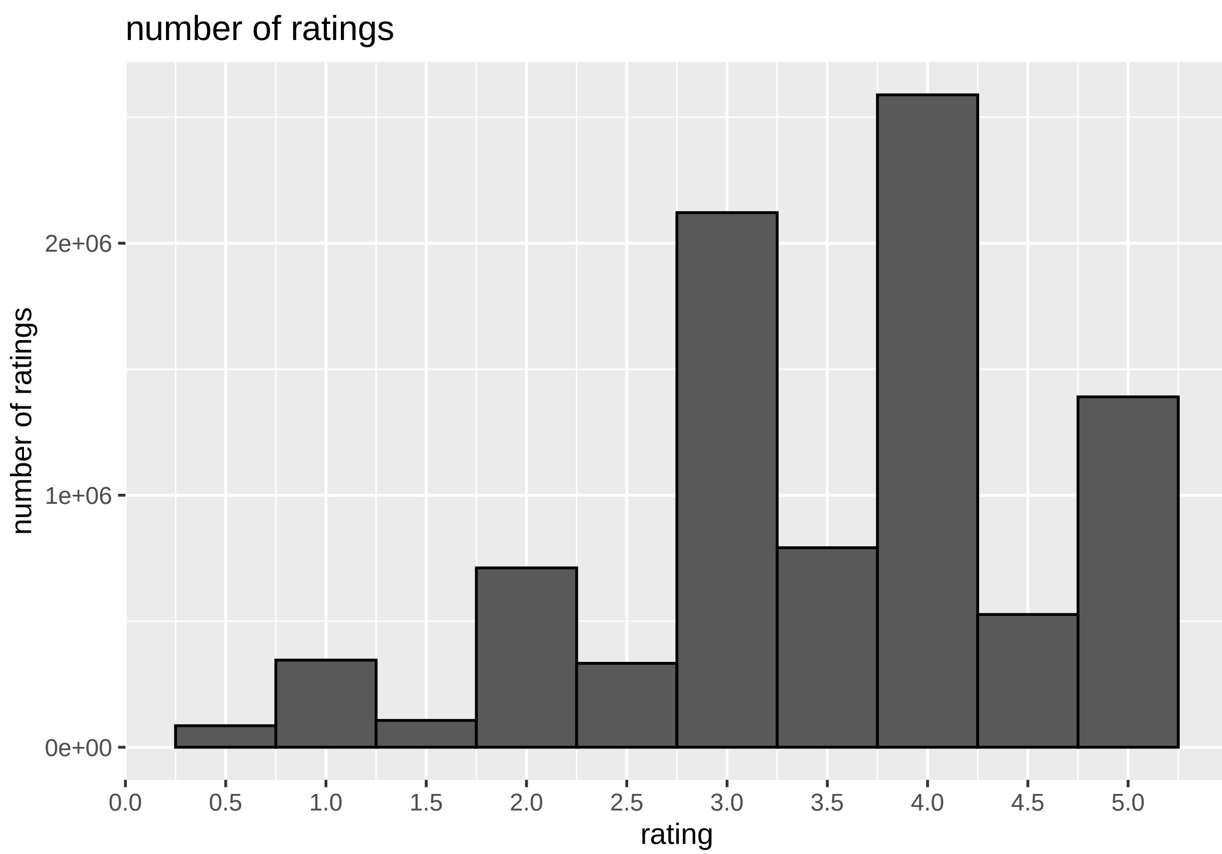
Distribution of user Ratings

```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("user Ratings count")
```



histogram of rating star number

```
edx%>% ggplot(aes(rating)) +
  geom_histogram( binwidth = 0.5,color = "black") +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  labs(x="rating", y="number of ratings") +
  ggtitle("number of ratings")
```



add more predictors to our data

After building our model based on moive and user as predictors, I am not satified with the result, so I go back create more predictors ——-year_rated: the year that user give rate; premier: the premier data of movie; s_genres: split the genres into separate lines. Then add all thess factors as predicters to create algorithm.

add rating year

```
edx <- mutate(edx, year_rated = year(as_datetime(timestamp)))
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 year_rated
## 1:                Comedy|Romance       1996
## 2:         Action|Crime|Thriller       1996
## 3:  Action|Drama|Sci-Fi|Thriller       1996
```

```
## 4:       Action|Adventure|Sci-Fi       1996
## 5: Action|Adventure|Drama|Sci-Fi       1996
## 6:       Children|Comedy|Fantasy       1996
```

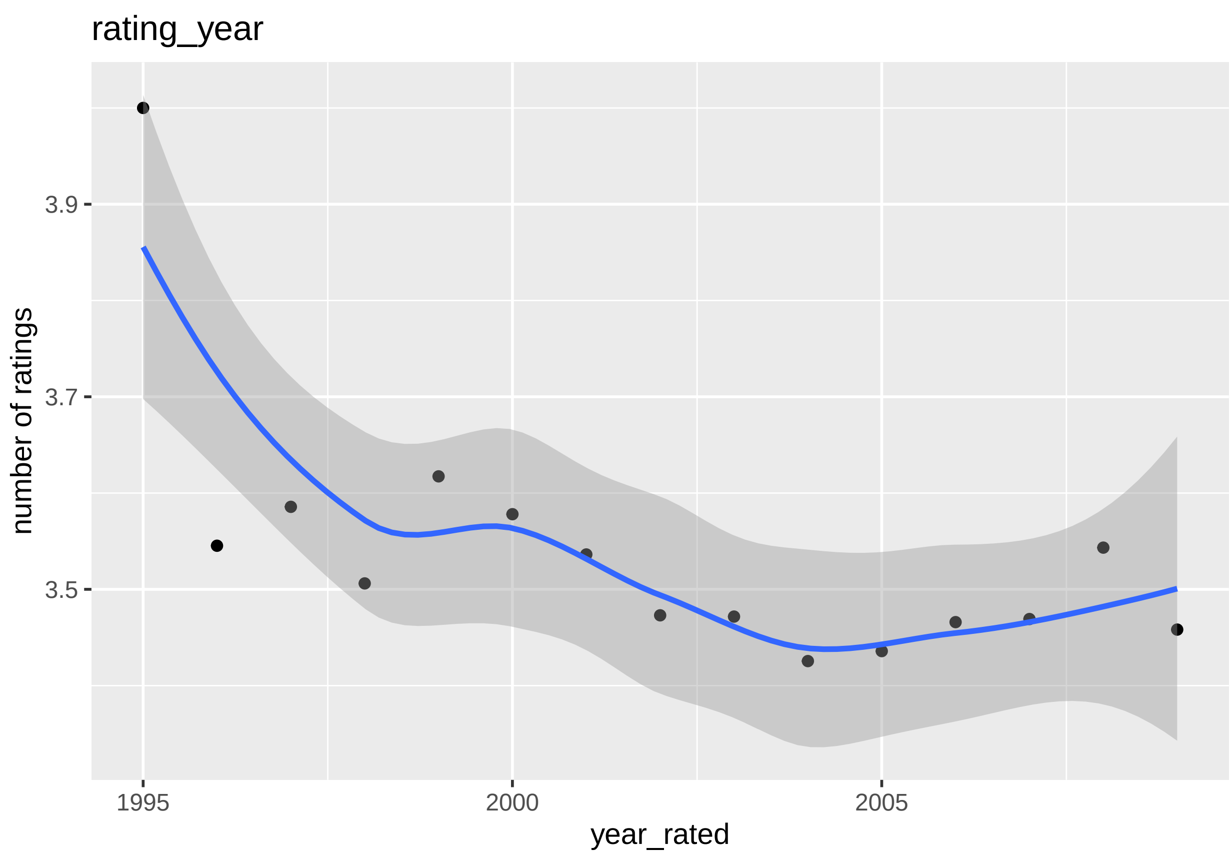
```
validation <- mutate(validation, year_rated = year(as_datetime(timestamp)))
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                  title
## 1:                                Dumb & Dumber (1994)
## 2:                                Jurassic Park (1993)
## 3:                                   Home Alone (1990)
## 4:                                      Rob Roy (1995)
## 5:                               Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres year_rated
## 1:                                Comedy       1996
## 2:       Action|Adventure|Sci-Fi|Thriller       1996
## 3:                       Children|Comedy       1996
## 4:               Action|Drama|Romance|War       1997
## 5:                           Crime|Drama       1997
## 6: Action|Adventure|Horror|Sci-Fi|Thriller       1997
```

rating vs year_rated

```
edx %>%
group_by(year_rated) %>%
summarize(rating = mean(rating)) %>%
ggplot(aes(year_rated, rating)) +
geom_point() +
geom_smooth() +
ggtitle("rating_year") +
xlab("year_rated") + ylab("number of ratings")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Extract the premier date

```
premier <- stringi::stri_extract(edx$title, regex = "(\\d{4})", comments = TRUE ) %>% as.numeric()
head(premier)
```

```
## [1] 1992 1995 1995 1994 1994 1994
```

```
vpremier <- stringi::stri_extract(validation$title, regex = "(\\d{4})", comments = TRUE ) %>% as.numeri
head(vpremier)
```

```
## [1] 1994 1993 1990 1995 1972 1997
```

Add the premier date

```
edx <- edx %>% mutate(premier_date = premier)
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)
```

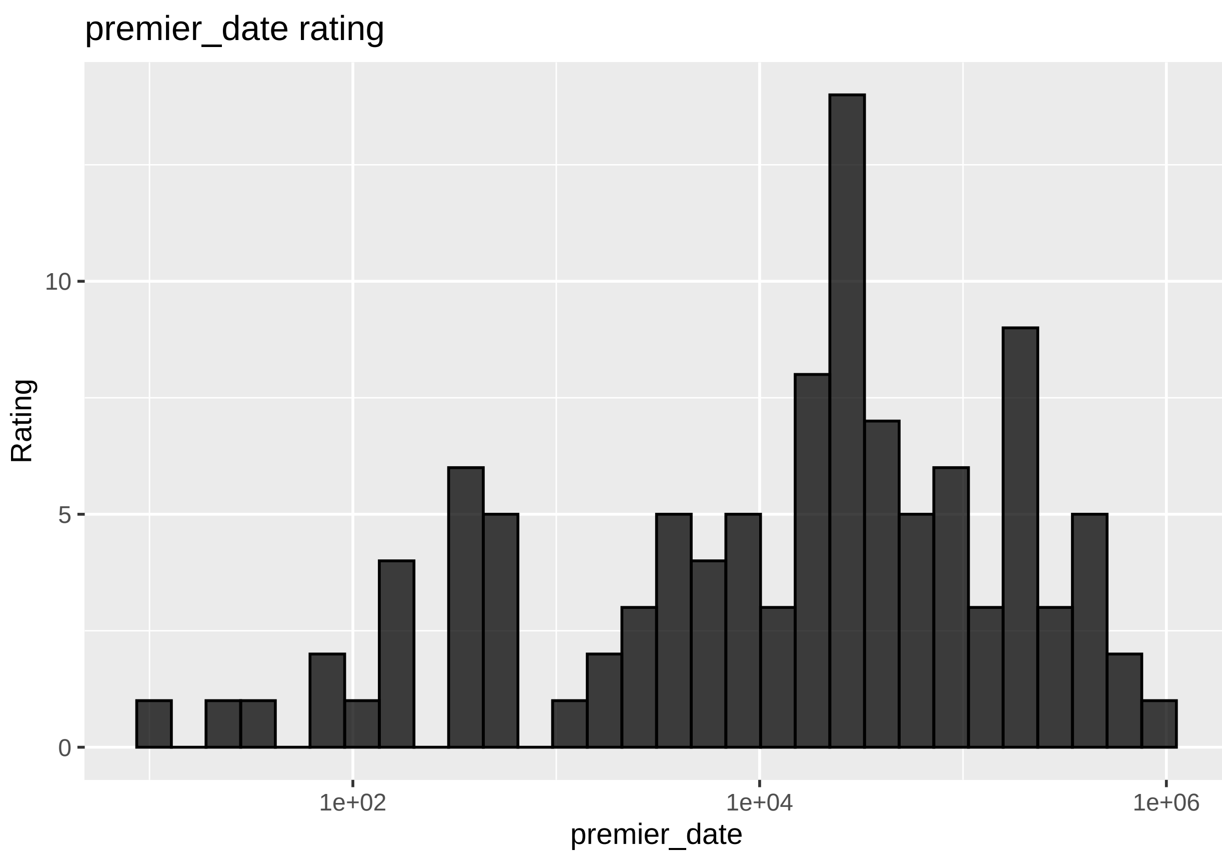
```
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres year_rated premier_date
## 1:                 Comedy|Romance       1996         1992
## 2:          Action|Crime|Thriller       1996         1995
## 3:  Action|Drama|Sci-Fi|Thriller       1996         1995
## 4:        Action|Adventure|Sci-Fi       1996         1994
## 5: Action|Adventure|Drama|Sci-Fi       1996         1994
## 6:        Children|Comedy|Fantasy       1996         1994
```

```
validation <- validation %>% mutate(premier_date = vpremier)
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                     title
## 1:                                   Dumb & Dumber (1994)
## 2:                                   Jurassic Park (1993)
## 3:                                      Home Alone (1990)
## 4:                                         Rob Roy (1995)
## 5:                                  Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres year_rated premier_date
## 1:                                Comedy       1996         1994
## 2:        Action|Adventure|Sci-Fi|Thriller       1996         1993
## 3:                       Children|Comedy       1996         1990
## 4:               Action|Drama|Romance|War       1997         1995
## 5:                           Crime|Drama       1997         1972
## 6: Action|Adventure|Horror|Sci-Fi|Thriller       1997         1997
```

plot premier date vs rating

```
edx %>%
  group_by(premier_date) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "black", alpha = 0.75) +
  scale_x_log10() +
  ggtitle("premier_date rating") +
  xlab("premier_date") + ylab("Rating")
```



copy column genres with a name of s_genres

```
edx <- edx %>% mutate(s_genres = genres)
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 year_rated premier_date
## 1:                Comedy|Romance       1996         1992
## 2:         Action|Crime|Thriller       1996         1995
## 3:  Action|Drama|Sci-Fi|Thriller       1996         1995
## 4:       Action|Adventure|Sci-Fi       1996         1994
## 5: Action|Adventure|Drama|Sci-Fi       1996         1994
## 6:       Children|Comedy|Fantasy       1996         1994
##                         s_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
```

```
validation <- validation %>% mutate(s_genres = genres)
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                 title
## 1:                                 Dumb & Dumber (1994)
## 2:                                 Jurassic Park (1993)
## 3:                                    Home Alone (1990)
## 4:                                       Rob Roy (1995)
## 5:                                Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres year_rated premier_date
## 1:                                Comedy       1996         1994
## 2:        Action|Adventure|Sci-Fi|Thriller       1996         1993
## 3:                         Children|Comedy       1996         1990
## 4:                Action|Drama|Romance|War       1997         1995
## 5:                             Crime|Drama       1997         1972
## 6: Action|Adventure|Horror|Sci-Fi|Thriller       1997         1997
##                                 s_genres
## 1:                                Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                         Children|Comedy
## 4:                Action|Drama|Romance|War
## 5:                             Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

split s_genres column

```
edx <-edx %>% separate_rows(s_genres, sep ="\\|")
head(edx)
```

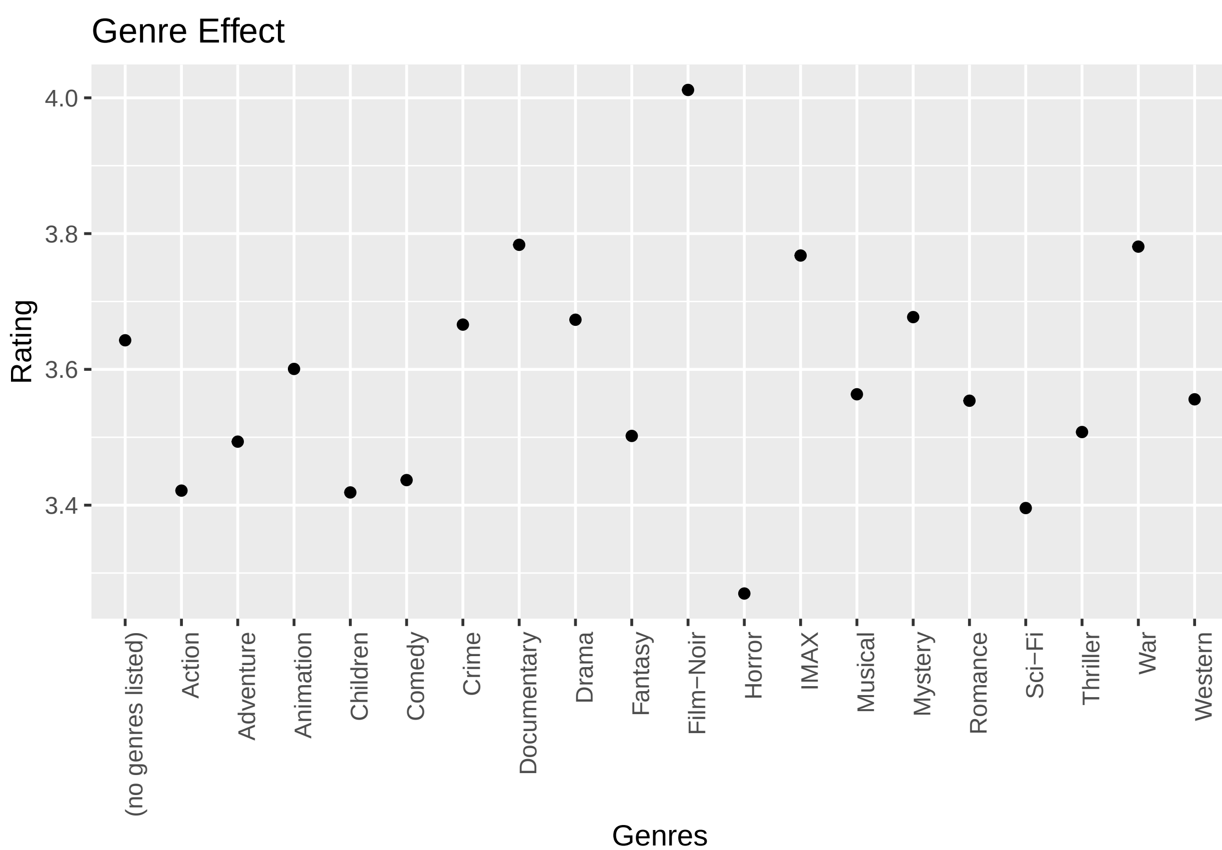
```
## # A tibble: 6 x 9
##   userId movieId rating timestamp title  genres year_rated premier_date s_genres
##    <int>   <dbl>  <dbl>     <int> <chr>  <chr>       <dbl>        <dbl> <chr>
## 1      1     122      5 838985046 Boome~ Comed~       1996         1992 Comedy
## 2      1     122      5 838985046 Boome~ Comed~       1996         1992 Romance
## 3      1     185      5 838983525 Net, ~ Actio~       1996         1995 Action
## 4      1     185      5 838983525 Net, ~ Actio~       1996         1995 Crime
## 5      1     185      5 838983525 Net, ~ Actio~       1996         1995 Thriller
## 6      1     292      5 838983421 Outbr~ Actio~       1996         1995 Action
```

```
validation <-validation %>% separate_rows(s_genres, sep ="\\|")
head(validation)
```

```
## # A tibble: 6 x 9
##   userId movieId rating timestamp title  genres year_rated premier_date s_genres
##    <int>   <dbl>  <dbl>     <int> <chr>  <chr>       <dbl>        <dbl> <chr>
## 1      1     231      5 838983392 Dumb ~ Comedy       1996         1994 Comedy
## 2      1     480      5 838983653 Juras~ Actio~       1996         1993 Action
## 3      1     480      5 838983653 Juras~ Actio~       1996         1993 Adventu~
## 4      1     480      5 838983653 Juras~ Actio~       1996         1993 Sci-Fi
## 5      1     480      5 838983653 Juras~ Actio~       1996         1993 Thriller
## 6      1     586      5 838984068 Home ~ Child~       1996         1990 Children
```

plot Genre Effect vs rating

```
edx %>% group_by(s_genres) %>%
  summarize(n = n(), avg = mean(rating)) %>%
  ggplot(aes(x = s_genres, y = avg)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Genre Effect") +
  xlab("Genres") + ylab("Rating")
```



The plot shows ratings are very different among genres.

Build the Recommendation System

Based on the course instruction, we should not use validation during the model building, so I split edx data set into train_set and test_set, and use them to build models.

split edx data into separate training and test

```
set.seed(1, sample.kind="Rounding")
```

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

```
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE) # set 20% of edx as test set
train_set <- edx[-test_index,]
head(train_set )
```

```
## # A tibble: 6 x 9
##   userId movieId rating timestamp title  genres year_rated premier_date s_genres
##    <int>   <dbl>  <dbl>     <int> <chr>  <chr>       <dbl>        <dbl> <chr>
## 1      1     122      5 838985046 Boome~ Comed~       1996         1992 Comedy
## 2      1     122      5 838985046 Boome~ Comed~       1996         1992 Romance
## 3      1     185      5 838983525 Net, ~ Actio~       1996         1995 Thriller
## 4      1     292      5 838983421 Outbr~ Actio~       1996         1995 Action
## 5      1     292      5 838983421 Outbr~ Actio~       1996         1995 Drama
## 6      1     292      5 838983421 Outbr~ Actio~       1996         1995 Sci-Fi
```

```
test_set <- edx[test_index,]
```

make sure all the user and movie and years found in test are in the train_set

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
head(test_set)
```

```
## # A tibble: 6 x 9
##   userId movieId rating timestamp title  genres year_rated premier_date s_genres
##    <int>   <dbl>  <dbl>     <int> <chr>  <chr>       <dbl>        <dbl> <chr>
## 1      1     185      5 838983525 Net, ~ Actio~       1996         1995 Action
## 2      1     185      5 838983525 Net, ~ Actio~       1996         1995 Crime
## 3      1     316      5 838983392 Starg~ Actio~       1996         1994 Adventu~
## 4      1     329      5 838983392 Star ~ Actio~       1996         1994 Sci-Fi
## 5      1     364      5 838983707 Lion ~ Adven~       1996         1994 Children
## 6      1     377      5 838983834 Speed~ Actio~       1996         1994 Thriller
```

Build models

build model 1, assumes the same rating for all movies and all users

compute the average

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

```
## [1] 3.526972
```

rmse of same rating of all movies and all users model

```
all_average_rmse <- RMSE(test_set$rating, mu_hat)
all_average_rmse
```

```
## [1] 1.051984
```

It is a high rmse, so it is not a good model. We must add more predictors to algorithm

store RMSE to data.frame "rmse_result"

```
rmse_results <- data_frame(method = "all_average",  RMSE = all_average_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
rmse_results
```

```
## # A tibble: 1 x 2
##   method       RMSE
##   <chr>       <dbl>
## 1 all_average  1.05
```

build mode2 movie effects

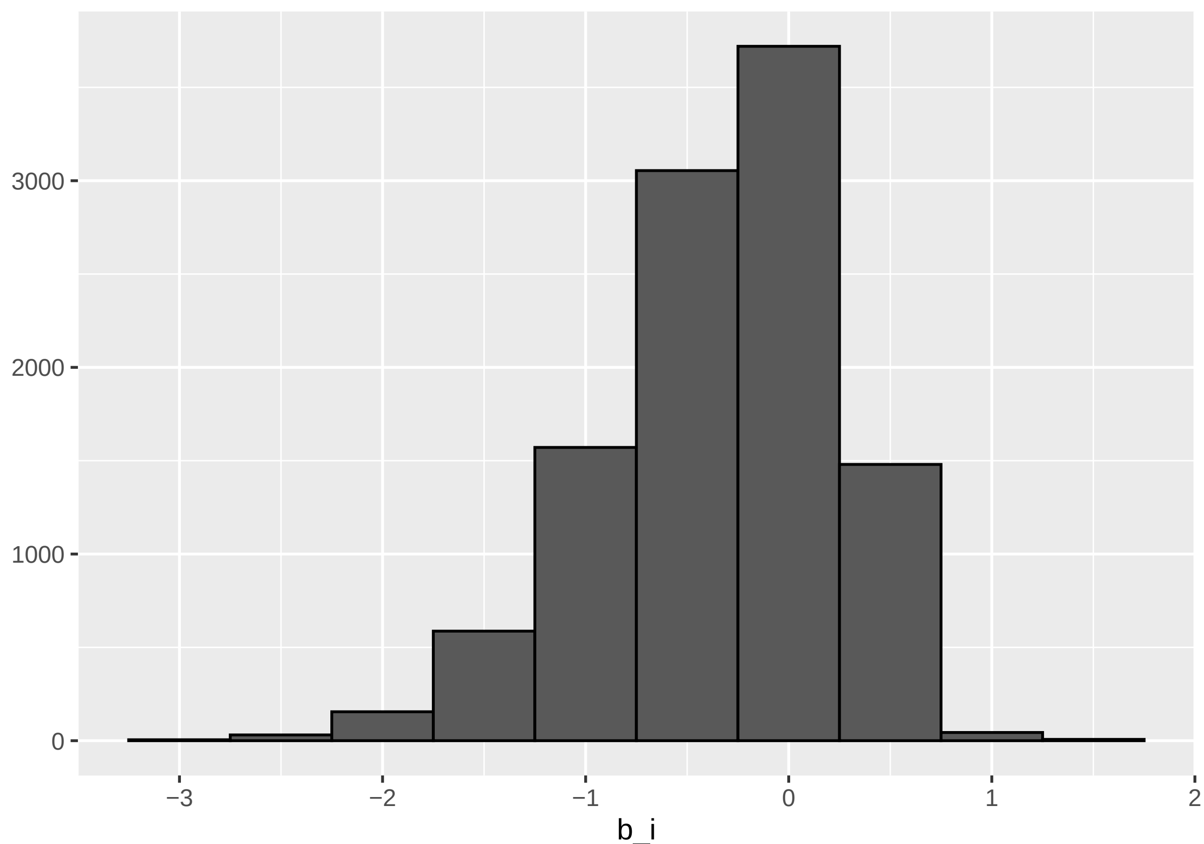
compute the average rating based on movies

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
movie_avgs
```

```
## # A tibble: 10,654 x 2
##    movieId    b_i
##      <dbl>  <dbl>
##  1       1  0.399
##  2       2 -0.323
##  3       3 -0.374
##  4       4 -0.656
##  5       5 -0.454
##  6       6  0.288
##  7       7 -0.159
##  8       8 -0.402
##  9       9 -0.528
## 10      10 -0.101
## # ... with 10,644 more rows
```

see distribution of bi(movie effect)

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



predicted rating of all average + movies effect model

```
movies_predicted_ratings <- mu_hat + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

head(movies_predicted_ratings)
```

```
## [1] 3.127485 3.127485 3.353322 3.337506 3.752519 3.526375
```

RMSE of all average + movies effect

```
movies_rmse <- RMSE(movies_predicted_ratings, test_set$rating)
movies_rmse
```

```
## [1] 0.9409964
```

Based on the previous data visualization, we see rationg count are different among movies. adding movies as a predictor, we do decrease rmse from 1.0519843 to 0.9409964

#add moviesBi_rmse to rmse_result data frame

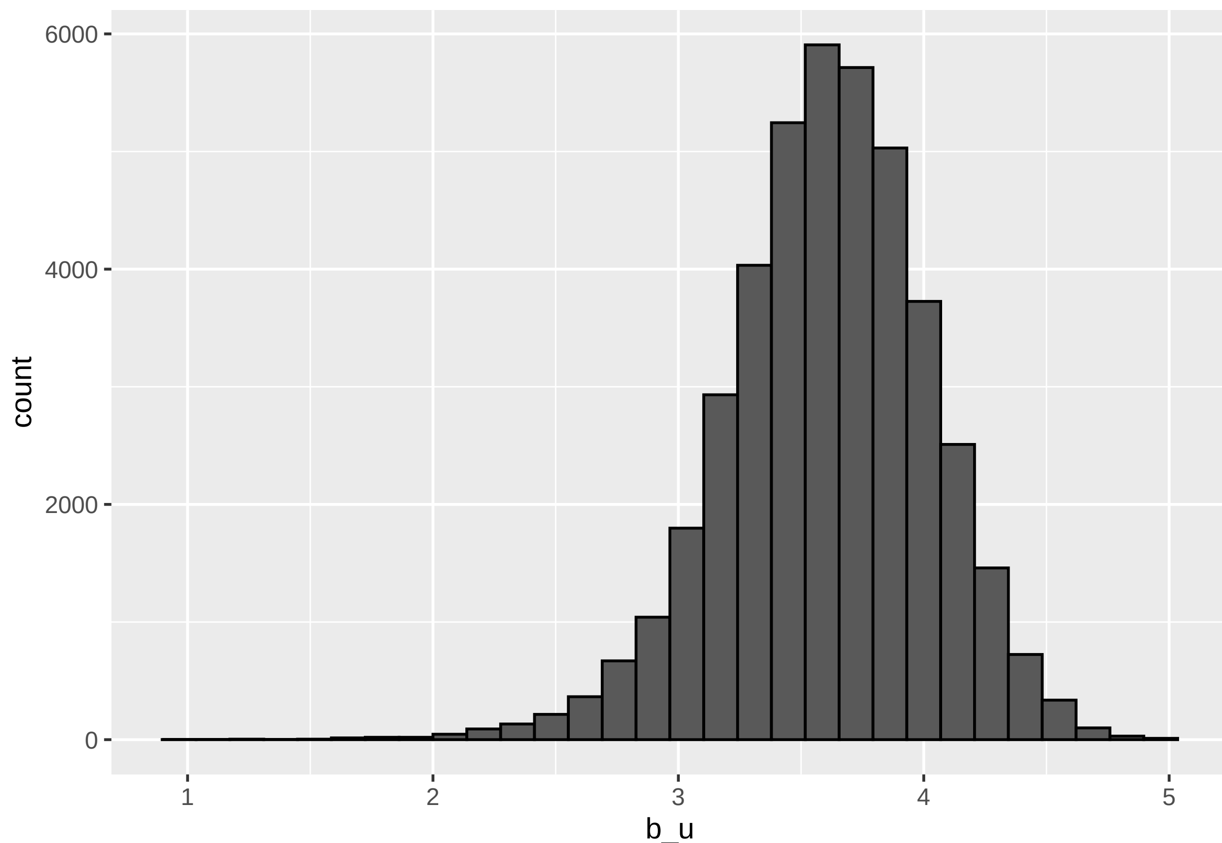
```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Model",
                                     RMSE = movies_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method       RMSE
##   <chr>       <dbl>
## 1 all_average 1.05
## 2 Movie Model 0.941
```

Modeling 3, movies+user effects

see distribution of bu (user effect) that user rated more than 100 movies

```
train_set %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



compute the average rating for user that have rated 100 or more movies

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

head(user_avgs)
```

```
## # A tibble: 6 x 2
##   userId     b_u
##    <int>   <dbl>
## 1      1  1.63
## 2      2 -0.210
## 3      3  0.301
## 4      4  0.789
## 5      5 -0.0153
## 6      6  0.325
```

construct predictors with movies+user effects model

```r
user_predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

head(user_predicted_ratings)
```

```
## [1] 4.758490 4.758490 4.984327 4.968511 5.383524 5.157380
```

RMSE of movies+user effects model

```r
movies_user_rmse <- RMSE(user_predicted_ratings, test_set$rating)
movies_user_rmse
```

```
## [1] 0.8574998
```

By adding user as a predictor, we got big improvement of our prediction model with rmse = 0.8574998

add movies_user_rmse to rmse_result data frame

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="movies and user model",
                                     RMSE = movies_user_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                 RMSE
##   <chr>                 <dbl>
## 1 all_average           1.05
## 2 Movie Model           0.941
## 3 movies and user model 0.857
```

Modeling 4, movies+user+rating year effects

compute the average rating based on year_rated

```r
year_avgs<- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(year_rated) %>%
  summarize(b_y = mean(rating - mu_hat - b_i-b_u))

head(year_avgs)
```

```
## # A tibble: 6 x 2
##   year_rated       b_y
##        <dbl>     <dbl>
## 1       1995  0.489
## 2       1996 -0.000439
## 3       1997 -0.000106
## 4       1998 -0.00153
## 5       1999  0.00358
## 6       2000  0.000996
```

construct predictors with movies+user+rating year effects model

```
year_predicted_ratings <- test_set%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_rated') %>%
  mutate(pred = mu_hat + b_i + b_u + b_y) %>%
  .$pred
head(year_predicted_ratings)
```

```
## [1] 4.758051 4.758051 4.983888 4.968072 5.383085 5.156941
```

RMSE of movies+user+rating year effects model

```
movies_user_year_rmse <- RMSE(year_predicted_ratings , test_set$rating)
movies_user_year_rmse
```

```
## [1] 0.8574946
```

The result show year_rated doed not improve prediction much, it is not a good predictor. Since it do decrease rmse a little, we will still keep it in the aglorium.

add movies_user_year_rmse to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="movies+user+yearrating model",
                                     RMSE = movies_user_year_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                        RMSE
##   <chr>                        <dbl>
## 1 all_average                  1.05
## 2 Movie Model                  0.941
## 3 movies and user model        0.857
## 4 movies+user+yearrating model 0.857
```

Modeling 5, movies+user+rating year+premier effects

compute the average rating based on movies+user+rating year+premier effects model

```
premier_avgs<- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_rated') %>%
  group_by(premier_date) %>%
  summarize(b_p = mean(rating - mu_hat - b_i - b_u - b_y))

head(premier_avgs)
```

```
## # A tibble: 6 x 2
##   premier_date     b_p
##          <dbl>   <dbl>
## 1         1000  0.0638
## 2         1138  0.0679
## 3         1408  0.0223
## 4         1492  0.112
## 5         1600 -0.0242
## 6         1732  0.130
```

construct predictors with movies+user+rating year+premier effects model

```
premier_predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_rated') %>%
  left_join(premier_avgs, by='premier_date') %>%
  mutate(pred = mu_hat + b_i + b_u + b_y +b_p) %>%
  .$pred

head(premier_predicted_ratings)
```

```
## [1] 4.720659 4.720659 4.949278 4.933462 5.348475 5.122330
```

RMSE of movies+user+rating year+premier effects model

```
movies_user_year_premier_rmse <- RMSE(premier_predicted_ratings, test_set$rating)
movies_user_year_premier_rmse
```

```
## [1] 0.8571341
```

add movies__user__year__premier__rmse to rmse__result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="movies+user+yearrating+premier model",
                                     RMSE = movies_user_year_premier_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                               RMSE
##   <chr>                               <dbl>
## 1 all_average                         1.05
## 2 Movie Model                         0.941
## 3 movies and user model               0.857
## 4 movies+user+yearrating model        0.857
## 5 movies+user+yearrating+premier model 0.857
```

Modeling 6, movies+user+rating year+premier+genres effects

compute the average rating based on movies+user+rating year+premier+genres effects model

```
genres_avgs<- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year_rated') %>%
  left_join(premier_avgs, by='premier_date') %>%
  group_by(s_genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u - b_y - b_p))

head(genres_avgs)
```

```
## # A tibble: 6 x 2
##   s_genres                b_g
##   <chr>                 <dbl>
## 1 (no genres listed)  0.229
## 2 Action             -0.0106
## 3 Adventure          -0.0149
## 4 Animation          -0.0164
## 5 Children           -0.0238
## 6 Comedy              0.00291
```

construct predictors with movies+user+rating year+premier+genres effects model

```
genres_predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
```

```
  left_join(year_avgs, by='year_rated') %>%
  left_join(premier_avgs, by='premier_date') %>%
  left_join(genres_avgs, by='s_genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_y +b_p + b_g) %>%
  .$pred

head(genres_predicted_ratings)
```

```
## [1] 4.710022 4.732424 4.934373 4.919984 5.324642 5.121680
```

RMSE of movies+user+rating year+premier+genres effects model

```
movies_user_year_premier_genres_rmse <- RMSE(genres_predicted_ratings, test_set$rating)
movies_user_year_premier_genres_rmse
```

```
## [1] 0.8570489
```

Even we have add all possible predictors, the rmse = 0.8570489 seems not good enough.

add movies_user_year_premier_genres_rmse to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="movies+user+yearrating+premier+genres model",
                                     RMSE = movies_user_year_premier_genres_rmse))
rmse_results
```

```
## # A tibble: 6 x 2
##   method                                      RMSE
##   <chr>                                      <dbl>
## 1 all_average                                1.05
## 2 Movie Model                                0.941
## 3 movies and user model                      0.857
## 4 movies+user+yearrating model               0.857
## 5 movies+user+yearrating+premier model       0.857
## 6 movies+user+yearrating+premier+genres model 0.857
```

In the data exploration part, we found some movies have very few number of ratings, and some users rate very few movies. this small sample size will cause large estimated error. In the following model,I will try regularication to reduce variability of the effect by penalizing large effects that come from small samples.

** Regularization effect model**

Modeling 7, Regularized_Movie_Model

add Penalized least squares =3

I randomly pick a lambda =3

```r
lambda <- 3

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

compute the average rating based on Regularized_Movie_Model

```r
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_ir = sum(rating - mu)/(n()+lambda), n_i = n())
```

construct predictors with Regularized_Movie_Model

```r
movie_reg_predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_ir) %>%
  pull(pred)

head(movie_reg_predicted_ratings)
```

```
## [1] 3.127522 3.127522 3.353335 3.337518 3.752510 3.526375
```

RMSE of Regularized_Movie_Model

```r
Regularized_Movie_Model<-RMSE(movie_reg_predicted_ratings , test_set$rating)
Regularized_Movie_Model
```

```
## [1] 0.9409851
```

Rmse is too large. I will try different lambda.

add rmse of Regularized_Movie_Model to rmse_result data frame

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized_Movie_Model",
                                     RMSE = Regularized_Movie_Model))
rmse_results
```

```
## # A tibble: 7 x 2
##   method                                       RMSE
##   <chr>                                       <dbl>
## 1 all_average                                 1.05
## 2 Movie Model                                 0.941
## 3 movies and user model                       0.857
## 4 movies+user+yearrating model                0.857
## 5 movies+user+yearrating+premier model        0.857
## 6 movies+user+yearrating+premier+genres model 0.857
## 7 Regularized_Movie_Model                     0.941
```

** Movie_reg_lambda_d model–Modeling 8, Regularized_Movie_Model with different tuning**

add Penalized least squares with different tuning

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

```
movie_reg_sum_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

head(movie_reg_sum_avgs)
```

```
## # A tibble: 6 x 3
##   movieId      s   n_i
##     <dbl>  <dbl> <int>
## 1       1 38109. 95427
## 2       2 -8355. 25842
## 3       3 -4198. 11222
## 4       4 -2481.  3784
## 5       5 -2332.  5139
## 6       6  8530. 29596
```

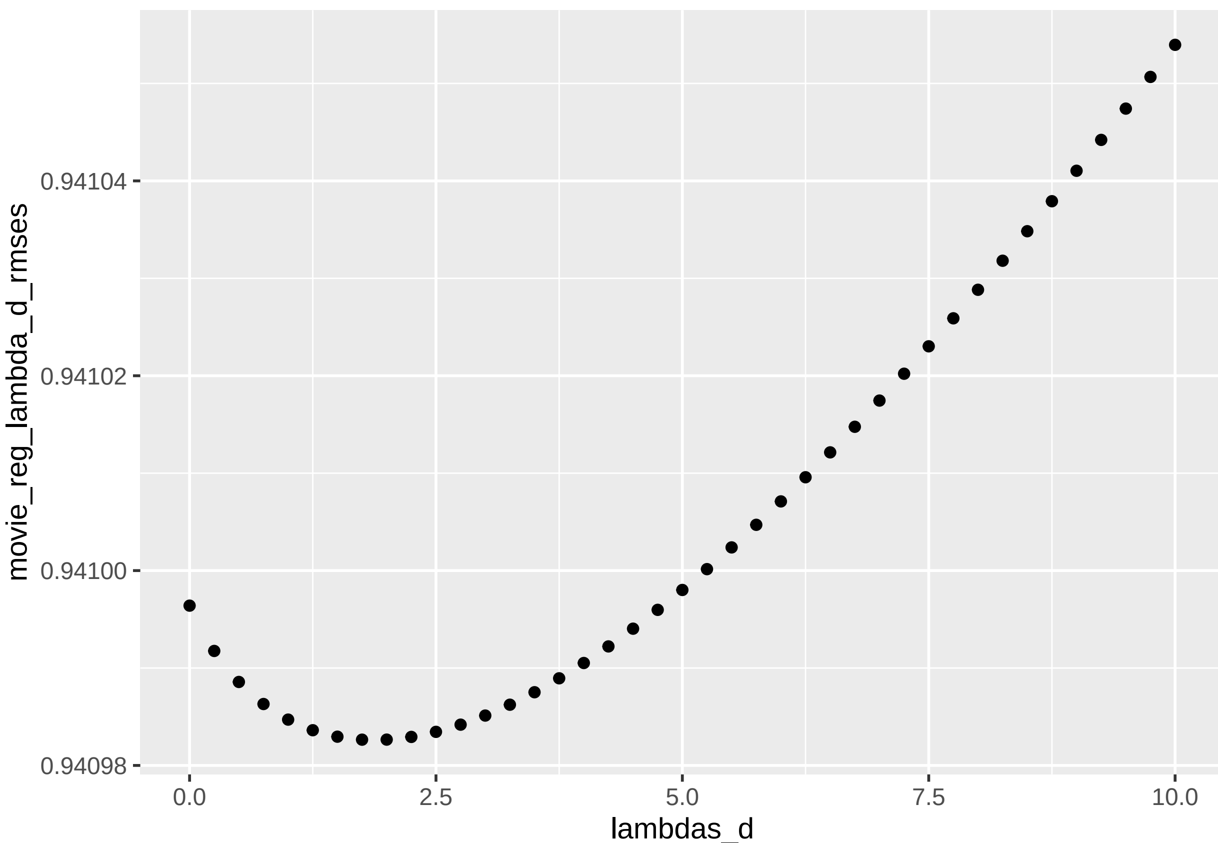
construct predictors with Movie_reg_lambda_d model

```
movie_reg_lambda_d_rmses <- sapply(lambdas_d, function(l){
  predicted_ratings <- test_set %>%
    left_join(movie_reg_sum_avgs, by='movieId') %>%
    mutate(b_ir = s/(n_i+l)) %>%
    mutate(pred = mu + b_ir) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

head(movie_reg_lambda_d_rmses)
```

```
## [1] 0.9409964 0.9409918 0.9409886 0.9409863 0.9409847 0.9409836
```

summary with corresponding rmse

```
qplot(lambdas_d, movie_reg_lambda_d_rmses)
```




```r
data.frame(lambdas_d, movie_reg_lambda_d_rmses)
```

```
##    lambdas_d movie_reg_lambda_d_rmses
## 1       0.00                0.9409964
## 2       0.25                0.9409918
## 3       0.50                0.9409886
## 4       0.75                0.9409863
## 5       1.00                0.9409847
## 6       1.25                0.9409836
## 7       1.50                0.9409830
## 8       1.75                0.9409826
## 9       2.00                0.9409827
## 10      2.25                0.9409829
## 11      2.50                0.9409834
## 12      2.75                0.9409842
## 13      3.00                0.9409851
## 14      3.25                0.9409862
## 15      3.50                0.9409875
## 16      3.75                0.9409889
## 17      4.00                0.9409905
## 18      4.25                0.9409922
## 19      4.50                0.9409940
## 20      4.75                0.9409960
## 21      5.00                0.9409980
## 22      5.25                0.9410001
```

```
## 23      5.50                 0.9410024
## 24      5.75                 0.9410047
## 25      6.00                 0.9410071
## 26      6.25                 0.9410096
## 27      6.50                 0.9410121
## 28      6.75                 0.9410148
## 29      7.00                 0.9410175
## 30      7.25                 0.9410202
## 31      7.50                 0.9410230
## 32      7.75                 0.9410259
## 33      8.00                 0.9410288
## 34      8.25                 0.9410318
## 35      8.50                 0.9410348
## 36      8.75                 0.9410379
## 37      9.00                 0.9410410
## 38      9.25                 0.9410442
## 39      9.50                 0.9410474
## 40      9.75                 0.9410507
## 41     10.00                 0.9410540
```

show the smallest rmse and the lambda lead to it.

```
lambdas_d[which.min(movie_reg_lambda_d_rmses)]
```

```
## [1] 1.75
```

```
movie_reg_lambda_d_rmses_model<- movie_reg_lambda_d_rmses[which.min(movie_reg_lambda_d_rmses)]

movie_reg_lambda_d_rmses_model
```

```
## [1] 0.9409826
```

The smallest rmse we get from lambda (0,10,0.25) tunning is 0.9409826, lambda=1.75 . Now, let's add all the predictors to the lambda tunining model

add movie_reg_lambda_d_rmses_model to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="movie_reg_lambda_d_rmses_model",
                                     RMSE = movie_reg_lambda_d_rmses_model))
rmse_results
```

```
## # A tibble: 8 x 2
##   method                                        RMSE
##   <chr>                                        <dbl>
## 1 all_average                                  1.05
## 2 Movie Model                                  0.941
## 3 movies and user model                        0.857
```

```
## 4 movies+user+yearrating model                0.857
## 5 movies+user+yearrating+premier model        0.857
## 6 movies+user+yearrating+premier+genres model 0.857
## 7 Regularized_Movie_Model                     0.941
## 8 movie_reg_lambda_d_rmses_model              0.941
```

Modeling 9, use all cross Regularization effects with different tuning

add Penalized least squares with different tuning

add all mivie+user+rate year+ premier+genres predictors

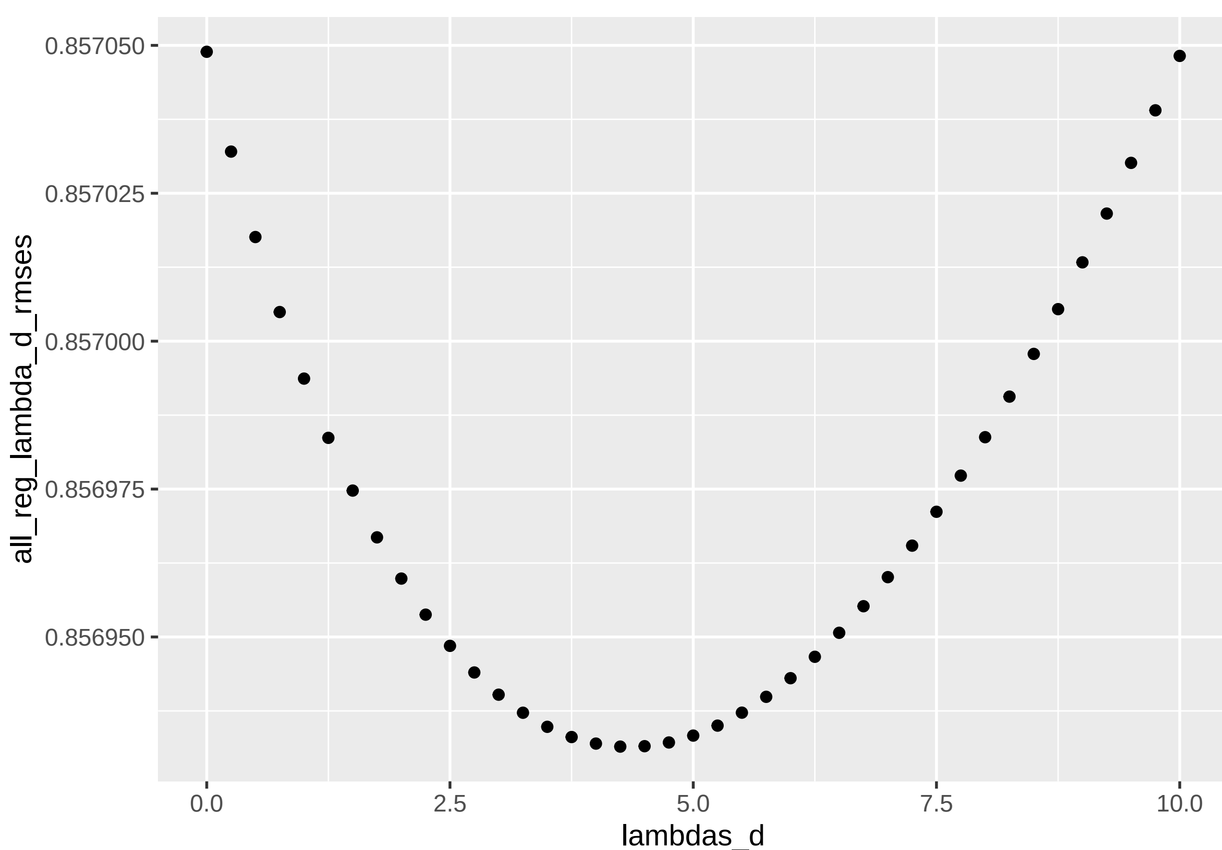
```
lambdas_d <- seq(0, 10, 0.25)

all_reg_lambda_d_rmses <- sapply(lambdas_d, function(l){

  mu <- mean(train_set$rating)

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

  b_ua <- train_set %>%
    left_join(b_ia, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_ua = sum(rating - b_ia - mu)/(n()+l))


  b_ya <- train_set %>%
    left_join(b_ia, by="movieId") %>%
    left_join(b_ua, by="userId") %>%
    group_by(year_rated) %>%
    summarize(b_ya = sum(rating - b_ia - mu - b_ua)/(n()+l))


  b_pa <- train_set %>%
    left_join(b_ia, by="movieId") %>%
    left_join(b_ua, by="userId") %>%
    left_join(b_ya, by="year_rated") %>%
    group_by(premier_date) %>%
    summarize(b_pa = sum(rating - b_ia - mu - b_ua - b_ya)/(n()+l))

  b_ga <- train_set %>%
    left_join(b_ia, by="movieId") %>%
    left_join(b_ua, by="userId") %>%
    left_join(b_ya, by="year_rated") %>%
    left_join(b_pa, by="premier_date") %>%
    group_by(s_genres) %>%
    summarize(b_ga = sum(rating - b_ia - mu - b_ua - b_ya- b_pa)/(n()+l))

  predicted_ratings <-
    test_set %>%
```

```
    left_join(b_ia, by = "movieId") %>%
    left_join(b_ua, by = "userId") %>%
    left_join(b_ya, by = "year_rated") %>%
    left_join(b_pa, by = "premier_date") %>%
    left_join(b_ga, by = "s_genres") %>%
    mutate(pred = mu + b_ia + b_ua + b_ya + b_pa + b_ga) %>%
    pull(pred)

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

head(all_reg_lambda_d_rmses)
```

```
## [1] 0.8570489 0.8570320 0.8570176 0.8570049 0.8569937 0.8569837
```

summary with corresponding rmse and find the lambda that get the smallest RMSE,

```
qplot(lambdas_d, all_reg_lambda_d_rmses)
```




```
lambdasmall <- lambdas_d [which.min(all_reg_lambda_d_rmses)]
lambdasmall
```

```
## [1] 4.25
```

show the smallest RMSE

```
all_reg_lambda_d_rmses_model <-all_reg_lambda_d_rmses[which.min(all_reg_lambda_d_rmses)]
all_reg_lambda_d_rmses_model
```

```
## [1] 0.8569315
```

add all_reg_lambda_d_rmses_model to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="all_reg_lambda_d_rmses_model",
                                     RMSE = all_reg_lambda_d_rmses_model))
rmse_results
```

```
## # A tibble: 9 x 2
##   method                                         RMSE
##   <chr>                                         <dbl>
## 1 all_average                                   1.05
## 2 Movie Model                                   0.941
## 3 movies and user model                         0.857
## 4 movies+user+yearrating model                  0.857
## 5 movies+user+yearrating+premier model          0.857
## 6 movies+user+yearrating+premier+genres model 0.857
## 7 Regularized_Movie_Model                       0.941
## 8 movie_reg_lambda_d_rmses_model                0.941
## 9 all_reg_lambda_d_rmses_model                  0.857
```

Including all predictors in our Regularization model with different tunning, I get a model with rmse of 0.8569315 , it is pretty good. we can use it as our final model. I will then use data set edx as a train set and data set validation as a test set, to fit this model.

final test, the edx set as train set, the validation set as the test set to run the final all_reg_lambda_d_rmses_model

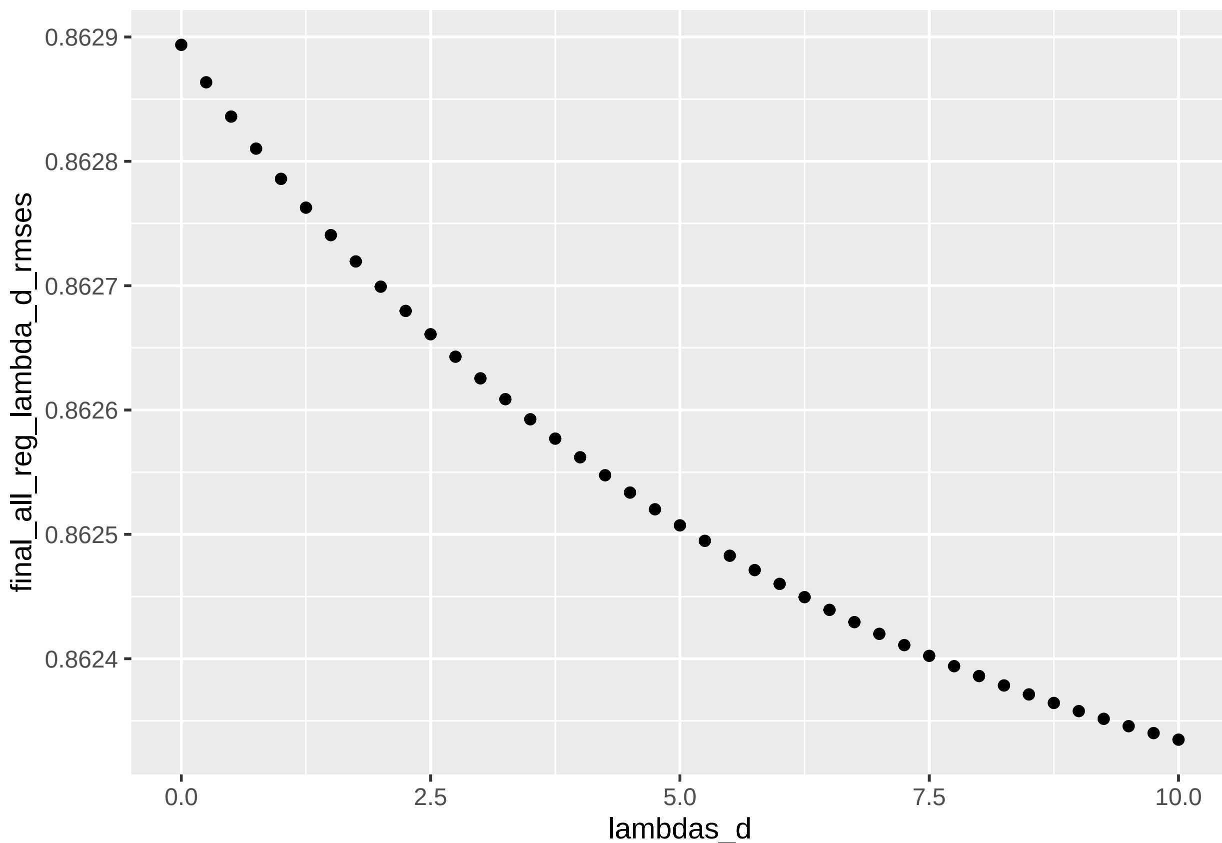
```
lambdas_d <- seq(0, 10, 0.25)

final_all_reg_lambda_d_rmses <- sapply(lambdas_d, function(l){

  fmu <- mean(edx$rating)

  b_if <- edx %>%
    group_by(movieId) %>%
    summarize(b_if = sum(rating - fmu)/(n()+l))

  b_uf <- edx %>%
    left_join(b_if, by="movieId") %>%
    group_by(userId) %>%
```

```
    summarize(b_uf = sum(rating - b_if - fmu)/(n()+l))


  b_yf <- edx %>%
    left_join(b_if, by="movieId") %>%
    left_join(b_uf, by="userId") %>%
    group_by(year_rated) %>%
    summarize(b_yf = sum(rating - b_if - fmu - b_uf)/(n()+l))


  b_pf <- edx %>%
    left_join(b_if, by="movieId") %>%
    left_join(b_uf, by="userId") %>%
    left_join(b_yf, by="year_rated") %>%
    group_by(premier_date) %>%
    summarize(b_pf = sum(rating - b_if - fmu - b_uf - b_yf)/(n()+l))

  b_gf <- edx %>%
    left_join(b_if, by="movieId") %>%
    left_join(b_uf, by="userId") %>%
    left_join(b_yf, by="year_rated") %>%
    left_join(b_pf, by="premier_date") %>%
    group_by(s_genres) %>%
    summarize(b_gf = sum(rating - b_if - fmu - b_uf - b_yf- b_pf)/(n()+l))


  fpredicted_ratings <-  validation %>%
    left_join(b_if, by = "movieId") %>%
    left_join(b_uf, by = "userId") %>%
    left_join(b_yf, by = "year_rated") %>%
    left_join(b_pf, by = "premier_date") %>%
    left_join(b_gf, by = "s_genres") %>%
    mutate(pred = fmu + b_if + b_uf + b_yf + b_pf + b_gf) %>%
    pull(pred)

  return(RMSE( fpredicted_ratings, validation$rating))
})

head(final_all_reg_lambda_d_rmses)
```

```
## [1] 0.8628937 0.8628636 0.8628360 0.8628102 0.8627859 0.8627627
```

find the lambda that get the smallest RMSE,

```
qplot(lambdas_d, final_all_reg_lambda_d_rmses)
```



From this plot, i find that maybe I still have possibility to get better prediction model if we set the tunning range wider.

```
final_lambdasmall <- lambdas_d [which.min(final_all_reg_lambda_d_rmses)]
final_lambdasmall
```

```
## [1] 10
```

```
data.frame(lambdas_d, final_all_reg_lambda_d_rmses)
```

```
##    lambdas_d final_all_reg_lambda_d_rmses
## 1       0.00                    0.8628937
## 2       0.25                    0.8628636
## 3       0.50                    0.8628360
## 4       0.75                    0.8628102
## 5       1.00                    0.8627859
## 6       1.25                    0.8627627
## 7       1.50                    0.8627407
## 8       1.75                    0.8627195
## 9       2.00                    0.8626992
## 10      2.25                    0.8626797
## 11      2.50                    0.8626609
## 12      2.75                    0.8626429
## 13      3.00                    0.8626255
## 14      3.25                    0.8626087
```

```
## 15      3.50                   0.8625926
## 16      3.75                   0.8625770
## 17      4.00                   0.8625620
## 18      4.25                   0.8625475
## 19      4.50                   0.8625336
## 20      4.75                   0.8625202
## 21      5.00                   0.8625072
## 22      5.25                   0.8624948
## 23      5.50                   0.8624828
## 24      5.75                   0.8624713
## 25      6.00                   0.8624602
## 26      6.25                   0.8624495
## 27      6.50                   0.8624393
## 28      6.75                   0.8624294
## 29      7.00                   0.8624200
## 30      7.25                   0.8624110
## 31      7.50                   0.8624023
## 32      7.75                   0.8623940
## 33      8.00                   0.8623861
## 34      8.25                   0.8623785
## 35      8.50                   0.8623713
## 36      8.75                   0.8623644
## 37      9.00                   0.8623579
## 38      9.25                   0.8623516
## 39      9.50                   0.8623457
## 40      9.75                   0.8623402
## 41     10.00                   0.8623349
```

show the smallest RMSE

```
final_all_reg_lambda_d_rmses_model <-final_all_reg_lambda_d_rmses[which.min(final_all_reg_lambda_d_rmses
final_all_reg_lambda_d_rmses_model
```

```
## [1] 0.8623349
```

It is good, my model get a rmse of 0.8623349

add final_all_reg_lambda_d_rmses to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="final_all_reg_lambda_d_rmses_model",
                                     RMSE = final_all_reg_lambda_d_rmses_model))
rmse_results
```

```
## # A tibble: 10 x 2
##    method                                        RMSE
##    <chr>                                        <dbl>
##  1 all_average                                  1.05
##  2 Movie Model                                  0.941
```

```
##  3 movies and user model                        0.857
##  4 movies+user+yearrating model                 0.857
##  5 movies+user+yearrating+premier model         0.857
##  6 movies+user+yearrating+premier+genres model 0.857
##  7 Regularized_Movie_Model                      0.941
##  8 movie_reg_lambda_d_rmses_model               0.941
##  9 all_reg_lambda_d_rmses_model                 0.857
## 10 final_all_reg_lambda_d_rmses_model           0.862
```

use wider ranger of lambdas, the edx set as training set, the validation set as the testing set to run the final all_reg_lambda_d_rmses_model

```
lambdas_w <- seq(0, 20, 0.25)

final_all_reg_lambda_w_rmses <- sapply(lambdas_w, function(l){

  fwmu <- mean(edx$rating)

  b_ifw <- edx %>%
    group_by(movieId) %>%
    summarize(b_ifw = sum(rating - fwmu)/(n()+l))

  b_ufw <- edx %>%
    left_join(b_ifw, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_ufw = sum(rating - b_ifw - fwmu)/(n()+l))


  b_yfw <- edx %>%
    left_join(b_ifw, by="movieId") %>%
    left_join(b_ufw, by="userId") %>%
    group_by(year_rated) %>%
    summarize(b_yfw = sum(rating - b_ifw - fwmu - b_ufw)/(n()+l))


  b_pfw <- edx %>%
    left_join(b_ifw, by="movieId") %>%
    left_join(b_ufw, by="userId") %>%
    left_join(b_yfw, by="year_rated") %>%
    group_by(premier_date) %>%
    summarize(b_pfw = sum(rating - b_ifw - fwmu - b_ufw - b_yfw)/(n()+l))

  b_gfw <- edx %>%
    left_join(b_ifw, by="movieId") %>%
    left_join(b_ufw, by="userId") %>%
    left_join(b_yfw, by="year_rated") %>%
    left_join(b_pfw, by="premier_date") %>%
    group_by(s_genres) %>%
    summarize(b_gfw = sum(rating - b_ifw - fwmu - b_ufw - b_yfw- b_pfw)/(n()+l))


  fwpredicted_ratings <-  validation %>%
```

```
    left_join(b_ifw, by = "movieId") %>%
    left_join(b_ufw, by = "userId") %>%
    left_join(b_yfw, by = "year_rated") %>%
    left_join(b_pfw, by = "premier_date") %>%
    left_join(b_gfw, by = "s_genres") %>%
    mutate(pred = fwmu + b_ifw + b_ufw + b_yfw + b_pfw + b_gfw) %>%
    pull(pred)

  return(RMSE(fwpredicted_ratings, validation$rating))
})

head(final_all_reg_lambda_w_rmses)
```

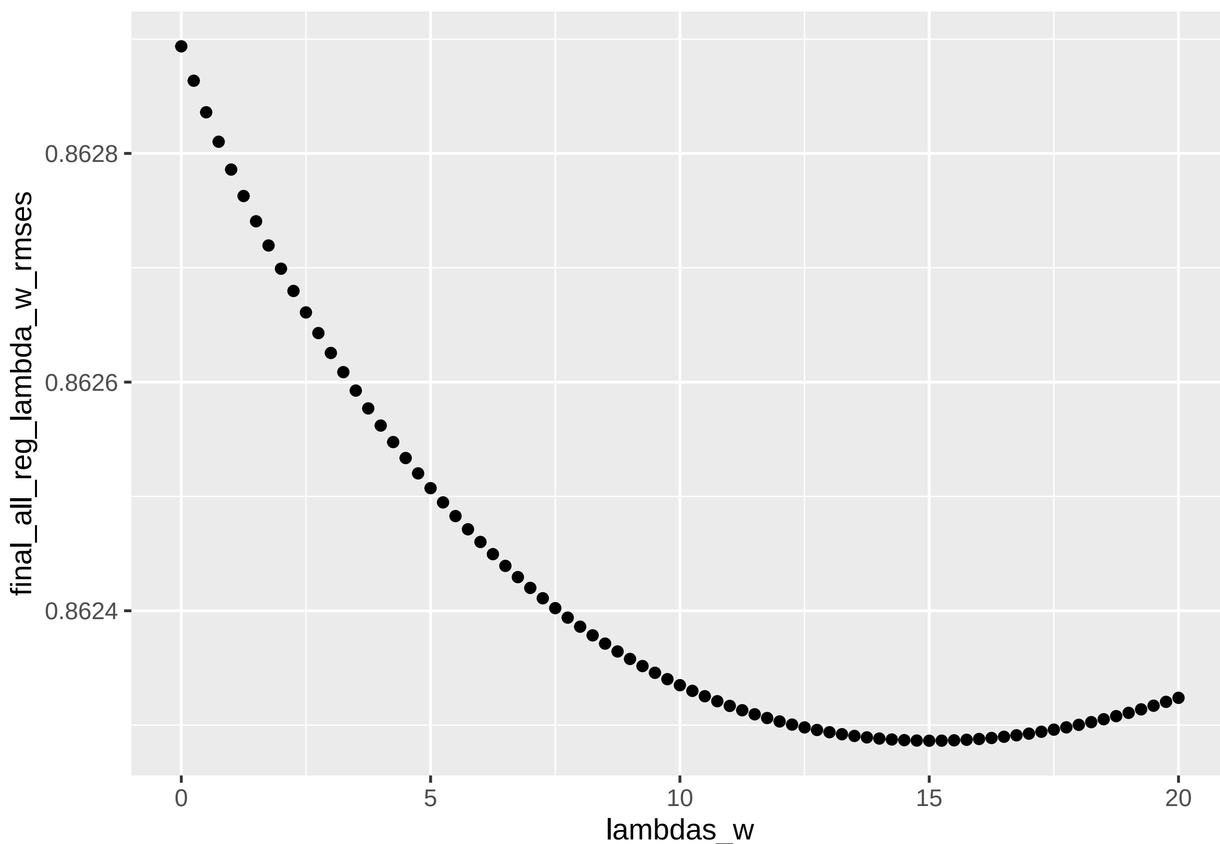
```
## [1] 0.8628937 0.8628636 0.8628360 0.8628102 0.8627859 0.8627627
```

find the lambda that get the smallest RMSE

```
final_w_lambdasmall <- lambdas_w [which.min(final_all_reg_lambda_w_rmses)]
final_w_lambdasmall
```

```
## [1] 15
```

```
qplot(lambdas_w, final_all_reg_lambda_w_rmses)
```



In this plot , it shows the smallest rmse is at lambdas_w = 15

```
data.frame(lambdas_w, final_all_reg_lambda_w_rmses)
```

```
##    lambdas_w final_all_reg_lambda_w_rmses
## 1       0.00                    0.8628937
## 2       0.25                    0.8628636
## 3       0.50                    0.8628360
## 4       0.75                    0.8628102
## 5       1.00                    0.8627859
## 6       1.25                    0.8627627
## 7       1.50                    0.8627407
## 8       1.75                    0.8627195
## 9       2.00                    0.8626992
## 10      2.25                    0.8626797
## 11      2.50                    0.8626609
## 12      2.75                    0.8626429
## 13      3.00                    0.8626255
## 14      3.25                    0.8626087
## 15      3.50                    0.8625926
## 16      3.75                    0.8625770
## 17      4.00                    0.8625620
## 18      4.25                    0.8625475
## 19      4.50                    0.8625336
## 20      4.75                    0.8625202
## 21      5.00                    0.8625072
## 22      5.25                    0.8624948
## 23      5.50                    0.8624828
## 24      5.75                    0.8624713
## 25      6.00                    0.8624602
## 26      6.25                    0.8624495
## 27      6.50                    0.8624393
## 28      6.75                    0.8624294
## 29      7.00                    0.8624200
## 30      7.25                    0.8624110
## 31      7.50                    0.8624023
## 32      7.75                    0.8623940
## 33      8.00                    0.8623861
## 34      8.25                    0.8623785
## 35      8.50                    0.8623713
## 36      8.75                    0.8623644
## 37      9.00                    0.8623579
## 38      9.25                    0.8623516
## 39      9.50                    0.8623457
## 40      9.75                    0.8623402
## 41     10.00                    0.8623349
## 42     10.25                    0.8623299
## 43     10.50                    0.8623253
## 44     10.75                    0.8623209
## 45     11.00                    0.8623168
## 46     11.25                    0.8623130
## 47     11.50                    0.8623095
## 48     11.75                    0.8623062
## 49     12.00                    0.8623032
```

```
## 50     12.25                    0.8623005
## 51     12.50                    0.8622980
## 52     12.75                    0.8622957
## 53     13.00                    0.8622938
## 54     13.25                    0.8622920
## 55     13.50                    0.8622905
## 56     13.75                    0.8622893
## 57     14.00                    0.8622882
## 58     14.25                    0.8622874
## 59     14.50                    0.8622868
## 60     14.75                    0.8622865
## 61     15.00                    0.8622863
## 62     15.25                    0.8622864
## 63     15.50                    0.8622867
## 64     15.75                    0.8622872
## 65     16.00                    0.8622878
## 66     16.25                    0.8622887
## 67     16.50                    0.8622898
## 68     16.75                    0.8622911
## 69     17.00                    0.8622925
## 70     17.25                    0.8622942
## 71     17.50                    0.8622960
## 72     17.75                    0.8622980
## 73     18.00                    0.8623002
## 74     18.25                    0.8623026
## 75     18.50                    0.8623051
## 76     18.75                    0.8623078
## 77     19.00                    0.8623107
## 78     19.25                    0.8623138
## 79     19.50                    0.8623170
## 80     19.75                    0.8623203
## 81     20.00                    0.8623239
```

show the smallest RMSE

```
final_all_reg_lambda_w_rmses_model <-final_all_reg_lambda_w_rmses[which.min(final_all_reg_lambda_w_rmse
final_all_reg_lambda_w_rmses_model
```

```
## [1] 0.8622863
```

After widering the tunning range of lambda, we did improved our prediction, we decreased rmse to 0.8622863. It is good.

add Regularized Movie_user_ratingyear_premier_Effec_Model1 to rmse_result data frame

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="final_all_reg_lambda_w_rmses_model",
                                     RMSE = final_all_reg_lambda_w_rmses_model))
rmse_results
```

```
## # A tibble: 11 x 2
##    method                                     RMSE
##    <chr>                                     <dbl>
##  1 all_average                               1.05 
##  2 Movie Model                               0.941
##  3 movies and user model                     0.857
##  4 movies+user+yearrating model              0.857
##  5 movies+user+yearrating+premier model      0.857
##  6 movies+user+yearrating+premier+genres model 0.857
##  7 Regularized_Movie_Model                   0.941
##  8 movie_reg_lambda_d_rmses_model            0.941
##  9 all_reg_lambda_d_rmses_model              0.857
## 10 final_all_reg_lambda_d_rmses_model        0.862
## 11 final_all_reg_lambda_w_rmses_model        0.862
```

III.result

To build prediction model, I try a variety of machine learning algorithms. Then comparing the predicted value with the actual outcome by loss function— root mean squared error (rmse).

Linear Model

In Linear Model, I begin with assumes the same rating for all movies and all users. I get rmse of 1.051984. I created 5 predictors for all data set, and build models by adding each predictors one by one . I get all_average model rmse 1.0519843 , movies model rmse 0.9409964 , movies and user model rmse 0.8574946 , movies+user+yearrating model rmse 0.8574946 , movies+user+yearrating+premier model rmse 0.8571341 , movies+user+yearrating+premier+genres model rmse 0.8570489 . I find some predictor for example user contribute more, some predictor like year_rated not. Even I included all possible predictors in my algorithm, still rmse is a little too big. Butit let me know the more predictors I included in building model, the better my prediction is.

Regularized Model

I try use regularization to penalize large effects that come from small samples. I randomly choose a penalized least squares rate lambda=3 and use only movieId as a predictor, my model get rmse of 0.9409851 . Not Good result. Then I set my Penalized least squares with different tuning(seq(0, 10, 0.25)), when lambda= 1.75 , I get my minimize rmse of 0.9409826. This almost make no improvement.

In the next model, I include all the possible predictors plus Penalized least squares tunning with seq(0, 10, 0.25), the minimize rmse of this model is 0.8569315 when lambda is 4.25. It is pretty good. I can accept it as a final model.

Final test

I use the data set validation to fit my final model. I get a rmse = 0.8623349 . When plotting lambdas_d vs final_all_reg_lambda_d_rmses, I find there is still a possibility to get smaller rmse if we set wider tunning range . I set lambda = seq(0, 20, 0.25), I get the smallest rmse = 0.8622863 at lambda= 15.

So for my project, I get final rmse = 0.8622863.

IV.conclusion

This preject is to build a model that can predict movie rating based on the rated movies data set edx. I use moviesId, user, year_rated, premier and genres as predictors. And find the more predictor included in the model, the more accurate prediction the model will give. Regularized Model which penalized those effects of smaller sample size get better prediction than liner model, I use it as the final model of my project. Test this final model with data set validation, I get a rmse of 0.8622863.

Limitation: I only try linear regression algorithms and regularization model, it need long time running on laptop, and is time spend.

In future work , I should also try K-nearest neighbors (kNN), Matrix Factorization and Principle Component Analysis in order to further decrease the RMSE.