Capstone Project - MovieLens

HarvardX - PH125.9x Capstone Project submitted in partial fullfilment of the requirements for the Data Science Professional Certificate

Phil Girurugwiro

2/24/2020

Contents

MovieLens Project

Introduction

Project Overview

The objective of the project is to build a movie recommendation system that would predict the user rating of any given movie in a database with a relatively good accuracy. MovieLens 10M Dataset, a data set of 10 million ratings of 10,000 movies by 72,000 users put together by GroupLens was used to accomplish this task. A training-test set approach was used to build the model on the training set and validate the accuracy of the model on the test set. To mitigate the risk of overtraining, the 10M dataset was divided in 3 sets. A training set, and two test sets. One test set was used to tune the model and the second test set was held out and used in the final validation of the model. The figure below shows how the partition was made. The objective is to build an algorithm that will result in the lowest possible residual mean squared error, RMSE, between the predicted ratings of movie i by user u, $\hat{y}_{u,i}$ and actual ratings, $y_{u,i}$. Specifically, the goal is to attain a RMSE of 0.8649 or below. The formula for the residual mean squared error is:

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

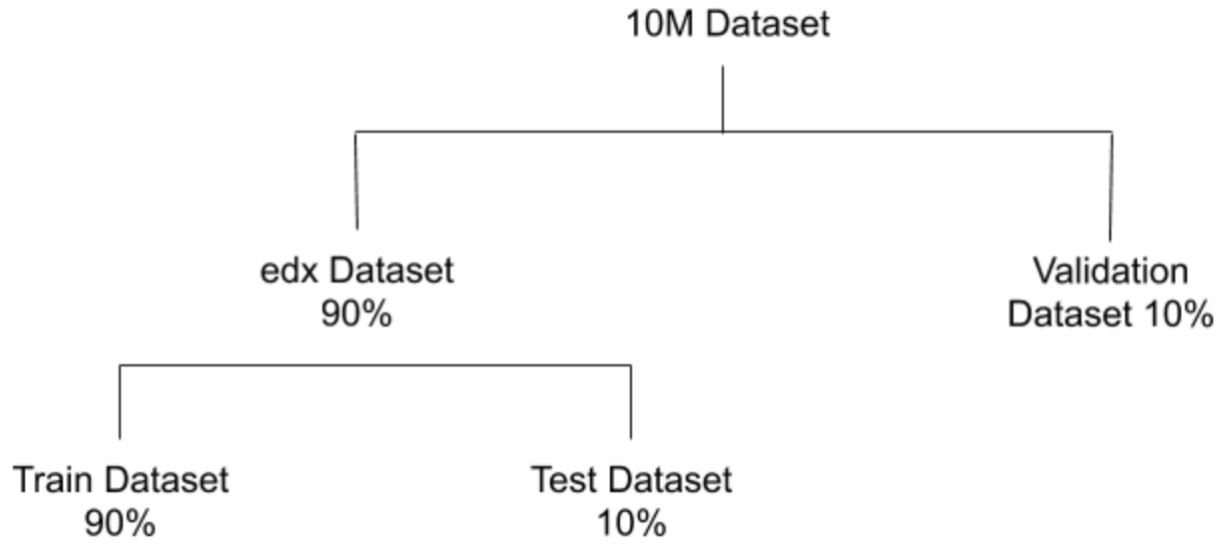


Figure 1: Dataset Partition

Downloading the Dataset

The code to download the dataset was provided by the project organisers and is reproduced below. The output of the code is two Datasets: **edx** which comprises of 90% of the original set and **validation** which comprises of 10%.

```
################################
# Create edx set, validation 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")

# 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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

#Change the rating time from the datestamp computer time (seconds since Jan 1, 1970) to actual rating d
library(lubridate)
movielens <- movielens %>% mutate(rating_date = as_datetime(timestamp), rating_year = as.character(forma
  mutate(release_year = str_extract(movielens$title,  "\\d{4}"))


# Validation set will be 10% of MovieLens data
set.seed(1, 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, movielens, removed)
```

The edx dataset is further split into the train and test sets which will be used to train and tune the model.

The final algorith will be validated on the validation set.

```
#CREATE THE TRAINING AND TESTING SETS: Train on 90% of data, and test on the other 10%

set.seed(2, sample.kind="Rounding")

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

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

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

```
rm(temp, test_index, removed)
```

Data Overview

Before training and building the model, we explore the dataset to understand the variables therein and conduct basic data analyses to gain any useful insights that might help us accomplish the task. This exploratory data analysis will be conducted on the **edx** dataset.

Exploratory Data Analysis

- Structure of the Dataset

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  9 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
##  $ title       : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres      : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Actio
##  $ rating_date : POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ rating_year : chr  "1996" "1996" "1996" "1996" ...
##  $ release_year: chr  "1992" "1995" "1995" "1994" ...
```

We notice that our Dataset is indeed a data frame that contains information on movies: titles, year of release, movie ID, and genre category. On the part of the user, the dataset only contains user ID. Finally, the dataset contains the rating that users gave to the movies.

- Summary Statistics

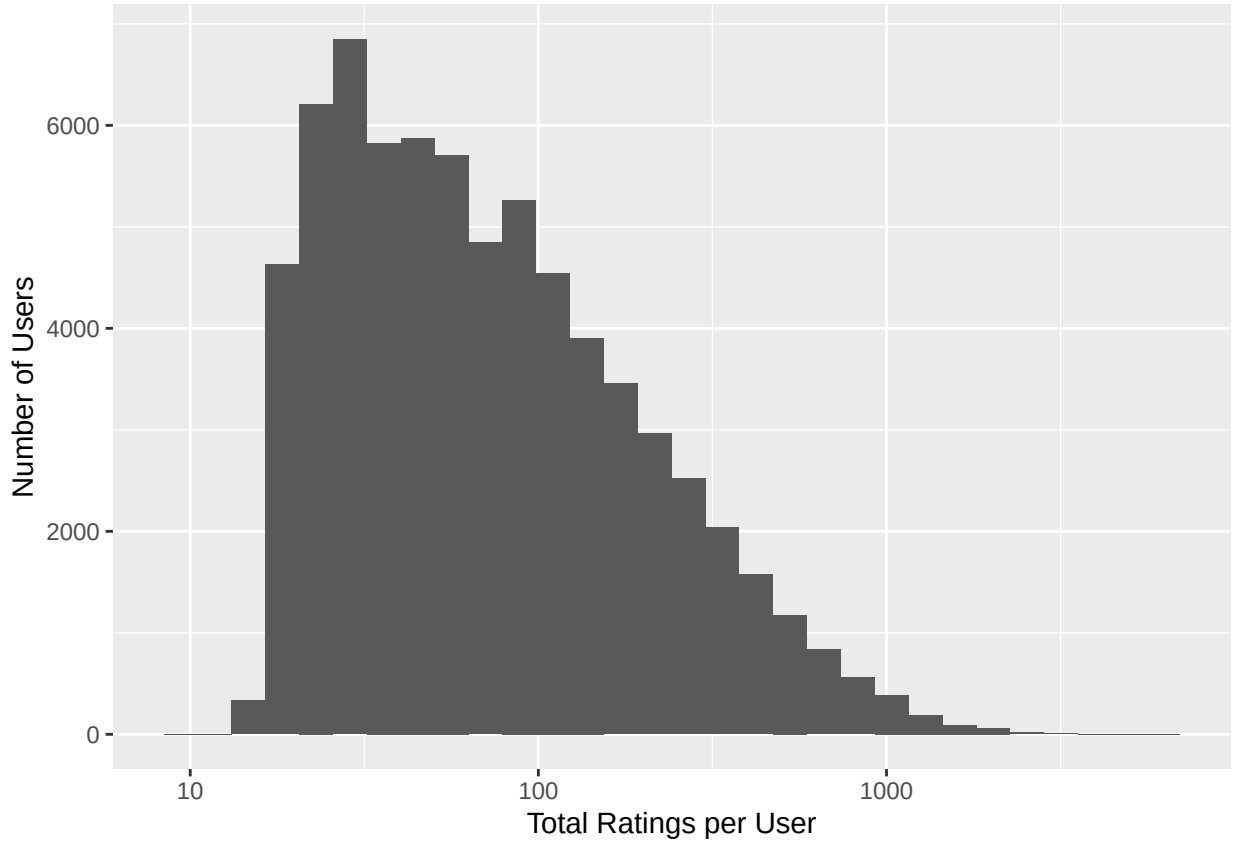
Total users and total movies in the dataset:

```
data.frame(users = n_distinct(edx$userId), movies = n_distinct(edx$movieId))
```

```
##   users movies
## 1 69878  10677
```

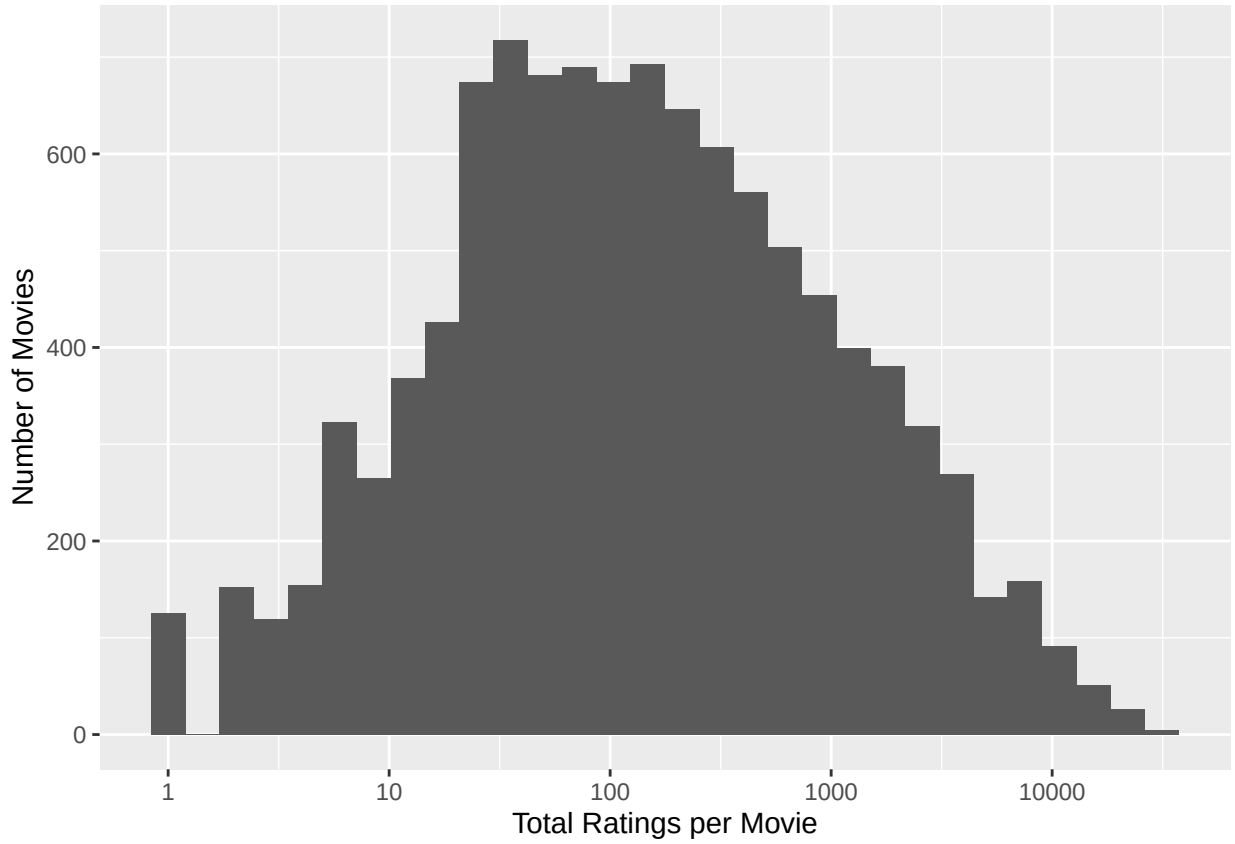
The total number of unique users is 69878 and the total number of movies is 10677 movies. If each user would to rate each movie, we'd end up with 69878 x 10677 ratings or 746M ratings. The edx dataset contains 9M ratings, which means that we only have 1.2% of the maximum possible ratings since not each movie was rated by all users. We can visualize this by looking at the distribution of ratings per movie and per user.

```
edx %>% group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("Total Ratings per User") +
  ylab("Number of Users")
```




```
edx %>% group_by(movieId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
```

```
  scale_x_log10() +
  xlab("Total Ratings per Movie") +
  ylab("Number of Movies")
```




```
edx %>% summarize(avg_rating = mean(rating))
```

```
##   avg_rating
## 1   3.512465
```

We can indeed see that the majority of users rated between 50 and 150 movies and the average rating across all users is 3.512

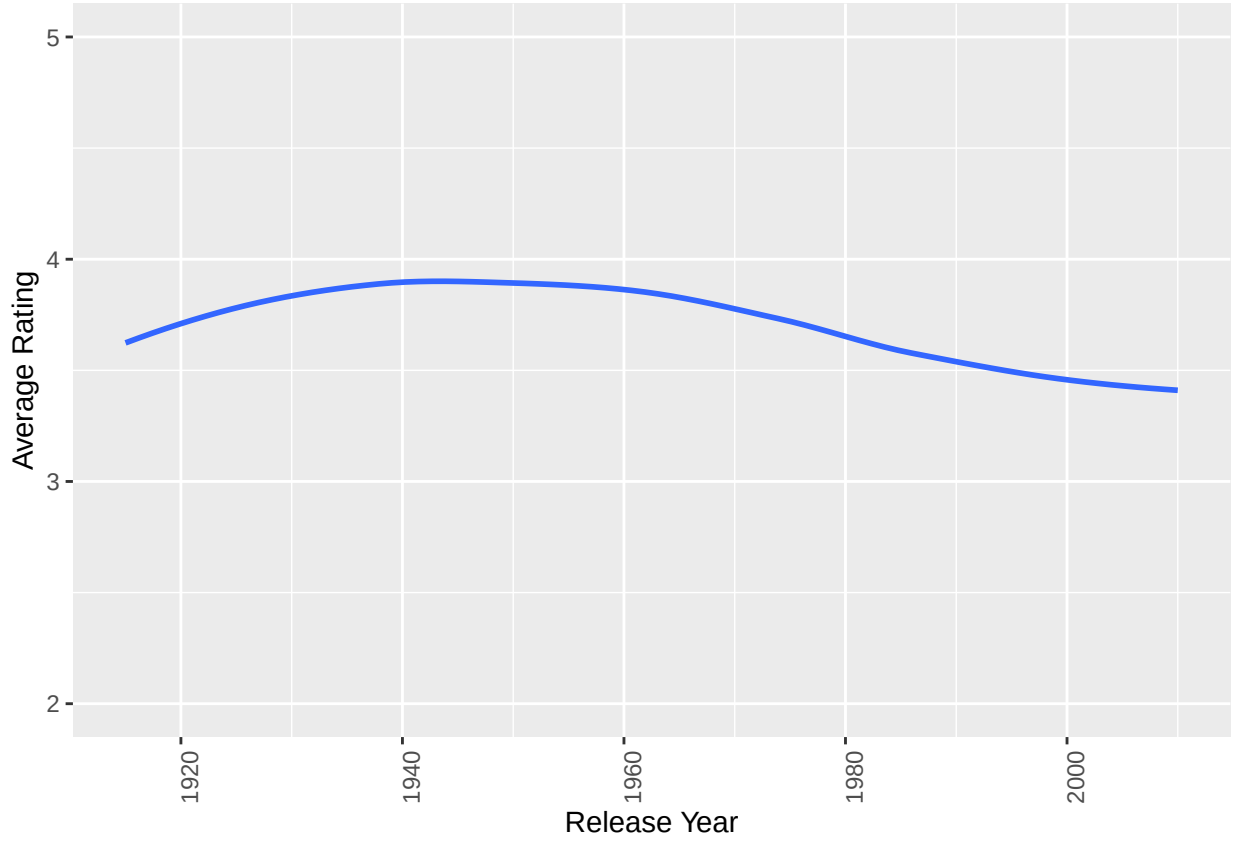
- Data Trends

Since the dataset contains two time variables: year of movie release and time of rating, it is worth considering the effects that time may have on the rating. For instance: do movies that came out in a certain time period have higher than average ranking? Does the rating of a movie go up or down as time goes by? Let's start with the average rating versus release year:

```
edx %>% group_by(release_year) %>% filter(release_year > 1900 & release_year < 2020) %>%
  summarize(AVG_ratings = mean(rating)) %>%
  ggplot(aes(as.numeric(release_year), AVG_ratings)) +
  geom_smooth(se = F) +
```

```
  theme(axis.text.x = element_text(angle = 90)) +
  scale_y_continuous(limits = c(2,5)) +
  xlab("Release Year")+
  ylab("Average Rating")
```

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

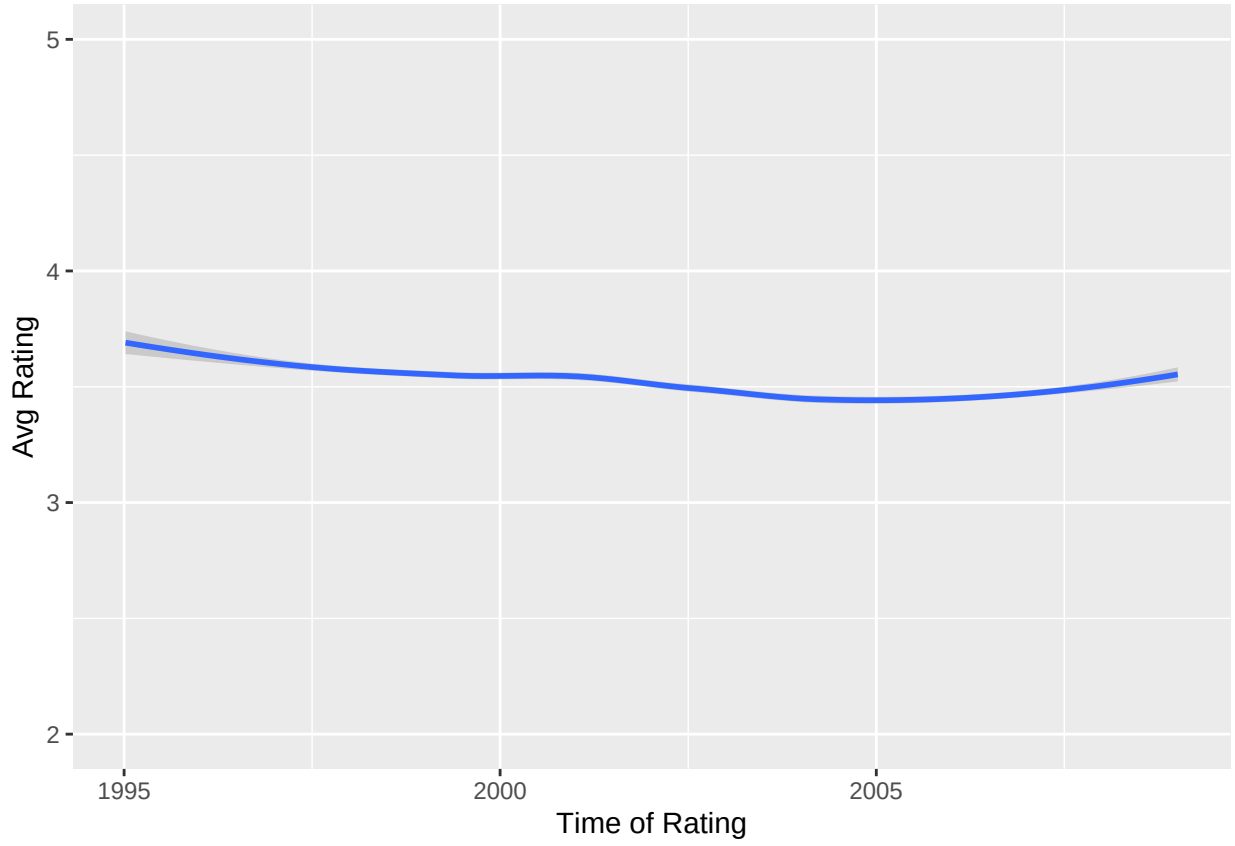


We observe that the average rating of the early to mid 1900s is higher than the most recent movies. One explanation is that the older movies are fewer and considered "classics" and tend to be rated higher, while newer movies are plenty, a mix of good and bad movies, which would tend to bring the average rating down.

Let's consider the effect of time of the rating.

```
edx %>% mutate(rating_date = round_date(rating_date, unit = "week")) %>%
  group_by(rating_date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(rating_date, rating)) +
  geom_smooth() +
  scale_y_continuous(limits = c(2,5)) +
  xlab("Time of Rating")+
  ylab("Avg Rating")
```

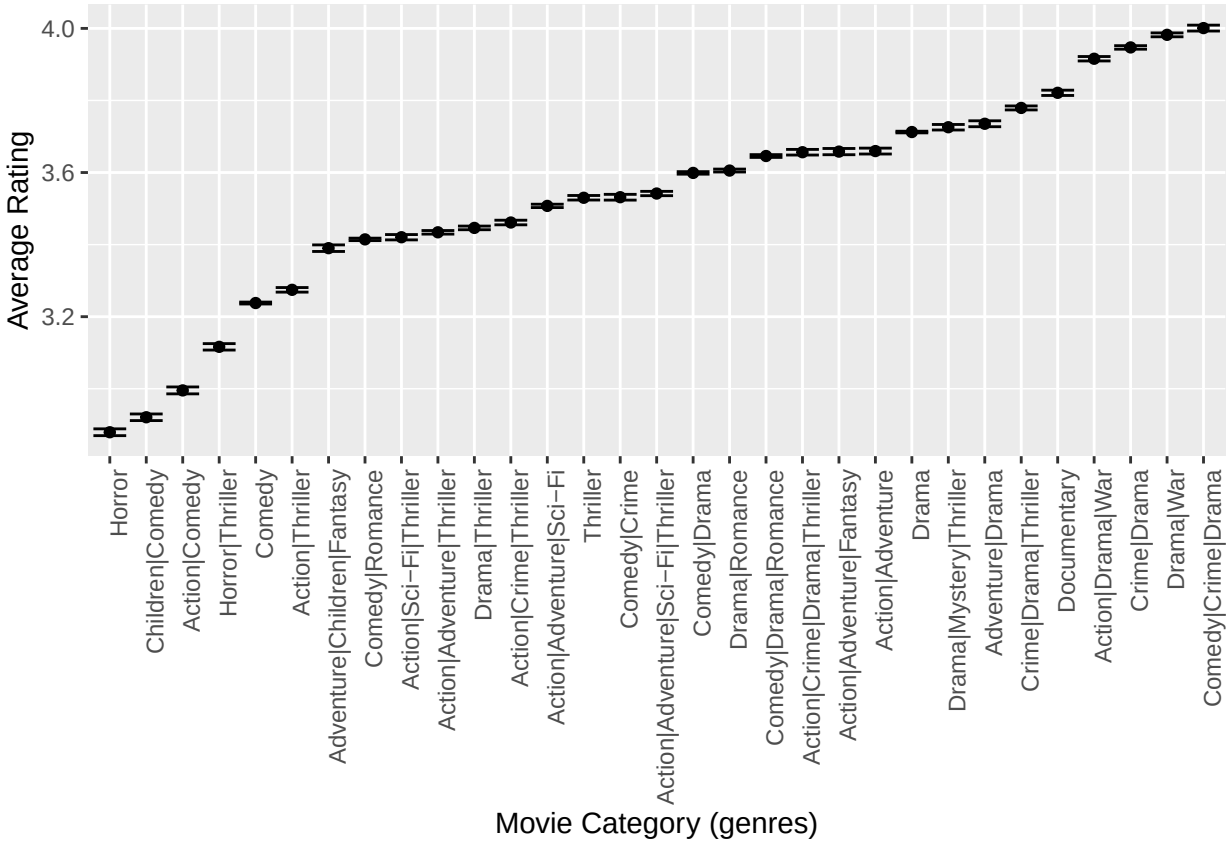
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Once again, we observe a similar behavior, this time indicating that users have more movies to rate in recent years compared to years prior, which in turn drives down the average rating.

Finally, let's consider the movie genres and see if some categories are likely to receive higher ratings compared to others.

```
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 50000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  xlab("Movie Category (genres)") +
  ylab("Average Rating")
```



Indeed there's a difference in rating for different gender categories.

Builiding a Recommendation System

From the data exploration above, it can be seen that individual movies, individual users, time of movie release, time of movie rating, and the movie genre category all play a role in the overall rating of the movie. All these factors will be taken into account in the building of the recommendation system.

Training the model consisted of several steps. On each step, the training set was used to train the model and the test set was used to evaluate the model by calculating the resulting RMSE. Recall the RMSE formula:

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

The following code was used to calculate the RMSE:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The following steps were taken:

- Predicting the rating to be simply equal to the average rating of all movies:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

- Considering the movie effect b_i

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

- Considering the user effect b_u

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

- Considering the time effect t_i

$$\hat{Y}_{u,i} = \mu + b_i + b_u + t_i + \epsilon_{u,i}$$

- Considering the genre effects g_e

$$\hat{Y}_{u,i} = \mu + b_i + b_u + t_i + g_e + \epsilon_{u,i}$$

- Regularization

We have seen in the exploratory data analysis section that some users only rated a few movies, and likewise, some movies only have a few users. We have also seen that the latest movies may have more ratings compared to the classics. Also, some genres have fewer movies compared to others. The number of ratings can have an impact to the overall average rating. To account for this effect, categories that have fewer ratings will be penalized by introducing a regularization parameter, λ. This parameter will weigh the average waiting by the number of available ratings for each consideration. Mathematically, regularization is performed as follows:

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

Model Builiding and Testing

Step 1. Simple Model

Predicting the rating as the average of ratings μ in the training set

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

```
mu <- mean(train_set$rating)
predicted_ratings_0 <- rep(mu, length(test_set$rating))
model_0_rmse <- RMSE(predicted_ratings_0, test_set$rating)
model_0_rmse
```

```
## [1] 1.06031
```

Giving each movie in the test set a rating that is equal to the overall average rating in the train set results in an RMSE of **1.06**.

Step 2. Add the Movie Effect

Predict the ratings in the test set with movie effect b_i considerations

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings_1 <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>% .$b_i
model_1_rmse <- RMSE(predicted_ratings_1, test_set$rating)
model_1_rmse
```

```
## [1] 0.9436226
```

Adding movie effects to the model improves the RMSE to 0.943.

Step 3. Add the User Effect

Predict the ratings in the test set with user effect b_u considerations

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

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

predicted_ratings_2 <- test_set %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(prediction =mu+b_i+b_u) %>%
  pull(prediction)
model_2_rmse <- RMSE(predicted_ratings_2, test_set$rating)
model_2_rmse
```

```
## [1] 0.8663835
```

With user effects added to the model, the new RMSE is now 0.8663

Step 4. Add the Timee Effect

Predict the ratings in the test set with time effect t_i considerations. Here, only the rating time was considered.

$$\hat{Y}_{u,i} = \mu + b_i + b_u + t_i + \epsilon_{u,i}$$

```
time_avgs <- train_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(movie_avgs, by= "movieId") %>%
  left_join(user_avgs, by= "userId") %>%
  group_by(week) %>%
  summarize(ti = mean(rating - b_i - b_u - mu))

predicted_ratings_3 <- test_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(time_avgs, by = "week") %>%
  mutate(prediction =mu+b_i+b_u+ti) %>%
  pull(prediction)
model_3_rmse <- RMSE(predicted_ratings_3, test_set$rating)
model_3_rmse
```

```
## [1] 0.8663111
```

By adding time of rating effect, the RMSE is very slightly improved to 0.8663

Step 5. Add the Genre Effect

Predict the ratings in the test set with movie effect g_e considerations

$$\hat{Y}_{u,i} = \mu + b_i + b_u + t_i + g_e + \epsilon_{u,i}$$

```
genre_avgs <- train_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(movie_avgs, by= "movieId") %>%
  left_join(user_avgs, by= "userId") %>%
  group_by(week) %>%
  left_join(time_avgs, by= "week") %>%
  ungroup() %>%
  group_by(genres) %>%
  summarize(ge = mean(rating - ti - b_i - b_u - mu))

predicted_ratings_4 <- test_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(time_avgs, by = "week") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(prediction =mu+b_i+b_u+ti+ge) %>%
  pull(prediction)
model_4_rmse <- RMSE(predicted_ratings_4, test_set$rating)
model_4_rmse
```

```
## [1] 0.8659941
```

The genre effect provides a further reduction in the RMSE to 0.8659

Step 6. Regularization

Predict the ratings in the test by regularizing all the features in previous steps.

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

The value for the parameter λ was chosen after running a loop function to choose the value that minimizes the RMSE function.

```
l <- 5.5

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

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

ti <- train_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(b_i, by= "movieId") %>%
  left_join(b_u, by= "userId") %>%
  group_by(week) %>%
  summarize(ti = sum(rating - b_i - b_u - mu)/(n()+l))

ge <- train_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(b_i, by= "movieId") %>%
  left_join(b_u, by= "userId") %>%
  group_by(week) %>%
  left_join(ti, by= "week") %>% ungroup() %>%
  group_by(genres) %>%
  summarize(ge = sum(rating - b_i - b_u - mu - ti)/(n()+l))


predicted_ratings_5 <- test_set %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(ge, by = "genres") %>%
  left_join(ti, by = "week") %>%
  mutate(pred = mu + b_i + b_u + ti + ge) %>%
  .$pred
model_5_rmse <-  RMSE(predicted_ratings_5, test_set$rating)
model_5_rmse
```

```
## [1] 0.8654096
```

The regularization of all the features used in the model provides a significant improvement of RMSE to 0.8654. This model will now be evaluated on the Validation set that was held out.

Step 7. Final Step: Testing the Model on the Validation Set

Predict the ratings in the test set with movie effect b_i considerations

```
final_ratings <-
  validation %>%
  mutate(week = round_date(rating_date, unit = "week")) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(ge, by = "genres") %>%
  left_join(ti, by = "week") %>%
  mutate(pred = mu + b_i + b_u + ti + ge) %>%
  .$pred

final_rmse <- RMSE(final_ratings, validation$rating)
final_rmse
```

```
## [1] 0.8647396
```

Results

```
rmse_results <- data_frame(Method= c("Just Average", "Movie Effect", "User Effect",
                                     "Time of Rating Effect", "Genre Effect",
                                     "Regularized Parameters", "Final RMSE/Validation Set"),
                           RMSE_result =c(model_0_rmse, model_1_rmse, model_2_rmse,
                                          model_3_rmse, model_4_rmse, model_5_rmse,
                                          final_rmse))
```

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

```
#SUMMARY TABLE
rmse_results %>% knitr::kable()
```

Method	RMSE_result
Just Average	1.0603099
Movie Effect	0.9436226
User Effect	0.8663835
Time of Rating Effect	0.8663111
Genre Effect	0.8659941
Regularized Parameters	0.8654096
Final RMSE/Validation Set	0.8647396

We observe that each feature consideration had a positive impact on the RMSE. The time feature had the least impact. While it would have been appropriate to consider each feature by itself, it was deemed efficient to build up the model by successively adding on the features. The end result satisfied the objective and goal of the project that was set forth in the beginning.

Conclusion

This project started off with large data set. The objective was to split the data set into a training set, a testing set, and a final evaluation set. A deep dive into the data set was done to study the structure of the data and its objects, exploratory data analysis was done to visualize patterns that may be helpful in building up the model. Finally, a model was build, taking into account the major features of the data set. The final model presented in this project achieves a RMSE value of 0.8647. This number proves satisfactory for the purpose of this project. However, it could be further improved by exploring other machine learning algorithms and considering other factors that were not explored in this project, e.g. the impact of the number of ratings that a user gives on a given day to the overall rating per user.

Reference Introduction to Data Science - Data Analysis and Prediction Algorithms with R by Rafael A. Irizarry