HarvardX: PH125.9x: Data Science: Capstone

A Capstone Project for the `Professional Certificate in Data Science` offered by Harvard University (HarvardX) via EdX

John King'athia Karuitha

30/11/2020

Contents

Abstract

In this exercise, I build a movie recommendation system using the movielens dataset as part of the professional certificate in data science certification offered by HarvardX - a Harvard University online program offered through EdX. The data consists of a training set (labelled edx) and a testing set (validation). The training set has about 10 million observations of 6 variables, while the testing set has about 1 million observations. By using regularization, I was able to lower the root mean squared error to 0.8649857.

Background

In this exercise, I use a subset of the `movielens` data set to build a movie recommendation system. A recommendation system suggests a good or a service to a customer based on their previous revealed preferences. Recommendation systems come in three versions;

```
## detach any loaded packages
if(!require(pacman)) install.packages("pacman", repos = "http://cran.us.r-project.org")
pacman::p_unload(pacman::p_loaded(), character.only = TRUE)

##################
## install 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(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(inspectdf)) install.packages("inspectdf", repos = "http://cran.us.r-project.org")
if(!require(anytime)) install.packages("anytime", repos = "http://cran.us.r-project.org")
if(!require(tibble)) install.packages("tibble", repos = "http://cran.us.r-project.org")
if(!require(skimr)) install.packages("skimr", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")

## Load required libraries
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(inspectdf)
library(anytime)
library(tibble)
library(skimr)
library(scales)
library(ggthemes)
```

- `Collaborative filtering recomendation systems` that attempt that predict and make recommendations to users based on the activities of other similar users. For instance, suppose two users A and B have a similar ratings pattern. User B rates a given good highly while user A has not rated it. Based on their past common tastes, the recommendation system could suggest the given good to user A.
- `Content filtering recomendation Systems` that makes suggestions based on the contents / characteristics of previous goods or services purchased. For instance, if user A has previously rated a movie by Mel Gibson highly, the recommendation system could recommend new movies by Mel Gibson, or suggest movies of a similar genre, or made by the same production company as the previously highly rated movies.
- `Hybrid Systems` that use both `collaborative filtering` and `content filtering` to make suggestions.

I start by splitting the data into a training set and test set as per the instructions. The task then is to use the training dataset to train a machine learning algorithm. After building the algorithm, I will then test it using the separate test dataset. The task proceeds as follows, in the next section (The Data) I load the dataset, split it into the training and test sets and describe the data. Next, I build the machine learning algorithm. I then test the algorithm on the testing data set. I conclude by evaluating the performance of the model.

The Data

I use a subset of the `movielens` dataset with 10 million observations (ratings). The original dataset is much larger, having 27 million ratings and is available here. First, I generate the dataset and then split the data into a training set and a testing set. I do not include in the main text the code for generating the dataset as it is available in the course website. The code for generating the data is provided by the course staff at HarvardX.

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

# Note: this process could take a couple of minutes

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")

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")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           #title = as.character(title),
                                           #genres = as.character(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 <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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 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, movielens, removed)
```

Generating the Data, Training Set and Testing Set

I do not show this step as it is taken straight from the course text (***see code above***). However, the `training` set `edx` consists of 9,000,055 observations of six variables. The `testing` or `validation` set consists of 999,999 observations for the same set of variables. Each row represents one customer who has rated one or more movies. The columns represents variables which are `userId`, `movieId`, `rating`, `timestamp`, `title`, and `genres` respectively.

Method/ Analysis

In this section, I wrangle and visualize the data and then run the regression models.

Wrangling the Training Set Data and Summary Statistics

The summary statistics for the training set are as follows. There are 69878 unique users in the dataset who have rated 10677 movies. Further, there are 797 genres or combinations of genres for each movie given that a movie could fall into more than one genre. The structure of the data is presented below.

```
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>
```

Next, I convert the time stamp variable into a sensible date and time.

```
library(anytime)
edx$timestamp <- anytime(edx$timestamp)
```

The summary statistics for the movie ratings are in the table below.

```
library(skimr)

# Skim the data for summary stats and make a tibble
skim(edx$rating) %>% tibble() %>%

  # Format table by removing unwanted columns
  select(-skim_type, -numeric.hist, - complete_rate) %>%

  ## Rename remaining columns
  rename(Variable = skim_variable, Missing = n_missing, Mean = numeric.mean,
```

```
        SD = numeric.sd, min = numeric.p0, Q1 = numeric.p25, Median = numeric.p50,

        Q3 = numeric.p75, Max = numeric.p100) %>%

  knitr::kable(caption = "Summary of the Movie Ratings")
```

Table 1: Summary of the Movie Ratings

Variable	Missing	Mean	SD	min	Q1	Median	Q3	Max
data	0	3.512465	1.060331	0.5	3	4	4	5

The distribution of categorical variables is shown in the table below, with drama the most common genre and pulp fiction the most recurring movie, meaning that it was the movie that was rated the most.

```
library(inspectdf)
inspectdf::inspect_cat(edx)
```

```
## # A tibble: 2 x 5
##   col_name   cnt common              common_pcnt levels
##   <chr>    <int> <chr>                     <dbl> <named list>
## 1 genres     797 Drama                     8.15  <tibble [797 x 3]>
## 2 title    10676 Pulp Fiction (1994)       0.348 <tibble [10,676 x 3]>
```

Further, the distribution of ratings by movies varies widely as the analysis of the first six movies shows.

```
## Distribution of ratings
edx %>%

  ## Group the movies by movieId
  group_by(movieId) %>%

  ## Get summary statistics
  skimr::skim(rating) %>%

  ## Get the first six movies
  head() %>%

  ## make a tibble
  tibble() %>%

  ## Clean the tibble
  select(-skim_type, -numeric.hist, -n_missing, -complete_rate) %>%

  ## Rename tibble columns
  rename(Variable = skim_variable, Mean = numeric.mean,
         SD = numeric.sd, Min = numeric.p0, Q1 = numeric.p25,
         Median = numeric.p50, Q3 = numeric.p75, Max = numeric.p100) %>%

  ## make a nice table
  knitr::kable(caption = "Summary statistics for the ratings variable on training set")
```

Table 2: Summary statistics for the ratings variable on training set

Variable	movieId	Mean	SD	Min	Q1	Median	Q3	Max
rating	1	3.927638	0.8971016	0.5	3	4	5.0	5
rating	2	3.205399	0.9507859	0.5	3	3	4.0	5
rating	3	3.146984	0.9993356	0.5	3	3	4.0	5
rating	4	2.864299	1.0924754	0.5	2	3	4.0	5
rating	5	3.068672	0.9637571	0.5	3	3	4.0	5
rating	6	3.815284	0.8853703	0.5	3	4	4.5	5

In addition, some movies have ratings from very few users which make the ratings of such movies unreliable as they draw from a small sample.

```
## Movies with very few ratings
edx %>%

  ## Count the movies
  count(movieId, title) %>%

  ## Arrange in ascending order
  arrange(n) %>%

  ## preview the first ten movies with few ratings
  head(10) %>%

  ## make a nice table
  knitr::kable(caption = "Sample of movies that have few ratings")
```

Table 3: Sample of movies that have few ratings

movieId	title	n
3191	Quarry, The (1998)	1
3226	Hellhounds on My Trail (1999)	1
3234	Train Ride to Hollywood (1978)	1
3356	Condo Painting (2000)	1
3383	Big Fella (1937)	1
3561	Stacy's Knights (1982)	1
3583	Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	1
4071	Dog Run (1996)	1
4075	Monkey's Tale, A (Les ChÃ¢teau des singes) (1999)	1
4820	Won't Anybody Listen? (2000)	1

Likewise, some users have made very few ratings.

```
## Users that have made very few ratings
edx %>%

  ## Count the users
  count(userId) %>%

  ## Arrange the users in ascending order of counts
  arrange(n) %>%
```

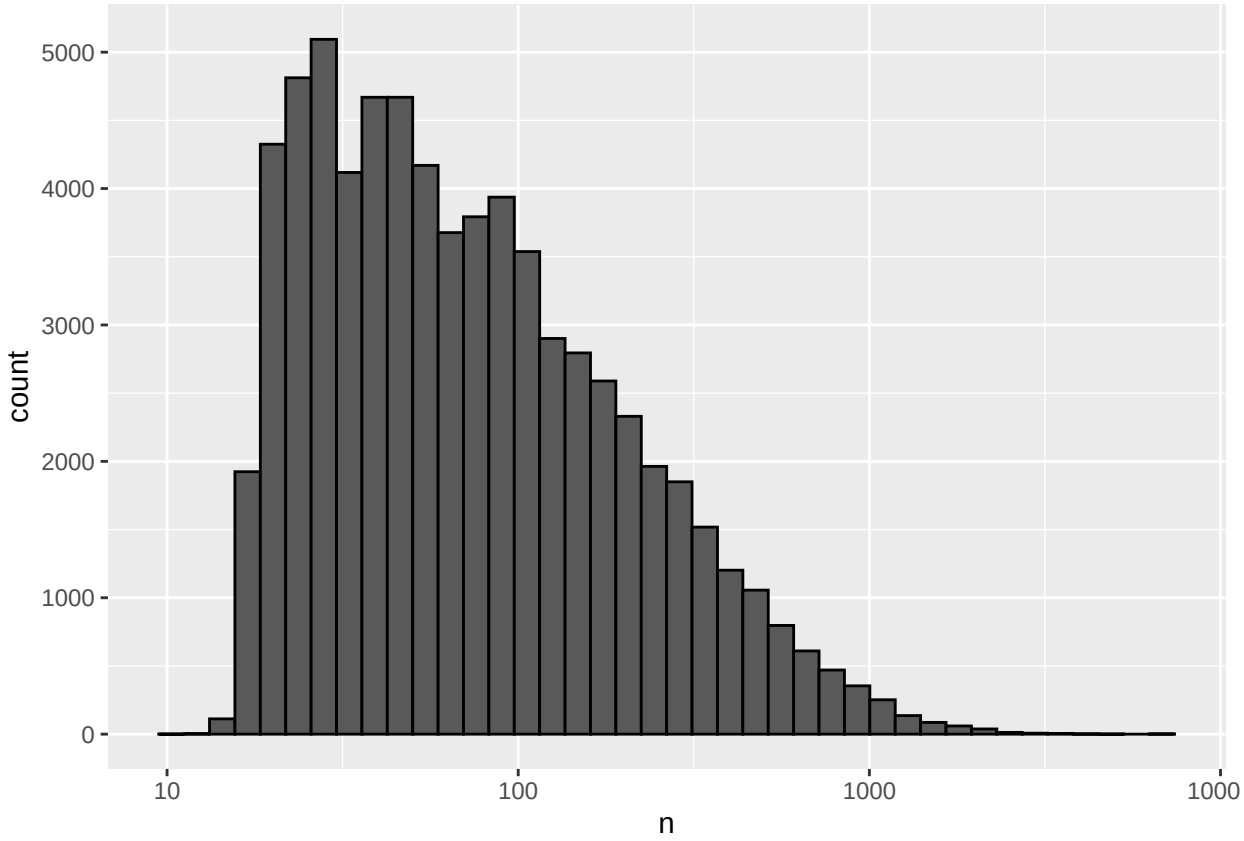
```
## Preview the first 10 users
head(10) %>%

## Make a neat table
knitr::kable(caption = "Sample of users that have made the fewest ratings")
```

Table 4: Sample of users that have made the fewest ratings

userId	n
62516	10
22170	12
15719	13
50608	13
901	14
1833	14
2476	14
5214	14
9689	14
10364	14

The bias in ratings is better seen in the graph below

```
## Visualization of users by count
edx %>%

  ## Count the number of users
  count(userId) %>%

  ## Plot the users by ratings
  ggplot(aes(x = n)) + geom_histogram(col = "black", bins = 40) +

  ## Log the scale of the users
  scale_x_log10()
```



The table below shows the distributions of ratings among the first six users.

```
## How the first six users have rated the movies
table(edx$userId, edx$rating) %>%

  ## Preview the first six users
  head() %>%

  ## make a neat table
  knitr::kable(caption = "Distribution of ratings among the first six users")
```

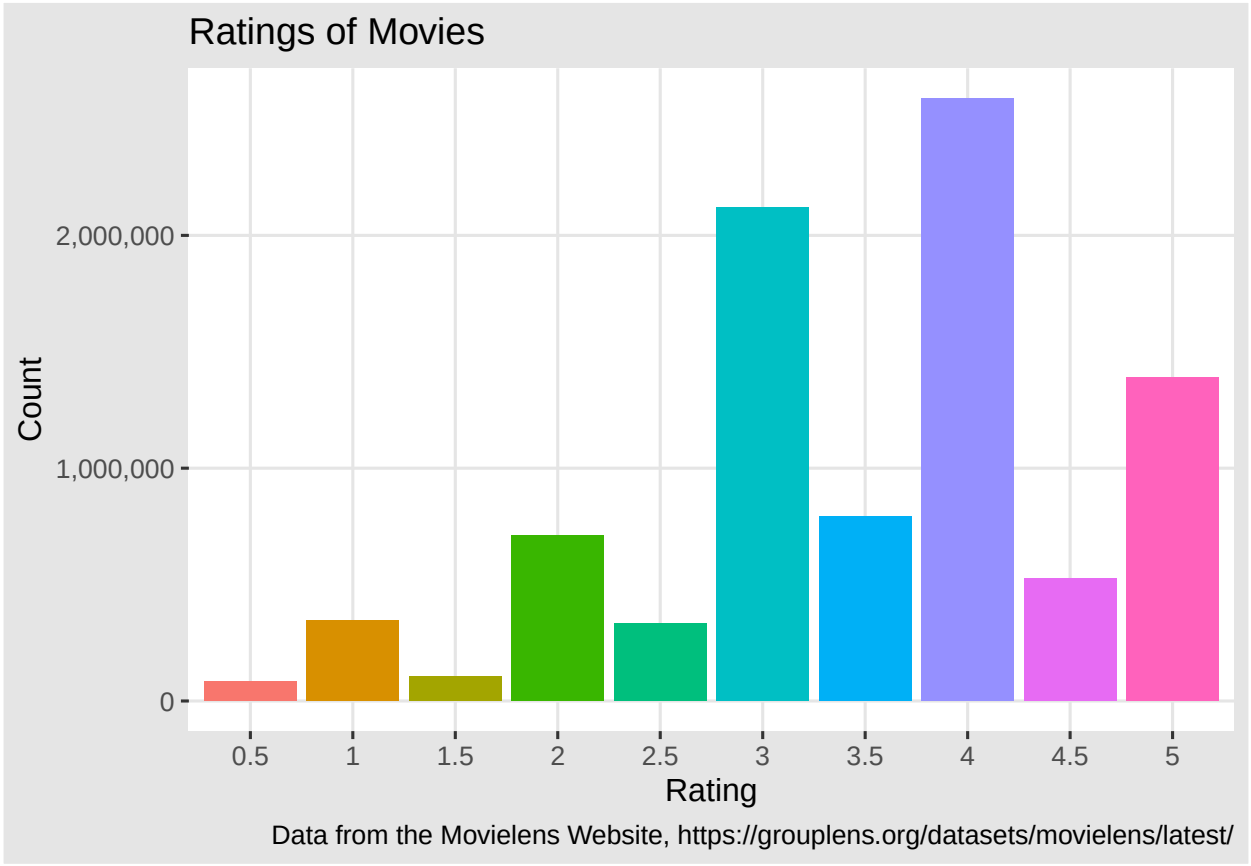
Table 5: Distribution of ratings among the first six users

0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
0	0	0	0	0	0	0	0	0	19
0	0	0	2	0	11	0	1	0	3
0	0	0	2	0	3	5	9	9	3
0	1	0	1	0	13	0	0	0	20
0	5	0	0	0	18	0	24	0	27
0	2	0	1	0	9	0	12	0	15

Data Visualization of the Training Dataset

I do a bar graph of the movie ratings below. Note that the ratings containing fractions are very few, consistent with `discretization` where people tend to round up or down to the nearest whole number.

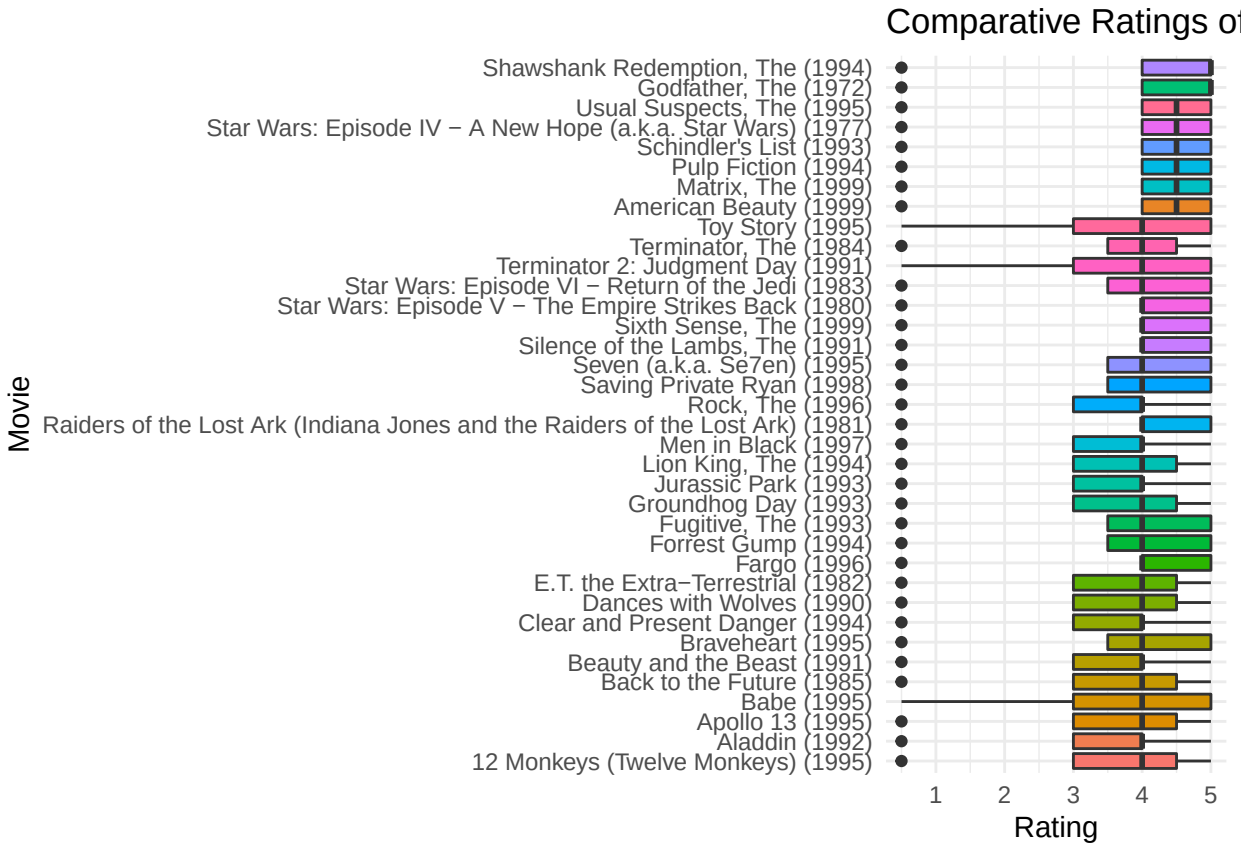
```
library(scales)
library(ggthemes)

# Plot bar graph of the distribution of ratings
edx %>% ggplot(aes(x = as.factor(rating), fill = as.factor(rating))) + geom_bar() +

  # Add a pleasant theme
  ggthemes::theme_igray() +

  # Remove legends
  theme(legend.position = "none") +

  # Convert the y-axis into numbers from scientific notation using the scales package
  scale_y_continuous(labels = scales::comma_format()) +

  # Label the X and Y-axis and add a caption
  labs(x = "Rating", y = "Count", title = "Ratings of Movies",

  caption = "Data from the Movielens Website, https://grouplens.org/datasets/movielens/latest/")
```



Next, I visualize the most popular movies and genres. I concentrate on the recent movies that have at least a median rating of 4 stars and have received at least 15,000 ratings.

```
# popular movies
edx %>%

  ## Group by movie and title.
```

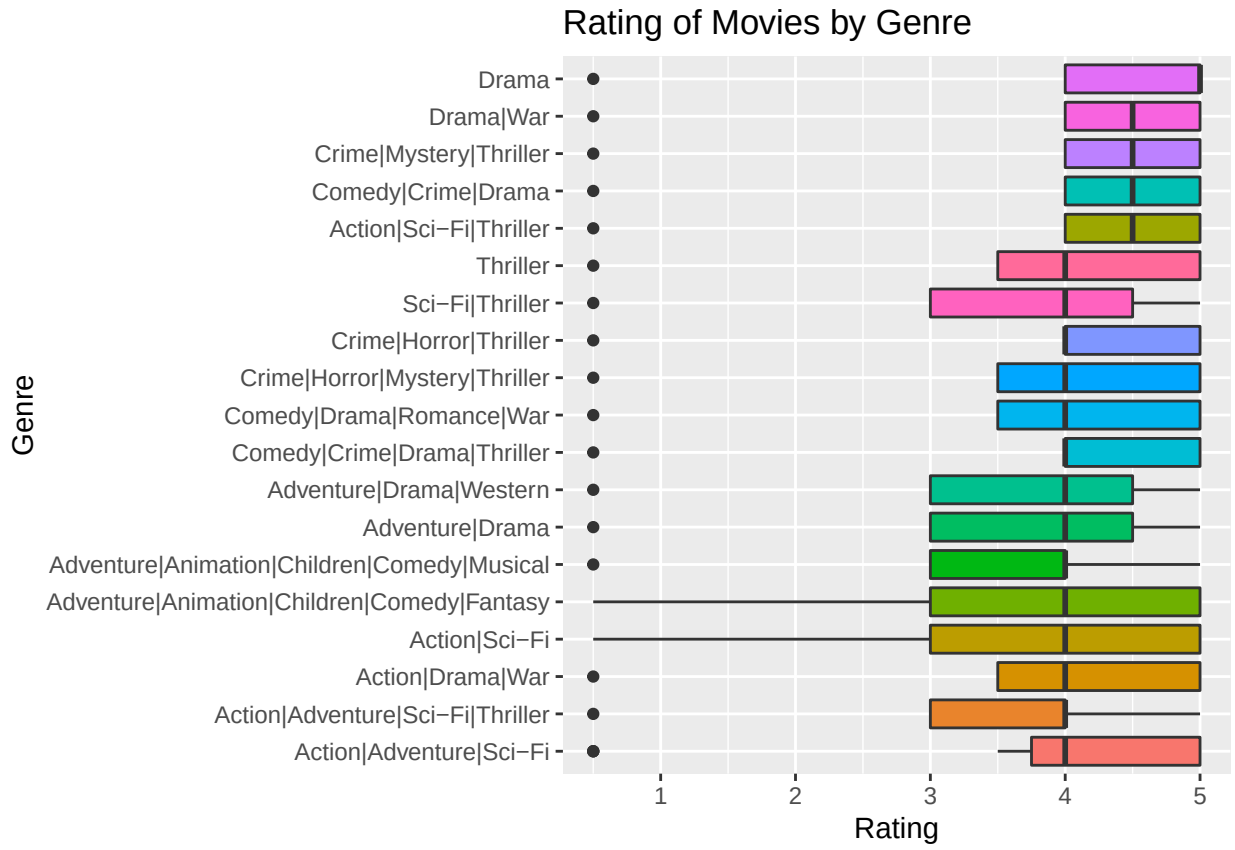
```
group_by(movieId, title) %>%

## Mutate timestamp to extract year.
mutate(movie_year = lubridate::year(timestamp)) %>%

## Filter the movies that have at least 15,000 ratings in 1993 or later.
filter(n() >= 15000 & movie_year >= 1993) %>%

## Do summary statistics for the movies.
summarise(min_rating = min(rating),
          q1_rating = quantile(rating, 0.25),
          median_rating = median(rating),
          q3_rating = quantile(rating, 0.75),
          max_rating = max(rating)) %>%

## Filter only those movies with a median rating of 4 and above.
filter(median_rating >= 4) %>%

group_by(movieId, title) %>%

## Convert the data to long format. These functions were previously done using gather.
pivot_longer(-c(1,2), names_to = "level", values_to = "rating") %>%

group_by(title, rating) %>%

## Plot ratings versus movie title.
ggplot(aes(x = reorder(title, rating, median), y = rating, fill = title)) +

## Add geom
geom_boxplot(show.legend = FALSE) +

## Flip the coordinate and add theme for pleasant theme.
coord_flip() + theme_minimal() +

## Add axis labels and title.
labs(x = "Movie", y = "Rating", title = "Comparative Ratings of Movies")
```



We now turn to the most popular genres. Here, I focus on genres that have at least 20,000 ratings and that have received a median rating of 4 and above.

```
## Popular genres
edx %>%

  ## Group by genres and title.
  group_by(genres, title) %>%

  ## Filter genres with at least 20,000 ratings.
  filter(n() >= 20000) %>%

  ## Summarise the ratings by genre data.
  summarise(min_rating = min(rating),
            q1_rating = quantile(rating, 0.25),
            median_rating = median(rating),
            q3_rating = quantile(rating, 0.75),
            max_rating = max(rating)) %>%

  ## Filter genres with no genres listed.
  filter(genres != "no genres listed", genres != "(no genres listed)") %>%

  ## Convert the data to long format as we did using gather.
  pivot_longer(-c("genres", "title"), names_to = "level", values_to = "value") %>%

  ## Group by genre
  group_by(genres) %>%
```

```
## Filter genres with a median rating of at least 4.
filter(median(value) >= 4) %>%

## Plot the data
ggplot(mapping = aes(x = reorder(genres, value, median), y = value, fill = genres)) +

## Add the geom and flip coordinates.
geom_boxplot(show.legend = FALSE) + coord_flip() +

## Add axis labels and title.
labs(x = "Genre", y = "Rating", title = "Rating of Movies by Genre")
```

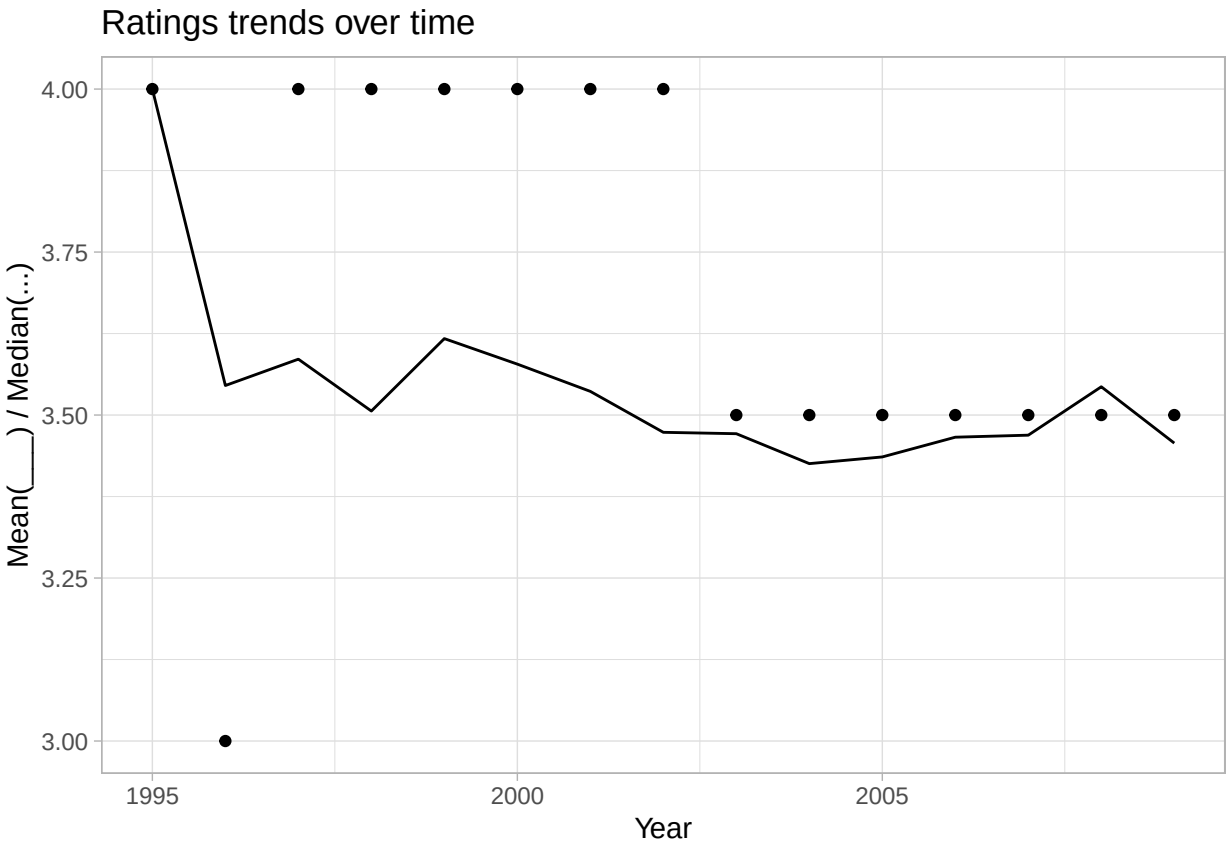


It appears that movies that cut across genres tend to get more ratings and also get generally higher median ratings than those that fall into one genre category.

Lastly, I examine the trends in ratings over time.

```
## Trends in movie ratings over time
edx %>%

  ## Create a year variable from the timestamp
  mutate(year = year(timestamp)) %>%

  ## Group by year
  group_by(year) %>%

  ## Summarize the mean and median rating over time
```

```
summarise(median = median(rating), mean = mean(rating)) %>%

## Plot the data
ggplot(mapping = aes(x = year, y = median)) +

## Add point geometry for median
geom_point() +

## Add line geometry for mean
geom_line(mapping = aes(x = year, y = mean)) +

## Add labels and titles
labs(x = "Year", y = "Mean(___) / Median(...)", title = "Ratings trends over time") +

## Add a pleasant theme
theme_light()
```



There appears to be downward decline in ratings averages over time which later stabilises.

Recomendation System

In this section, I run regression models- starting with the basic regression models and then adding penalty terms for movies that have very few ratings.

Baseline model

I start by constructing the evaluation function of the root mean squared error (RMSE). Then, I define a baseline model which is the average rating of movies. The baseline model has a root mean squared error of 1.061202 on the validation / test set.

```
## RMSE function
rmse <- function(actual, predicted){
  sqrt(mean((actual - predicted) ^ 2))
}

## Prediction using the average only ----
## Here I define a baseline for assessing model accuracy with a RMSE of 1.06 on the training set
rmse(edx$rating, mean(edx$rating))
```

```
## [1] 1.060331
```

```
## Here I define a baseline for assessing model accuracy with a RMSE of 1.06 on the test set
rmse(validation$rating, mean(validation$rating))
```

```
## [1] 1.061202
```

Next, I incorporate the user effects, build the model on the training set and use the validation set to evaluate it. The RMSE is 0.9439087 on the validation set.

```
## Incorporate user effects ----
## Defining the mean of ratings
mu <- mean(edx$rating)

## Computing the user specific coefficient bi.
bi <- edx %>% group_by(movieId) %>%

  summarise(bi = mean(rating - mu)) %>%

  ungroup()

## Combining the user effects with the validation set for  evaluation
user_effect <- validation %>%

  left_join(bi, by = "movieId")

## Evaluation on the test set
rmse(validation$rating, user_effect$bi + mu)
```

```
## [1] 0.9439087
```

Next, I incorporate the movie effects. The RMSE on the test set in this case improves to 0.8653488.

```
## Incorporate movie effects ----
## Combining the training set with the computed user effects bi and computing
## The movie effects bm.
bm <- edx %>%

  left_join(bi, by = "movieId") %>%

  group_by(userId) %>%

  summarise(bm = mean(rating - mu - bi, na.rm = TRUE)) %>%
```

```
  ungroup()

## Combining the computed movie effect with the validation set for evaluation
user_movie_effect <- user_effect %>%

  left_join(bm, by = "userId")

## The evaluation
rmse(validation$rating, mu + user_movie_effect$bi + user_movie_effect$bm)
```

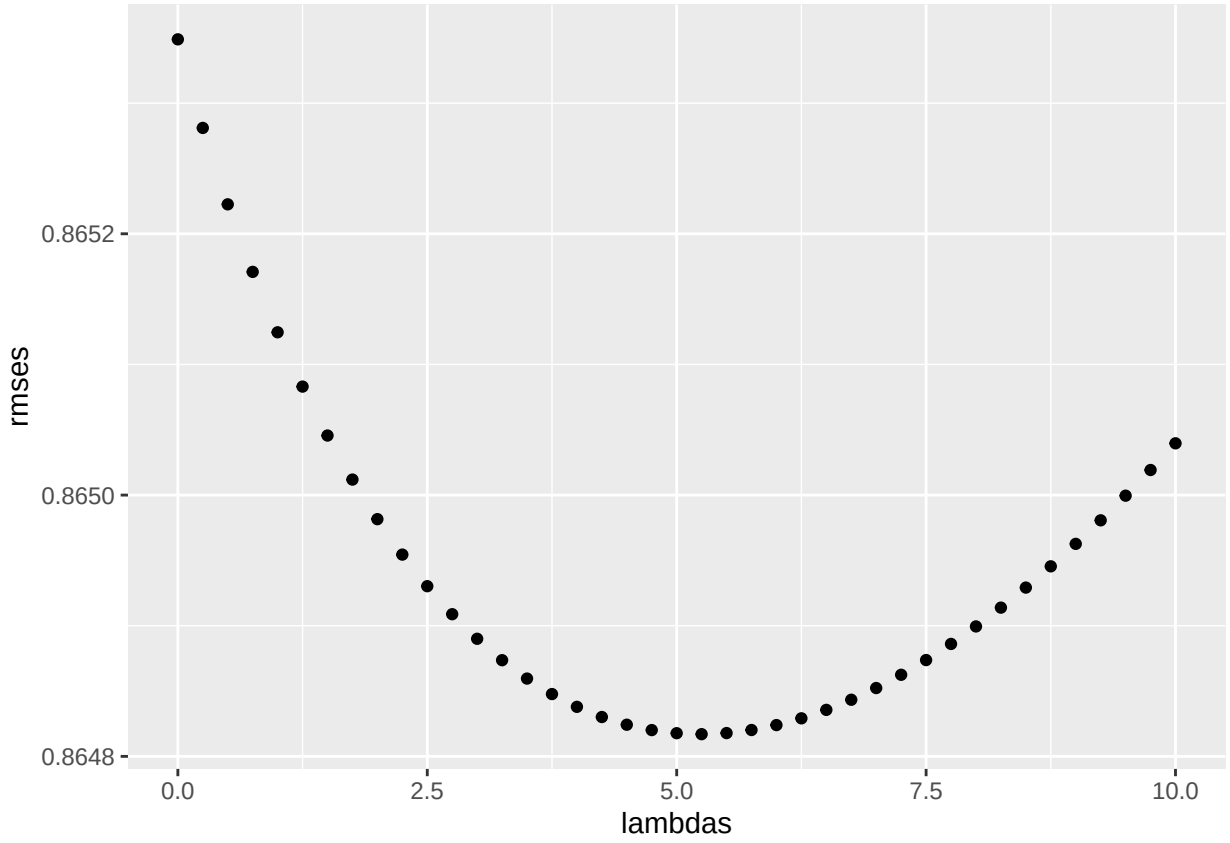
```
## [1] 0.8653488
```

Regularization

Regularized Movie and user effects

I have noted that some movies are rated by very few users making such estimates unstable. Likewise, some users rate very few movies. In this section, I deal with the movies that are subject to these anomalies by implementing regularization to penalize extreme ratings that come from a small sample sizes.

```
# lambda is a tuning parameter
# Use cross-validation to choose it.
lambdas <- seq(0, 10, 0.25)
# For each lambda,find b_i & b_u, followed by rating prediction & testing
# note:the below code could take some time
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

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

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

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

  return(RMSE(validation$rating,predicted_ratings))
})
# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, rmses)
```




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

```
## [1] 5.25
```

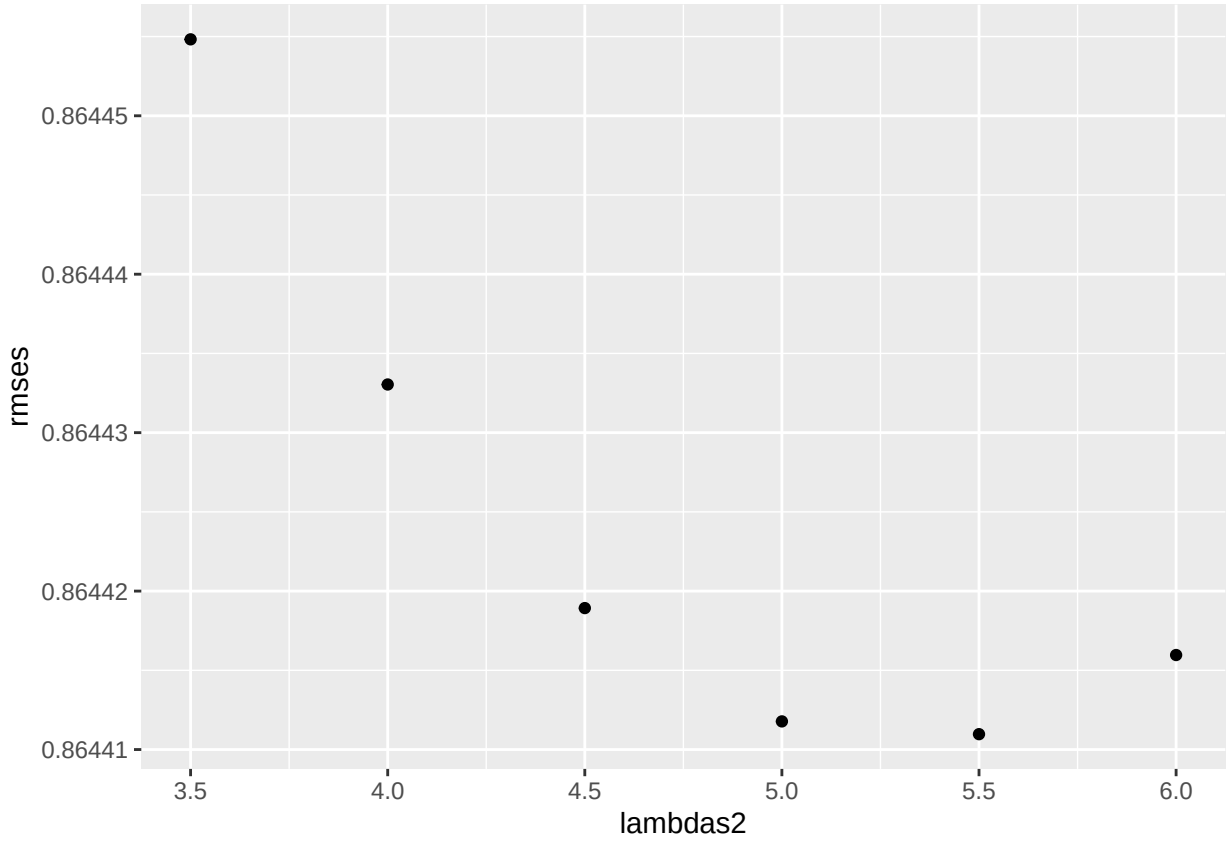
```
min(rmses)
```

```
## [1] 0.864817
```

Regularised movie, user, year, and genre effects

```
# b_y and b_g are the year and genre effects. I try a series of lambdas.
lambdas2 <- seq(3.5, 6, 0.5)
# Setting up the function
rmses <- sapply(lambdas2, function(l){

  ## Library for converting time columns
  library(anytime)

  ## The baseline prediction
  mu <- mean(edx$rating)

  ## Regularized movie effect
  b_i <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+l))
```

```
  ## Regularized user effect
  b_u <- edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu)/(n()+l))

  ## Regularized user and movie effects
  b_y <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(year = lubridate::year(timestamp)) %>%
    group_by(year) %>%
    summarise(b_y = sum(rating - mu - b_i - b_u)/(n()+l), n_y = n())

  ## Regularized movie, user and year effects
  b_g <- edx %>%
    mutate(year = lubridate::year(timestamp)) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+l), n_g = n())

  ## Regularized movie, user, year, and genre effects.
  predicted_ratings <- validation %>%
    mutate(year = lubridate::year(anytime(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred

  ## Computing the RMSEs.
  return(rmse(validation$rating, predicted_ratings))
})
# Compute new predictions using the optimal lambda
# Test and save results
qplot(lambdas2, rmses)
```



The minimum RMSE is achieved at a lambda of 5.5 giving the following RMSE of 0.864411.

```
min(rmses)
```

```
## [1] 0.864411
```

See next page.

Results

The table below shows the results of the regression models. Clearly, regularized regressions do better than regular regressions. Overall, all the models do better than the baseline of guessing the average by a substantial margin. The major limitation is computing power where the models do take time to run, and in some cases the computer RAM is not adequate.

```
library(tibble)

tribble(~ No, ~ Model, ~ RMSE, ~ Comment,
        "1", "Naive mean guess", "1.0606506", "The guess has high RMSE",
        "2", "Movie effect", "0.9437046", "Does better than guessing",
        "3", "Movie and user effects", "0.8655329", "Improvement over the movie effect",
        "4", "Regularized movie and user effects", "0.8649857", "More improvement",
        "5", "Regularized movie, user, year, and genre effects", "0.864411", "Marginal improvement") %>%

  knitr::kable(caption = "Summary of Machine Learning Model and Performance")
```

Table 6: Summary of Machine Learning Model and Performance

No	Model	RMSE	Comment
1	Naive mean guess	1.0606506	The guess has high RMSE
2	Movie effect	0.9437046	Does better than guessing
3	Movie and user effects	0.8655329	Improvement over the movie effect
4	Regularized movie and user effects	0.8649857	More improvement
5	Regularized movie, user, year, and genre effects	0.864411	Marginal improvement

Conclusion

In this exercise I have build a movie recommendation system. I started with the naive system based on the average rating to a model that have a much lower RMSE that incorporates movie effects and users effects. I have also added the genre effect but the added reduction to RMSE is very low - to 0.864411. It is possible to also do a better recommendation system using the `recommenderlab` package but it was not in the scope of my analysis. In future, I plan to learn the use of recomenderlab for training recommendation system models.