HarvardX PH125.9x - Capstone - Movielens Project

Kwaku Owusu-Tieku (kowusu01@gmail.com)

3/9/2022

Introduction

In this project, I present a movie recommendation system that can be used for predicting movie ratings. Recommendation systems are used by companies such as Netflix, Amazon, and other large online retailers to suggest items for users based on some parameters. For instance, items can be suggested to users based on popularity.

In the case of movie recommendation, we could assume that a user might like a movie simply because many people like that movie. In other cases we could analyze user profiles for attributes such as gender, age group, etc. to find similarities among users and recommend items based on their similarities. For instance, we might recommend Horror movies to males if our analysis reveal that males tend to watch more Horror movies than females.

In our dataset, we have a selected number of users and a set of movies. Not all users have watched or rated every movie. In fact, no single user has rated every movie. The challenge here is to assign (predict) ratings to movies that a user has not rated. The assumption here is that a higher rating means the user liked the movie. Therefore, if our prediction assigns a high rating to a movie for a user, we take that to mean the user will most likely like that movie, hence, we can recommend that movie to the user.

Source Code and Repository

The source code for the entire project can be found on github at https://github.com/kowusu01/KOwusu.Tieku.HarvardX.Capstone.Movielens

Development Environment Used

OS: Windows 10 x64 (build 19041)
Machine: Dell Latitude, i5 Dual Core @ 2.4GHz and 2.5GHz
Memory: 16GB
R: version 4.1.2 (2021-11-01)

The Dataset

The dataset used in this project is the ml-10m.zip containing 10million records. This dataset can be found at http://files.grouplens.org/datasets/movielens/ml-10m.zip. As per standard data science practice, the data has been partitioned into training (the edx set), and hold-out (validation set). All training and tuning will be peformed on the edx set.

Quick Glance at the Dataset

To begin the analysis, the movielens data is downloaded and unzipped. There are two important files in this download - movies.dat and ratings.dat. These files can be viewed using any standard text viewer.

movies.dat
A quick peek at the **movies.dat** shows that the file is a delimeted with double colon (::) as the separator. Each line is unique movie having the movieId, the title, and the list of genres for that movie. Notice that the genre is also a delimeted string with | as the separator.

ratings.dat
The next file in the downloaded zip is the **ratings.dat**. This file contains each rating a user has given a movie. Again this is a delimited file with :: as the separator. Each line contains userId, the movieId, the rating that was provided, and the timestamp.

Data Wrangling

Required Wrangling

In the dataset, each rating is an observation, therefore the data in the *ratings.dat* is joined with the dataset in the *movies.dat*. After some data wrangling, the resulting data looks as below. All features are converted to the appropriate types. For instance, the movieId and userId must be converted from character to numeric. The movie title and genres by default might be loaded as factors; they are converted to characters. The following table shows a sample of the data after initial wrangling.

```
Rows: 9,000,055
Columns: 6
$ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
$ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
$ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
$ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
$ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
$ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

Table 1: Sample records after joining movies and ratings

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical

Additional Wrangling

The following data wrangling tasks were performed.

Split Movie Genres. The movie genre column is also broken down into individual items and added as dummy variables. For instance, if a movie is listed under genres Action and Comedy, it will have 1 under those two columns and 0 for the rest.

Fixing Problematic Genre Names. Close inspection of the genres also revealed that some genres contain hyphens which could cause problems when used as variables names. These genres are **'Sci-Fi'** and **'Film-Noir'**. The hyphens were removed. Another genre, **'(no genres listed)'** contain spaces and could cause problems later. The genre was renamed to *NoGenres*. The code used to perform that task is listed below:

```r
###############################################################
# split genres and create column for each genre
###############################################################
fnProcessGenres <- function(dataset){

  dataset <- dataset %>% mutate(genre = str_split(genres, "\\|" )) %>%
    unnest(cols = genre) %>% mutate(is_used=1)
  dataset <- dataset %>%
    pivot_wider(names_from = genre, values_from=is_used, values_fill = 0)

  (genre_names <- names(dataset))

  ## rename the following genres

  if(which(genre_names == "Sci-Fi") > 0)
    names(dataset)[which(genre_names == "Sci-Fi")] <- "SciFi"

  if (which(genre_names == "Film-Noir") > 0)
    names(dataset)[which(genre_names == "Film-Noir")] <- "FilmNoir"

  if(which(genre_names == "(no genres listed)") > 0)
    names(dataset)[which(genre_names == "(no genres listed)")] <- "NoGenre"

  dataset %>% select(-c("genres") )

}


# 1. split genres into columns and add to the dataset
edx <- fnProcessGenres(edx)

# 2. extract movie release year from titles
edx <-  edx %>%
  mutate(release_year = str_extract_all(title, pattern = "\\(\\d{4}\\)"))

edx$release_year <- edx$release_year %>% str_replace(pattern= "\\(", "")
edx$release_year <- edx$release_year %>% str_replace(pattern= "\\)", "")
edx <- edx %>% mutate(release_year = as.integer(release_year))

# convert review timestamp to datetime
edx <- edx %>% mutate(timestamp = as_datetime(timestamp))
```

The following table shows the data after the wrangling.

```
Rows: 9,000,055
Columns: 26
$ userId       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ movieId      <int> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42~
$ rating       <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
$ timestamp    <dttm> 1996-08-02 11:24:06, 1996-08-02 10:58:45, 1996-08-02 10:~
$ release_year <int> 1992, 1995, 1995, 1994, 1994, 1994, 1994, 1994, 1994, 199~
$ title        <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",~
$ Comedy       <int> 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, ~
$ Romance      <int> 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, ~
$ Action       <int> 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, ~
$ Crime        <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ~
$ Thriller     <int> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, ~
$ Drama        <int> 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, ~
$ SciFi        <int> 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ~
$ Adventure    <int> 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ~
$ Children     <int> 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, ~
$ Fantasy      <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ~
$ War          <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ~
$ Animation    <int> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, ~
$ Musical      <int> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, ~
$ Western      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ Mystery      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ FilmNoir     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ Horror       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ Documentary  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ IMAX         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ NoGenre      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

Data Exploratory

In this section, I explore the data to gain understanding of the general properties of the data. In most cases, I will be exploring the training data(edx) because it is assumed that the data distribution is very similar, if not the same in both training and validation sets.

Before exploring, I ask a few questions that I attempt to answer using the data.

- For instance, what are the patterns in the user rating?
- What are the top most rated movies?
- Who are the most active users in terms of number of rating?
- Do some people just give 5-star rating to every movie and others just rating every movie poorly?
- What is the overall rating average for all movies? What are the average rating per movie and per user?

The following table shows the total number of records, the total number of users, number of movies in both the training and validation datasets.

Table 2: Items Count in the Datasets

Item	TrainingSet	ValidationSet
Unique Records	9000055	999999
Unique Users	69878	68534
Unique Movies	10677	9809

Null Values

A simple query to the training dataframe shows that there are no null (NA) values in either the edx or validation datasets. This is good news since we don't need to exclude any invalid data.

```
sum( is.na(edx) )
```

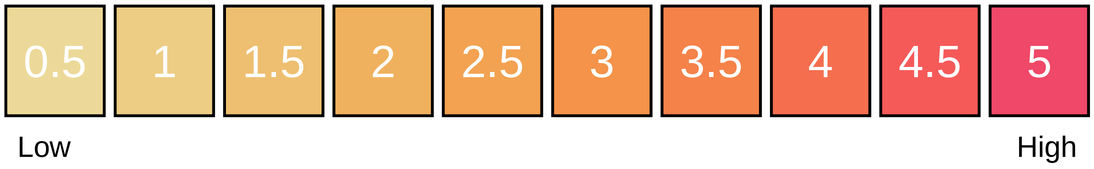
```
[1] 0
```

```
sum( is.na(validation) )
```

```
[1] 0
```

Overall Mean, Median

Using the unique() and the summary() functions, we see that overall there are ten unique ratings, given by users with a minimum of `0.5` and max of `5`. No rating of zero (0) is given. The overall mean for movie rating is `3.51`, with a median of `4`. This means most users seem to be generous and give pretty high ratings with **4.0** being the most predominant rating.



```
unique(edx$rating) %>% sort()
```

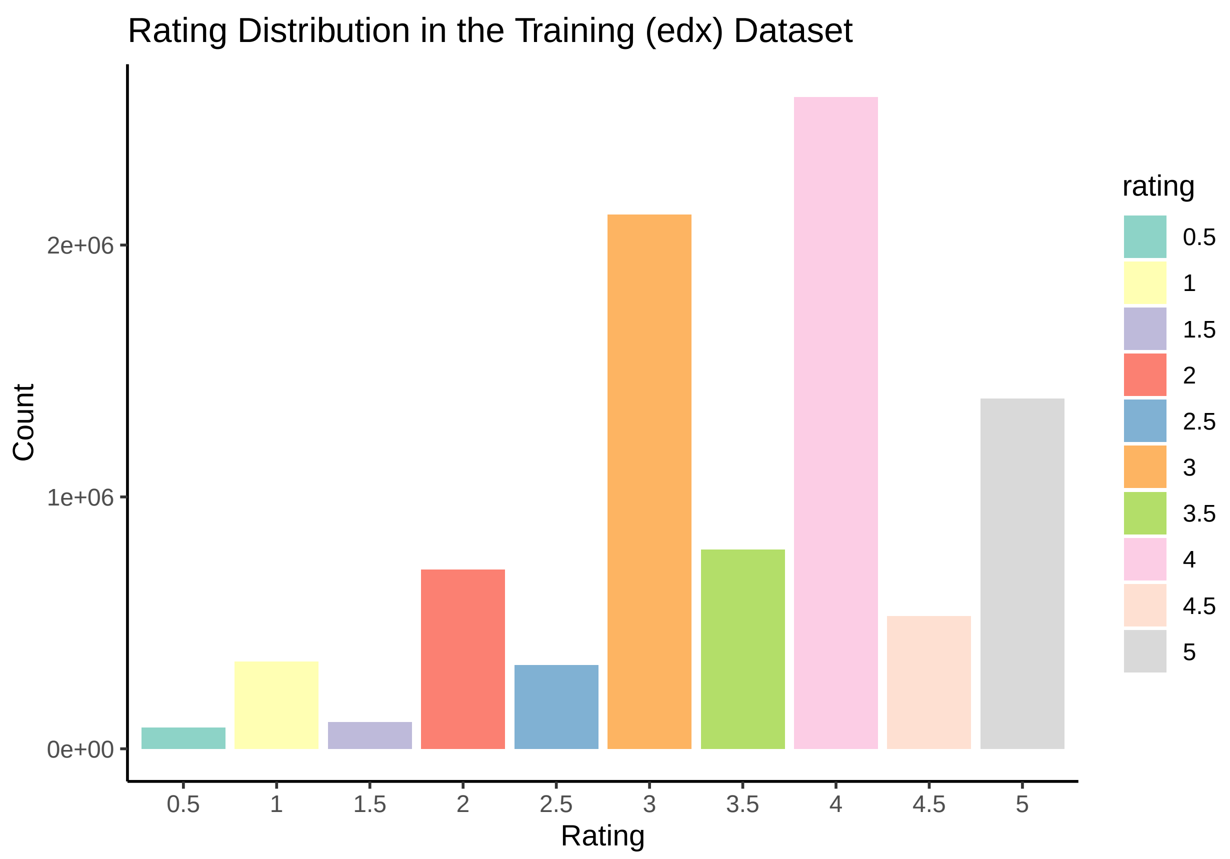
```
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
summary(edx$rating)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.500   3.000   4.000   3.512   4.000   5.000 
```

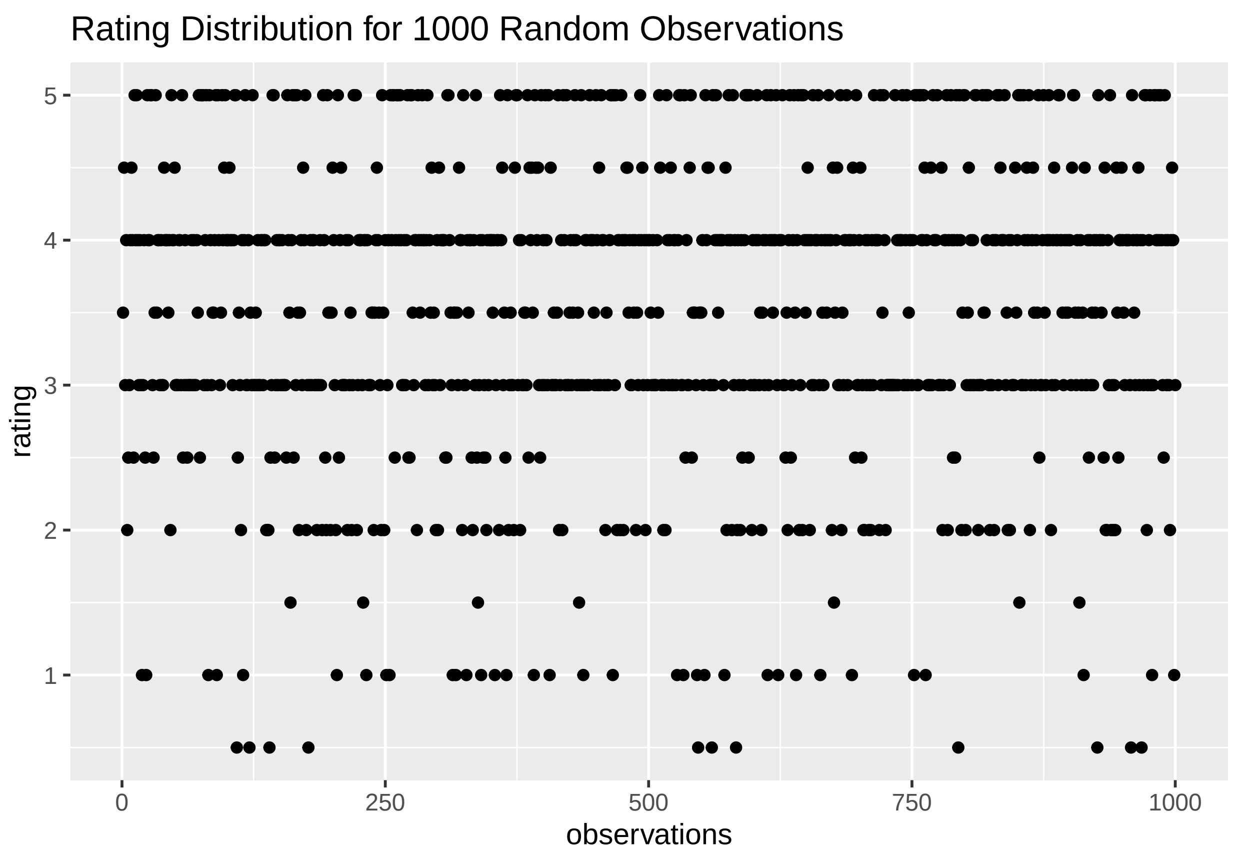
The rating distribution can also be visualized using a simple histogram.

```
edx %>% group_by(rating) %>% summarise(n=n()) %>%
  select(rating, n) %>%
  mutate(rating = factor(rating)) %>%
  ggplot() + aes(x=rating, y=n, fill=rating) + geom_col() +
        scale_fill_manual(values = my_colors_20)    +
  labs(title = 'Rating Distribution in the Training (edx) Dataset',
       x="Rating", y="Count") +
  theme_classic()
```


Rating Distribution in the Training (edx) Dataset
Count
2e+06
1e+06
0e+00
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Rating
rating
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Another easy way to visualize the distribution is to plot a sample rating. A small sample of 1000 random observations confirm that 3.0, 4.0, and 5.0 are the most common ratings as indicated by the almost solid lines on the plot below.

```
sample.size <- 1000
sample.data <- sample_n(edx, sample.size) %>%
  select(rating) %>%
  mutate(x = seq(1:sample.size))

sample.data %>% ggplot() +
  geom_point(aes(x = x, y = rating), show.legend = FALSE) +
  labs(title = 'Rating Distribution for 1000 Random Observations', x = "observations", y="rating")
```



Variations in Users Rating Behavior

We know from experience that users exhibit different behaviors when it comes to rating items, whether it's movies, items purchased or whatever. For some users, ratings movies may be habitual, meaning they rate almost every movie they watch, others rate only those movies they like, while others only rate movies they really hated just to warn other people. Of course, there are some users who don't rate at all.

Among users who make the effort to rate movies, some are very generous and give every movie a 5 star rating, others may give a poor or average rating to every movie. These differences are called biases; the exact reasons for these biases are not known, buts it's important to be aware of them when making predictions about how users may rate future movies. In this section, I explore the data to find some of these biases.

Most Active Users vs Least Active Users

I have selected top ten most active users and ten least active users by number of rating and shown in the table. From the table, we see a huge difference in the number of movies rated between the two groups with most active users in thousands while least active users in tens. There is also a difference in average ratings with some of the least active users having even higher averages than active users. This is important to note because users with few ratings may be outliers and can affect our predictions.

Table 3: 10 Most Active and 10 Least Active Users by No. of Ratings

userId	num_ratings
Most Active	
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023
19635	3771
3817	3733
63134	3371
58357	3361
27584	3142
Least Active	
62516	10
22170	12
15719	13
50608	13
901	14
1833	14
2476	14
5214	14
9689	14
10364	14

Most Rated and Least rated Movies

The same is done for movies; the top and bottom ten movies by number of ratings are shown below in the table. Again, we see a vast difference in the two groups. Some movies in the least rated group received only one rating while those in the most rated group have received tens of thousands of ratings. So there is potential outliers here.

Table 4: Ten Most Rated and Ten Least Rated Movies by No. of Ratings

movieId	num_ratings
Most Rated	
296	31362
356	31079
593	30382
480	29360
318	28015
110	26212
457	25998
589	25984
260	25672
150	24284
Least Rated	
3191	1
3226	1
3234	1
3356	1
3383	1
3561	1
3583	1
4071	1
4075	1
4820	1

Most Rated Movies

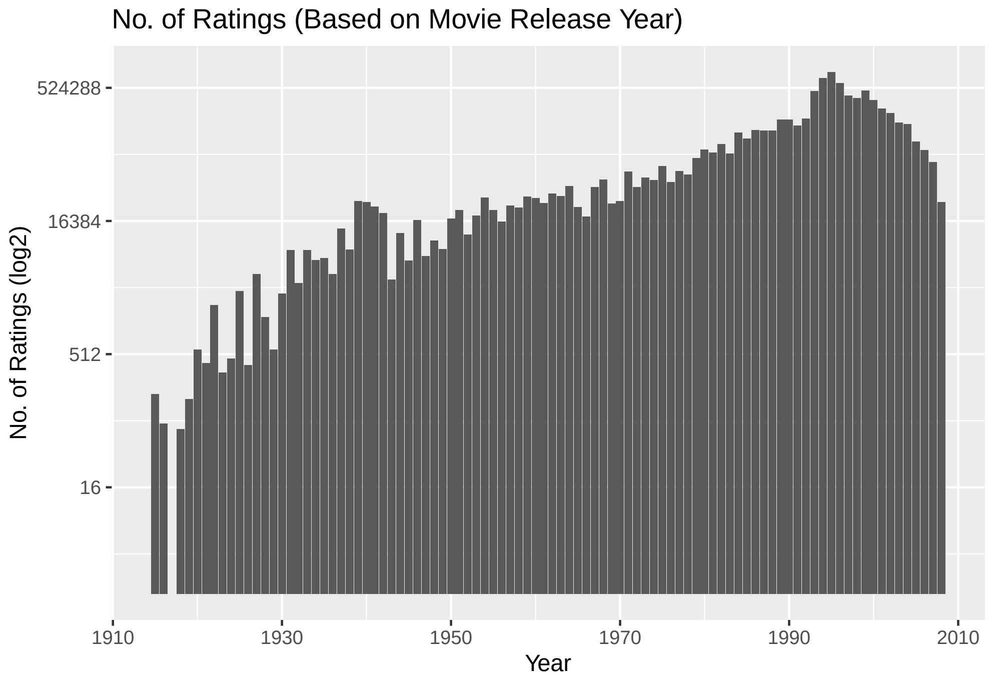
To get an idea of most rated movies based on average rating, I exclude movies that have received less than 50 ratings. This provides a better view of what users think are the best movies. The table below lists the top 10 movies by average ratings.

Table 5: Top 10 Movies by Avg. Ratings

movieId	avg_rating	title
318	4.455131	Shawshank Redemption, The (1994)
858	4.415366	Godfather, The (1972)
50	4.365854	Usual Suspects, The (1995)
527	4.363493	Schindler's List (1993)
912	4.320424	Casablanca (1942)
904	4.318651	Rear Window (1954)
922	4.315880	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)
1212	4.311426	Third Man, The (1949)
3435	4.310817	Double Indemnity (1944)
1178	4.308721	Paths of Glory (1957)

Ratings Distribution Based on Movie Release Year

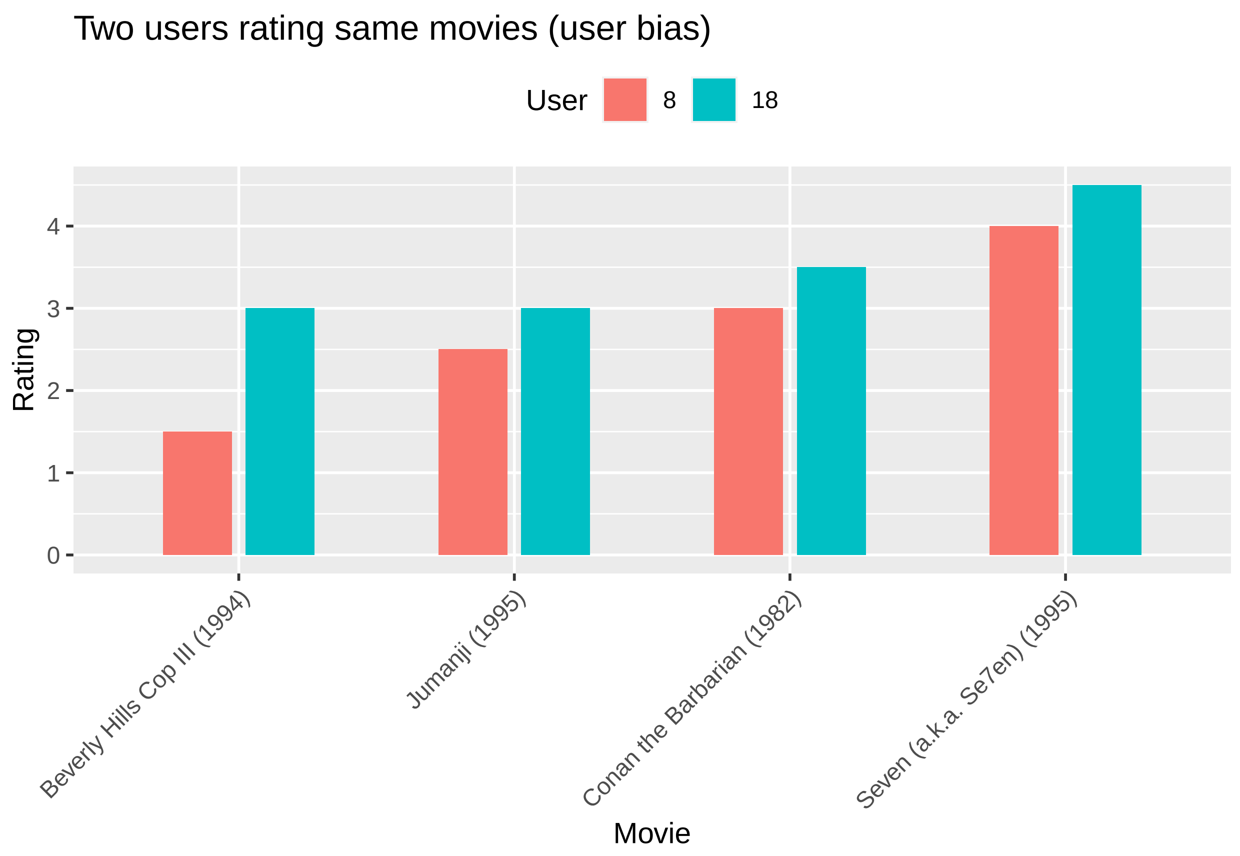
Number of Rating Per Release Year. From the data, the ratings are not evenly distributed based movie release year. As seen from the table below some movies released in certain years receive more ratings than others. In general, the number of ratings are increasing with time, with most rating being given to movies released in the 90s. In fact 1995 movies have the most rating than all years. However, as the graph shows, the number of ratings start to decrease for more recent movies (after 1995). The decrease in the number of rating for the newer movies may be due to the fact that most people have not seen the new movies yet.



Some users give higher ratings than others

As an illustration, I selected four random movies, two users who have watched all four movies, and observed their rating. See the table and table and chart below. From the figures, user **18** gives higher ratings to all four movies than user **8**.

	User Ratings			
userId	Beverly Hills Cop III (1994)	Conan the Barbarian (1982)	Jumanji (1995)	Seven (a.k.a. Se7en) (1995)
8	1.5	3.0	2.5	4.0
18	3.0	3.5	3.0	4.5

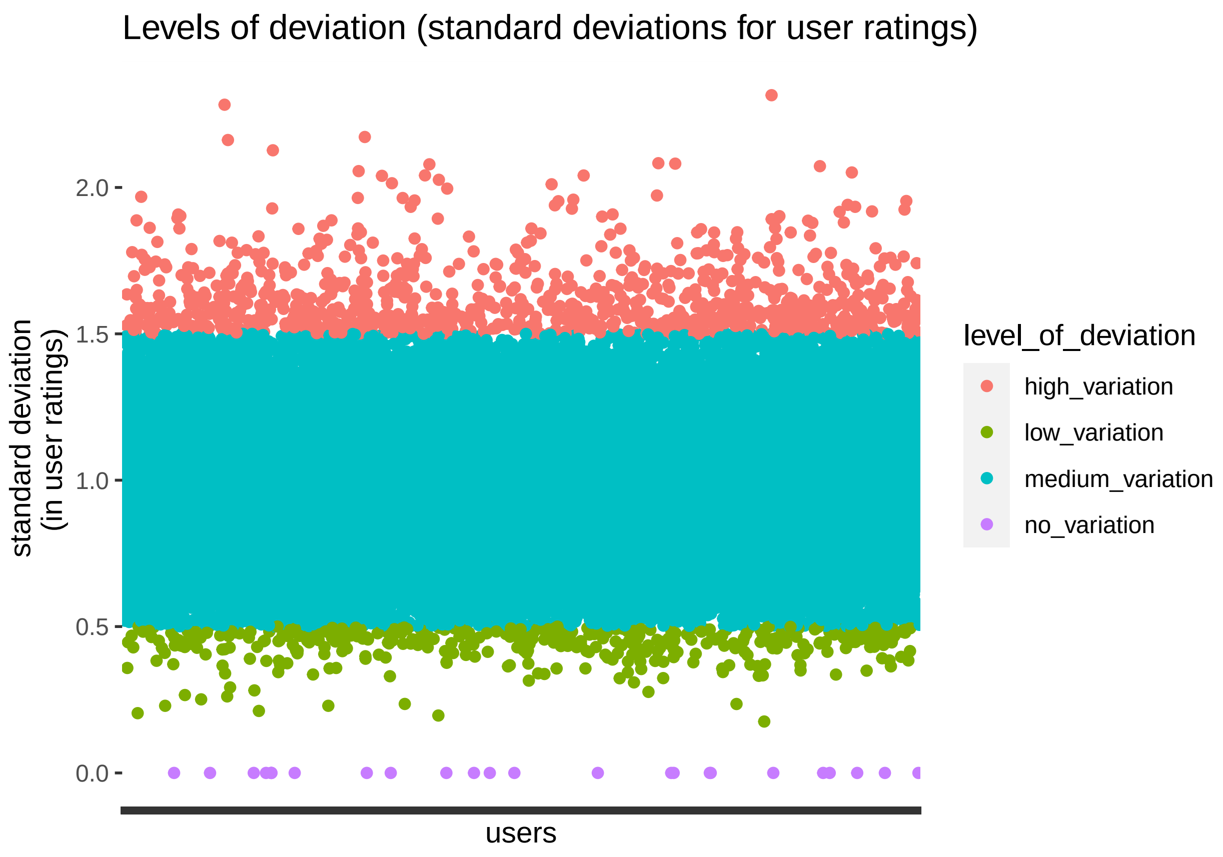


Some users give the same rating to every movie

Another form of bias result from some users tendency to give the same rating to every movie. When a user gives the same rating to every movie, the variation (the standard deviation) is zero. This type of data is not useful for algorithms looking for patterns and such data is often removed from the dataset.

Here I start by computing the standard deviation for each user's ratings and plot a sample of it.

```r
# user bias - some users tend to give the same rating for every movie regardless

# we start by calculating standard deviations for each user's ratings
user_rating_sds <- edx %>% select(userId, movieId, rating) %>% group_by(userId) %>%
  summarise(n=n(), sd=sd(rating)) %>% arrange(desc(sd))

# see users and the standard deviation in their ratings
user_rating_sds %>% select(userId, sd) %>% arrange((sd)) %>% slice(1:1000) %>% view()

# now lets plot a sample of the standard deviations
sd_sample_index <- sample(nrow(user_rating_sds), 50000)
sd_sample <- user_rating_sds[sd_sample_index, ]

sd_sample <- sd_sample %>%
  mutate(level_of_deviation=if_else(sd==0, "no_variation", if_else(sd<=0.5, "low_variation",
                                  if_else(sd<=1.5, "medium_variation", "high_variation"))))

ratings_sd_chart <- sd_sample %>% ggplot(aes(x=factor(userId), y=sd)) +
  geom_point(aes(color=level_of_deviation)) +
  labs(title = "Levels of deviation (stanbdard deviation for user ratings)") +
  ylab("stanard deviation") + xlab("") +
  theme(axis.text.x=element_blank())
```



From the chart above we see that the majority of the user ratings have variations (standard deviation) between **0.5** and **1.5**. There are obviously some with no variation (standard deviation of 0, as seen at the bottom part of the chart), and some are above **2.0**.

Exploring further, the users that give same ratings to all movies are identified. The table below lists the users with zero standard deviations and favorite their ratings, including how many movies each has rated.

Table 6: Users with zero standard deviations in ratings

userId	favorite_rating	num_ratings
1	5.0	19
3457	1.0	19
4771	3.0	21
7984	5.0	17
11884	5.0	18
13027	5.0	29
13496	0.5	17
13513	5.0	17
13524	5.0	20
15575	5.0	29
18965	5.0	49
22045	5.0	18
22964	4.0	17
24176	1.0	131
24490	1.0	17
26308	5.0	17
27831	5.0	20
29110	3.0	185
30519	5.0	17
31598	4.0	23
33017	4.0	43
35184	5.0	23
42649	5.0	20
48146	0.5	25
49209	2.5	18
49438	3.0	30
49862	0.5	17
52674	5.0	14
52749	5.0	85
54009	5.0	27
58344	4.0	17
62815	0.5	20
63381	0.5	18
65873	5.0	19
68379	5.0	56
71422	5.0	16

From the table above, it's clear that the number of users with zero standard deviation is a tiny percentage compared to the thousands of users in the dataset. We can also see that these users have rated fewer movies compared to other users in the dataset. So maybe, the lack of variability may be due to the fact that they have not rated many movies. Hopefully, once they rate more movies, their ratings might exhibit more variability.

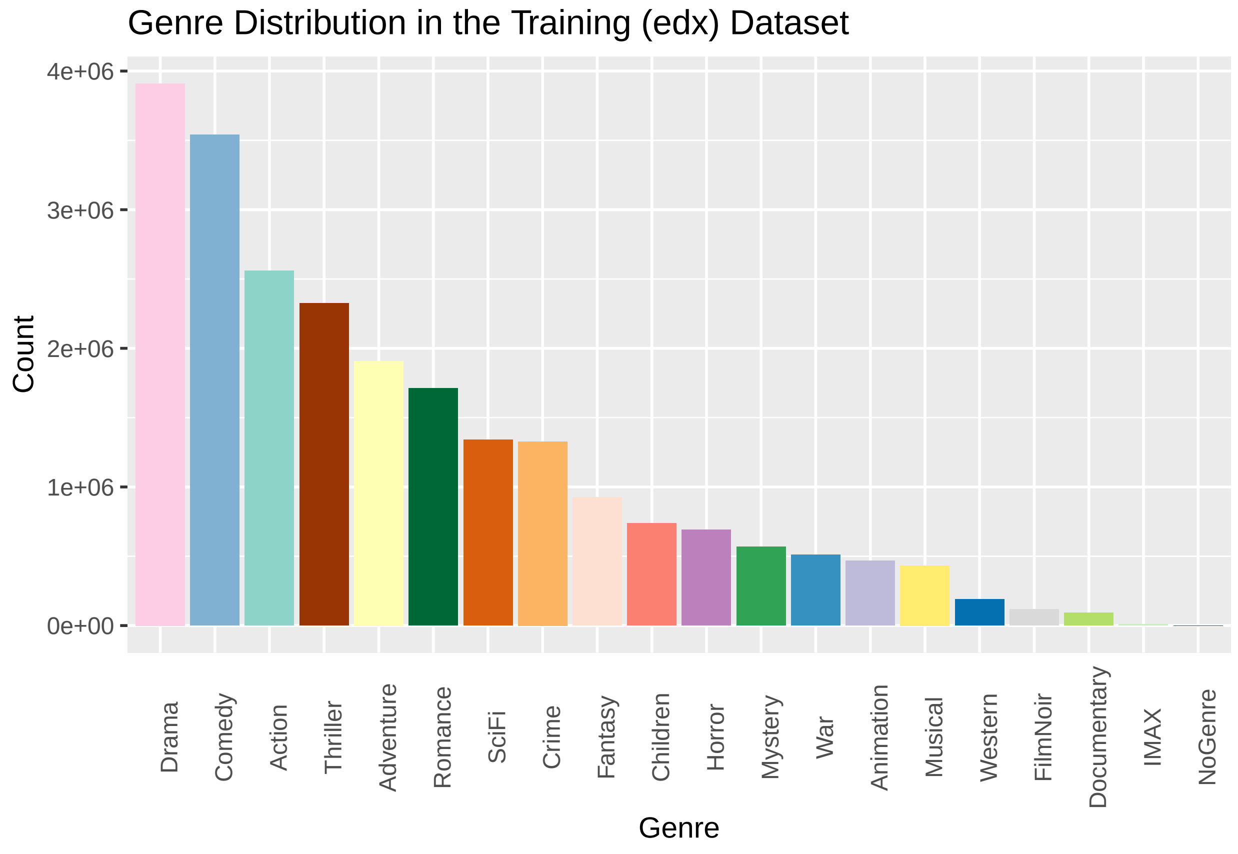
Movie Genres

Movie genres can also influence movie ratings. Some people like Action movies while others like Comedy. As mentioned in the Data Wrangling section, the genres have been broken down into columns representing the individual genres. The following table shows the frequency of each genre in the edx dataset.

Table 7: Movies with Sample Genres

userId	movieId	rating	Comedy	Drama	Action	Thriller	Adventure	Crime	Romance	War
1	122	5	1	0	0	0	0	0	1	0
1	185	5	0	0	1	1	0	1	0	0
1	292	5	0	1	1	1	0	0	0	0
1	316	5	0	0	1	0	1	0	0	0
1	329	5	0	1	1	0	1	0	0	0
1	355	5	1	0	0	0	0	0	0	0
1	356	5	1	1	0	0	0	0	1	1
1	362	5	0	0	0	0	1	0	1	0
1	364	5	0	1	0	0	1	0	0	0
1	370	5	1	0	1	0	0	0	0	0

From the table and the graph below, it can be seen that Drama and Comedy are the most frequent genres in the dataset than any other genre. Genres such as Western, Film Noir, and Documentary are less common.



Model, Approach, and Analysis

Approach

The model used in this analysis will be based on the linear model described in the course work[1]. This model trains an algorithm using user and movie effects (biases).

The main steps are outlined below and also shown the subsequent figure following the steps.

0. the data is first partitioned into edx and validation sets
1. from the edx dataset, set aside 10% for testing during the training phase
2. the remaining 90% will be used in cross validation; on the remaining 90%, use 10-fold cross validation to create train/test sets (cross validation train/test pairs)
3. on each cv train/test, calculate the biases, calculate rmse across the given set of lambda values
4. find the *"best"* lambda from each cv from step 3
5. average all the *"best"* lambdas to obtain best final *best_lambda*
6. fit a final model by computing all the biases on the entire edx dataset using *best_lambda* from step 5
7. use the final model from step 6, and apply it to final validation set to make final predictions

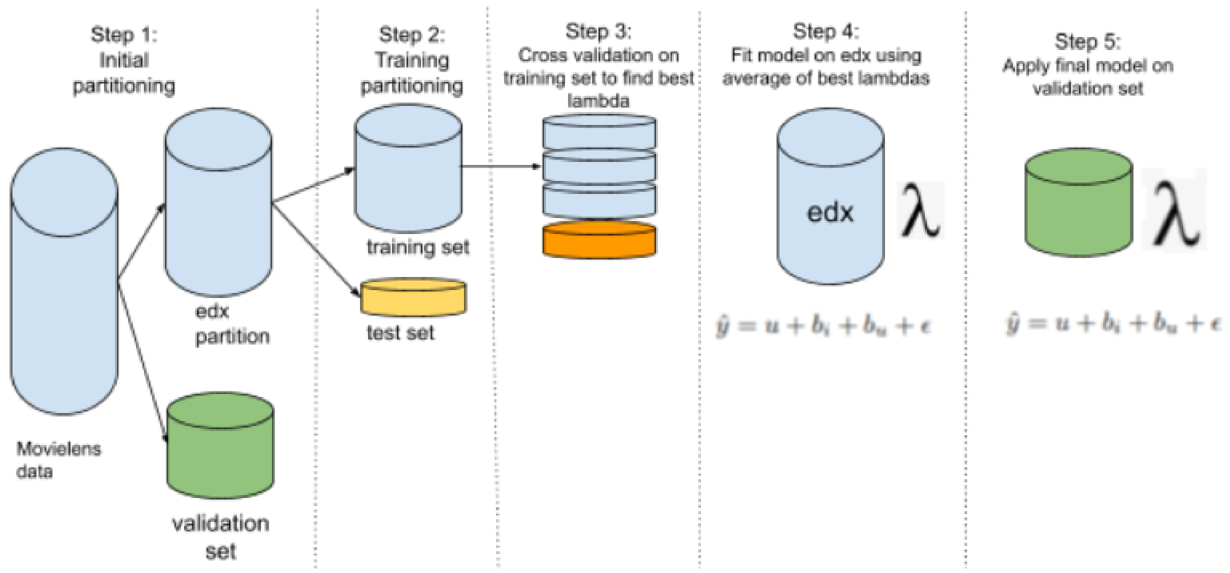


Figure 1: Modeling Approach

[1]Irizarry, Rafael A., (2021), Introduction to Data Science Data Analysis and Prediction Algorithms with R, Chapter 34, Section 34.7 - Recommendation systems. https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems/

Data Partitioning

The training dataset, (edx), as already mentioned, is partitioned into 90/10 (train/test) to train and test the models.

Building the Model - Predictors (Features) and Response

Even though it is clear that the main variable we are attempting to predict (the response) is movie rating, the question of predictors is a bit tricky. For instance we know that userId, and movieId are two predictors for movie ratings, but userId for instance is not just a single variable because we do not have only one user with a single bias and a single coefficient. Instead we have several users each having a different bias and a different coefficient. In fact, we can look at the data as unique movies, whose ratings are influenced by multiple user biases plus biases from variations in the movie types, and movie metadata such as cast, genre, etc.

The Base Model - Regularized Linear Model

The base model, is a regularized linear model (Penalized Least Squares) based on biases. There are user biases (bias due to human behavior) and movie bias (bias that may result from some attributes of the movie). There could be other biases due to the time the movie was released (holidays vs non-holidays), the geographic area in which the movie is released, etc. These biases can affect how many ratings a movie may get and how high or low the ratings are.

The predictive model, taken from the Data Science course textbook, is based on user and movie biases and is shown below:

$\hat{y} = u + b_i + b_u + \epsilon$

where u is the overall mean rating (in this case from the training dataset), bi is the movie bias, b_u is the user bias, and ϵ is the independent irreducible error.

The movie bias b_i, for each movie, is computed by subtracting the overall mean rating u from y (the observed rating for each movie), $b_i = y - u$ and adding them together.

The user bias is computed by subtracting the movie average u and movie bias b_i from the rating y for each user, and summing it to obtain the total bias.

Regularization is brought into the equation to reduce the effect of outliers (e.g movies with very few ratings, and users who have rated few movies). The model is then used to predict ratings in the test set.

Evaluating Model Performance

There are several ways to measure the performance of a regression model. In this analysis, I use RMSE as a measure of model fitness. RMSE (residual mean square error) is the typical error we make when making a prediction [2].

What does this mean? Given y as the empirical (actual or observed) values of the response, and $\hat{y}$ as the predicted values, how much does $\hat{y}$ deviate from y? If our prediction is very good, then $\hat{y}$ - y, (the residual) will be very small, meaning the prediction and actual values are very close. As the name suggests, it is the mean (average) error we expect to make in our prediction.

RMSE is measured in the same units as the response, in the case of movie rating, an RMSE of 1 means on average we can expect to be off by 1 rating in our prediction.

[2] James, G., Witten, D., Hastie, T., Tibshirani, R., (2017), Introduction of Statistical Learning with Applications in R, "Springer Text in Statistics"

Training the Model

In the regularized model, the main tuning parameter is lambda. This parameter is used to reduce the effect of outliers. In order to avoid overfitting, the edx dataset was partitioned into 90/10 training and test set. The following shows the number of records in the subsequent train and test set.

```
nrow(train_set)
```

```
[1] 8100057
```

```
nrow(test_set)
```

```
[1] 899998
```

Next, the caret createDataPartition() function was used to create 10 sets of train/test partitions, I call these cross validation train/test sets. A model was trained for each of these cross validation train/tests sets (more like manual cross validation) over a given set of lambda values to find an optimized value for lambda.

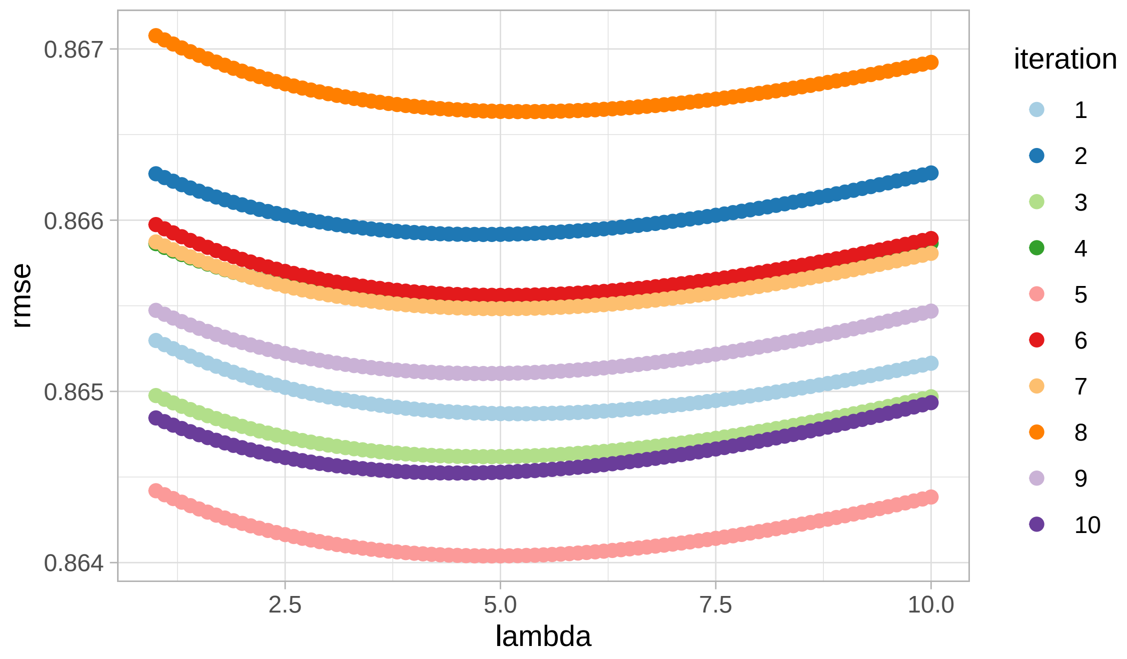
This allowed me to pick the minimum lambda value for each cross validation iteration. The process resulted in ten minimum values for lambda. These values were then averaged to obtain a final optimized lambda value to be used in the final model. The following code shows parts of the code for setting up the cross validation.

```
# perform cross validation.
#
# since the edx dataset is so large we can't use the standard cross validation,
# it won't fit in memory, so here we do it manually.

# 1. create to cross validation train/test sets
CV.FOLDS   <- 10
indexes <- lapply(1:CV.FOLDS, FUN=function(x){fnCreateIndexes(train_set)})

# 2. create a list of lambdas to find best lambda from each cross validation train/test
lambdas <- seq(1, 10, 0.1)

# 3. use sapply to execute the function that perform the entire train/test scenario
(cv_rmse_list <- lapply(indexes, fnTrainAndTest, dataset=train_set, lambdas=lambdas))
```

The following is a list of rmses returned from two of the cross validation iterations.

```
[[1]]
 [1] 0.8652973 0.8652730 0.8652496 0.8652273 0.8652058 0.8651853 0.8651657
 [8] 0.8651469 0.8651289 0.8651116 0.8650952 0.8650794 0.8650644 0.8650501
[15] 0.8650365 0.8650235 0.8650111 0.8649994 0.8649882 0.8649777 0.8649677
[22] 0.8649583 0.8649495 0.8649411 0.8649333 0.8649260 0.8649192 0.8649129
[29] 0.8649070 0.8649017 0.8648967 0.8648922 0.8648882 0.8648845 0.8648813
[36] 0.8648785 0.8648760 0.8648740 0.8648723 0.8648711 0.8648701 0.8648696
[43] 0.8648694 0.8648695 0.8648699 0.8648707 0.8648718 0.8648733 0.8648750
[50] 0.8648771 0.8648794 0.8648820 0.8648849 0.8648881 0.8648916 0.8648954
[57] 0.8648994 0.8649037 0.8649082 0.8649130 0.8649180 0.8649233 0.8649288
[64] 0.8649345 0.8649405 0.8649467 0.8649531 0.8649597 0.8649666 0.8649736
[71] 0.8649809 0.8649883 0.8649960 0.8650039 0.8650119 0.8650202 0.8650286
```

```
[78] 0.8650372 0.8650460 0.8650550 0.8650641 0.8650734 0.8650829 0.8650925
[85] 0.8651023 0.8651123 0.8651224 0.8651327 0.8651431 0.8651537 0.8651644

[[2]]
 [1] 0.8662710 0.8662488 0.8662276 0.8662074 0.8661880 0.8661696 0.8661521
 [8] 0.8661353 0.8661194 0.8661042 0.8660897 0.8660760 0.8660630 0.8660506
[15] 0.8660390 0.8660279 0.8660175 0.8660076 0.8659984 0.8659897 0.8659815
[22] 0.8659740 0.8659669 0.8659604 0.8659543 0.8659488 0.8659437 0.8659391
[29] 0.8659349 0.8659312 0.8659280 0.8659251 0.8659227 0.8659207 0.8659191
[36] 0.8659179 0.8659170 0.8659166 0.8659165 0.8659167 0.8659173 0.8659183
[43] 0.8659196 0.8659212 0.8659231 0.8659254 0.8659280 0.8659309 0.8659340
[50] 0.8659375 0.8659413 0.8659453 0.8659496 0.8659542 0.8659590 0.8659642
[57] 0.8659695 0.8659751 0.8659810 0.8659871 0.8659935 0.8660000 0.8660068
[64] 0.8660139 0.8660211 0.8660286 0.8660363 0.8660442 0.8660523 0.8660606
[71] 0.8660691 0.8660778 0.8660866 0.8660957 0.8661050 0.8661144 0.8661240
[78] 0.8661338 0.8661438 0.8661539 0.8661642 0.8661747 0.8661853 0.8661961
[85] 0.8662070 0.8662181 0.8662294 0.8662407 0.8662523 0.8662640 0.8662758
```



The ten best lambdas from the cross validation are:

```
 [1] 5.2 4.8 4.8 4.8 4.9 5.0 5.0 5.3 4.8 4.5
```

The average of the ten best lambdas is `4.91`. With this lambda, the rmse on the test set is `4.91`.

The Final Model & Results

A final model was fitted on the entire edx dataset to create the final user and movies biases using the optimized lambda of `4.91`. The following tables show samples of the final user and movie biases.

These biases (effects) were then applied to the validation set to make final predictions. The final rmse on the validation is `0.8648184`.

Table 8: Final user effects

userId	b_u
1	1.3342181
2	-0.1834749
3	0.2286596
4	0.5718031
5	0.0803612
6	0.3069534
7	0.0227535
8	0.1993029
9	0.1885776
10	0.0803042
11	0.6166684
12	0.1125889
13	0.0141983
14	-0.1026846
16	0.2114713

Table 9: Final movie effects

movieId	b_i
1	0.4150868
2	-0.3069260
3	-0.3652265
4	-0.6461541
5	-0.4434531
6	0.3026987
7	-0.1537719
8	-0.3756268
9	-0.5135532
10	-0.0866125
11	0.1608593
12	-0.9473862
13	-0.3252787
14	-0.0691912
15	-0.7885403

Conclusion

In this analysis, I trained a model described in our course textbook by applying cross validation to get best best lambda. I was able to get a good rmse, `0.8648184` without overfitting. Even though the model performance is pretty good, improvement could be achieved by including the genres, rating timestamp, and release dates as features in the prediction. Also other methods like PCA and SVD could be used to add more residuals to the model to improve it.

TECHNICAL INFORMATION

Executing the Scripts

Note:
This project was developed and tested in Windows. It has not been tested under any other operating system.

The easiest way to execute the script is to clone the repo from github and use RStudio. When you clone the repo, you will get the entire folder structure needed to execute the scripts.

Cloning from github
1. git clone https://github.com/kowusu01/KOwusu.Tieku.HarvardX.Capstone.Movielens
2. go to RStudio and navigate to the project folder, open the project in RStudio.
3. open the movielens.R script
4. select all and Run

Executing the scripts attached to the submission
If you decide not to clone the repo and instead just to run the files attached in the submission, follow the steps:
1. download the attached files to your preferred location
2. in the same location where you copied the files, create a folder called rda, the script requires it
3. go to RStudio and navigate to your folder
4. open the movielens.R script
5. select all and Run

Executing the report
The report relies on objects created from the main script. You must first run the main script before executing the report.

REFERENCES

1. Irizarry, Rafael A., (2021), Introduction to Data Science Data Analysis and Prediction Algorithms with R,
   URL: https://rafalab.github.io/dsbook/

2. James, G., Witten, D., Hastie, T., Tibshirani, R., (2017), Introduction of Statistical Learning with Applications in R, "Springer Text in Statistics".

3. Marwick, B., Boettiger, C. & Mullen, L., (2017), Packaging Data Analytical Work Reproducibly Using R (and Friends),
   URL: https://www.researchgate.net/publication/345666324_Packaging_data_analytical_work_reproducibly_using_R_and_friends

4. Nwanganga, F & Chapple, Mike, (2020), Practical Machine Learning in R, "Wiley".

5. Zhu, Hao. (2020), Create Awesome LaTeX Table with knitr::kable and kableExtra,
   URL: https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_pdf.pdf