PH125.9x Capstone Project 1: MovieLens

Somnath Saha

12/06/2021

Contents

1 Overview

This project is part of the final course - **Data Science: Capstone** in HarvardX's multi-part **Data Science Professional Certificate** series offered via the edX platform.

A movie recommendation system is developed using the tools - the powerful R language & it's associated libraries on the RStudio - and the expert training provided in this series. The 10M version of the MovieLens dataset is used to train the prediction model. The data is cleaned up, analyzed and used to develop a model that can predict movie ratings within a target RMSE.

1.1 About the dataset

GroupLens is a research group in the *Department of Computer Science and Engineering* at the *University of Minnesota*. The 10M version of the MovieLens dataset is provided by GroupLens that contains **10000054** ratings and **95580** tags applied to **10681** movies by **71567** users of the online movie recommender service MovieLens.
Go to movielens website: *https://grouplens.org/datasets/movielens/10m*
Access dataset: *http://files.grouplens.org/datasets/movielens/ml-10m.zip*

1.2 Goal of the project

The goal of the project is train a model that attains an RMSE value of less than **0.86490** on the final validation set. The final validation set will neither be analyzed for its contents nor be used to train the model in any manner. Although RMSE values greater than the **0.86490** would also fetch some passing score as per the project goals but the work done in the project has successfully developed a model of RMSE in the perfect target of below **0.86490**.

1.3 Key steps to reach the final model

The initial parts of the project will help to gain insights into the data through tabular and visual means. This is done in R language and use the publicly available powerful *R* libraries like ggplot. Next, models are trained with the different available effects independently and in cumulative fashion. Studying and leveraging the observations from such models, a final model is developed with the proper order of effects as well as acoomodating linearization to compensate for any outliers. A final model is developed using the training set which gives an RMSE $<$ 0.86490 on the validation step.

2 Ready the data

2.1 Fetch the data

The following code is made availavle by the course to fetch the data from the grouplens website and extract a working dataset (edx) and a final validation dataset (validation) out of the dataset.

```
# Install any missing packages
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")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")

##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
# dl <- tempfile()
# download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
#
# ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
#                  col.names = c("userId", "movieId", "rating", "timestamp"))
#
# movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
# colnames(movies) <- c("movieId", "title", "genres")
#
# # if using R 4.0 or later
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
#                                            title = as.character(title),
#                                            genres = as.character(genres))
# movielens <- read.csv("/Users/ssaha/Documents/datascience/data-science-harvardx/capstone/project_movi
# movielens <- left_join(ratings, movies, by = "movieId")
movielens <- read.csv("movielensRed.csv")
#movielens <- read.csv("~/Documents/datascience/data-science-harvardx/movielens.csv")


# Validation set will be 10% of MovieLens data
set.seed(27, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

```
rm(dl, ratings, movies, test_index, temp, removed) #add movielens for production
```

2.2 Cleanup the data for more valuable features

A re-usable function is defined to extract and add new columns for release and review years from already available data. The edX dataset is processed here. The validation dataset will be processed after the final model is developed.

```
add_release_and_review_years <- function(dataset) {
  # Trim and split title (year) column into title and year columns
  dataset <- dataset %>% mutate(title = str_trim(title)) %>%
            extract(title, c("title_temp", "release_year"),
                    regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F) %>%
            mutate(release_year = if_else((str_length((release_year)) > 4),
                    as.integer(str_split(release_year, "-", simplify = T)[1]),
                    as.integer(release_year))) %>%
            mutate(title = if_else(is.na(title_temp), title, title_temp)) %>%
            select(-title_temp)

   # Convert timestamp column into date format, removing time data & get review year from it
  dataset <- dataset %>% mutate(review_date = round_date(as_datetime(timestamp), unit = "week"))
  dataset <- dataset %>% mutate(review_year = year(review_date))
  dataset
}

# Add the release and review years to edx set as well
edx <- add_release_and_review_years(edx)
```

3 Data analysis and visualization

3.1 Brief look at edx dataset

3.1.1 Structure of edx dataset: Column Names & Types

```
str(edx)
```

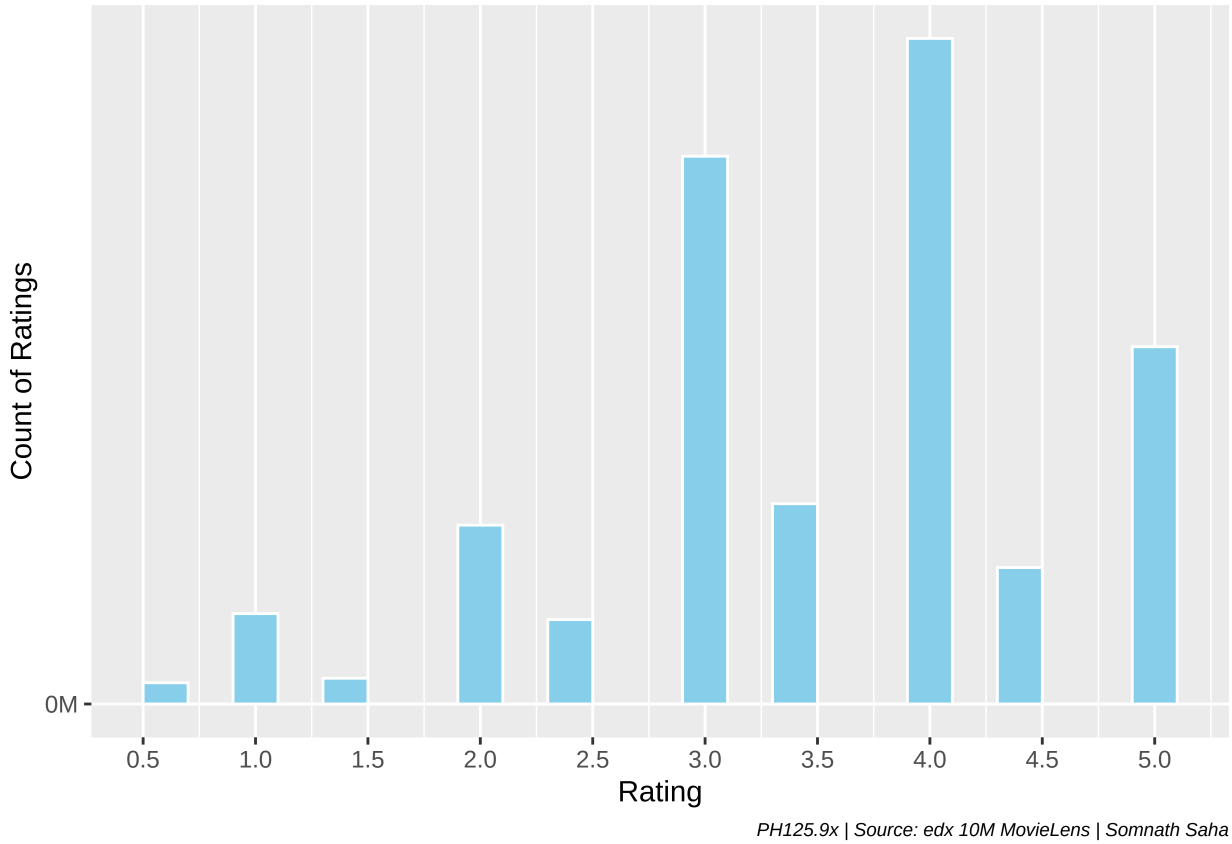
```
## 'data.frame':    92217 obs. of  9 variables:
##  $ userId      : int  7 8 8 8 8 11 14 18 18 18 ...
##  $ movieId     : int  1276 256 4299 6977 7438 1258 2012 349 380 1028 ...
##  $ rating      : num  4.5 2.5 3.5 4 4 3 3 3.5 4 3 ...
##  $ timestamp   : int  1054292598 1115860190 1111624114 1116549866 1111624051 945878259 1133574782 11
##  $ title       : chr  "Cool Hand Luke" "Junior" "Knight's Tale, A" "New Jack City" ...
##  $ release_year: int  1967 1994 2001 1991 2004 1980 1990 1994 1994 1964 ...
##  $ genres      : chr  "Drama" "Comedy|Sci-Fi" "Action|Adventure|Comedy" "Action|Crime|Drama" ...
##  $ review_date : POSIXct, format: "2003-06-01" "2005-05-15" ...
##  $ review_year : num  2003 2005 2005 2005 2005 ...
```

3.1.2 Data in edx dataset: First few rows

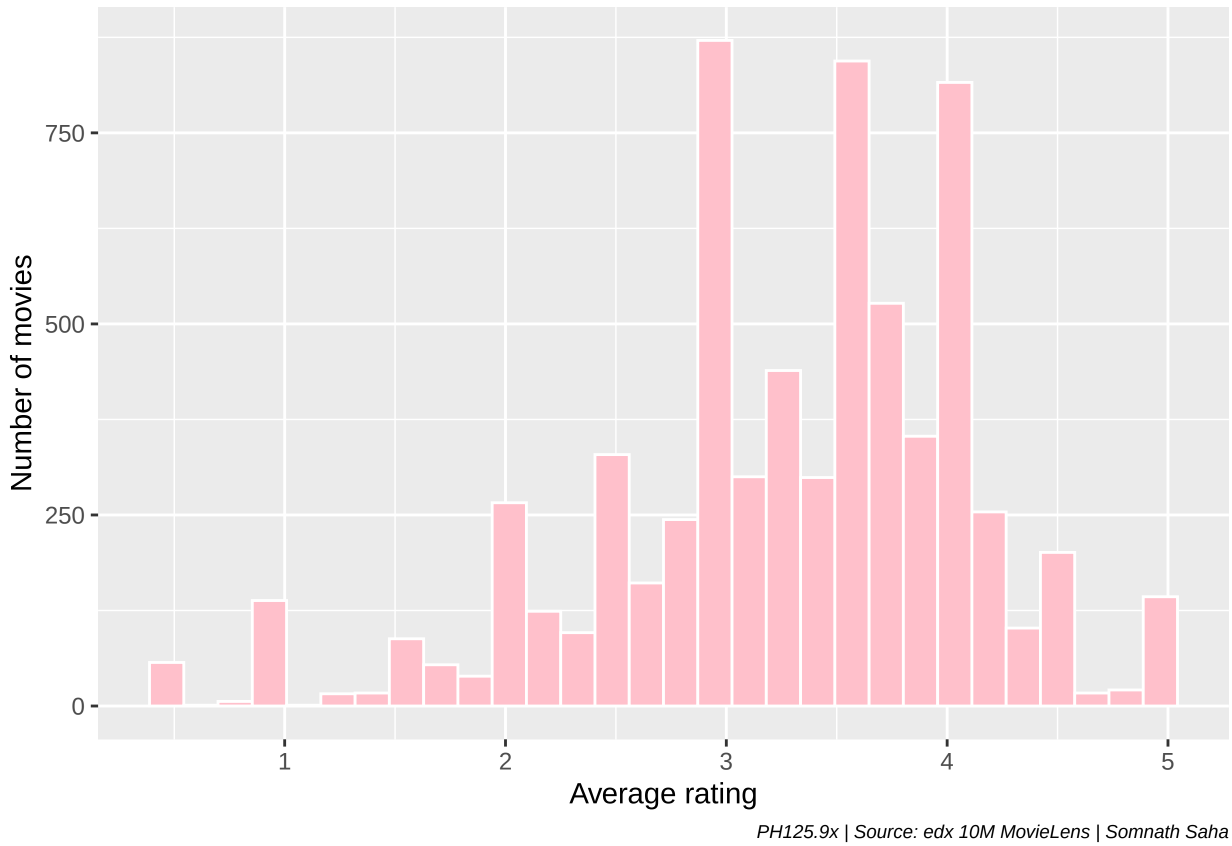
```
head(edx)
```

```
##   userId movieId rating  timestamp             title release_year
## 2      7    1276    4.5 1054292598    Cool Hand Luke         1967
## 3      8     256    2.5 1115860190            Junior         1994
## 5      8    4299    3.5 1111624114  Knight's Tale, A         2001
## 6      8    6977    4.0 1116549866     New Jack City         1991
## 7      8    7438    4.0 1111624051 Kill Bill: Vol. 2         2004
## 8     11    1258    3.0  945878259      Shining, The         1980
##                    genres review_date review_year
## 2                   Drama  2003-06-01        2003
## 3           Comedy|Sci-Fi  2005-05-15        2005
## 5 Action|Adventure|Comedy  2005-03-27        2005
## 6      Action|Crime|Drama  2005-05-22        2005
## 7   Action|Drama|Thriller  2005-03-27        2005
## 8         Horror|Thriller  1999-12-26        1999
```

3.2 Plot distribution of ratings in the edx dataset



3.3 Plot average rating by movie in the edx dataset



3.4 Separate individual genres and ranking them by the total number of ratings in the edx dataset

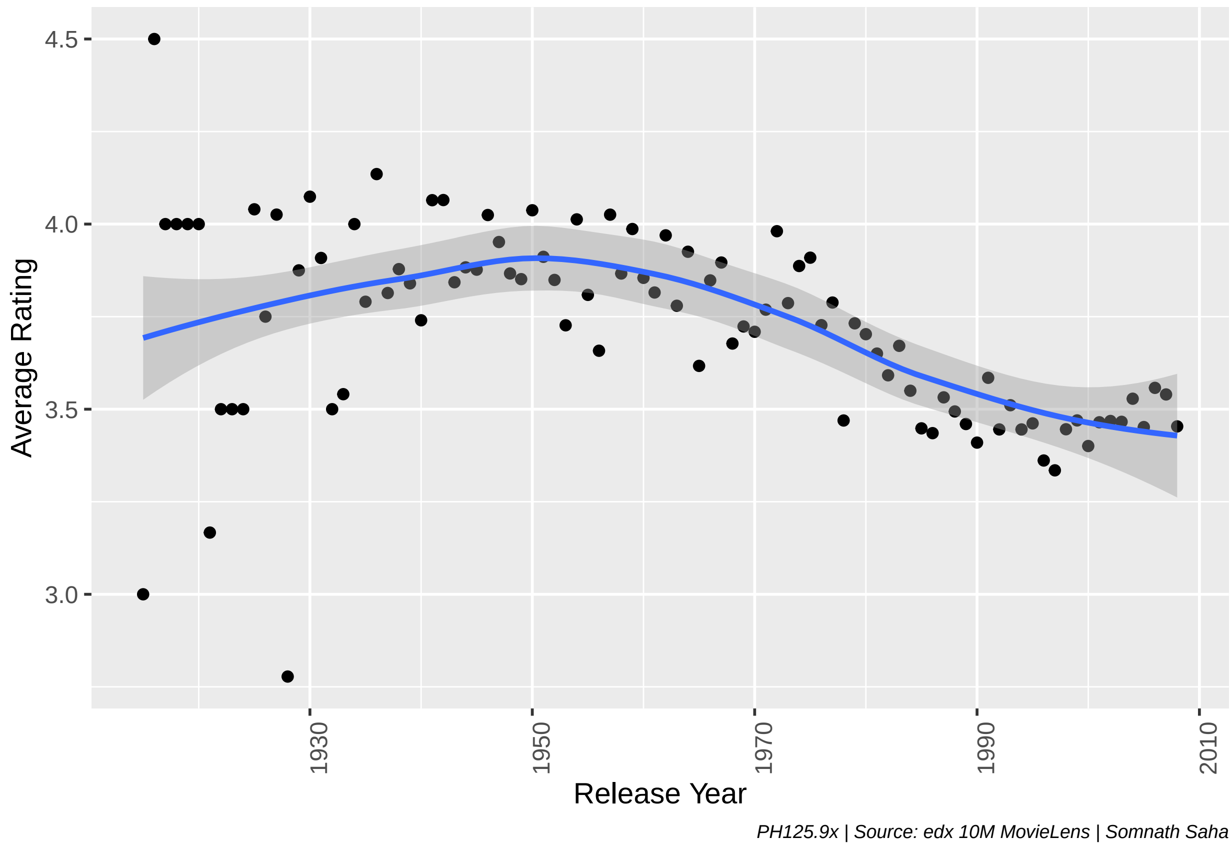
3.5 Plot average rating by genre for genre combinations with at least 50,000 ratings

3.6 Group and list top 10 movie titles based on number of ratings

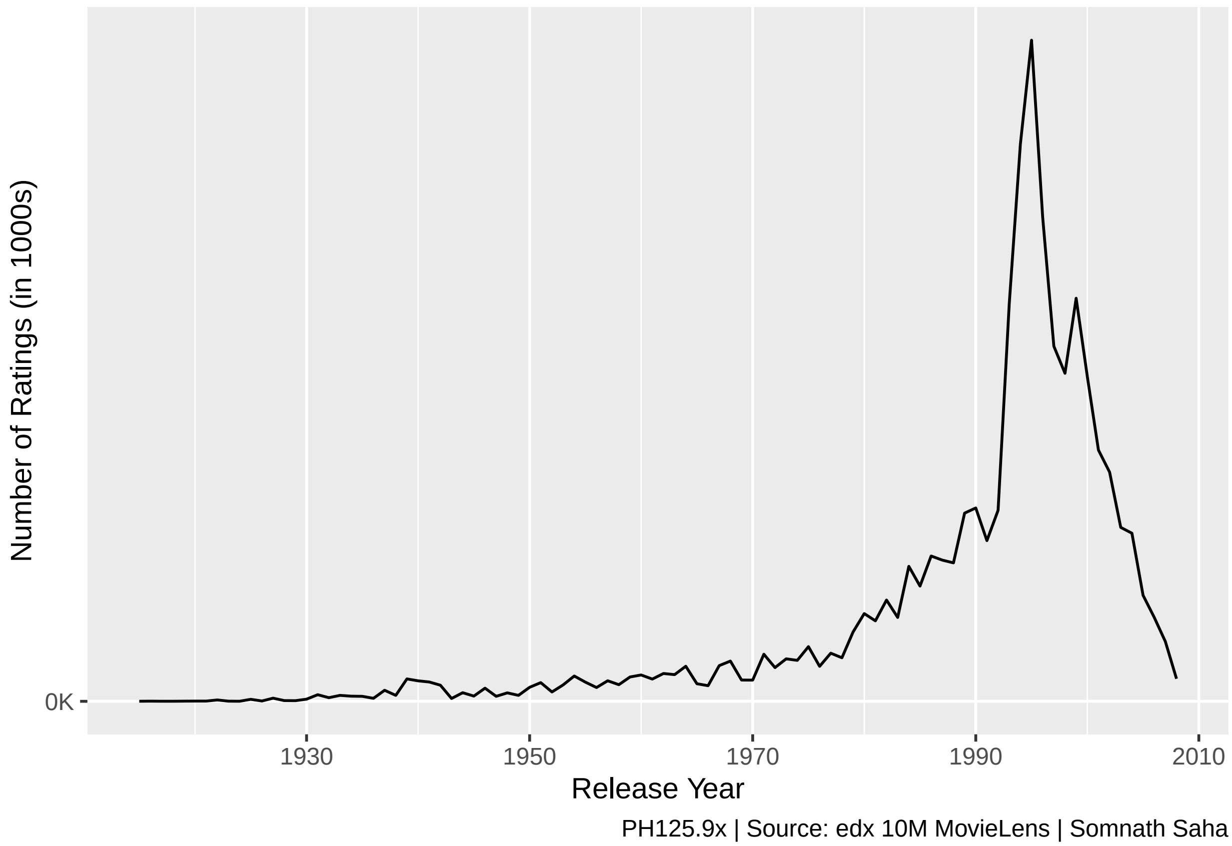
Table 1: Top 10 movie titles based on number of ratings

Movie Name	No. of Ratings	Avg Rating
Pulp Fiction	324	4.152778
Jurassic Park	317	3.643533
Silence of the Lambs, The	317	4.178233
Forrest Gump	308	3.944805
Braveheart	278	4.122302
Independence Day (a.k.a. ID4)	273	3.333333
Shawshank Redemption, The	272	4.351103
Terminator 2: Judgment Day	271	3.924354
Apollo 13	266	3.921053
Schindler's List	255	4.327451
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	255	4.103922
Toy Story	255	3.929412

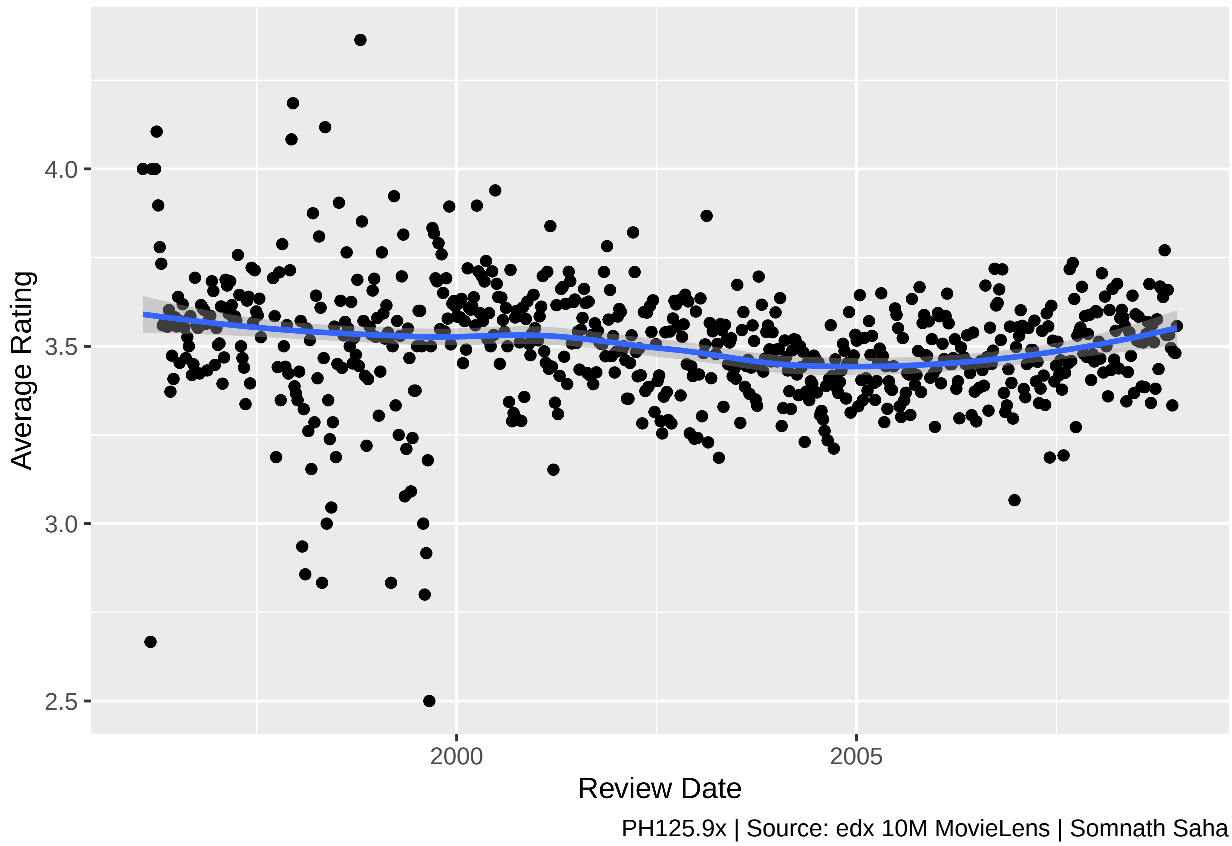
3.7 Plot average rating by year of release in the edx dataset



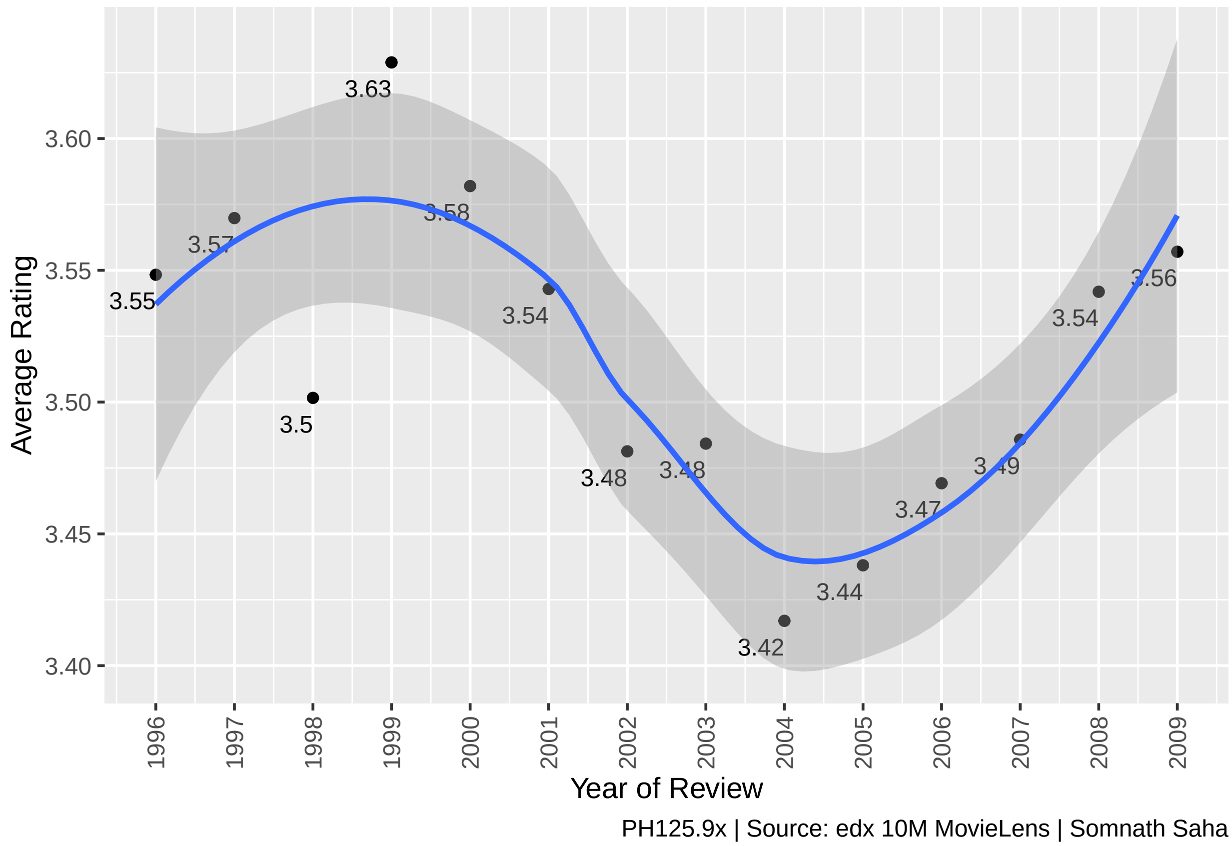
3.8 Plot number of ratings by year of release in the edx dataset



3.9 Plot average rating by date of review in the edx dataset



3.10 Plot average rating by year of review in the edx dataset



4 Creation of training and test datasets

The edx dataset is divided into a training set and test set to develop and test different models. 90% of the data as the training set is used to train the models and these models are tested on the remaining 10% of the test dataset. The model with the best prediction (lowest RMSE) on the test set is used as final model to predict ratings for the validation set.

```
set.seed(27, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]

# Keep only those rows in edx_test set which have
# movieId and userId existing in edx_train dataset
edx_test <- edx_test %>%
            semi_join(edx_train, by = "movieId") %>%
            semi_join(edx_train, by = "userId")

# Do the same for the validation set as well
validation <- validation %>%
              semi_join(edx_train, by = "movieId") %>%
              semi_join(edx_train, by = "userId")
```

Training Set: edx_train

```
## 'data.frame':    82994 obs. of  9 variables:
##  $ userId      : int  7 8 8 8 11 14 18 18 18 18 ...
##  $ movieId     : int  1276 4299 6977 7438 1258 2012 349 380 1028 2081 ...
##  $ rating      : num  4.5 3.5 4 4 3 3 3.5 4 3 2 ...
##  $ timestamp   : int  1054292598 1111624114 1116549866 1111624051 945878259 1133574782 1111545957 111
##  $ title       : chr  "Cool Hand Luke" "Knight's Tale, A" "New Jack City" "Kill Bill: Vol. 2" ...
##  $ release_year: int  1967 2001 1991 2004 1980 1990 1994 1994 1964 1989 ...
##  $ genres      : chr  "Drama" "Action|Adventure|Comedy" "Action|Crime|Drama" "Action|Drama|Thriller"
##  $ review_date : POSIXct, format: "2003-06-01" "2005-03-27" ...
##  $ review_year : num  2003 2005 2005 2005 1999 ...
```

Test Set: edx_test

```
## 'data.frame':    6910 obs. of  9 variables:
##  $ userId      : int  8 18 34 38 38 65 73 73 78 137 ...
##  $ movieId     : int  256 4703 990 2763 6886 2284 1348 2186 1097 5009 ...
##  $ rating      : num  2.5 4 3 4.5 4 4 4 4 3 3 ...
##  $ timestamp   : int  1115860190 1111547361 982513173 1170301544 1171228662 970393581 974297494 9742
##  $ title       : chr  "Junior" "Chocolat" "Maximum Risk" "Thomas Crown Affair, The" ...
##  $ release_year: int  1994 1988 1996 1999 2003 1994 1922 1951 1982 2001 ...
##  $ genres      : chr  "Comedy|Sci-Fi" "Drama" "Action|Adventure|Thriller" "Action|Mystery" ...
##  $ review_date : POSIXct, format: "2005-05-15" "2005-03-20" ...
##  $ review_year : num  2005 2005 2001 2007 2007 ...
```

RMSE function that will be used for all the different models

By definition,

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(actualData - predictedData)^2}$$

R function defined as -

```
RMSE <- function(actual_data, predicted_data){
  sqrt(mean((actual_data - predicted_data)^2))
}
```

5 Development of different models and their performance

5.1 Model 1.0: Simple mean of training set

```
# Get the mean rating as prediction value
mu <- mean(edx_train$rating)
# Get the RMSE for test set using mu as prediction value
rmse_model1 <- RMSE(edx_test$rating, mu)
```

Table 2: Simple Mean based Model

method	RMSE
Model 1.0: Simple mean of training set	1.054878

5.2 Model 2.x: Models with single effects

5.2.1 Model 2.1: Mean with movie effect

```
#Train on the edx training set to find movie effect
movie_effects     <-  edx_train %>%
                      group_by(movieId) %>%
                      summarize(b_m = mean(rating - mu))

#Generate predictions on the edx test set
predicted_ratings <-  mu + (edx_test %>%
                              left_join(movie_effects, by='movieId') %>%
                              .$b_m)

#Find RMSE for this model
rmse_model_b_m <- RMSE(edx_test$rating, predicted_ratings)
```

Table 3: RMSE Values for 2.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612

5.2.2 Model 2.2: Mean with user effect

```
#Train on the edx training set to find user effect
user_effects <-     edx_train %>%
                    group_by(userId) %>%
                    summarize(b_u = mean(rating - mu))

#Generate predictions on the edx test set
predicted_ratings <-  mu + (edx_test %>%
                              left_join(user_effects, by='userId') %>%
                              .$b_u)

#Find RMSE for this model
rmse_model_b_u <- RMSE(edx_test$rating, predicted_ratings)
```

Table 4: RMSE Values for 2.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005

5.2.3 Model 2.3: Mean with genre effect

```
#Train on the edx training set to find genres effect
genre_effects   <-  edx_train %>%
                    group_by(genres) %>%
                    summarize(b_g = mean(rating - mu))

#Generate predictions on the edx test set
predicted_ratings <-  mu + (edx_test %>%
                              left_join(genre_effects, by = 'genres') %>%
                              .$b_g)

#Find RMSE for this model
rmse_model_b_g <- RMSE(edx_test$rating, predicted_ratings)
```

Table 5: RMSE Values for 2.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850

5.2.4 Model 2.4: Mean with release year effect

```
#Train on the edx training set to find user effect
release_yr_effects <- edx_train %>%
                      group_by(release_year) %>%
                      summarize(b_yr_release = mean(rating - mu))

#Generate predictions on the edx test set
predicted_ratings <-  mu + (edx_test %>%
                              left_join(release_yr_effects, by='release_year') %>%
                              .$b_yr_release)

#Find RMSE for this model
rmse_model_b_yr_release <- RMSE(edx_test$rating, predicted_ratings)
```

Table 6: RMSE Values for 2.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696

5.2.5 Model 2.5: Mean with review year effect

```
#Train on the edx training set to find user effect
review_yr_effects <-  edx_train %>%
                      group_by(review_year) %>%
                      summarize(b_yr_review = mean(rating - mu))

#Generate predictions on the edx test set
predicted_ratings <-  mu + (edx_test %>%
                              left_join(review_yr_effects, by='review_year') %>%
                              .$b_yr_review)

#Find RMSE for this model
rmse_model_b_yr_review <- RMSE(edx_test$rating, predicted_ratings)
```

Table 7: RMSE Values for 2.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834

5.3 Model 3.x: Study of models involving two effects in different forms

5.3.1 Model 3.1: Mean with movie and user effect taken independently

```r
# Combine Model 2.1 & Model 2.2
# Generate predictions on the edx test set
predicted_ratings <-  edx_test %>%
                      left_join(movie_effects, by='movieId') %>%
                      left_join(user_effects, by='userId') %>%
                      mutate(pred = mu + b_m + b_u) %>%
                      .$pred

# Find RMSE for this model
rmse_model_3_1 <- RMSE(edx_test$rating, predicted_ratings)
```

Table 8: RMSE Values for 3.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595

5.3.2 Model 3.2: Mean with movie and cumulative user effect

```r
# Add user effect to Model 2.1 (Movie effects model)
user_effects_comb <-    edx_train %>%
                        left_join(movie_effects, by='movieId') %>%
                        group_by(userId) %>%
                        summarize(b_u = mean(rating - mu - b_m))

# Generate predictions on the edx test set
predicted_ratings <-  edx_test %>%
                      left_join(movie_effects, by='movieId') %>%
                      left_join(user_effects_comb, by='userId') %>%
                      mutate(pred = mu + b_m + b_u) %>%
                      .$pred

# Find RMSE for this model
rmse_model_3_2 <- RMSE(edx_test$rating, predicted_ratings)
```

Table 9: RMSE Values for 3.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612

method	RMSE
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462

5.3.3 Model 3.3: Mean with user and cumulative movie effect

```
#Train on the edx training set to find movie effect including user effect
combined_user_movie_effects <-    edx_train %>%
                                  left_join(user_effects, by='userId') %>%
                                  group_by(movieId) %>%
                                  summarize(b_m = mean(rating - mu - b_u))

#Generate predictions on the edx test set
predicted_ratings <-  edx_test %>%
                      left_join(user_effects, by='userId') %>%
                      left_join(combined_user_movie_effects, by='movieId') %>%
                      mutate(pred = mu + b_m + b_u) %>%
                      .$pred

#Find RMSE for this model
rmse_model_3_3 <- RMSE(edx_test$rating, predicted_ratings)
```

Table 10: RMSE Values for 3.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789

5.3.4 Model 3.x Observations

It can be observed from the 3.x models that the order in which the features are chosen makes a difference. It is better to choose movie effects first and then the user effects in cumulative manner instead of training with them independently or choosing them in a different order.

Next, the effects would be chosen in increasing order of their RMSE values based on their independent effect models. Thus, the sequence would be movie, user, genre, release year & review year effects.

5.4 Model 4.x: Towards the model involving all the effects

5.4.1 Model 4.1: Model of movie, user, genre effects in respective order

```
# Add genre effect to Model 3.2 that already includes movie and user effects
genre_effects_comb <- edx_train %>%
                      left_join(movie_effects, by = "movieId") %>%
                      left_join(user_effects_comb, by = "userId") %>%
                      group_by(genres) %>%
                      summarise(b_g = mean(rating - mu - b_m - b_u))

# Predict ratings adjusting for movie, user and genre effects
predicted_ratings <-  edx_test %>%
                      left_join(movie_effects, by = "movieId") %>%
                      left_join(user_effects_comb, by = "userId") %>%
                      left_join(genre_effects_comb, by = "genres") %>%
                      mutate(pred = mu + b_m + b_u + b_g) %>%
                      pull(pred)

# Calculate RMSE based on genre effects model
rmse_model_4_1 <- RMSE(predicted_ratings, edx_test$rating)
```

Table 11: RMSE Values for 4.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792

5.4.2 Model 4.2: Model of movie, user, genre & release year effects in respective order

```
# Estimate release year effect (b_yr_release)
release_year_effects_comb <-  edx_train %>%
                              left_join(movie_effects, by = "movieId") %>%
                              left_join(user_effects_comb, by = "userId") %>%
                              left_join(genre_effects_comb, by = "genres") %>%
                              group_by(release_year) %>%
                              summarise(b_yr_release = mean(rating - mu - b_m - b_u - b_g))

# Predict ratings adjusting for movie, user, genre and year effects
predicted_ratings <-  edx_test %>%
                      left_join(movie_effects, by = "movieId") %>%
                      left_join(user_effects_comb, by = "userId") %>%
```

```
                       left_join(genre_effects_comb, by = "genres") %>%
                       left_join(release_year_effects_comb, by = "release_year") %>%
                       mutate(pred = mu + b_m + b_u + b_g + b_yr_release) %>%
                       pull(pred)

# Calculate RMSE based on year effects model
rmse_model_4_2 <- RMSE(predicted_ratings, edx_test$rating)
```

Table 12: RMSE Values for 4.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792
Model 4.2: Mean + Genre + Movie + User + Release Yr Effect	1.0507776

5.4.3 Model 4.3: Model of movie, user, genre effects, release year & review year effects in respective order

```
# Estimate review date effect (b_yr_review)
review_yr_effects_comb <- edx_train %>%
                           left_join(movie_effects, by = "movieId") %>%
                           left_join(user_effects_comb, by = "userId") %>%
                           left_join(genre_effects_comb, by = "genres") %>%
                           left_join(release_year_effects_comb, by = "release_year") %>%
                           group_by(review_year) %>%
                           summarise(b_yr_review = mean(rating - mu - b_m -
                                                            b_u - b_g - b_yr_release))
```

Table 13: RMSE Values for 4.x Model series

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792

method	RMSE
Model 4.2: Mean + Genre + Movie + User + Release Yr Effect	1.0507776
Model 4.3: Mean + Genre + Movie + User + Release Yr + Review Yr Effect	1.0506968

5.5 Model 5.0: Effects of Regularization on the model

Regularization is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. Thus, to avoid overfit we will apply regularization and study its influence on the RMSE.

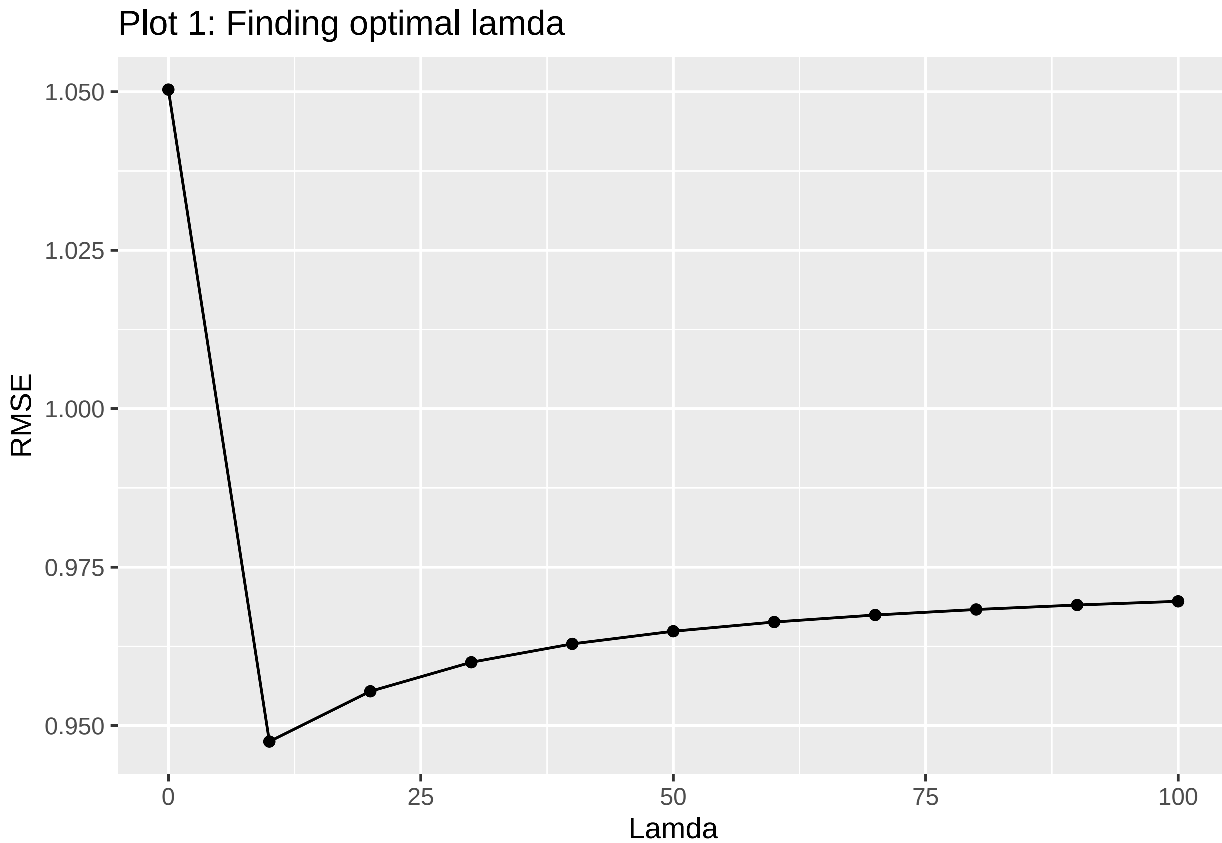
5.5.1 Define function to find RMSE for given lambda on a predefined model

Regularization process involves finding out an optimal lambda to tune the model. A common function is defined to find the RMSE for particular lambda on Model 3.2.

```
find_rmse_for_lambda <- function(l) {
  #Train on the edx training set to find user effect including movie effect
  combined_movie_user_effects <-  edx_train %>%
                                  left_join(movie_effects, by='movieId') %>%
                                  group_by(userId) %>%
                                  summarize(b_u = sum(rating - mu - b_m)/(n()+l))
  #Generate predictions on the edx test set
  predicted_ratings <-  edx_test %>%
                        left_join(movie_effects, by='movieId') %>%
                        left_join(combined_movie_user_effects, by='userId') %>%
                        mutate(pred = mu + b_m + b_u) %>%
                        .$pred
  #Find RMSE for this model
  RMSE(edx_test$rating, predicted_ratings)
}
```

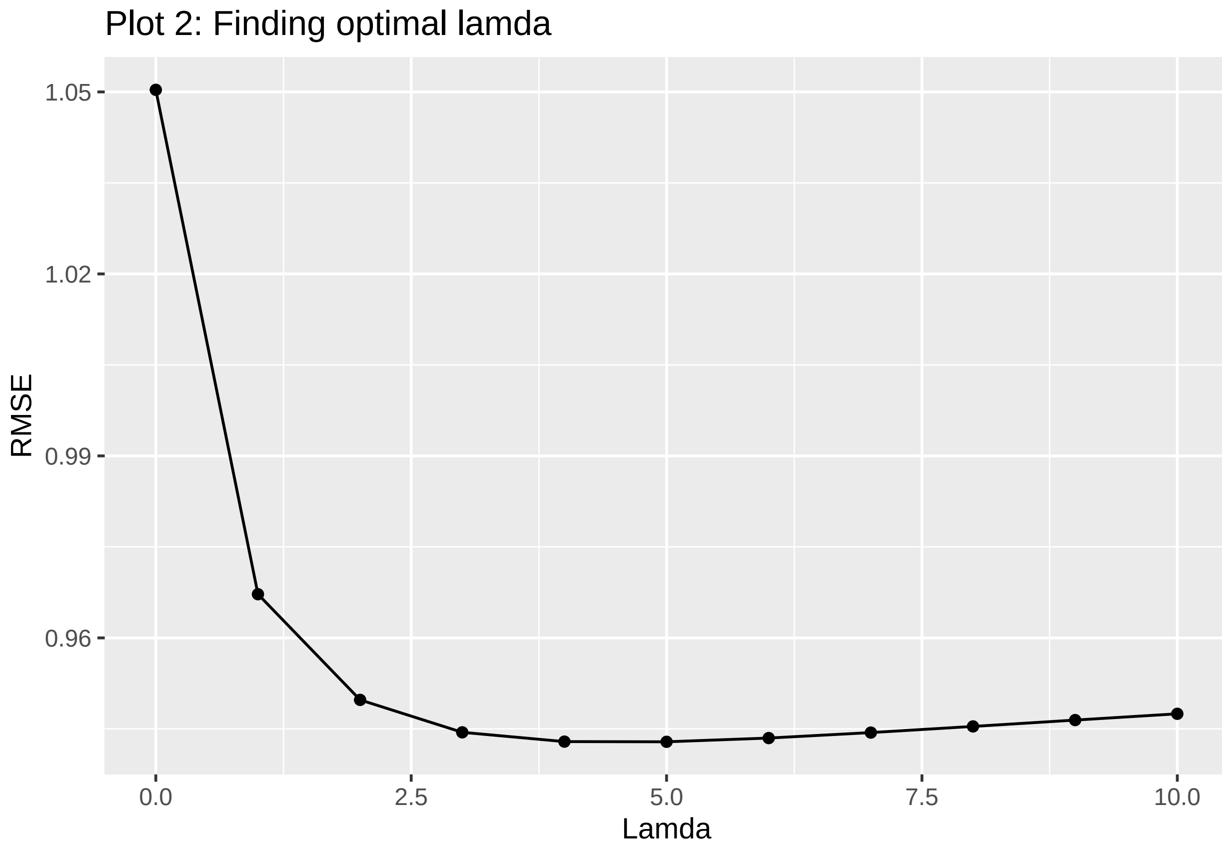
5.5.2 Finding the optimal value of lambda

```
# Towards optimal lambda - Step 1
lambdas <- seq(0, 100, 10)
rmses <- sapply(lambdas, function(l) {  find_rmse_for_lambda(l) })
qplot(x = lambdas, y = rmses, main = "Plot 1: Finding optimal lamda",
      xlab = "Lamda", ylab = "RMSE", geom = c("point", "line"))
```



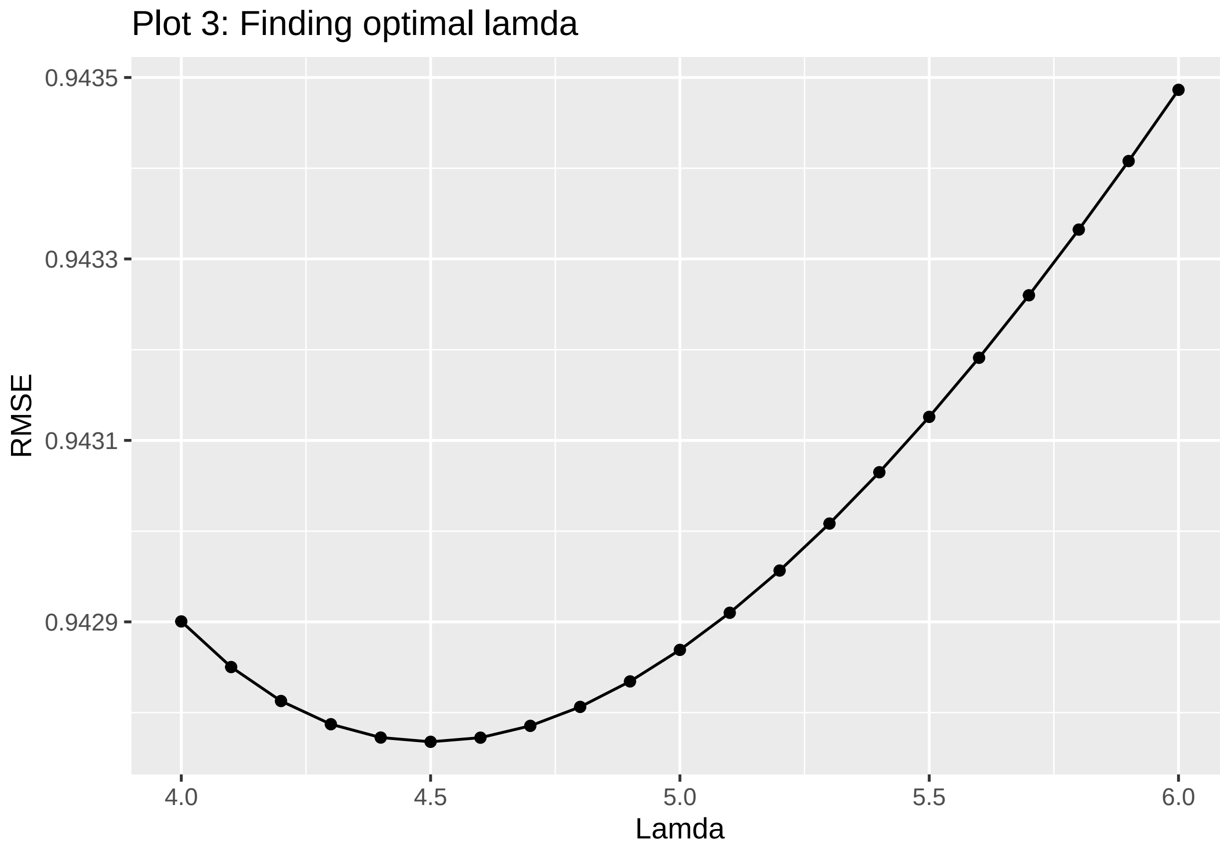
Observation: The optimal lambda seems to lie in the range of (0,10).

```
# Towards optimal lambda - Step 2
lambdas <- seq(0, 10, 1)
rmses <- sapply(lambdas, function(l) {  find_rmse_for_lambda(l) })
qplot(x = lambdas, y = rmses, main = "Plot 2: Finding optimal lamda",
      xlab = "Lamda", ylab = "RMSE", geom = c("point", "line"))
```



Observation: The optimal lambda lies between 4 and 6. Next, we'll find RMSE for models with lambda between 4 & 6 in steps of 0.1. The lambda with minimum RMSE is the optimal lambda for the above model.

```
# Towards optimal lambda - Step 3
lambdas <- seq(4, 6, 0.1)
rmses <- sapply(lambdas, function(l) {  find_rmse_for_lambda(l) })
qplot(x = lambdas, y = rmses, main = "Plot 3: Finding optimal lamda",
      xlab = "Lamda", ylab = "RMSE", geom = c("point", "line"))
```




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

Observation: The optimal lambda is 4.5 that gives an RMSE of `r, min_rmse`.

Table 14: RMSE Values for All Models

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792
Model 4.2: Mean + Genre + Movie + User + Release Yr Effect	1.0507776
Model 4.3: Mean + Genre + Movie + User + Release Yr + Review Yr Effect	1.0506968
Model 5.0: Mean + Regularized Movie + Cum. User Effect	0.9427678

Observation: As observed in the regularized model, the RMSE value is 0.9427678 against a non-regularized model that had an RMSE of 1.0503462. Therefore, we will use regularization along with all the features to train our final model.

5.6 Model 6.0: Model with all features and regularization

```
# Regularise model, predict ratings and calculate RMSE for passed value of lambda
train_predict_get_rmse <- function(l, trainSet, testSet)
{
  b_m   <-        trainSet %>%
                  group_by(movieId) %>%
                  summarise(b_m = sum(rating - mu)/(n()+l))
  b_u   <-        trainSet %>%
                  left_join(b_m, by="movieId") %>%
                  group_by(userId) %>%
                  summarise(b_u = sum(rating - b_m - mu)/(n()+l))
  b_g   <-        trainSet %>%
                  left_join(b_m, by="movieId") %>%
                  left_join(b_u, by="userId") %>%
                  group_by(genres) %>%
                  summarise(b_g = sum(rating - b_m - b_u - mu)/(n()+l))
  b_yr_release <- trainSet %>%
                  left_join(b_m, by="movieId") %>%
                  left_join(b_u, by="userId") %>%
                  left_join(b_g, by="genres") %>%
                  group_by(release_year) %>%
                  summarise(b_yr_release = sum(rating - b_m - b_u - b_g - mu)/(n()+l))
  b_yr_review <-  trainSet %>%
                  left_join(b_m, by="movieId") %>%
                  left_join(b_u, by="userId") %>%
                  left_join(b_g, by="genres") %>%
                  left_join(b_yr_release, by="release_year") %>%
                  group_by(review_year) %>%
                  summarise(b_yr_review = sum(rating - b_m - b_u - b_g - mu)/(n()+l))

  predicted_ratings <-  testSet %>%
                        left_join(b_m, by="movieId") %>%
                        left_join(b_u, by="userId") %>%
                        left_join(b_g, by="genres") %>%
                        left_join(b_yr_release, by="release_year") %>%
                        left_join(b_yr_review, by="review_year") %>%
                        mutate(pred = mu + b_m + b_u + b_g + b_yr_release + b_yr_review) %>%
                        pull(pred)

  return (RMSE(predicted_ratings, testSet$rating))
}
# Generate a sequence of values for lambda ranging from 4 to 6 with 0.1 increments
inc <- 0.1
lambdas <- seq(4, 6, inc)

# Get RMSE values for all the lambdas
rmses <- sapply(lambdas, function(l) {  train_predict_get_rmse(l, edx_train, edx_test) })

# Assign optimal tuning parameter (lambda)
optimal_lambda <- lambdas[which.min(rmses)]

# Minimum RMSE achieved
```

```r
regularised_rmse <- min(rmses)

#Add the new RMSE to the RMSE Table
rmse_table <- bind_rows(rmse_table,
                        data_frame(method="Model 6: Regularized Model of Movie + User + Genre + Release
                                   RMSE = regularised_rmse ))
rmse_table %>% knitr::kable(caption = "RMSE Values for All Models")
```

Table 15: RMSE Values for All Models

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792
Model 4.2: Mean + Genre + Movie + User + Release Yr Effect	1.0507776
Model 4.3: Mean + Genre + Movie + User + Release Yr + Review Yr Effect	1.0506968
Model 5.0: Mean + Regularized Movie + Cum. User Effect	0.9427678
Model 6: Regularized Model of Movie + User + Genre + Release Yr + Review Yr Effect	0.9214858

6 RMSE on validation set with final chosen model

```r
# RMSE Objectives:
# ================================================================================
# 0 points: No RMSE
# 5 points: RMSE >= 0.90000 AND/OR the reported RMSE is the result of overtraining
#           (validation set - the final hold-out test set - ratings used
#           for anything except reporting the final RMSE value) AND/OR the reported RMSE
#           is the result of simply copying and running code provided in previous courses
#           in the series.
# 10 points: 0.86550 <= RMSE <= 0.89999
# 15 points: 0.86500 <= RMSE <= 0.86549
# 20 points: 0.86490 <= RMSE <= 0.86499
# 25 points: RMSE < 0.86490
# --------------------------------------------------------------------------------

# Keep only those rows in validation set which have
# movieId and userId existing in edx_train dataset
```

```r
MovieRecommenderModel <- function(predictInputSet)
{
      # Add release and review years that are used in our model
      predictInputSet <- add_release_and_review_years(predictInputSet)
```

```
      # Predict using the values of the various feature effects found
      predicted_ratings <-  predictInputSet %>%
                        left_join(b_m, by="movieId") %>%
                        left_join(b_u, by="userId") %>%
                        left_join(b_g, by="genres") %>%
                        left_join(b_yr_release, by="release_year") %>%
                        left_join(b_yr_review, by="review_year") %>%
                        mutate(pred = mu + b_m + b_u + b_g + b_yr_release + b_yr_review) %>%
                        pull(pred)

      #Return the predictions
      predicted_ratings
}

predicted_ratings <- MovieRecommenderModel(validation)
final_rmse_validation_set <- RMSE(validation$rating, predicted_ratings)

# Add the new RMSE to the RMSE Table
rmse_table <- bind_rows(rmse_table,
                        data_frame(method="RMSE on validation set using Model 5",
                                   RMSE = final_rmse_validation_set))
rmse_table %>% knitr::kable(caption = "RMSE Values for All Models")
```

Table 16: RMSE Values for All Models

method	RMSE
Model 1.0: Simple mean of training set	1.0548776
Model 2.1: Mean with movie effect	0.9756612
Model 2.2: Mean with user effect	1.1540005
Model 2.3: Mean with genre effect	1.0176850
Model 2.4: Mean with release year effect	1.0430696
Model 2.5: Mean with review year effect	1.0533834
Model 3.1: Mean + Movie + User Effect	1.1125595
Model 3.2: Mean + Movie + Cum. User Effect	1.0503462
Model 3.3: Mean + User + Cum. Movie Effect	1.1063789
Model 4.1: Mean + Movie + User + Genre Effect	1.0508792
Model 4.2: Mean + Genre + Movie + User + Release Yr Effect	1.0507776
Model 4.3: Mean + Genre + Movie + User + Release Yr + Review Yr Effect	1.0506968
Model 5.0: Mean + Regularized Movie + Cum. User Effect	0.9427678
Model 6: Regularized Model of Movie + User + Genre + Release Yr + Review Yr Effect	0.9214858
RMSE on validation set using Model 5	0.9190067

7 Conclusions

The MovieRecommenderModel developed predicts values on the validation set with an RMSE of 0.9190067. The model includes the movie, user, genre, release year and review year effects along with regularization.

8 References

[1] Introduction to Data Science, Rafael A. Irizarry
[2] http://www.sthda.com/english/wiki/qplot-quick-plot-with-ggplot2-r-software-and-data-visualization
[3] https://bookdown.org/yihui/rmarkdown-cookbook/kable.html
[4] https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet