# HarvardX Data Science Capstone: MovieLens Report

*Justin Nielson*

*May 29, 2019*

## Contents



## 1. Introduction

The HarvardX Data Science Capstone MovieLens project is a movie recommendation system using machine learning algorithms made famous by the one million dollar Netflix competition launched on october 2006.

At the end of the challenge in September 2009, Netflix awarded the Grand Prize to a developer team "BellKor's Pragmatic Chaos" with a winning solution with a Root Mean Squared Error(RMSE) of about 0.857 that increased the accuracy of the company's recommendation engine by 10%.

Using the 10M version of MovieLens data split into edx training set and 10% validation set, I used some of the machine learning techniques that went into the winning solution for the Netflix competition. The ML model used with the smallest RMSE was a multiple linear regression model using regularized movie and user effects at 0.8641362.

## 2. Overview

This report contains sections for data exploration, visualization, preprocessing, evaluated machine learning algorithms, and RMSE analysis sections including methods that were used to transform the data to create the best predicive model.

The results and conclusion sections and the end includes final thoughts on the MovieLens project and suggestions to improve the predictive model using more advanced machine learning algorithms and matrix factorization methods.

### 2.1. Loading libraries and data

```
# Loading packages for data exploration, visualization, preprocessing,
# machine learning algorithms, and RMSE analysis

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

```
## Loading required package: tidyverse

## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang

## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2

## -- Attaching packages ------------------------------------------------------------ tidyverse 1.2.1 --

## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- 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
```

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

```
## Loading required package: 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 object is masked from 'package:base':
##
##     date
```

```r
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: knitr
```

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

```
## Loading required package: rmarkdown
```

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

```
## Loading required package: tinytex
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(ggplot2)
library(knitr)
library(rmarkdown)
library(tinytex)

# MovieLens 10M data table:
# https://grouplens.org/data tables/movielens/10m/
# http://files.grouplens.org/data tables/movielens/ml-10m.zip

# Since I am using using R 3.6.0 I downloaded edx.rds and validation.rds data tables from
# HarvardX_Capstone_MovieLens Google Drive
# https://drive.google.com/drive/folders/1IZcBBX0OmL9wu9AdzMBFUG8GoPbGQ38D

movielens <- readRDS("edx.rds", refhook = NULL)
validation <- readRDS("validation.rds", refhook = NULL)

# Validation set will be 10% of MovieLens data
# if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
set.seed(1, sample.kind = "Rounding")
```

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

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set

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

# Add rows removed from validation set back into edx set

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

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

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

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

```
## Warning in rm(dl, ratings, movies, test_index, temp, movielens, removed):
## object 'dl' not found

## Warning in rm(dl, ratings, movies, test_index, temp, movielens, removed):
## object 'ratings' not found

## Warning in rm(dl, ratings, movies, test_index, temp, movielens, removed):
## object 'movies' not found
```

## 3. Executive Summary

Since I amd using R version 3.6.0, there were issues reading the MovieLens files directly from GroupLens so a RDS copy of the edx and validation datasets were provided in a HarvardX_Capstone_MovieLens Google Drive. Link is below. https://drive.google.com/drive/folders/1IZcBBX0OmL9wu9AdzMBFUG8GoPbGQ38D

The edX dataset is a subset of the MovieLens 10M data table made of 6 variables and a total of 8,100,065 observations and the validation dataset represents approximately 10% or 899,990 observations and contains the same 6 variables.

Each observations or row contains the variables or columns "userid", "movieId", "rating", "timestamp", "title", "genres".

Quantitative or numeric variables include the following:

*** userId - the unique identifier for the user.**

*** movieId - the unique identifier for the movie.**

Table 1: Head of edx dataset

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 1 | 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy\|Romance |
| 4 | 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action\|Drama\|Sci-Fi\|Thriller |
| 5 | 1 | 316 | 5 | 838983392 | Stargate (1994) | Action\|Adventure\|Sci-Fi |
| 6 | 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action\|Adventure\|Drama\|Sci-Fi |
| 7 | 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children\|Comedy\|Fantasy |
| 8 | 1 | 356 | 5 | 838983653 | Forrest Gump (1994) | Comedy\|Drama\|Romance\|War |

Table 2: Summary stats for edx dataset

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| | Min. : 1 | Min. : 1 | Min. :0.500 | Min. :7.897e+08 | Length:8100065 | Length:8100065 |
| | 1st Qu.:18127 | 1st Qu.: 648 | 1st Qu.:3.000 | 1st Qu.:9.468e+08 | Class :character | Class :character |
| | Median :35732 | Median : 1834 | Median :4.000 | Median :1.035e+09 | Mode :character | Mode :character |
| | Mean :35870 | Mean : 4120 | Mean :3.512 | Mean :1.033e+09 | NA | NA |
| | 3rd Qu.:53607 | 3rd Qu.: 3624 | 3rd Qu.:4.000 | 3rd Qu.:1.127e+09 | NA | NA |
| | Max. :71567 | Max. :65133 | Max. :5.000 | Max. :1.231e+09 | NA | NA |

*** timestamp - unix timestamp when the user rating was provided in seconds since Unix Epoch on January 1st, 1970 at UTC.**

Qualitative or non-numeric variables include the following:

*** title: movie title with year of release in ()**

*** genres: genres associated with the movie separated by**

The variable or column "rating" is the outcome we want to predict, y.

*** rating : a numeric rating between 0 and 5 for the movie in 0.5 increments.**

*MovieLens edx dataset*

```
edx_head <- head(edx)

kable(edx_head, "latex", booktabs = T, caption = "Head of edx dataset")
```

```
edx_summary <- summary(edx)
kable(edx_summary, "latex", booktabs = T, caption = "Summary stats for edx dataset")
```

```
edx_unique <- data.table(validation %>% summarize(users = n_distinct(userId),
                                                   movies = n_distinct(movieId)))
kable(edx_unique, "latex", booktabs = T, caption = "Edx dataset - unique userIds and movieIds")
```

*MovieLens validation dataset*

Table 3: Edx dataset - unique userIds and movieIds

| users | movies |
|---|---|
| 68159 | 9701 |

Table 4: Head of validation dataset

| userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|
| 1 | 185 | 5 | 838983525 | Net, The (1995) | Action\|Crime |
| 2 | 260 | 5 | 868244562 | Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) | Action\|Adven |
| 2 | 590 | 5 | 868245608 | Dances with Wolves (1990) | Adventure\|Dr |
| 2 | 1049 | 3 | 868245920 | Ghost and the Darkness, The (1996) | Action\|Adven |
| 2 | 1210 | 4 | 868245644 | Star Wars: Episode VI - Return of the Jedi (1983) | Action\|Adven |
| 3 | 1148 | 4 | 1133571121 | Wallace & Gromit: The Wrong Trousers (1993) | Animation\|Cl |

```
val_head <- head(validation)

kable(val_head, "latex", booktabs = T, caption = "Head of validation dataset")
```

```
val_summary <- summary(validation)
kable(val_summary, "latex", booktabs = T, caption = "Summary stats for validation dataset")
```

```
val_unique <- data.table(validation %>% summarize(users = n_distinct(userId),
                                                  movies = n_distinct(movieId)))
kable(val_unique, "latex", booktabs = T, caption = "Validation dataset - unique userIds and movieIds")
```

## 4. Methods and Analysis:

### 4.1. Data exploration and visualization

The MovieLens edx and validation datasets contain a unix timestamp field for when the user rating was provided in seconds since Jan. 1, 1970 at UTC. In order to use the variable as a factor for visualization and modeling, I needed to transform it into standard datatime.

I used the as_datetime function in the lubridate package to mutate in the datatime format. I then created a scatterplot of y = average ratings vs x = date grouped by month and added geom_smooth line option for improved visualization of the date trend of ratings.

As you can see from the chart below, the variability of the average monthly rating decreased over time and converged to an average monthly rating of approximately 3.5 after 2005 when there was a higher frequency of ratings and movies in the edx dataset.

```
edx <- data.table(edx)
edx$timestamp <- as_datetime(edx$timestamp)

validation <- data.table(validation)
validation$timestamp <- as_datetime(validation$timestamp)
```

Table 5: Summary stats for validation dataset

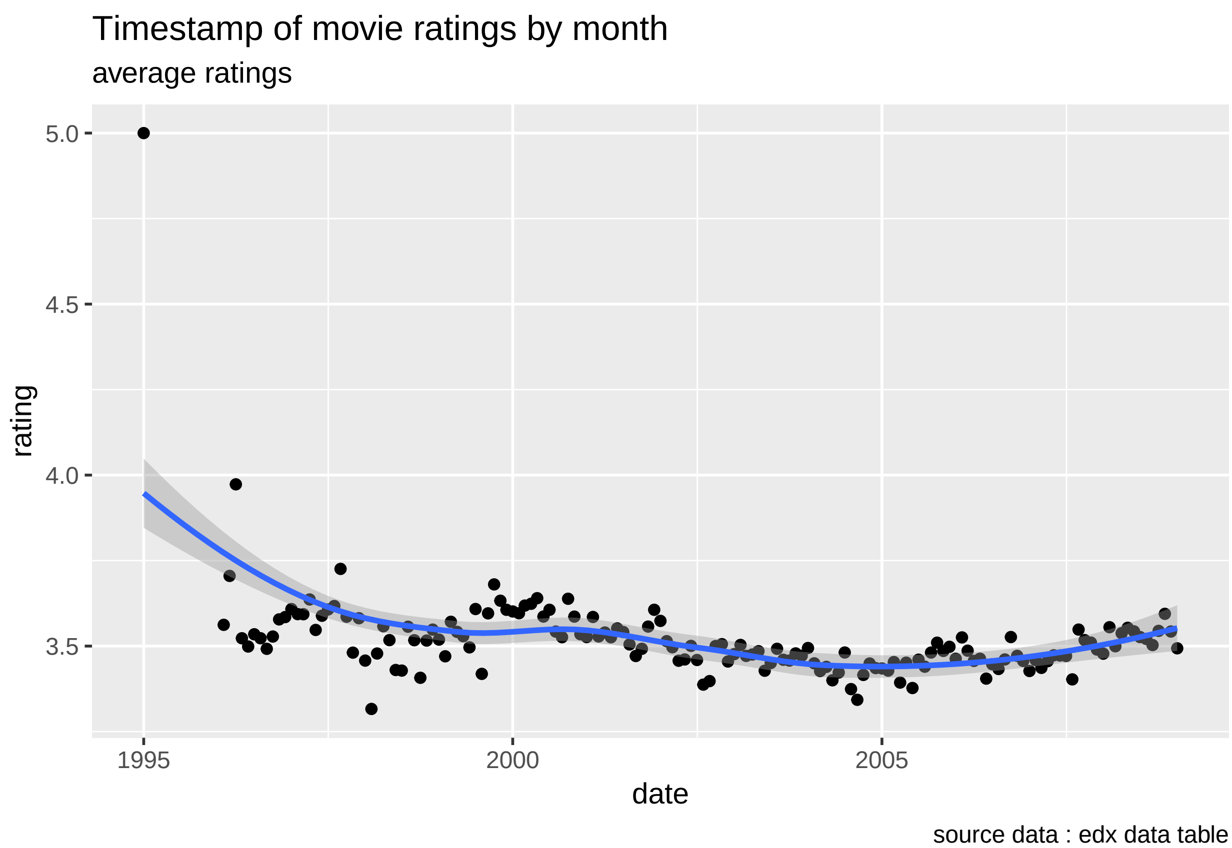
| userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|
| Min. : 1 | Min. : 1 | Min. :0.500 | Min. :7.897e+08 | Length:899990 | Length:899990 |
| 1st Qu.:18105 | 1st Qu.: 648 | 1st Qu.:3.000 | 1st Qu.:9.468e+08 | Class :character | Class :character |
| Median :35761 | Median : 1834 | Median :4.000 | Median :1.036e+09 | Mode :character | Mode :character |
| Mean :35868 | Mean : 4133 | Mean :3.513 | Mean :1.033e+09 | NA | NA |
| 3rd Qu.:53598 | 3rd Qu.: 3638 | 3rd Qu.:4.000 | 3rd Qu.:1.127e+09 | NA | NA |
| Max. :71567 | Max. :65130 | Max. :5.000 | Max. :1.231e+09 | NA | NA |

Table 6: Validation dataset - unique userIds and movieIds

| users | movies |
|---|---|
| 68159 | 9701 |

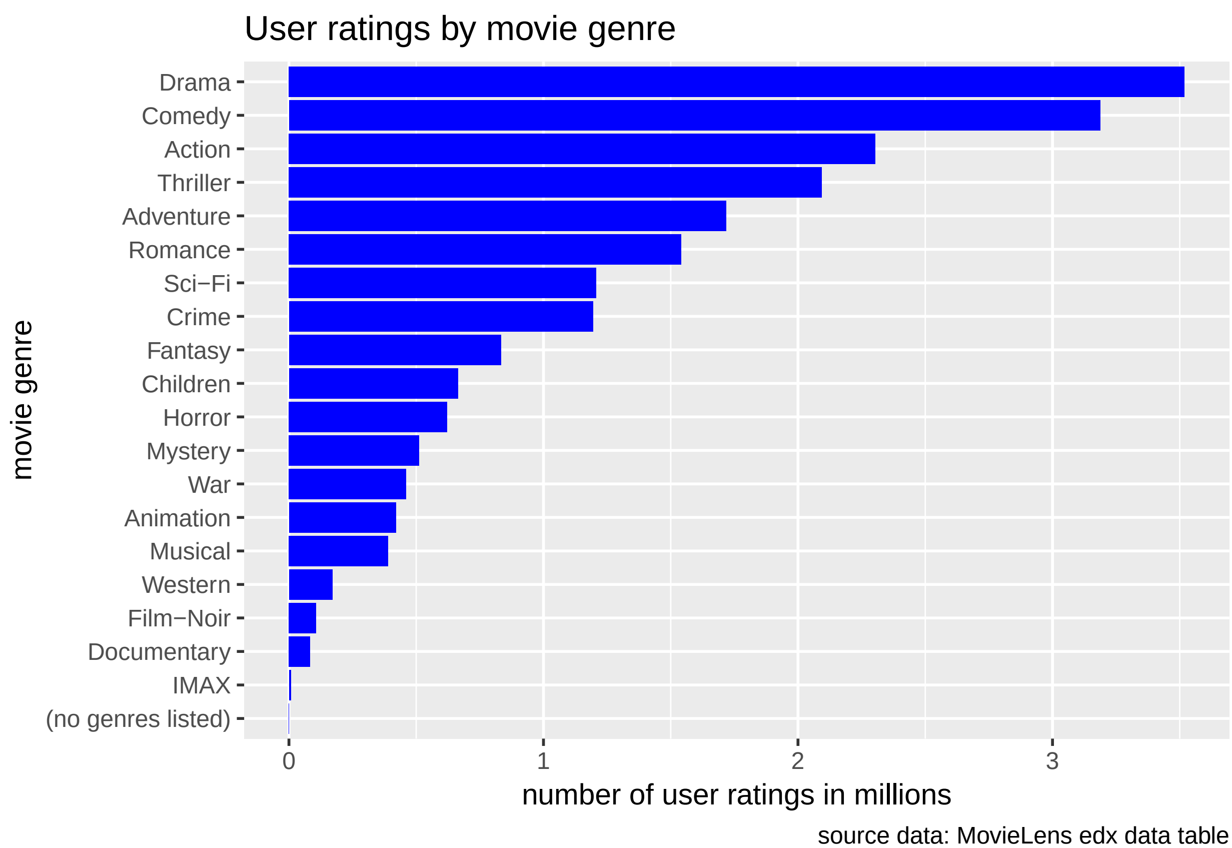
```
edx %>%
  mutate(date = round_date(timestamp, unit = "month")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp of movie ratings by month")+
  labs(subtitle = "average ratings",
       caption = "source data : edx data table")
```

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



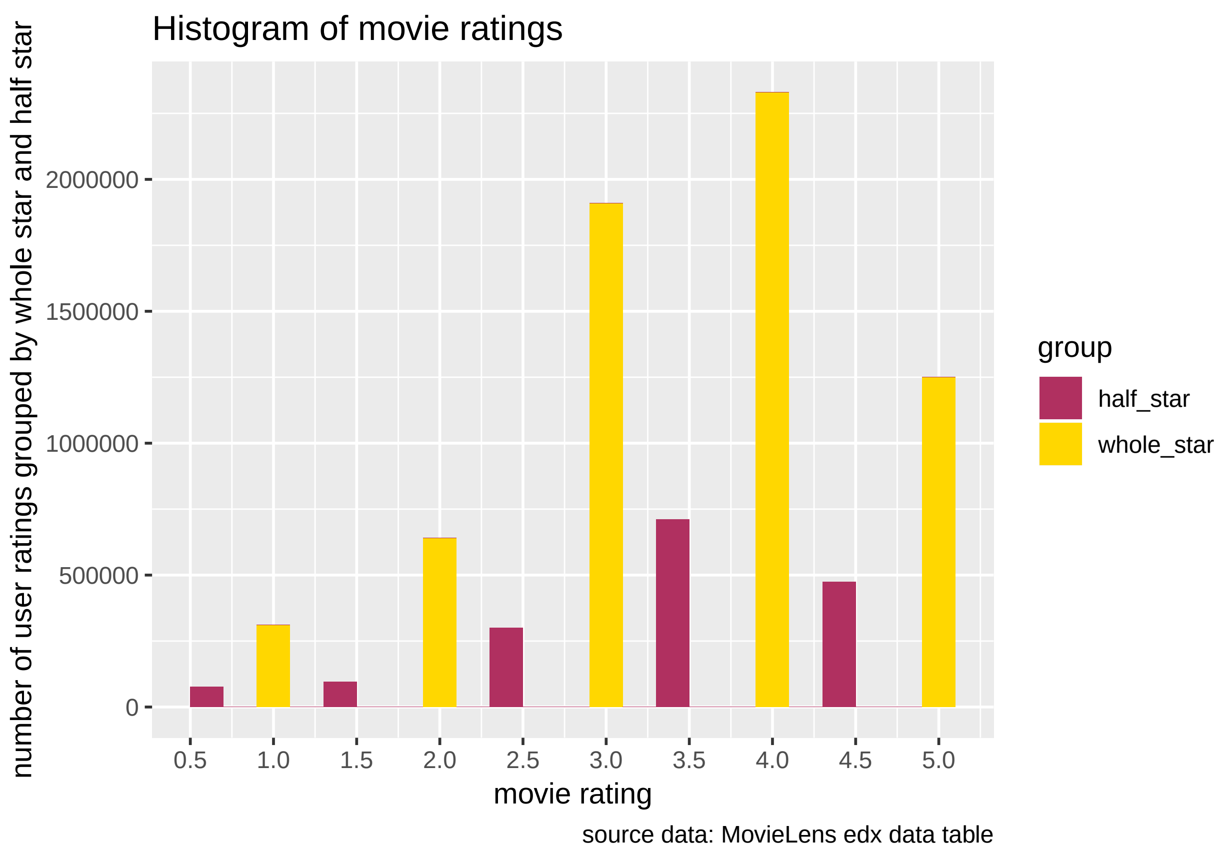

Grouping user ratings by movie genre for the edx dataset, the genres of "Drama" and "Comedy" were the most popular with over three million user ratings followed by "Action" and "Thriller" with over 2 million user ratings. Documentary and IMAX genre movies were the least rated by users in the edx dataset.

```r
# Bar chart of user ratings by movie genre for the edx data table
movie_genre <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

movie_genre <- data.table(movie_genre)
movie_genre <- movie_genre[order(-count),]

ggplot(data=movie_genre, aes(x=reorder(movie_genre$genres,movie_genre$count),
                             y=sapply(movie_genre$count, function(y) y/1000000),
                             fill=I("blue"))) +
  geom_bar(position="dodge",stat="identity") +
  coord_flip() +
  labs(x="movie genre", y="number of user ratings in millions",
       caption = "source data: MovieLens edx data table") +
  ggtitle("User ratings by movie genre")
```




Exploring whole and half star ratings for the edx dataset, the most popular rating is 4.0 with over two million user ratings followed by 3.0 and 5.0. Half star ratings in general are less popular than whole ratings as shown in the below histogram.

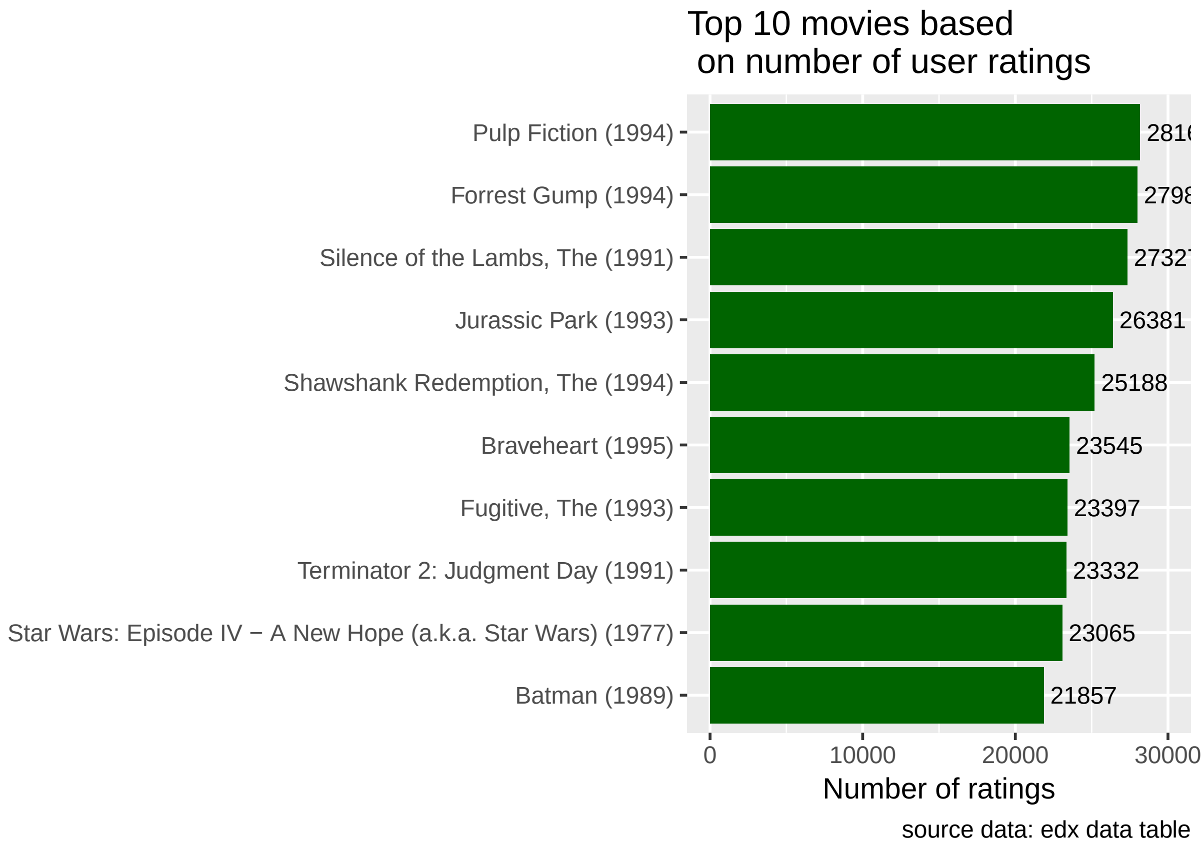
```
# Histogram of movie ratings grouped by whole star and half star ratings for the edx data table
group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                    edx$rating == 4 | edx$rating == 5) ,
                 "whole_star",
                 "half_star")

edx_ratings <- data.table(edx$rating, group)

ggplot(edx_ratings, aes(x= edx$rating, fill = group)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  scale_fill_manual(values = c("half_star"="maroon", "whole_star"="gold")) +
  labs(x="movie rating", y="number of user ratings grouped by whole star and half star",
       caption = "source data: MovieLens edx data table") +
  ggtitle("Histogram of movie ratings")
```




Ranking the top 10 movies by number of user ratings in the edx dataset, Pulp Fiction (1994) had the most user ratings followed by Forest Gump (1994), Silence of the Lambs, The (1991), Jurassic Park (1993), and Shawshank Redemption, The (1994). These top 5 movies are also part of the Drama, Comedy, or Action genres which ranked at the top of user ratings by genre.

Interestingly, movies released in the 1990's dominated the top 10 movies with all but the bottom two Star Wars: Episode IV - A New Hope (a.k.a Star Wars)(1977) and Batman (1989) not in that decade.

```
# Create top_movies dataframe from edx data table which contains the top 10 movies by number of user ra

top_movies <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(10,count) %>%
  arrange(desc(count))

# Bar chart of top_movies

top_movies %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="dark green") + coord_flip(y=c(0, 30000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Top 10 movies based \n on number of user ratings" , caption = "source data: edx data tabl
```




### 4.2. Data preprocessing and transformation

The dependent variables userId and movieId should be treated as factors for linear regression modeling purposes.To perform this transformation we make a copies of edx dataset and renamed as training_set and validation dataset and rename as test_set.

```
training_set <- edx

training_set$userId <- as.factor(training_set$userId)
training_set$movieId <- as.factor(training_set$movieId)
```

I added a column to the training_set for date which is the month and year for the rating and set it as a factor.

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

training_set$date <- as.factor(training_set$date)
```

Due to the large size of the edx dataset I selected the training_set columns with the quantitative dependent variables to by used in the linear regression modeling process userId, movieId, date, and the predictive outcome variable y, rating.

```r
training_set <- training_set %>% select(userId, movieId, date, rating)
```

I transformed the validation dataset the same as the training_set with columns userId, movieId, date, and rating as the test_set.

```r
test_set <- validation

test_set$userId <- as.factor(test_set$userId)
test_set$movieId <- as.factor(validation$movieId)

test_set <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "month"))

test_set$date <- as.factor(test_set$date)

test_set <- test_set %>% select(userId, movieId, date, rating)
```

### 4.3. Evaluated Machine Learning Algorithms

#### 4.3.1 Baseline Naive Model

The baseline naive model that I will be trying to beat with more advanced linear regression models is generated by using the average rating ($\mu$) of 3.51245564078807 on the training_set to be predicted into the test_set.

The *baseline-naive model* is calculated as follows:

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

***   is the "true" rating for all movies.**

***   (epsilon) are independent errors sampled from the same distribution centered at 0.**

The resulting RMSE for the baseline model is 1.060054 which is a poor model and a marginally better predictive model than random guessing at 1.177464 RMSE. I will try to improve on this model using dependent variables movieId, userId, and date in linear regression models along with regularization methods.

```r
# Random guessing rating across all movies.
random_guess <- rep(3, nrow(test_set))
RMSE_random <- RMSE(test_set$rating, random_guess)

rmse_table <- data_frame(Method = "Random guessing rating across all movies" , RMSE = RMSE_random)
```

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

```r
rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 7: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.177464 |

```
# Baseline using average rating across all movies.
mu_hat_baseline <- mean(training_set$rating)
RMSE_baseline <- RMSE(test_set$rating, mu_hat_baseline)

rmse_table <- rbind(rmse_table,
                    data_frame(Method = "Naive model using mean rating across all movies", RMSE = RMSE_

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 8: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.177464 |
| Naive model using mean rating across all movies | 1.060054 |

### 4.3.2 Simple Linear Regression Model

I followed the same approach to build the linear regression models as the simplest recommendation systems described by professor Rafa Irizarry in the dsbook sourced from the github page https://rafalab.github.io/dsbook/.

The simple linear regression model that I will generate first uses the rating by movieId bias or the *movie effect* $b_i$ dependent variable on the training_set to predict the `rating` ($Y_{u,i}$ for the test_set. According to professor Irizarry, statistics textbooks refer to to the $b_s$ as effects. However, in the Netflix challenge papers, they refer to them as "bias", thus the $b$ notation.

The *movie-specific effect model* is calculated as follows:

$$Y_i = \mu + b_i + \varepsilon_i$$

where:

***   is the "true" rating for all movies.**

*** bi effects or bias, movie-specific effect.**

***   (epsilon) are independent errors sampled from the same distribution centered at 0.**

The resulting RMSE from this *movie-specific effect model* on the test_set was an improvement on the baseline naive model RMSE at 0.9429615. We can likely do better with adding more dependent variables to the model.

```
# Fitting Simple Regression Model to the edx training set.

# The average of all movie ratings mu of the training_set.
mu <- mean(training_set$rating)
```

```r
# Group average ratings by movieId on the training set.
movie_avgs <- training_set %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))

# Predicted ratings using movie-specific effect.
predicted_ratings_bi <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$bi
```

```
## Warning: Column `movieId` joining factors with different levels, coercing
## to character vector
```

```r
movie_model <- RMSE(predicted_ratings_bi, test_set$rating)

rmse_table <- rbind(rmse_table,
                    data_frame(Method = "Predicted movie ratings regression model using movie-specific e

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 9: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.1774645 |
| Naive model using mean rating across all movies | 1.0600537 |
| Predicted movie ratings regression model using movie-specific effect | 0.9429615 |

#### 4.3.2 Multiple Linear Regression Models

The next regression model that I will generate uses the rating by movieId bias or the *movie effect* ($m_i$) and userId bias or *user effect* ($b_u$) dependent variables on the training_set to predict the `rating` $Y_{u,i}$ for the test_set.

The *movie-user-specific effect model* is calculated as follows:

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

where:

***  is the "true" rating for all movies.**

*** bi effects or bias, movie-specific effect.**

*** bu effects or bias, user-specific effect.**

***  (epsilon) are independent errors sampled from the same distribution centered at 0.**

The resulting RMSE from this *movie-user-specific effect model* on the test_set was an improvement on the *movie-specific effect model* RMSE at 0.8646843. Let's see if we can do better with adding the last dependent variable date to the model.

```r
# Fitting Multiple Regression Models to the edx training set.

# Predicted movie ratings regression model using both movieId and userId averages.

user_avgs <- training_set  %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi))

predicted_ratings_bi_bu <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + bi + bu) %>%
  .$pred
```

```
## Warning: Column `movieId` joining factors with different levels, coercing
## to character vector
```

```
## Warning: Column `userId` joining factors with different levels, coercing to
## character vector
```

```r
movie_user_model <- RMSE(predicted_ratings_bi_bu, test_set$rating)

rmse_table <- rbind(rmse_table,
                    data_frame(Method = "Predicted movie ratings regression model using movie-user-speci

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 10: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.1774645 |
| Naive model using mean rating across all movies | 1.0600537 |
| Predicted movie ratings regression model using movie-specific effect | 0.9429615 |
| Predicted movie ratings regression model using movie-user-specific effect | 0.8646843 |

The last linear regression model that I will generate uses the rating by movieId bias or the *movie effect* ($b_i$), userId bias or *user effect* ($b_u$), and date bias or *date effect* ($b_d$) dependent variables on the training_set to predict the `rating` $Y_{i,u,d}$ for the test_set.

The *movie-user-date-specific effect model* is calculated as follows:

$$Y_{i,u,d} = \mu + b_i + b_u + b_d + \varepsilon_{i,u,d}$$

where:

***   is the "true" rating for all movies.**

*** bi effects or bias, movie-specific effect.**

*** bu effects or bias, user-specific effect.**

*** bd effects or bias, date-specific effect.**

***   (epsilon) are independent errors sampled from the same distribution centered at 0.**

The resulting RMSE from this *movie-user-date-specific effect model* on the test_set was a neligible improvement on the *movie-user-specific effect model* RMSE at 0.8646637. Next step is if regularization method improves the model RMSE.

```
# Predicted movie ratings regression model using movieId, userId, and date averages.

date_avgs <- training_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(date) %>%
  summarize(bd = mean(rating - mu - bi - bu))

predicted_ratings_bi_bu_bd <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(date_avgs, by='date') %>%
  mutate(pred = mu + bi + bu + bd) %>%
  .$pred
```

```
## Warning: Column `movieId` joining factors with different levels, coercing
## to character vector
```

```
## Warning: Column `userId` joining factors with different levels, coercing to
## character vector
```

```
movie_user_date_model <- RMSE(predicted_ratings_bi_bu_bd, test_set$rating)

rmse_table <- rbind(rmse_table,
                    data_frame(Method = "Predicted movie ratings regression model using movie-user-date-

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 11: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.1774645 |
| Naive model using mean rating across all movies | 1.0600537 |
| Predicted movie ratings regression model using movie-specific effect | 0.9429615 |
| Predicted movie ratings regression model using movie-user-specific effect | 0.8646843 |
| Predicted movie ratings regression model using movie-user-date-specific effect | 0.8646637 |

#### 4.3.3 Regularization

I followed the same approach for the regularization method for recommendation systems described by professor Irizarry in the dsbook sourced from the github page https://rafalab.github.io/dsbook/.

The formula for the regularization method on the *movie-user-specific effect model* is as follows.

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

The *movie-user-specific effect model* with the regularization method on the test_set to seeks to minimize the RMSE using cross validation to pick a

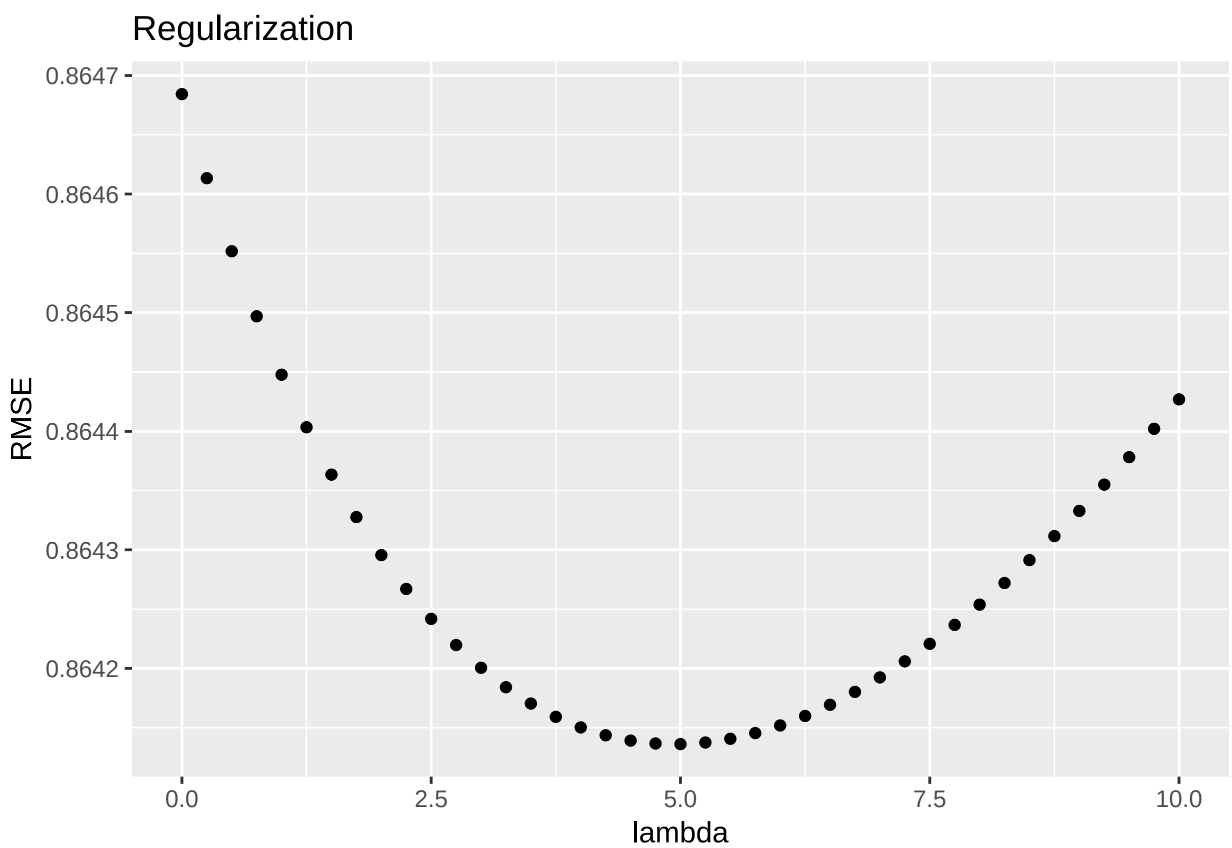
$$\lambda$$

. In this case, the

$$\lambda$$

that optimized (minimized) the RMSE was 5.

```
# Regularization using movieId and userId dependent variables regression prediction model.

lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(training_set$rating)

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

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

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

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

qplot(lambdas, rmses,
      main = "Regularization",
      xlab = "lambda", ylab = "RMSE") # lambda vs RMSE
```




```
lambda_opt <- lambdas[which.min(rmses)]
lambda_opt # lambda which optimizes the model (minimizes RMSE) which is 5 in this case
```

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

```
rmse_table <- rbind(rmse_table,
                    data_frame(Method = "Regularization of ratings regression prediction model using mov
                               RMSE = min(rmses)))

rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 12: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.1774645 |
| Naive model using mean rating across all movies | 1.0600537 |
| Predicted movie ratings regression model using movie-specific effect | 0.9429615 |
| Predicted movie ratings regression model using movie-user-specific effect | 0.8646843 |
| Predicted movie ratings regression model using movie-user-date-specific effect | 0.8646637 |
| Regularization of ratings regression prediction model using movieId and userid averages | 0.8641362 |

## 5. Results:

The minimized RMSE is obtained from the *movie-user-specific effect model* using the regularization method with a $\lambda = 5$. The resulting minimized RMSE was 0.8641362, which was close to the Netflix challenge winning solution.

More advanced ML algorithms with matrix factorization, dimension reduction, ensamble methods could improve upon the RMSE although computational resources were limited to my personal laptop. Also, there was limited time to research different analytic modeling techniques such as *random forest*,*decision tree*, *support vector machine*, *gbm*, *neural network*, *ensamble*, and others. The large size of the MovieLens dataset made most of the ML libraries unusable unless a smaller subset of the dataset was used which was not allowed for this particular project.

```
rmse_table %>% knitr::kable(caption = "RMSEs")
```

Table 13: RMSEs

| Method | RMSE |
|---|---|
| Random guessing rating across all movies | 1.1774645 |
| Naive model using mean rating across all movies | 1.0600537 |
| Predicted movie ratings regression model using movie-specific effect | 0.9429615 |
| Predicted movie ratings regression model using movie-user-specific effect | 0.8646843 |
| Predicted movie ratings regression model using movie-user-date-specific effect | 0.8646637 |
| Regularization of ratings regression prediction model using movieId and userid averages | 0.8641362 |

## 6. Conclusion:

The main learning objective of the HarvardX: Introduction to Data Science program was to give aspiring data scientists like myself the tools in R to run analytic models using machine learning to make predictions and solve real world problems. This wapurls a fascinating journey over the past six months and I look forward to improving my data science skills in R and Python through my work and personally through Kaggle competitions that I have an interest in.

My "Harvard_Data_Science_MovieLens Github repository" is **in this link**

## *References*


*** Irizzary,R., 2019. Introduction to Data Science,**

**github page,https://rafalab.github.io/dsbook/**

*** Koren, Y., 2009. The BellKor Solution to the Netflix Grand Prize.**

**Netflix prize documentation,**

**https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf**