HarvardX: PH125.9x Data Science
MovieLens Rating Prediction Project

Denis Korolskii

November 17, 2019

Overview

MovieLens Project of the HarvardX: PH125.9x Data Science: Capstone course. Current task is to create recommendation system using MovieLens dataset. Also, current task is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

Introduction

A recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications. Recommender systems are utilized in a variety of areas, and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. For this project we will focus on create a movie recommendation system using the 10M version of MovieLens dataset, collected by GroupLens Research.

##Executive summary

The goal is to train a machine learning algorithm using the inputs of a provided training subset to predict movie ratings in a validation set. The evaluation of algorithm performance is the Root Mean Square Error. RMSE is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSE represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. These deviations are called residuals when the calculations are performed over the data sample that was used for estimation and are called errors (or prediction errors) when computed out-of-sample. The RMSE serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset and not between datasets, as it is scale-dependent. RMSE is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data. In general, a lower RMSD is better than a higher one. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.86550. The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

Finally, the best resulting model will be used to predict the movie ratings. "It will be usefull to utilize and load several packages from CRAN. As per the project guidelines, the dataset will be split into a training and validation set (10%), and the training set will then be further split into a train/test set with the test set being 10% of the training set."

Dataset

The MovieLens dataset is automatically downloaded

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

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
title = as.character(title),
genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

In order to predict in the most possible accurate way the movie rating of the users that haven't seen the movie yet, the he MovieLens dataset will be splitted into 2 subsets that will be the "edx", a training subset to train the algorithm, and "validation" a subset to test the movie ratings.

```
# The Validation subset will be 10% of the MovieLens data.
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 subset:
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)
```

Algorithm development is to be carried out on the "edx" subset only, as "validation" subset will be used to test the final algorithm.

Metadata Source: http://files.grouplens.org/datasets/movielens/ml-10m-README.html

Summary This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided. The data are contained in three files, movies.dat, ratings.dat and tags.dat. Also included are scripts for generating subsets of the data to support five-fold cross-validation of rating predictions. More details about the contents and use of all these files follows. This and other GroupLens data sets are publicly available for download at

GroupLens Data Sets. All ratings are contained in the file ratings.dat. Each line of this file represents one rating of one movie by one user, and has the following format: UserID::MovieID::Rating::Timestamp The lines within this file are ordered first by UserID, then, within user, by MovieID. Ratings are made on a 5-star scale, with half-star increments. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. Tags Data File Structure All tags are contained in the file tags.dat. Each line of this file represents one tag applied to one movie by one user, and has the following format: UserID::MovieID::Tag::Timestamp The lines within this file are ordered first by UserID, then, within user, by MovieID. Tags are user generated metadata about movies. Each tag is typically a single word, or short phrase. The meaning, value and purpose of a particular tag is determined by each user. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. Movies Data File Structure Movie information is contained in the file movies.dat. Each line of this file represents one movie, and has the following format: MovieID::Title::Genres MovieID is the real MovieLens id. Movie titles, by policy, should be entered identically to those found in IMDB, including year of release. However, they are entered manually, so errors and inconsistencies may exist. Genres are a pipe-separated list, and are selected from the following: Action Adventure Animation Children's Comedy Crime Documentary Drama Fantasy Film-Noir Horror Musical Mystery Romance Sci-Fi Thriller War Western

Methods and Analysis

Data Analysis

To get familiar with the dataset, we find the first rows of "edx" subset as below. The subset contain the six variables "userID", "movieID", "rating", "timestamp", "title", and "genres". Each row represent a single rating of a user for a single movie.

```r
#load libraries
library(ggplot2)
library(tidyverse)
library(caret)
library(lubridate)
```

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

## The following object is masked from 'package:base':
##
##     date
```

```r
library(stringr)
```

Preprocessing

Testing for any N/A

```r
colSums(is.na(edx))
```

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

```
#Converting timestamp to year
edx <- mutate(edx, UserTag_year = year(as_datetime(timestamp)))
```

```
#Extracting "year" from "title" data field
edx <- mutate(edx, movie_year = as.numeric(str_sub(title,-5,-2)) )
```

Exploring dataset

check for duplicates/double elements and comparing with metadata option. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Each user is represented by an id, and no other information is provided.

```
#Checking dataset whether it includes users below metadata limit option or not
uid<-edx%>%group_by(userId)%>%summarize(number = n(), min=min(rating),max=max(rating)) %>%
arrange(desc(number))%>%filter(number<=19)

#This part of dataset (users who made less than 19 ratings)
#does not accord the metadata option (more than 20 ratings per user).
nrow(uid)
```

```
## [1] 3458
```

Checking for any doubles. Users who wathched same film several times.

```
double_watching <- edx %>% group_by(userId,title)%>%
summarize (number = n(), min=min(rating), max=max(rating)) %>% arrange(desc(number)) %>% filter(number>

nrow(double_watching)
```

```
## [1] 16
```

```
#The number of users who wathched same film several times is insignificant.
```

Testing for duplicate movie titles with several movieId

```
dupl_title_movieId<-edx%>%group_by(title)%>%summarize(min=min(movieId),
max=max(movieId)) %>%filter(min!=max)

head(dupl_title_movieId)
```

```
## # A tibble: 1 x 3
##   title                     min   max
##   <chr>                   <dbl> <dbl>
## 1 War of the Worlds (2005) 34048 64997
```

```
#Only one movie has several movieId
```

Checking timestamps for inconsistencies

```r
edx<-edx%>%mutate(Tag_year_delay=UserTag_year-movie_year)
nrow(filter(edx,edx$Tag_year_delay<=-1))
```

```
## [1] 179
```

```r
head(filter(edx,edx$Tag_year_delay<=-1))
```

```
##   userId movieId rating timestamp                  title          genres
## 1    785     981      3 844464462 Dangerous Ground (1997)           Drama
## 2   1468     879      2 841308568       Relic, The (1997) Horror|Thriller
## 3   1583     981      1 842861387 Dangerous Ground (1997)           Drama
## 4   1766     870      5 839441876     Gone Fishin' (1997)          Comedy
## 5   1766     879      5 840812687       Relic, The (1997) Horror|Thriller
## 6   1766     981      3 841947226 Dangerous Ground (1997)           Drama
##   UserTag_year movie_year Tag_year_delay
## 1         1996       1997             -1
## 2         1996       1997             -1
## 3         1996       1997             -1
## 4         1996       1997             -1
## 5         1996       1997             -1
## 6         1996       1997             -1
```

```r
#Some movies rated earlier than its premier year
```

Observing dataset

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres UserTag_year movie_year Tag_year_delay
## 1                Comedy|Romance         1996       1992              4
## 2         Action|Crime|Thriller         1996       1995              1
## 3                        Comedy         1996       1994              2
## 4  Action|Drama|Sci-Fi|Thriller         1996       1995              1
## 5       Action|Adventure|Sci-Fi         1996       1994              2
## 6 Action|Adventure|Drama|Sci-Fi         1996       1994              2
```

```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
```

```
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title               genres            UserTag_year    movie_year
##  Length:9000061     Length:9000061     Min.   :1995   Min.   :1915
##  Class :character   Class :character   1st Qu.:2000   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :2002   Median :1994
##                                        Mean   :2002   Mean   :1990
##                                        3rd Qu.:2005   3rd Qu.:1998
##                                        Max.   :2009   Max.   :2008
##  Tag_year_delay
##  Min.   :-2.00
##  1st Qu.: 2.00
##  Median : 7.00
##  Mean   :11.98
##  3rd Qu.:16.00
##  Max.   :93.00
```

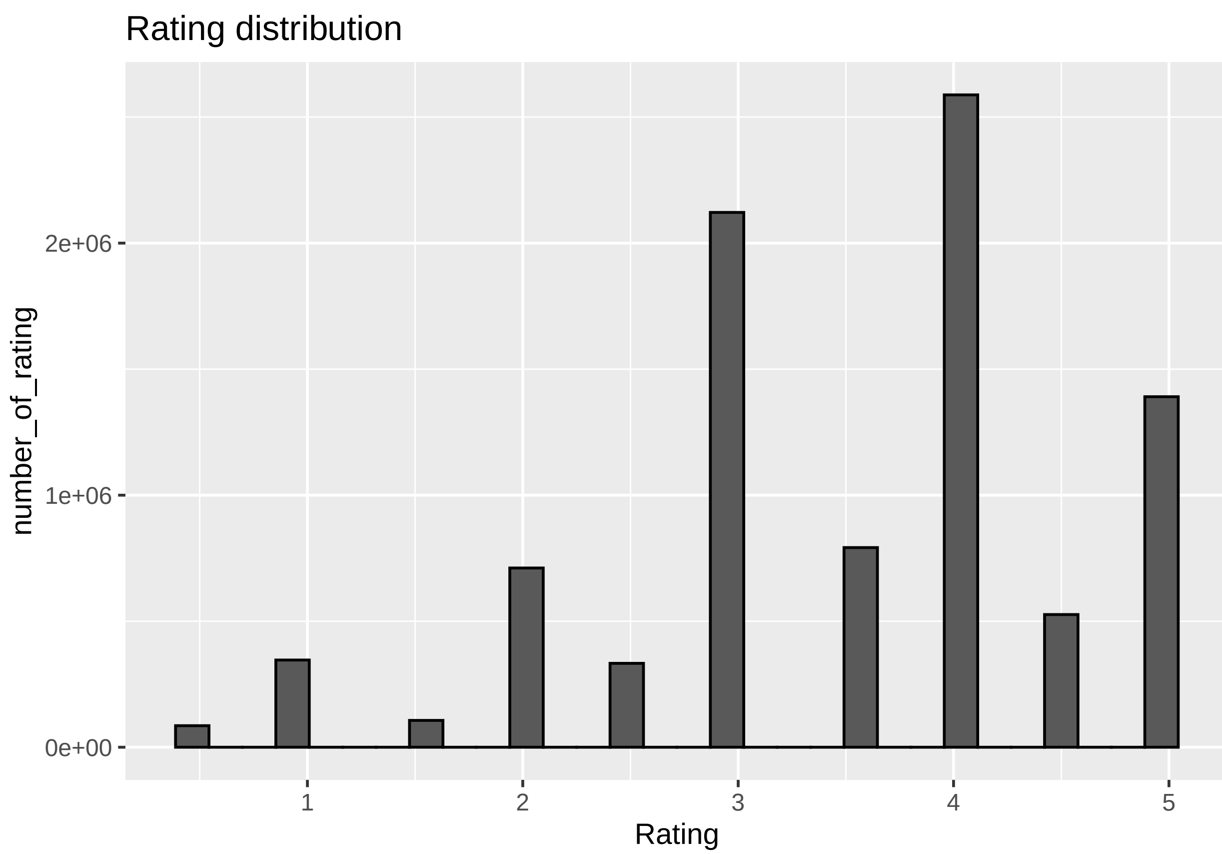
User activity characteristics

```
head( edx%>% group_by(userId) %>% summarize(number = n(),
min=min(rating), max=max(rating)) %>% arrange (desc(number)))
```

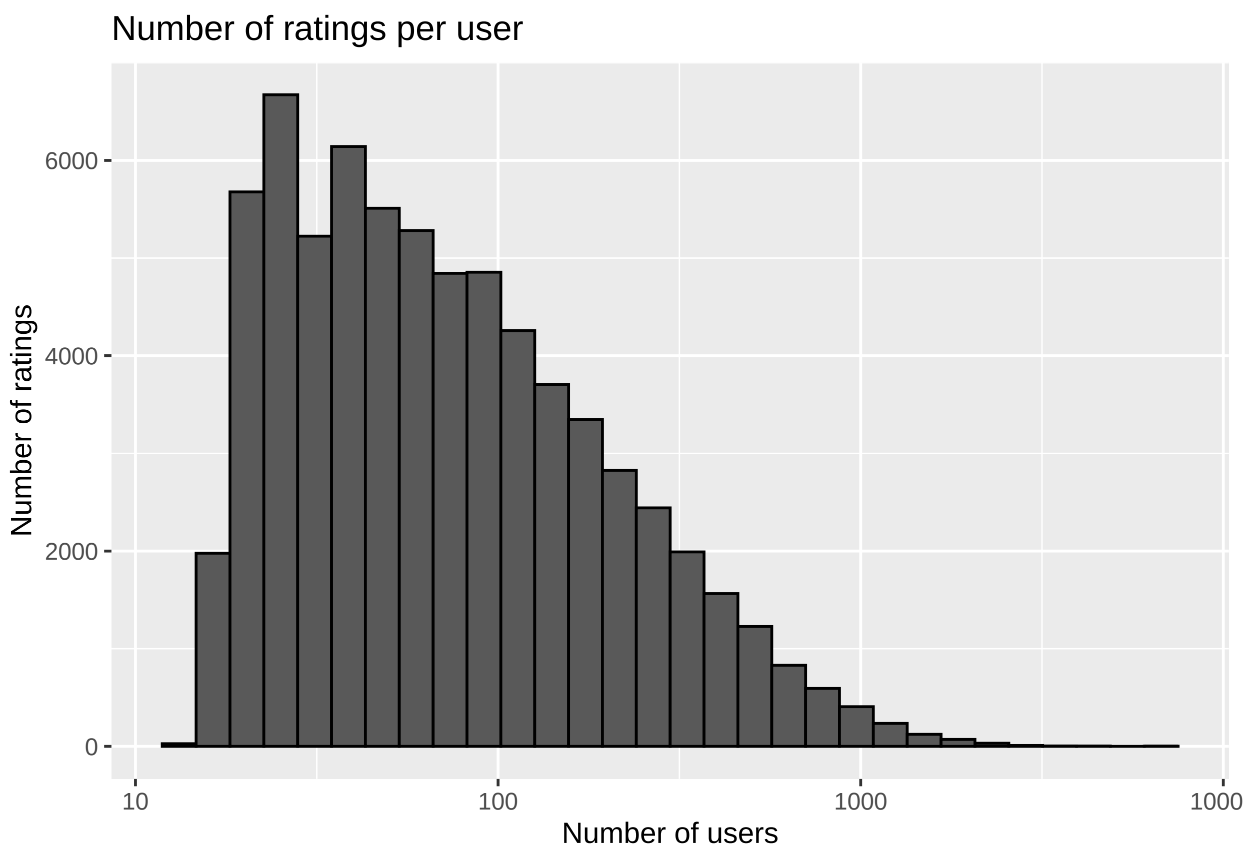
```
## # A tibble: 6 x 4
##   userId number   min   max
##    <int>  <int> <dbl> <dbl>
## 1  59269   6637   0.5     5
## 2  67385   6376   1       5
## 3  14463   4637   1       5
## 4  68259   4056   0.5     5
## 5  27468   4018   1       5
## 6  19635   3740   1       5
```

```
#Table shows extremal user activity
```

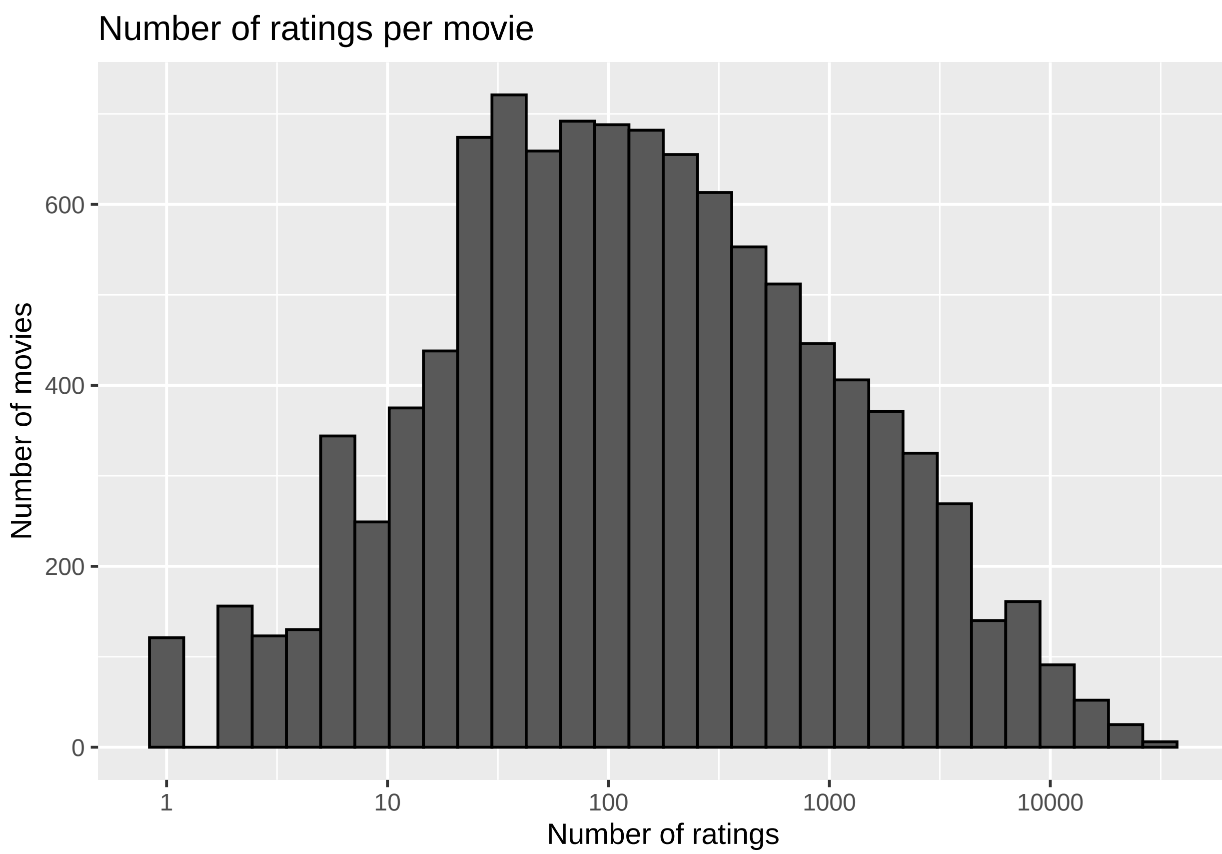
```
#Histogram.Rating distribution

edx %>% ggplot(aes(rating)) +
  geom_histogram(bins = 30, color = "black") +
  labs(title = "Rating distribution",
       x = "Rating",
       y = "number_of_rating")
```




```r
# Number of ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(x=n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of users") +
  ylab("Number of ratings") +
  ggtitle("Number of ratings per user")
```




```r
#Number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie")
```

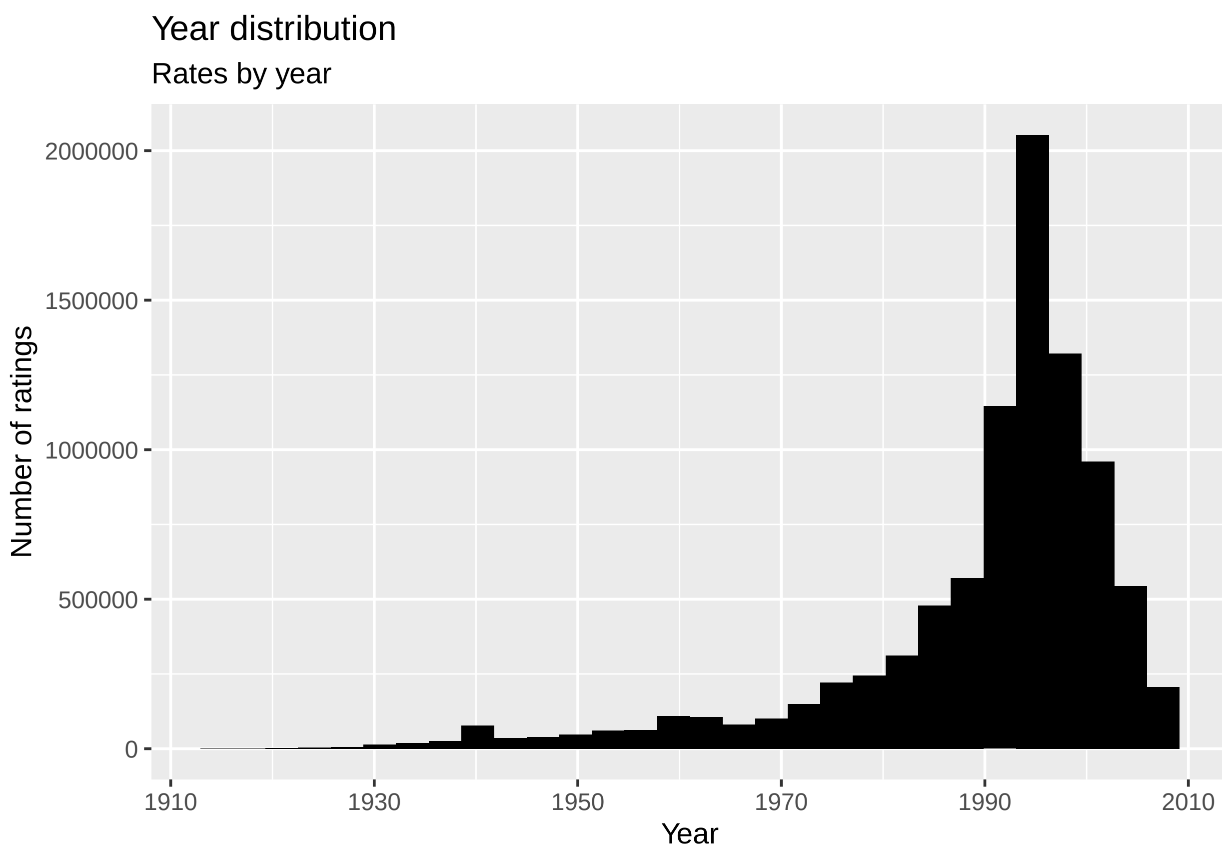


Unique elements of dataset

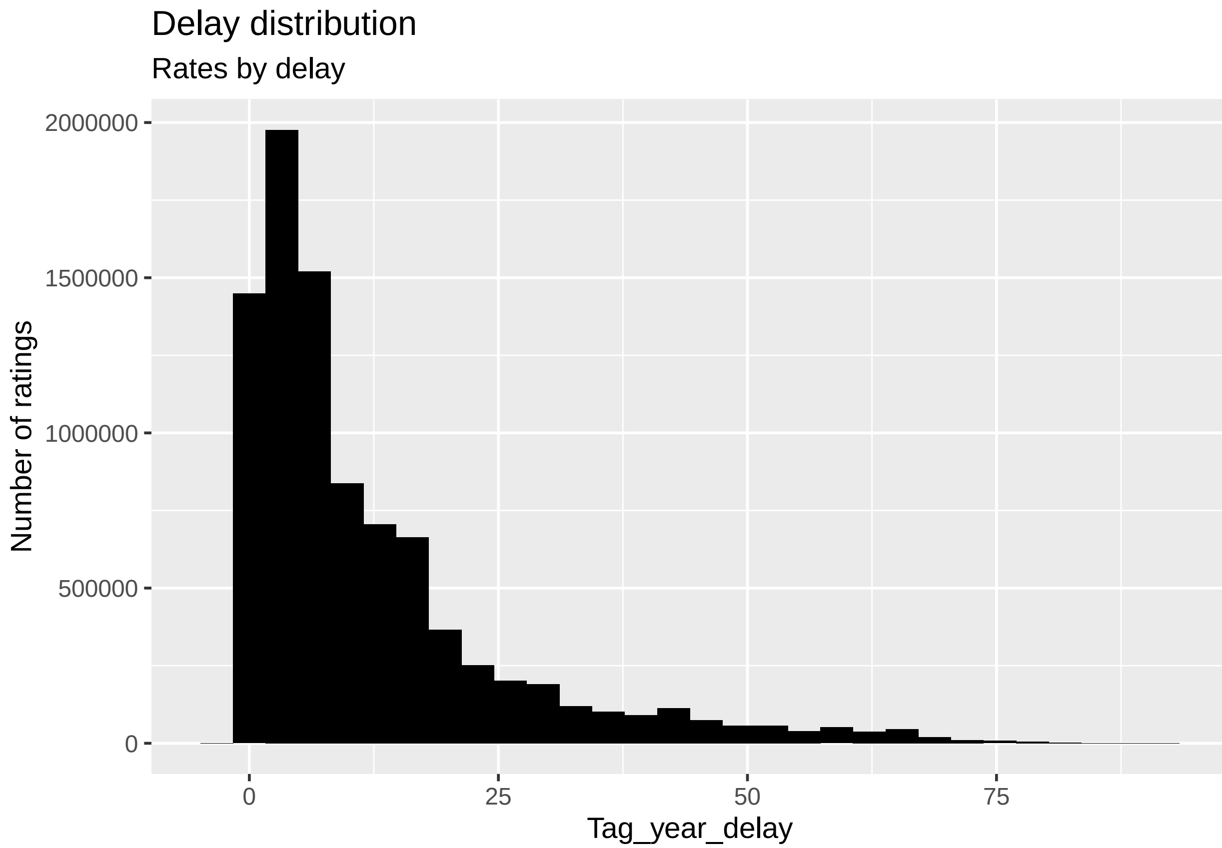
```
edx %>%
  summarize(dist_Users = n_distinct(userId),
            dist_Movies = n_distinct(movieId),
            dist_Genres = n_distinct(genres))
```

```
##   dist_Users dist_Movies dist_Genres
## 1      69878       10677         797
```

```
#Movie premier year distribution
edx %>% ggplot(aes(movie_year)) +
  geom_histogram(bins = 30, fill = "black") +
  labs(title = "Year distribution",
       subtitle = "Rates by year",
       x = "Year",
       y = "Number of ratings")
```




```r
#Distance distribution between year of movie premier and rate
edx %>% ggplot(aes(Tag_year_delay)) +
  geom_histogram( bins = 30,fill = "black") +
  labs(title = "Delay distribution",
       subtitle = "Rates by delay",
       x = "Tag_year_delay",
       y = "Number of ratings")
```



Regression model

Prediction the same rating for all movies regardless of user Average movie effect

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

b_i on the training set

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

predicted ratings

```
predicted_ratings_bi <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

avg movie + user effect

b_u on the training set

```r
user_avgs <- edx %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

predicted ratings

```r
predicted_ratings_bu <- validation %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

avg movie + user + time effect

extracting year of given rating from timestamp on validation dataset

```r
validation1 <- validation %>%
  mutate(UserTag_year = year(as_datetime(timestamp)))
```

Time effects (b_t) on the training set

```r
UserTag_year_avgs <- edx %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(UserTag_year) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
```

predicted ratings

```r
predicted_ratings_bt <- validation1 %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(UserTag_year_avgs, by="UserTag_year") %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred
```

calculating RMSE for current models

```r
rmse_1 <- RMSE(validation$rating,predicted_ratings_bi)

rmse_result <- data_frame(method = "Avg movie rating model 1", RMSE = rmse_1)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```r
rmse_result %>% knitr::kable()
```

method	RMSE
Avg movie rating model 1	0.9437046

```
rmse_2 <- RMSE(validation$rating,predicted_ratings_bu)
rmse_result <- data_frame(method = "Avg movie + user effect model 2", RMSE = rmse_2)
rmse_result %>% knitr::kable()
```

method	RMSE
Avg movie + user effect model 2	0.8655329

```
rmse_3 <- RMSE(validation$rating,predicted_ratings_bt)
rmse_result <- data_frame(method = "Avg movie + user effect + time effect model 3", RMSE = rmse_3)
rmse_result %>% knitr::kable()
```

method	RMSE
Avg movie + user effect + time effect model 3	0.8655313

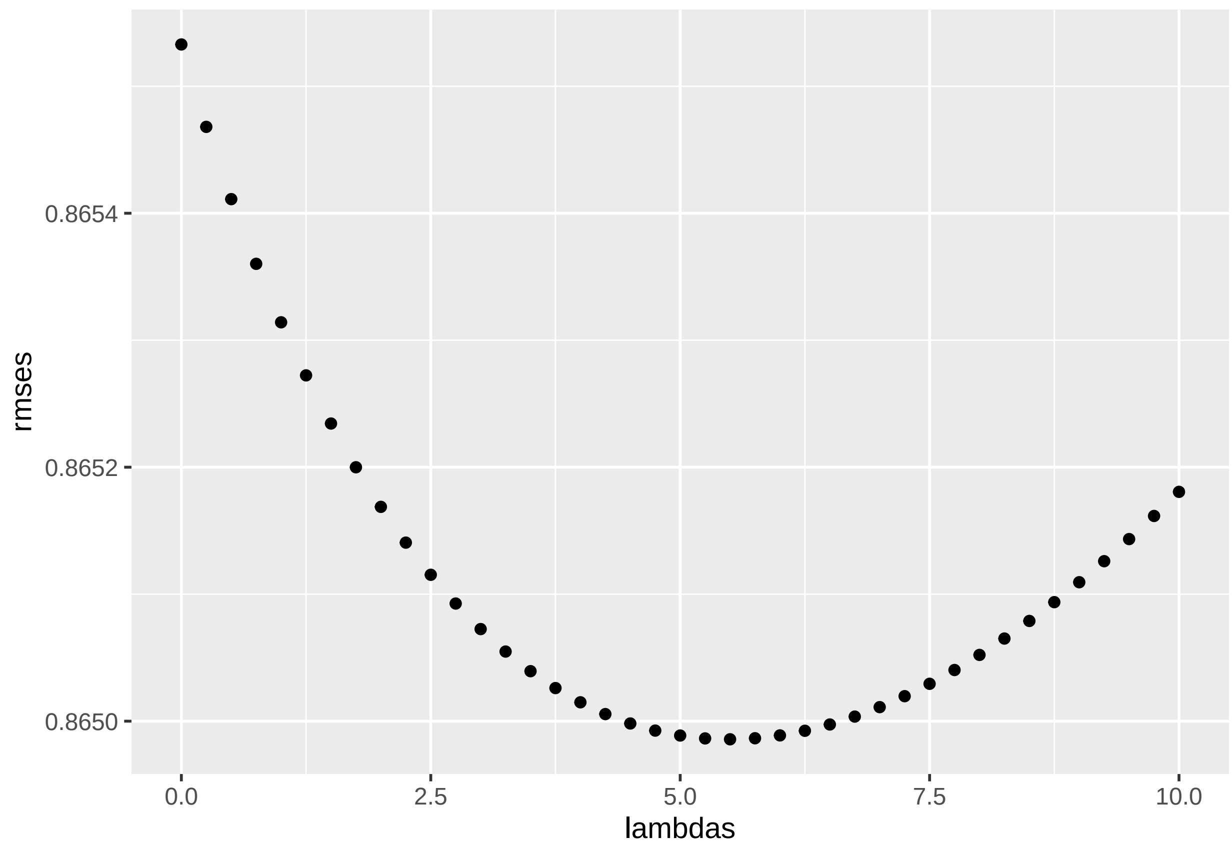
Regularization

Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach.

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

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

  mu_reg <- mean(edx$rating)

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

  b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))

  predicted_ratings_b_i_u <-
    validation %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg) %>%
    .$pred

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

plotting results

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




```
rmse_4 <- min(rmses)
rmse_result <- data_frame (method = "Avg movie + user effect + time effect + regularization model 4", RI
rmse_result %>% knitr::kable()
```

method	RMSE
Avg movie + user effect + time effect + regularization model 4 # Results	0.8649857

Calculated RMSE of all methods below

```
rmse_result <- data_frame(method = "Average movie + user effect + time effect + regularization model 4"
rmse_results <- bind_rows(rmse_result,
 data_frame(method="Average movie + user effect + time effect model 3",RMSE = rmse_3))
rmse_results <- bind_rows(rmse_results,
 data_frame(method="Average movie + user effect  2",RMSE = rmse_2))
rmse_results <- bind_rows(rmse_results,
 data_frame(method="Average movie   1",RMSE = rmse_1))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie + user effect + time effect + regularization model 4	0.8649857
Average movie + user effect + time effect model 3	0.8655313
Average movie + user effect 2	0.8655329
Average movie 1	0.9437046

Conclusion

This MovieLens project just successfully examined to predict movie rating. The model evaluation performance through the RMSE (root mean squared error) showed that the Linear regression model with regularized effects on users and movies are useful to predict ratings on the validation set. Current model effeciency approximately 0.86499

Appendix - Enviroment

```r
print("Operating System:")
```

```
## [1] "Operating System:"
```

```r
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```