MovieLens - Movie Recommendation System - Project Report

Sam Seatt

6/09/2019

Contents

About this Project

Project GitHub Repository: https://github.com/samseatt/movielens-mrs

To remain consistent with the project-inception artifacts, I have used R version 3.5.2 (2018-12-20) for developing and running my scripts.

The Readme file (https://raw.githubusercontent.com/samseatt/movielens-mrs/master/Readme.txt) in the GithHub repository describes how to set up this project (if you are interested beyond the artifact attached with this submission). Basically you can download my RStudio project file in the GitHub repository, or you may simply Run model.R file or Knit movielens-project-report.Rmd file from any RStudio project, provided you have some basic R libraries installed.

An errata file may also be posted in that repository in case I discover any errors post-submission.

In the interest of execution time of these scripts, I have also saved the curated training and validation sets online in an AWS database as RData objects (edx.rda and validation.rda):

- https://adaprise-01.s3.amazonaws.com/edx/edx.rda

- https://adaprise-01.s3.amazonaws.com/edx/validation.rda

The scripts then load the objects listed above into the local environment.

You will need to make appropriate edits in the code depending on the option you want to run and where you want to get the dataset from.

Before committing to run these artifacts, be aware that the data download (over the internet, in this case) and several operations on that full dataset (including some transformations, plots and model training) take significant time. The entire end-to-end script (either when you run model.R in its entirety, or Knit the full movielens-mrs.Rmd) takes about 10 minutes to run on my Macbook Pro laptop.

Executive Summary

The goal of this project is to develop a well-performing movie recommendation system. The main objective of such a system is to use collected/generated data (subset of MovieLens dataset, in this case) to train a model that will then predict a list of top movies that a given user will like to watch.

In addition to developing the machine learning models, data is analyzed, explored and visualized in order gain several insights. A copy of the dataset was then further curated to aid in further visualizations, and to more conveniently exploit (in the future) the insights learned from this data.

The MovieLens solution is an example of a recommender system (or recommendation system). These systems seek to predict the rating or preference a user would give to an item. (source: Recommender Systems Handbook, Springer, 2011, pages 1-35)

Specifically, in this case, in content-based recommendation systems, users and items (movies in this case, or items to purchase in other classic examples) have profiles that describe their characteristics. The system then recommends the items whose profile matches, or best matches, the user's profile. (source: towardsdatascienc.com)

The Netflix Challenge

According to a wired magazine article ("This is how Netflix's top-secret recommendation system works"), "more than 80 per cent of the TV shows people watch on Netflix [around 2017] are discovered through the platform's recommendation system. This project attempts to retrace some of the actions taken when the contestants and engineers developed that system. The objective is to achieve an RMSE <= 0.87750 using the 10M version of the MovieLens data provided by the course staff.

In real life recommendation systems are ubiquitous, especially on the internet and they are not limited to movie and product recommendation.

Goals and Objectives

The objective of my (trained) models is to predict the rating, given a movie (represented by a movie ID) and a user (represented by a user ID). More specifically, at prediction time the model will take a movie ID and a user ID, and predict what rating this particular user will give to the given move.

The basic underlying strategy boils down to filling in the ratings for the unfilled slots all the movie/user combinations, based on the small subset of previously filled spots (by actual users).

The end goal is then to use pattern/model i.e. trained slots to filter the best rated movies for a user and recommend them to her.

The metric, or performance used here is RMSE (the root mean squared error, predicted on the previously allocated validation dataset).

Process, and Model Selection

The training approach or strategy is to not use linear regression due to the large dataset and its combinatorial vastness. Instead the approach taken (using mean rating and "effects") is developed to practically achieve the same effect as linear regression.

I start with a random baseline model where the model always gives an average rating. I then add movie effects to my model. This is followed by adding the user effect, and both user and movie effects. Then I add regularization. Regularization allows me to This incremental approach will also help me understand the contribution of each of these effects as well as that of regularization, including the specific hyperparameters tried (lambda in this case) to achieve best results.

Prior to the modeling exercise I explored and visualized the data I later use in training. The hope is that this allows me and the reader to get an intuition of the independent variables (including the movie itself, the user desiring a rating, the time a rating is given, and the genre(s) the movei belongs to) and their effects on the final rating a movie receives from a user. This exploration also brings some insights on what other independent effects have the potential to further improve the RMSE and thus provide even more valuable movie suggestions. We need to be cognizant though about which parameters to use, and which we simply ignore, in order to avoid overtraining the model and to not unnecessarily slow its training (and potentially even the prediction-time) performance.

Finally, as mentioned, established techniques like linear regression are not applied here directly due to the relatively large size of dataset and the fact that the raw models will tantamount to predicting every rating for every user/movie combination. Doing so will require computing resources beyond what's available to me locally in the student setting.

The Dataset

This project uses the 10M version of the MovieLens datasets (https://grouplens.org/datasets/movielens/10m/) as specified in the assignment instructions. This dataset "contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens."

This dataset the horizontally split into a training (edx) and a validation (validation) subsets.

Each row in the dataset represents a unique rating assigned to one movie by one user. The rating can thus be considered as the dependent variable as it depends on the user, the movie, and possibly other dependent variables in the system like timestamp and genre

The dataset contains the following columns:

userId: User ID – an ID representing a user, unique for each individual user of the system

movieId: Movie ID – an ID representing a movie, unique for each movie in the system

rating: The rating given to a single movie by a single user. Allowed values are 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 and 5.

timestamp: A system timestamp indicating the date and time the rating was submitted

title: A text string containing the title of the movie and the year it was released

genres: A list of genres the movie belongs to. Genres are a pipe-separated list, and are selected from the following: * Action * Adventure * Animation * Children's * Comedy * Crime * Documentary * Drama * Fantasy * Film-Noir * Horror * Musical * Mystery * Romance * Sci-Fi * Thriller * War * Western

Key Steps Performed

This subsection highlights the key steps performed during this analysis. The output of these steps is furnished in the section that follows.

- 10M MovieLens dataset is loaded, cleaned, partitioned and stored as RData objects in the cloud.

- The data is analyzed in details to understand relationships and help in proper input selection and model selection.
- The models are run with increased complexity, as follows:
  - Baseline model that simply predicts an average rating
  - With movie effect
  - With move and user effects
  - With movie and user effects, and best regularization results selected
  - With movie and user effects, and regularization using handpicked (best) lambda
- The results are then evaluated primarily for their RMSE values
- Final model is selected and presented.
- Recommendations are made for further possible improvements in future iterations of this work.

Methods and Analysis

Process and techniques used here include: * data cleansing * data exploration * data visualization * ingights gained * modeling approach

These are described in following sub-sections in more or less the sequence in which they are applied.

Including the requisite libraries

The project uses tidyverse which is a set of packages that whare common API and design/development principles. Following core packages from the tidyverse ecosystem are used in this project: ggplot2 (for data visualization), dplyr (for data manipulation), tidyr (for data tidying), readr (for data import), and stringr (for strings). In addition to these core packages the following tidyverse packages are also used: lubridate (for date/times). Caret package is used for data loading and machine learning tasks. reshape2 package is used to reshape data when needed. knitr package is used to Knit this document.

Data partitioning and loading

A 10M rows version of the MovieLens dataset is loaded from grouplens.org repository. The data is partitioned into training (edx) and validation data sets. The code needed to do the loading and partition is provided as a prerequisites to this project. This code is saved in my GitHub project at <>. This script completes the following tasks:

- Downloads the zip file: “http://files.grouplens.org/datasets/movielens/ml-10m.zip”
- Unzips the dataset and curates the data into the required columns
- Splits the dataset into training (edx) and validation datasets
- Makes sure that the keys (movieId and userId) are consistent between traiing and validation by ensuring that the users and movies in the validation set are also present in the training set.
- After the load, the training and validation datasets are, respectively, available as ‘edx’ and ‘validation’ R session variables

I used the provided code verbatim to load the data. This keeps things consistent betweem my work and the expectations of those grading it.

Additionally, I also save the two datasets in permanet storage as edx.rda and validation.rda RData (RDA) files. This allows them to be retrieved even after the R session has been lost, e.g. when Knitting this document in PDF.

I use the saved RDA files to retrieve the edx and validation datasets:

```r
edx_path <- "https://adaprise-01.s3.amazonaws.com/edx/edx.rda"
validation_path <- "https://adaprise-01.s3.amazonaws.com/edx/validation.rda"
if (!exists("edx")) {
  load(url(edx_path))
}
```

```
if (!exists("validation")) {
  load(url(validation_path))
}
```

Inspect the loaded dataset

Used head() and str() to check the data columns and their format. This information is useful in determining what data transformation will be required (during the data cleaning step) for proper ML modeling.

The following shows the columns of the training set data frame and the type of data contains within the columns:

```
## '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
```

Shows the distribution within the columns. userId and movieId are ID columns and not numeric values. These can be treated as factors.

Ratings timestamps are computer readable and less suitable for human viewing e.g. in plots. These will be reshaped later during further analysis.

Genres are unstructured. They will also be tidied down the road for additional analysis of any dependence while predicting movie ratings: one may have reason to think that users are likely to rate their favorite genras higher.

Movies and users

```
##   n_users n_movies
## 1   69878    10677
```

The training dataset contains 9000055 rows i.e. 9000055 individual ratings by 69878 unique users for 10677 unique movies.

Define common function

The most important function is the RMSE calculaton function: RMSE(). This function is again copied here verbatim from the formula provided in the class for use with this assignment. This to ensure the RMSE is calculated according to the rules of this project.

This simple function takes two arrays of equal size actual or true ratings, and the ratings we have predicted, for each rating of the set we want to evaluate (in other words, our validation set). The method then calculateds the root mean squared error (RMSE) in the standard way.

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

Data analysis and explorative visualizations

Check data distributions, especially ratings stats and trends

First find all the possible ratings that can be given (for this large data this is same as all possible ratings that the users in the datasets have given):

```r
sort(unique(edx$rating))
```

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

If we assign ratings randomly (from all the possible ratings available to give: i.e. 0.5 to 5 in 0.5 increments, with 0 or anyting not divisible by half not allowed values), then this is the average and median respectively that will be given:

```r
summary(unique(edx$rating))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.500   1.625   2.750   2.750   3.875   5.000
```

See the the basic stats of the ratings column:

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

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

Comparing the above two summaries tell us that movies are getting higher ratings on the average in this dataset compared to the ratings a random, evenly-weighted process will assign.

Clearly, people are rating movies higher than random system using central limit theorem around values normlized around possible outcomes (rather than achieved outcomes i.e. actual ratings). This most likely appears to be human psyhcology, but let's look for any trends, causes and confounders that may be present - for example, if good movies get rated more or seen more than average or bad movies, then we may want to know that.

So, first check for trends and biases in the data, like movie distribution bias e.g. if good movies are simply getting more ratings because they are being watched more. Or if users feel too good about a movie right after watching it - i.e. moves are made to entertain us i.e. make us feel good to begin with; or if users tend to give good ratings in order to feel good while tending to only give bad rating when they are really unhappy with the movie.

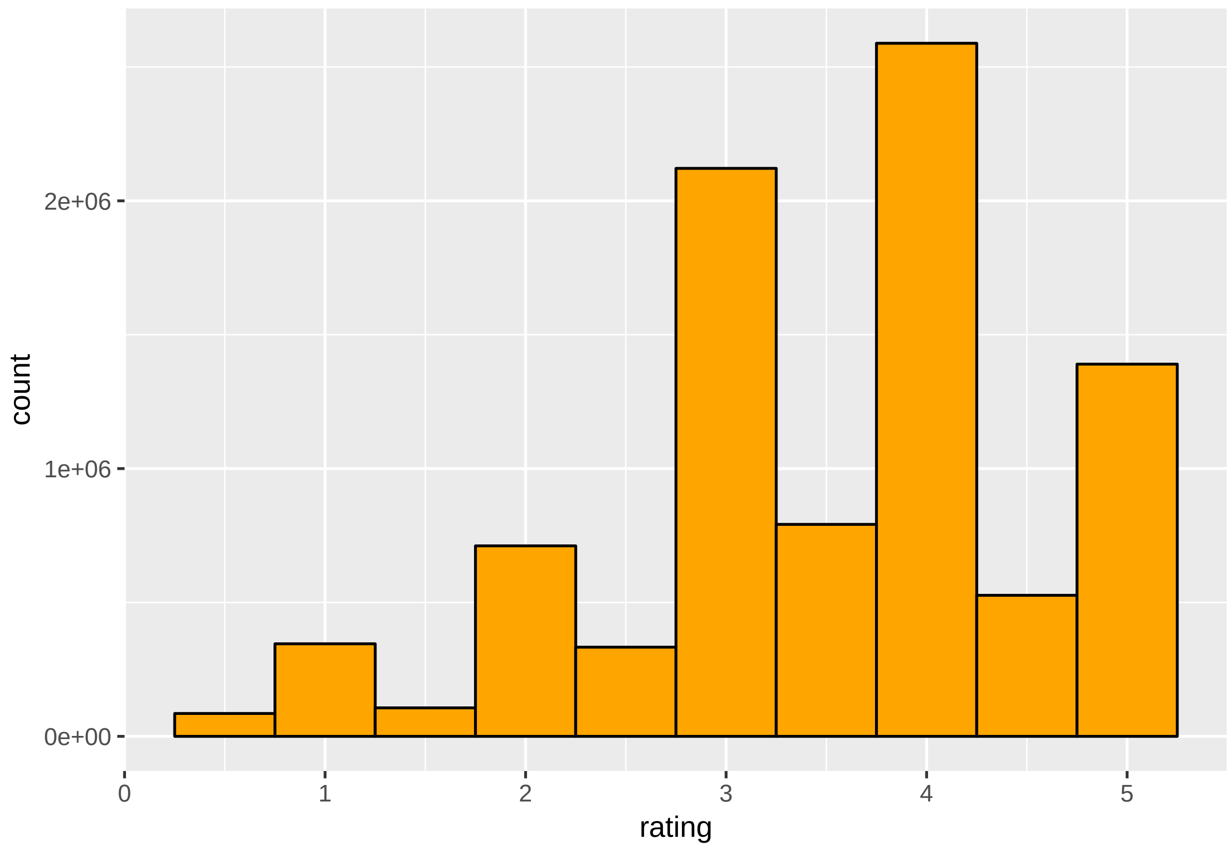
Here bias is being analyzed for the purpose of understanding the data trends; however these two biases are closely related to the two biases ('movie effect' and 'user effect') that will be used below during the machine learning model development exercise.

We all have witnessed the human-nature phenomenon online where ratings (perticularly product ratings) somehow tend to be either too good or too bad, and proportionally less in the middle. Let's confirm this trend as well:

This may not be a very useful exercise for model-development purpose, but this extra intuition around the data can be helpful when interpreting results.

The following histogram helps us to see the distribution of each possible rating given to movies.

```r
edx %>% ggplot(aes(x=rating)) +
  geom_histogram(color="black", fill="orange", binwidth=0.5)
```



It shows that not only the ratings tend to be higher i.e. not centered at around 2.75, but also (as noted in the Machine Learning module) whole number ratings are much more common than half-point ratings (0.5, 1.5, 2.5, 3.5 and 4.5).

Stats related to movies and users in the dataset

Show the average rating and rating frequency of each specifid movie.

- lowest rated movies:

```
edx %>% group_by(movieId) %>%
  summarize(title = first(title), average = mean(rating), count = n()) %>%
  arrange(average)
```

```
## # A tibble: 10,677 x 4
##    movieId title                                   average count
##      <dbl> <chr>                                     <dbl> <int>
##  1    5805 Besotted (2001)                           0.5       2
##  2    8394 Hi-Line, The (1999)                       0.5       1
##  3   61768 Accused (Anklaget) (2005)                 0.5       1
##  4   63828 Confessions of a Superhero (2007)         0.5       1
##  5   64999 War of the Worlds 2: The Next Wave (2008) 0.5       2
##  6    8859 SuperBabies: Baby Geniuses 2 (2004)       0.795    56
##  7    7282 Hip Hop Witch, Da (2000)                  0.821    14
##  8   61348 Disaster Movie (2008)                     0.859    32
##  9    6483 From Justin to Kelly (2003)               0.902   199
## 10     604 Criminals (1996)                          1         2
## # ... with 10,667 more rows
```

- highest rated movies:

```
edx %>% group_by(movieId) %>%
  summarize(title = first(title), average = mean(rating), count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 4
##    movieId title                                              average count
##      <dbl> <chr>                                                <dbl> <int>
##  1     296 Pulp Fiction (1994)                                   4.15 31362
##  2     356 Forrest Gump (1994)                                   4.01 31079
##  3     593 Silence of the Lambs, The (1991)                      4.20 30382
##  4     480 Jurassic Park (1993)                                  3.66 29360
##  5     318 Shawshank Redemption, The (1994)                      4.46 28015
##  6     110 Braveheart (1995)                                     4.08 26212
##  7     457 Fugitive, The (1993)                                  4.01 25998
##  8     589 Terminator 2: Judgment Day (1991)                     3.93 25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star W~    4.22 25672
## 10     150 Apollo 13 (1995)                                      3.89 24284
## # ... with 10,667 more rows
```

- movies rated the least number of times (among those that were rated at least once, since otherwise they won’t be in our dataset):

```
edx %>% group_by(movieId) %>%
  summarize(title = first(title), average = mean(rating), count = n()) %>%
  arrange(count)
```

```
## # A tibble: 10,677 x 4
##    movieId title                                              average count
##      <dbl> <chr>                                                <dbl> <int>
##  1    3191 Quarry, The (1998)                                     3.5     1
##  2    3226 Hellhounds on My Trail (1999)                          5       1
##  3    3234 Train Ride to Hollywood (1978)                         3       1
##  4    3356 Condo Painting (2000)                                  3       1
##  5    3383 Big Fella (1937)                                       3       1
##  6    3561 Stacy's Knights (1982)                                 1       1
##  7    3583 Black Tights (1-2-3-4 ou Les Collants noirs) (196~     3       1
##  8    4071 Dog Run (1996)                                         1       1
##  9    4075 Monkey's Tale, A (Les Château des singes) (1999)       1       1
## 10    4820 Won't Anybody Listen? (2000)                           2       1
## # ... with 10,667 more rows
```

- movies rated the most number of times:

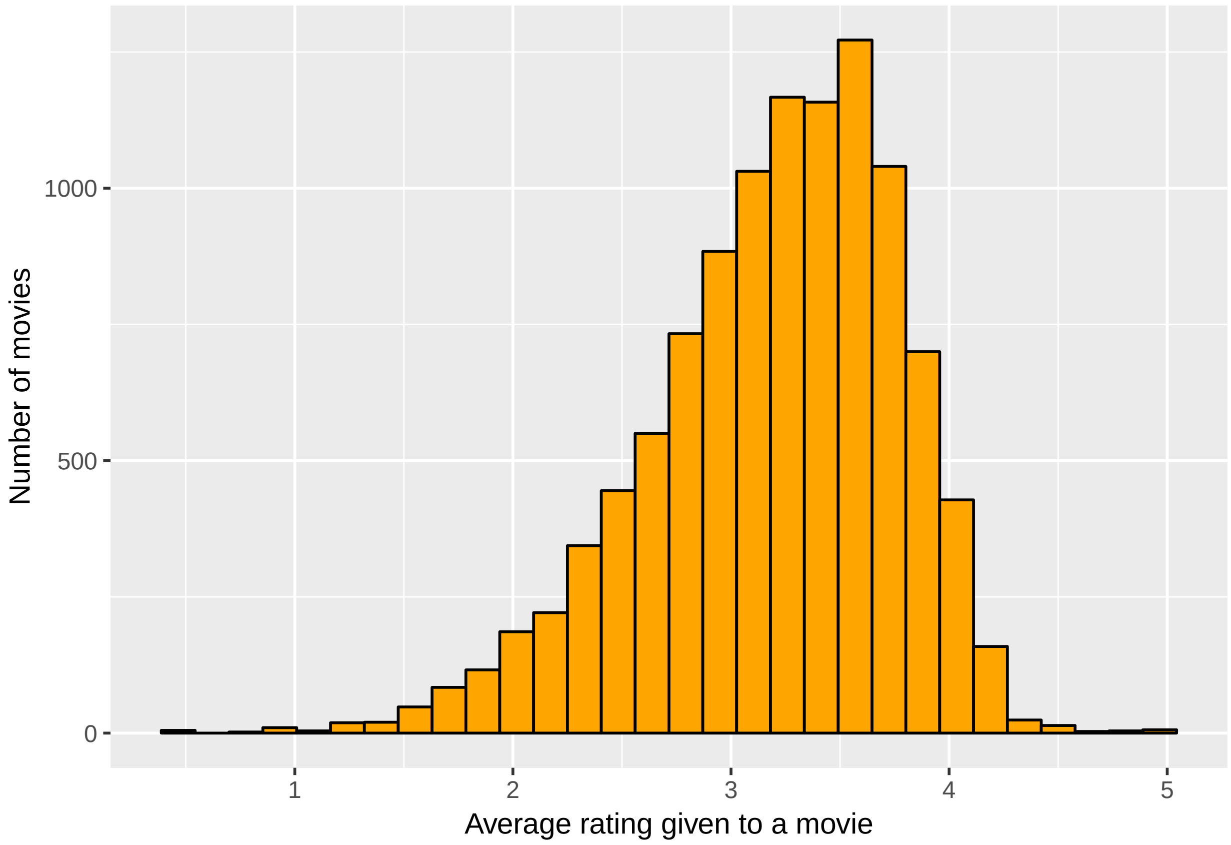
```
edx %>% group_by(movieId) %>%
  summarize(title = first(title), average = mean(rating), count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 4
##    movieId title                                              average count
##      <dbl> <chr>                                                <dbl> <int>
##  1     296 Pulp Fiction (1994)                                   4.15 31362
##  2     356 Forrest Gump (1994)                                   4.01 31079
##  3     593 Silence of the Lambs, The (1991)                      4.20 30382
##  4     480 Jurassic Park (1993)                                  3.66 29360
##  5     318 Shawshank Redemption, The (1994)                      4.46 28015
##  6     110 Braveheart (1995)                                  4.08 26212
##  7     457 Fugitive, The (1993)                               4.01 25998
##  8     589 Terminator 2: Judgment Day (1991)                  3.93 25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star W~    4.22 25672
## 10     150 Apollo 13 (1995)                                   3.89 24284
## # ... with 10,667 more rows
```

Movie and user effects in the dataset

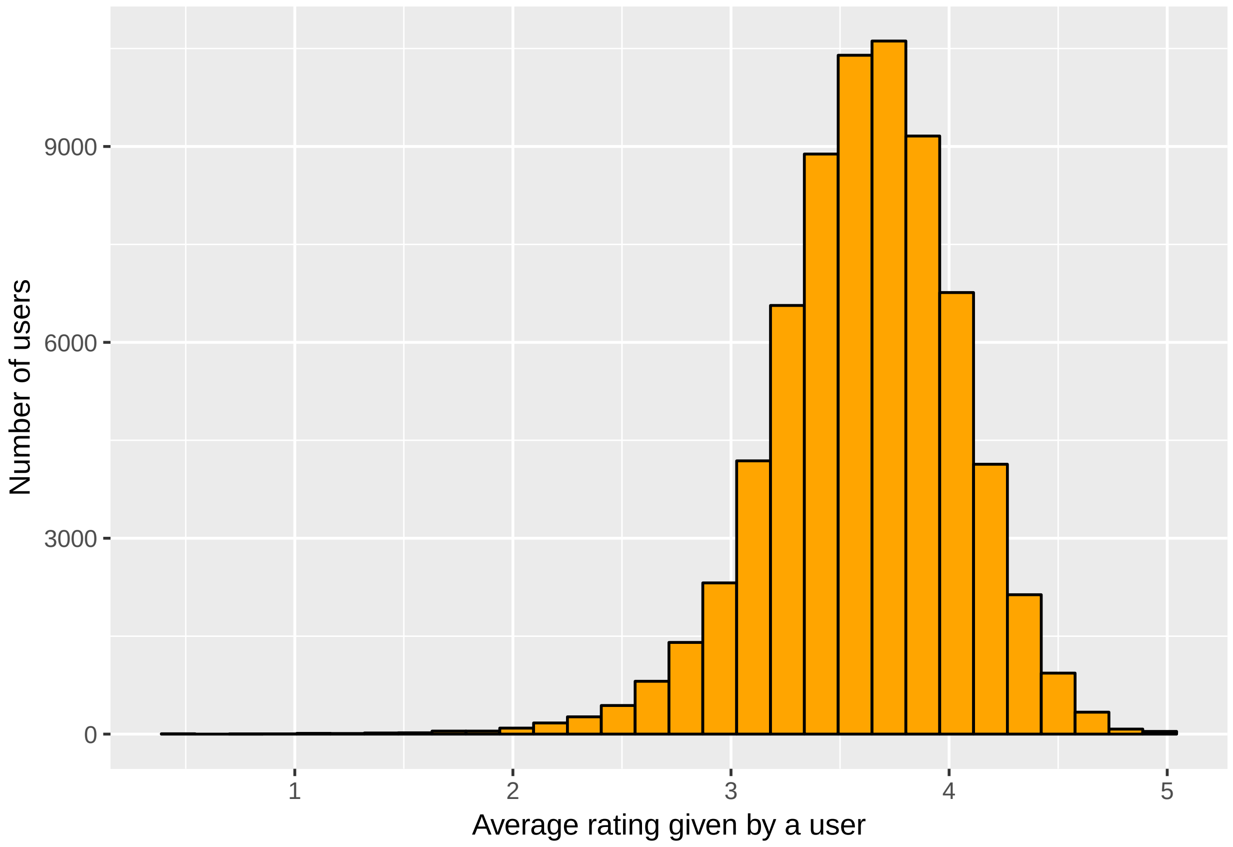
The following graph shows the distribution of movies that receive a particular rating. The distribution is centered close to the rating of 3.5. The histogram tells us that all movies are not rated similarly even by an average user.

```
edx %>% group_by(movieId) %>%
  summarize(title = first(title), average = mean(rating), count = n()) %>%
  arrange(average) %>%
  ggplot(aes(x=average)) +
  geom_histogram(color="black", fill="orange") +
  xlab("Average rating given to a movie") +
  ylab("Number of movies")
```



Similarly users are not equal when it comes to prividing ratings - some users being more stingy overall, than other users.

```
edx %>% group_by(userId) %>%
  summarize(user = first(userId), average = mean(rating), count = n()) %>%
  arrange(average) %>%
  ggplot(aes(x=average)) +
  geom_histogram(color="black", fill="orange") +
  xlab("Average rating given by a user") +
  ylab("Number of users")
```



The above two insights will later be exploited when building the ML model using movie effects and user effects, respectively.

We have now built some intuition around the following characteristics of the domain in which we will run our models:

- Not all movies get watched by equal frequendies.
- Not all movies have similar ratings
- Not all the users rate movies with equal frequencies….
- Not all the users rate movies the same.

Insights around data generation and data collection in general

How sparse are the ratings

Of course, not all movies get rated. In fact each user can watch only a small fraction of all the movies available on NetFlix get watched by each user, and a fraction of movies watched gets rated, and not all users rate a lot of movies (as seen above).

Percentage of ratings given (out of all ratings possible in this dataset)

```
nrow(edx) / (length(unique(edx$movieId)) * length(unique(edx$userId)))
```

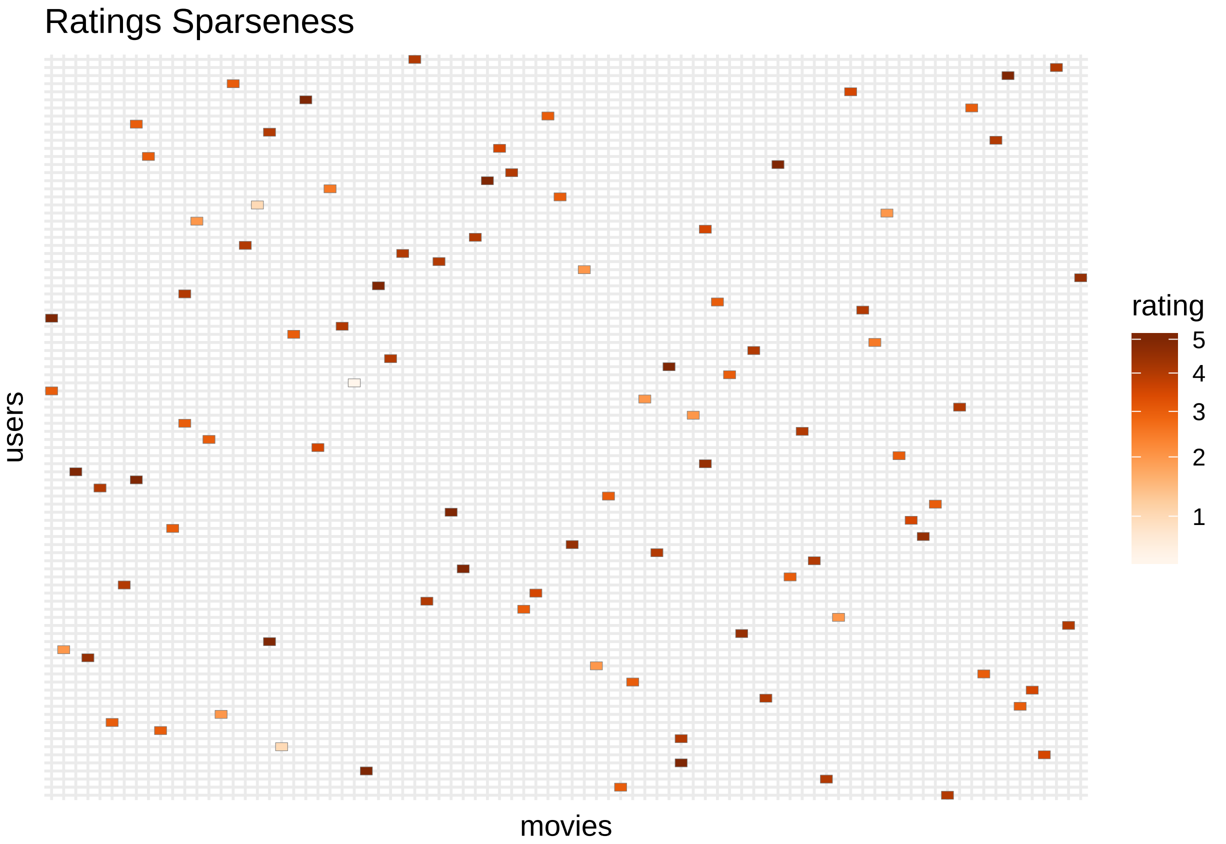
```
## [1] 0.012063
```

Before we explore this statistic, first let's create a smaller dataset to allow faster plotting or broader trends. The following dataset (edx_lite) is a random slice 1/10000th of the edx dataset.

```
set.seed(2)
edx_lite_index <- createDataPartition(y = edx$rating, times = 1, p = 0.00001, list = FALSE)
edx_lite <- edx[edx_lite_index,]
```

Now use this dataset to visualize the sparseness. The empty grid cells represent unrated combinations while in filled grid cells with the color gradient corresponding to the rating value itself:

```
edx_lite %>% ggplot(aes(as.factor(movieId), as.factor(userId), fill = rating)) +
  geom_tile(color = "grey50") +
  scale_fill_gradientn(colors = RColorBrewer::brewer.pal(9, "Oranges"), trans = "sqrt") +
  theme_minimal() +  theme(axis.text.x = element_blank(),
                           axis.text.y = element_blank()) +
  ggtitle("Ratings Sparseness") +
  ylab("users") +
  xlab("movies")
```



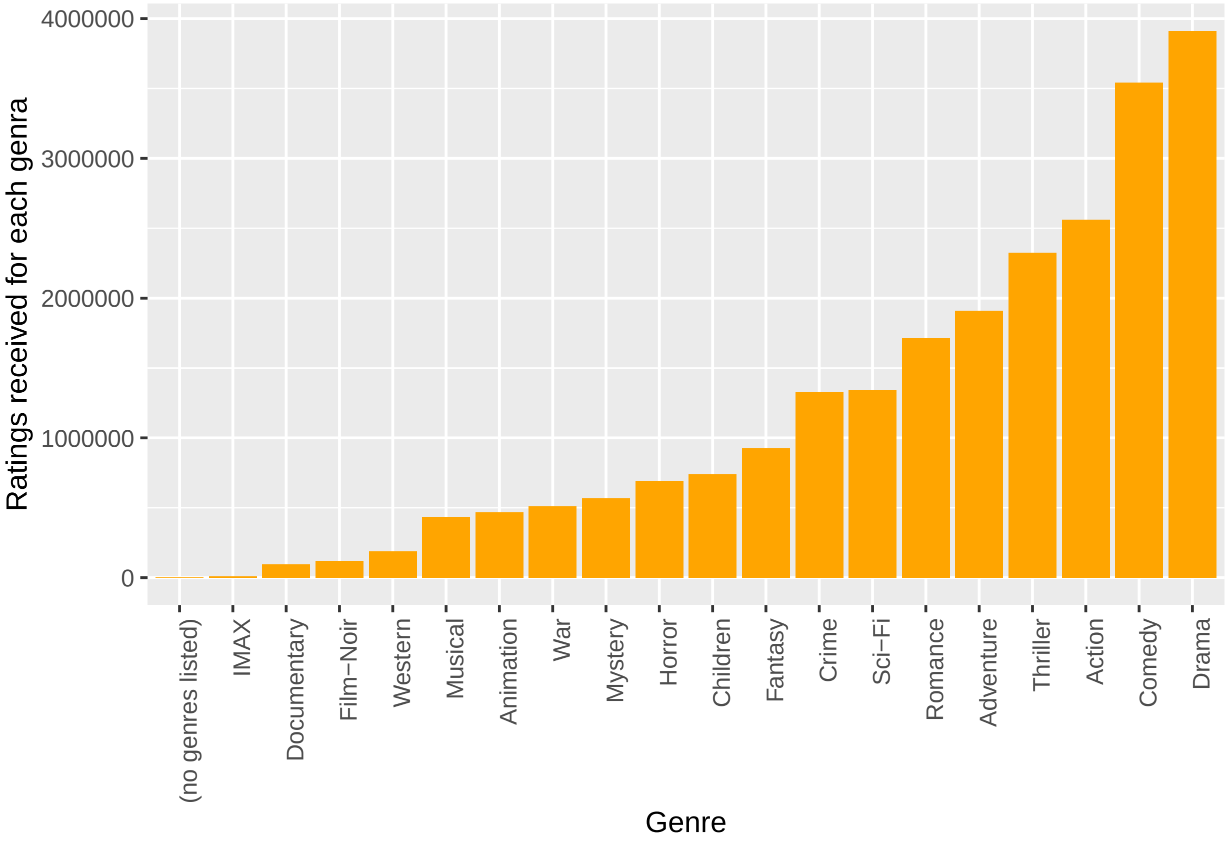
The above stat and plot show that the ratings are fairly sparse with about 1.2 % (percent) of all movies rated. But it is not as sparse as one can can think i.e. a single average user cannot watch 1% of all the movies in the system, let alone go ahead and then rate each one of them. This, however, makes more sense when we the fact that the database only contains (a) just the movies that have been rated (that is, that were at least watched during the data collection period, and (b) relatively less consequentially, only those users are listed who have rated at least one movie. We need to be aware of this characteristic/limitation of the dataset when we use it to describe the world.

Exploiting generas, dates and any other information from the dataset

Distribution of movie genres

```
by_genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>% summarize(count = n())

options(scipen=10000)
as_data_frame(by_genres) %>%
  mutate(genres = reorder(genres, count)) %>%
  ggplot(aes(x=genres, y=count)) +
  geom_bar(position="dodge", stat="identity", fill="orange") +
  labs(x="Genre", y="Ratings received for each genra") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Additional data reshaping

In order to be able to exploit and analyze these other parameters, first I further reshape the dataset, without affecting the original rows and columns. The modified data will not be used for the competition (staying with the ground rules for this assignment, only the originally provided dataset will be used in training and RMSE calculation purposes).

In the new dataset I expand the genre information into a tidy format - each column represented the boolean presence or absence of a specific genra in a given movie.

I also expand the dates in order to explore time-related parameters (e.g. how old a movie is when it is rated) for the purposes of improving future models. To achieve this I use stringr string manipulaton extract the release year from the title and convert it into the date format. I also convert the rating timestamp to a date.

These two dates allow me to calculate how old a movie is when it is rated, and consequently, how age of a movie affects its rating, or more importantly how this insight/training can be used to recommend the right movies for a given user.

The new dataset will be called edx_x (edx - expanded).

I then similarly expand the validation dataset to validation_x

Finally, for faster plotting (in some plotting scenarios), I will also create a reduced rows version of this new dataset and call it edx_x_lite.

```
edx_x <- mutate(edx, time = as.Date.POSIXct(timestamp),
                release_time = as.Date(ISOdate(gsub("\\).*", "", gsub(".*\\(", "", title)), 1, 1, 0, 0,
                action = grepl("Action", genres),
                adventure = grepl("Adventure", genres),
                animation = grepl("Animation", genres),
                children = grepl("Children", genres),
                comedy = grepl("Comedy", genres),
                crime = grepl("Crime", genres),
                documentary = grepl("Documentary", genres),
                drama = grepl("Drama", genres),
                fantasy = grepl("Fantasy", genres),
                film_noir = grepl("Film-Noir", genres),
                horror = grepl("Horror", genres),
                imax = grepl("IMAX", genres),
                musical = grepl("Musical", genres),
                mystery = grepl("Mystery", genres),
                romance = grepl("Romance", genres),
                sci_fi = grepl("Sci-Fi", genres),
                thriller = grepl("Thriller", genres),
                war = grepl("War", genres),
                western = grepl("Western", genres)
)

# Similarly further curate and expand validation
validation_x <- mutate(validation, time = as.Date.POSIXct(timestamp),
                release_time = as.Date(ISOdate(gsub("\\).*", "", gsub(".*\\(", "", title)), 1, 1, 0, 0,
                action = grepl("Action", genres),
                adventure = grepl("Adventure", genres),
                animation = grepl("Animation", genres),
                children = grepl("Children", genres),
                comedy = grepl("Comedy", genres),
                crime = grepl("Crime", genres),
                documentary = grepl("Documentary", genres),
                drama = grepl("Drama", genres),
                fantasy = grepl("Fantasy", genres),
                film_noir = grepl("Film-Noir", genres),
                horror = grepl("Horror", genres),
                imax = grepl("IMAX", genres),
                musical = grepl("Musical", genres),
                mystery = grepl("Mystery", genres),
                romance = grepl("Romance", genres),
                sci_fi = grepl("Sci-Fi", genres),
                thriller = grepl("Thriller", genres),
                war = grepl("War", genres),
                western = grepl("Western", genres)
)



str(edx_x)
```

```
## 'data.frame':    9000055 obs. of  27 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
##  $ time        : Date, format: "1996-08-02" "1996-08-02" ...
##  $ release_time: Date, format: "1992-01-01" "1995-01-01" ...
##  $ action      : logi  FALSE TRUE TRUE TRUE TRUE FALSE ...
##  $ adventure   : logi  FALSE FALSE FALSE TRUE TRUE FALSE ...
##  $ animation   : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ children    : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
##  $ comedy      : logi  TRUE FALSE FALSE FALSE FALSE TRUE ...
##  $ crime       : logi  FALSE TRUE FALSE FALSE FALSE FALSE ...
##  $ documentary : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ drama       : logi  FALSE FALSE TRUE FALSE TRUE FALSE ...
##  $ fantasy     : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
##  $ film_noir   : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ horror      : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ imax        : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ musical     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ mystery     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ romance     : logi  TRUE FALSE FALSE FALSE FALSE FALSE ...
##  $ sci_fi      : logi  FALSE FALSE TRUE TRUE TRUE FALSE ...
##  $ thriller    : logi  FALSE TRUE TRUE FALSE FALSE FALSE ...
##  $ war         : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ western     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
summary(edx_x)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 789652009
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 946768283
##  Median :35738   Median : 1834   Median :4.000   Median :1035493918
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1032615907
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1126750881
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1231131736
##     title              genres               time
##  Length:9000055     Length:9000055     Min.   :1995-01-09
##  Class :character   Class :character   1st Qu.:2000-01-01
##  Mode  :character   Mode  :character   Median :2002-10-24
##                                        Mean   :2002-09-21
##                                        3rd Qu.:2005-09-15
##                                        Max.   :2009-01-05
##   release_time          action        adventure       animation
##  Min.   :1915-01-01   Mode :logical   Mode :logical   Mode :logical
##  1st Qu.:1987-01-01   FALSE:6439510   FALSE:7091163   FALSE:8532887
##  Median :1994-01-01   TRUE :2560545   TRUE :1908892   TRUE :467168
##  Mean   :1990-03-22
##  3rd Qu.:1998-01-01
##  Max.   :2008-01-01
##   children        comedy          crime         documentary
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:8262061   FALSE:5459125   FALSE:7672340   FALSE:8906989
##  TRUE :737994    TRUE :3540930   TRUE :1327715   TRUE :93066
##
##
##
##    drama          fantasy        film_noir        horror
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:5089928   FALSE:8074418   FALSE:8881514   FALSE:8308570
##  TRUE :3910127   TRUE :925637    TRUE :118541    TRUE :691485
##
##
##
##     imax          musical         mystery         romance
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:8991874   FALSE:8566975   FALSE:8431723   FALSE:7287955
##  TRUE :8181      TRUE :433080    TRUE :568332    TRUE :1712100
##
##
##
##    sci_fi         thriller          war           western
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:7658872   FALSE:6674156   FALSE:8488908   FALSE:8810661
##  TRUE :1341183   TRUE :2325899   TRUE :511147    TRUE :189394
##
##
##
```

```
summary(validation_x)
```

```
##       userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 789652009
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 946673613
##  Median :35768   Median : 1827   Median :4.000   Median :1035246862
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1032520380
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1126641056
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1231131028
##     title              genres               time
##  Length:999999      Length:999999      Min.   :1995-01-09
##  Class :character   Class :character   1st Qu.:1999-12-31
##  Mode  :character   Mode  :character   Median :2002-10-22
##                                        Mean   :2002-09-19
##                                        3rd Qu.:2005-09-13
##                                        Max.   :2009-01-05
##   release_time          action        adventure       animation
##  Min.   :1915-01-01   Mode :logical   Mode :logical   Mode :logical
##  1st Qu.:1987-01-01   FALSE:715195    FALSE:787817    FALSE:948055
##  Median :1994-01-01   TRUE :284804    TRUE :212182    TRUE :51944
##  Mean   :1990-03-15
##  3rd Qu.:1998-01-01
##  Max.   :2008-01-01
##   children       comedy            crime         documentary
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:917844    FALSE:606861    FALSE:852757    FALSE:989611
##  TRUE :82155     TRUE :393138    TRUE :147242    TRUE :10388
##
##
##
##    drama           fantasy        film_noir         horror
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:565928    FALSE:897154    FALSE:986948    FALSE:923259
##  TRUE :434071    TRUE :102845    TRUE :13051     TRUE :76740
##
##
##
##     imax           musical         mystery         romance
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:999100    FALSE:951905    FALSE:937387    FALSE:810216
##  TRUE :899       TRUE :48094     TRUE :62612     TRUE :189783
##
##
##
##    sci_fi          thriller          war           western
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:850693    FALSE:741463    FALSE:943083    FALSE:978934
##  TRUE :149306    TRUE :258536    TRUE :56916     TRUE :21065
##
##
##
```

```r
set.seed(2)
edx_x_lite_index <- createDataPartition(y = edx_x$rating, times = 1, p = 0.00001, list = FALSE)
edx_x_lite <- edx[edx_x_lite_index,]

summary(edx_x_lite)
```

```
##       userId         movieId            rating        timestamp
##  Min.   :  543   Min.   :  110.0   Min.   :0.500   Min.   : 829936108
##  1st Qu.:16999   1st Qu.:  474.2   1st Qu.:3.000   1st Qu.: 945054252
##  Median :33094   Median : 1604.0   Median :3.750   Median :1009110917
##  Mean   :36254   Mean   : 2824.3   Mean   :3.549   Mean   :1023856753
##  3rd Qu.:57230   3rd Qu.: 3283.8   3rd Qu.:4.000   3rd Qu.:1112327646
##  Max.   :70819   Max.   :39292.0   Max.   :5.000   Max.   :1225575076
##     title              genres
##  Length:92          Length:92
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

With these additional / spread out columns, now I can calculate the time lapse between the release of a movie and the day the rating was received. This information parameter can then potentially be fed into an ML model to account for a bias that we can call a "new movie effect". Of course, should we decide to use such a parameter then it would be beneficial for very old movies that were release decades before NetFlix came into existence, to have their effects truncated. Movie release date can also be used as a parameter in itself to gauge if particular user tends to like newer movies, or all movies relatively equally.

For example, for each rating I can calculate how old (in number of days) the movie being rated was.

```
head(edx_x$time - edx_x$release_time)
```

```
## Time differences in days
## [1] 1675  579  579  944  944  944
```

Similarly I can see the summary statistics of both the movie release time distribution, and user's watching/rating time distribution.

```
summary(edx_x$time)
```

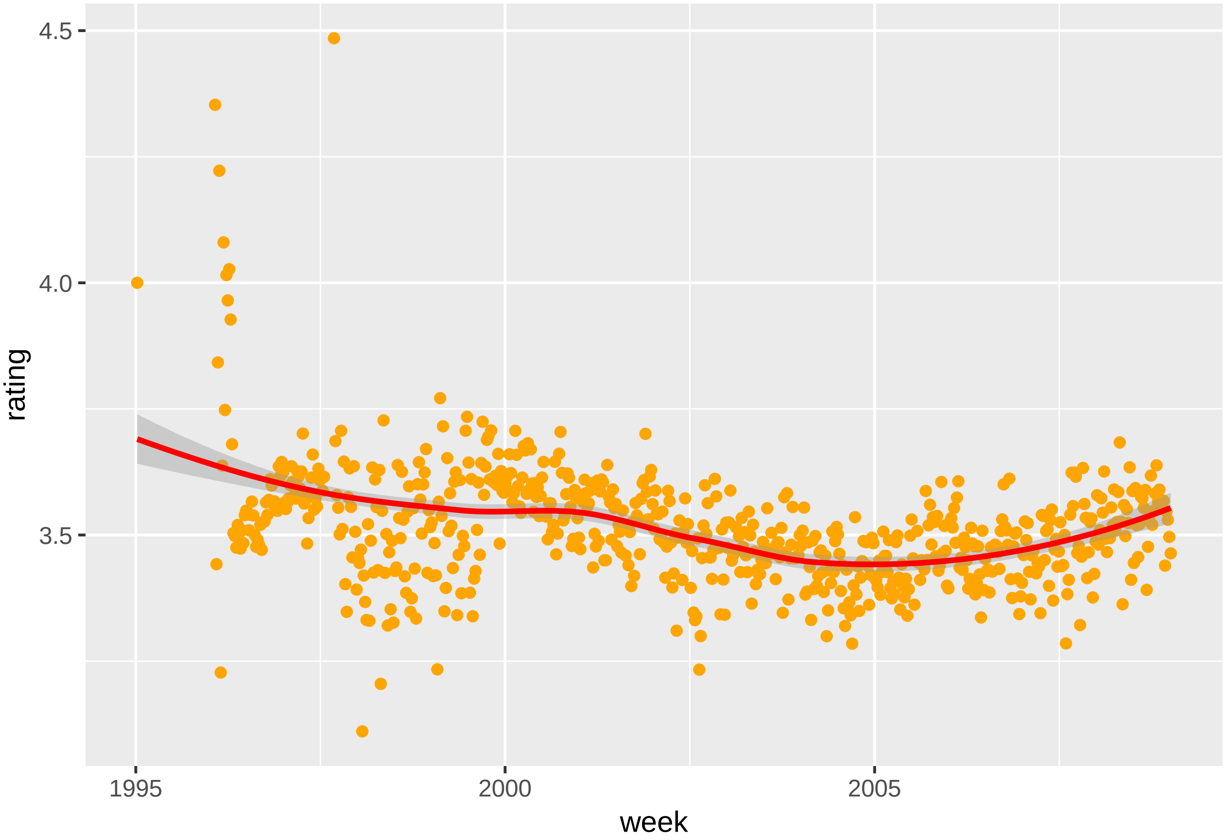
```
##         Min.      1st Qu.       Median         Mean      3rd Qu. 
## "1995-01-09" "2000-01-01" "2002-10-24" "2002-09-21" "2005-09-15" 
##         Max. 
## "2009-01-05"
```

```
summary(edx_x$release_time)
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu. 
## "1915-01-01" "1987-01-01" "1994-01-01" "1990-03-22" "1998-01-01" 
##         Max. 
## "2008-01-01"
```

Also check how users rated movies over time. In addition to understanding viewing trends over years, this may also help us understand how the data was collected: consistently, or over haphazard chunks. Weekly intervals are used since a week should be representative of each different day of the week (as weekends represent a different viewing pattern for a general user).

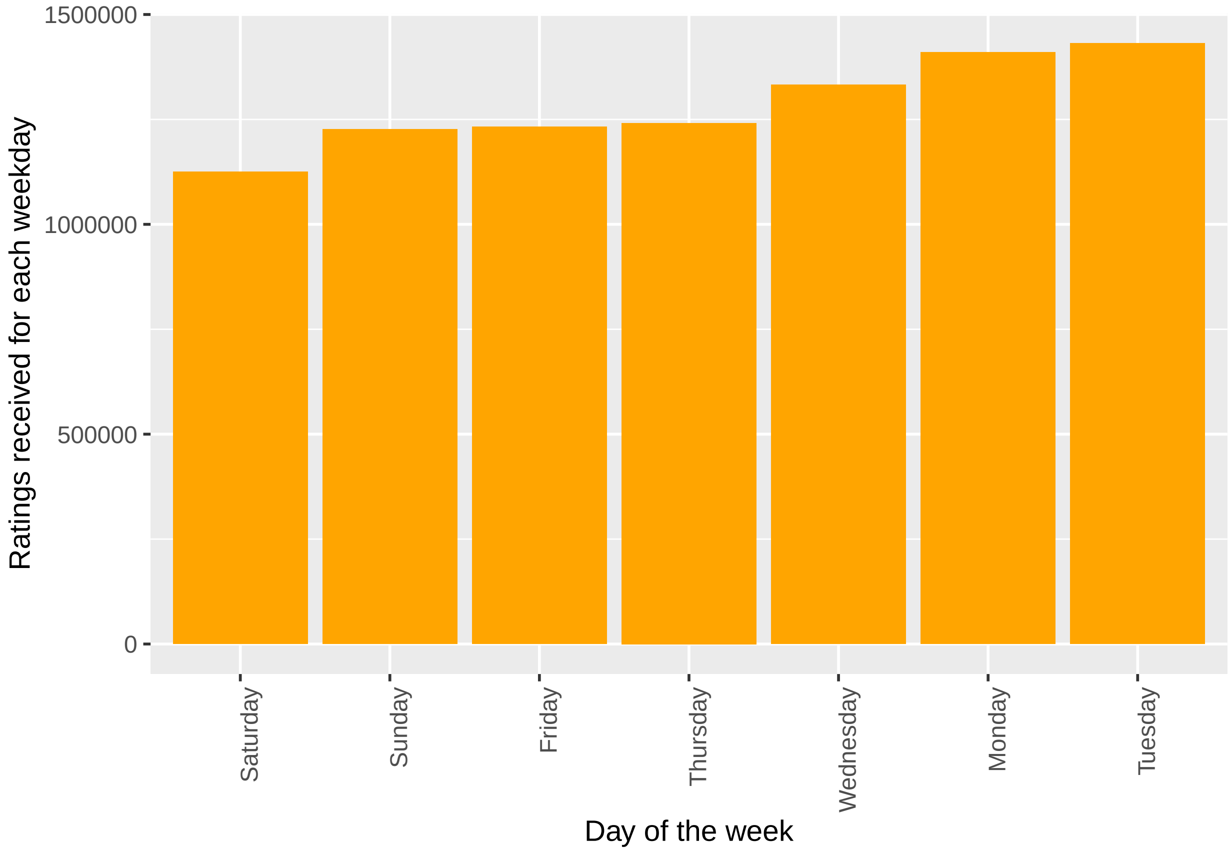
```
edx %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week")) %>% group_by(week) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(week, rating)) +
  geom_point(color="orange") +
  geom_smooth(color="red", method = "loess")
```



Now let's see if there is a specific pattern over different weekdays.

It could be useful to use this information to suggest a movie to the user based on the day of the week they are looking to watch it. However, such additonal data if not too meaningful could also contribute towards over training.

```
edx_x %>%
  mutate(rating_day = weekdays(as.Date(edx_x$time, origin = "1960-10-01"))) %>%
  group_by(rating_day) %>%
  summarize(count = n()) %>%
  mutate(rating_day = reorder(rating_day, count)) %>%
  ggplot(aes(x=rating_day, y=count)) +
  geom_bar(position="dodge", stat="identity", fill="orange") +
  labs(x="Day of the week", y="Ratings received for each weekday") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



It turns out that the rating are more or less evenly split over all weekdays, and the slight variation that do exist does not seem to follow a meaningful pattern. So I will not proceed with any enhancement in the model that will account for the weekday when suggesting movies to a user.

For the end purpose of this report, I will leave the title/year alone and focus my training on movie effect, user effect, and regularization first. I am going to take a look at the timestamp to see generally when the ratings in this dataset were received: data collection window, and any seasonal effects on the ratings or on the quality of movies watched in a given season. This probably will not amount be very useful training parameter(s), and we need to be congnizant of potentially data dredging when reaching such investigatory lengths. But it doesn't hurt to play with these while experimenting with the models.

Modeling approach

I will try multiple models with incrementally greater complexity, until a competitive RMSE is achieved. I will collect the results of each model tried and present the results in the following (results) section.

Due to the large sioze of this dataset (10 million records) linear regression and various other regression ML models will not be feasible to run. I could reduce the size of the dataset, but this will render my results inconsistent with others. So I am not taking this approach.

The approach taken here will be the same as in the class. Use an explicit model and try various effects and the regularization technique.

Results and Outcomes

Model 1: baseline

we predict the same rating for all the movies regardless of the user or the movie quality or genre (we set that 'same' rating as the the average rating of all the movies i.e. the mean rating for the entier set - i.e. in this over-simplistic model, each movie is considered to have this average rating). Hence, this attempt gives us the baseline error if we do nothing so to speak.

Define mu_hat as the average rating of all the movies i.e., this is the rating this naive model 'predicts' for each movie. (Of course we take the mean of the traiing (edx) set and not the validation set as this average calculation is akin ot traiing on the training set. Though it shouldn't make much difference anyway due to the law of averages as we studied.

```
# Define mu_hat as the average rating of all the movies i.e., this is the rating this naive
# model 'predicts' for each movie. (Of course we take the mean of the traiing (edx) set and
# not the validation set as this average calculation is akin ot traiing on the training set.
# Though it shouldn't make much difference anywat due to the law of averages as we studied
# in the earleir courses.)
avg_rating <- mean(edx$rating)

# We compare the average against the actual to compare the RMSE
# This, according to statistics theory, get to be around 1.
baseline_rmse <- RMSE(validation$rating, avg_rating)
baseline_rmse
```

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

```
# Add the results to a tibble for future evaluation e.g. when presenting a comparative analysis
# in the RMD report.
rmse_results <- tibble(method = "baseline", RMSE = baseline_rmse)
```

Now we try to incrementally improve our model to hopefully get progressively lower RMSEs i.e. better predictions.

Esentially what we are predicting here is what a given user will predict a given movie. This does not appear to have a practical value by itself (other than winning a competetion). However, the way Netflix (or some other move recommender system, or a recommender system in general) can use these results is by predicting a given user's rating; then finding say the top 5 or 10 or more movies from these predictions; then presenting those to the user as recommendations that the user would like. For our evaluation purposes, we comnpare this model against the validation set; in real life the test is if the user really feels that the list presented were indeed the movies she liked when she saw them. In practice we can for example poll random users to see how the recommendations worked for them, or more practically, we can add further logic in our system to see how mony of the movies recommended did the users actually see, actually finish, and actually rated higher, etc. This last part is not required for this project, but it is indeed important to know if, and how, and how efficiently (delay etc.) our models are going to be used during prediction time.

Model 2 - Modeling movie effects

This is my second attempt to improve the model through training. Here I improve my model by adding move effects to my model.

```
# Basically in this code I calculate the average rating of each movie in the training set
# I also subtract the overall average rating of all movies (mu) from each move average to get
# an number that is normalized around mu, like if was shown in the machine learning class
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# So I have basically trained the model such that individual rating of the movie is what the model
# predicts. So it can predict the overall good movies for the user, but with no regard to the user's
# individual preference

# Use the trained bias to predict the validation
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Calculate RMSE and add it to the results tibble
movie_effects_rmse <- RMSE(validation$rating, predicted_ratings)
movie_effects_rmse # 0.9439087
```

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

```
rmse_results <- bind_rows(rmse_results, tibble(method = "movie effects", RMSE = movie_effects_rmse))
```

Since the above approach is still simplistic, as we don't account for user's preference and just give her the best overall movies. This helps, but as we see, for this reason, fails to give us an acceptable RMSE

Model 3 - Modeling user effects

This is my third attempt to improve the model through training. Here I improve my model by adding user effects (in addition to movie effects) to this model model.

```
# To add the user effect as bias, first calculate the average rating given by each user.
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict the ratings on validation set by including both biases: the move effect and the
# recently calculated user effect.
# In other words, for every user-movie combination predicted (e.g. when predictong a movies for
# a given single use), we bias the movie by how well it received ratings for other users and how
# well each of the user, rating tha movie, has rated other movies.
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate RMSE and add it to the results tibble
movie_user_effects_rmse <- RMSE(validation$rating, predicted_ratings)
movie_user_effects_rmse # 0.8653488
```

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

```
rmse_results <- bind_rows(rmse_results, tibble(method = "movie and user effects",
                                               RMSE = movie_user_effects_rmse))
```

NOTE: We already get the desired RMSE with the user effects included (above)

But let's try to make it even better by using regularization

Model 4 - Modeling Regularization + Movie Effect + User Effect

First let's note that I have two options to calculate lambda

Calculate the RMSE using a series of lambdas. Then pick the model that provided best RMSE.

This is less efficient in computing time, but gives me a slightly better RMSE than what I get by calculating lambda beforehand using either the full training set, a subset of the training set, or the validation set.

```r
## First create a range of lambdas from which to pick the best one
lambdas <- seq(0, 10, 0.25)

# Function to calculate a set of RMSEs using a series of lambdas
# The function calculates an RMSE for each lambda using movie effects + user effects +
# regularization using that lambda
#
# The result is an array containing the RMSE obtained using each of the lambda in the sequence
# of lambdas passed in the first argument
rmses <- sapply(lambdas, function(l){

  # Calculate average rating for normalization purpose, as done before
  mu <- mean(edx$rating)

  # Calculate the first bias to use: movie effect, and regularization with lambda l
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  # Calculate the second bias to use: user effect, and regularization with lambda l
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + l))

  # Predict the rating using this model with the biases regularized over a particular lambda in the ran
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # Calculate the RMSE for predictions for each lambda; return this list of RMSEs
  return(RMSE(validation$rating, predicted_ratings))
})

# Find the minimum of all the calculated RMSEs. This gives us the best RMSE using movie effects,
# plus user effects, plus the best lambda from the finite list of lambdas
movie_user_effects_reg_rmse = min(rmses)
movie_user_effects_reg_rmse # 0.8648170
```

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

```r
rmse_results <- bind_rows(rmse_results, tibble(method = "movie and user effects with regularization",
                                               RMSE = movie_user_effects_reg_rmse))
```

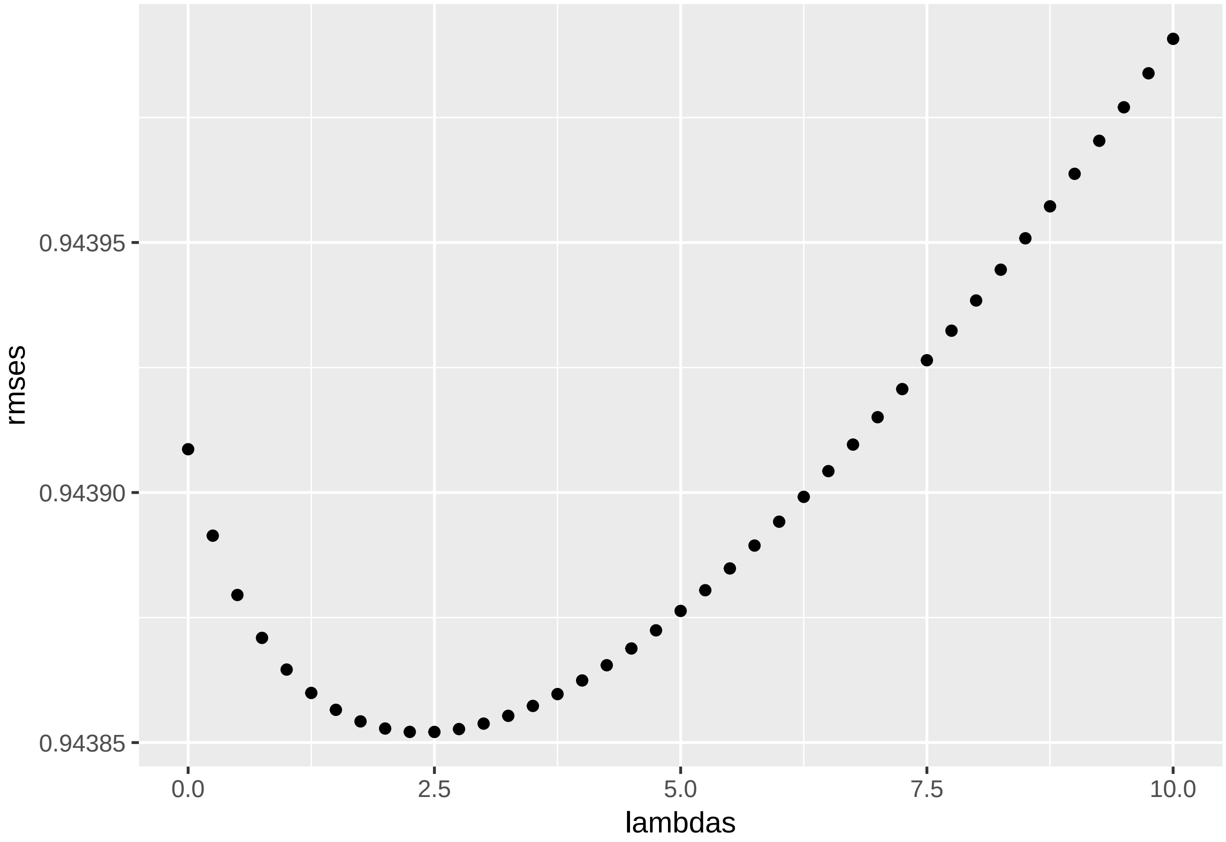
This gives even better results and significantly better result than required for this assignment, albeit at the well-justified cost of some additional calculations across all the lambdas

However, I can also calculate/pick the lambda first and then run the analysis as follows:

Model 5 - Modeling Regularization + Movie Effect + User Effect - with picked lambda

```
#########################################################
# Function to pick the best lambda
#########################################################
mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

eval <- validation
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- eval %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(eval$rating, predicted_ratings))
})
qplot(lambdas, rmses)
```



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

mu <- mean(edx$rating)

# I can calculate the bias for this lambda
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda))

# Calculate the second bias to use: user effect, and regularization with lambda l
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n() + lambda))

# Predict the rating using this model with the biases regularized over a particular lambda in the range
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

movie_user_effects_reg2_rmse = RMSE(validation$rating, predicted_ratings)
movie_user_effects_reg2_rmse # 0.8649303
```

```
## [1] 0.8649303
```

```
rmse_results <- bind_rows(rmse_results, tibble(method = "movie and user effects 2",
                                               RMSE = movie_user_effects_reg2_rmse))
```

Also I can demonstrate, by plugging 0 for lambda in the above code, that lambda of 0 gives me the same result as I got without regularization. This is just a little cross-checking to validate my own code.

Display the results:

```
## [1] 3.512465
```

```
## [1] 4
```

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

method	RMSE
baseline	1.0612018
movie effects	0.9439087
movie and user effects	0.8653488
movie and user effects with regularization	0.8648170
movie and user effects 2	0.8649303

Conclusion

Using movie effect, user effect and regularization with a properly tuned lambda provided an RMSE of 0.86482. This beats the required target of <= 0.87750.

This is a significant improvement to the baseline, and if implemented will certainly provide benefit for the end users. The two aforementioned effects and regularization imply that the improvement is achieved by preferring movies that are better rated, by preferring movies that are liked by users who rate similar to that user, and by not preferring movies with outlier and infrequent ratings and thus reducing overfitting.

This project was a great learning exercise that allowed me to consolidate what I learned in this data science specialization, and specifically in its machine learning class. However, the RMSE of our final model could be further improved based on the recommendation provided in the following subsection. Such ongoing improvements can be attempted in the future iterations of this project.

Further Recommendations, and Future Considerations

In addition to RMSE, precision and specificity may also be important tradeoffs depending upon the context in which something is being recommended in general. For example, if something being recommended is time consuming (for example, buying ticket and going somewhere solely on the basis of a recommendation) then accuracy of recommendation may be a factor that may be valued more, while if the recommendation is reminding an important action to perform (for example, an important accessory to an item being purchased, or a treatment option based on matching a patient's profile with a disease profile) that may rather not be missed, then specificity may be valued more in the recommended outcome.

Additional independent parameters and effects discovered and curated during analysis could be used to further enhance the model and improve its RMSE further. These include the genera(s) the movie belongs vs. the genera(s) that the user prefers, how old the movie is when it is rated, and the time of the day or day of the week the movie is watched (rated).

Performance (large, cross-connected dataset) prevents me from trying more standardized (out-of-box) models like linear regression, nearest neighbors, SVD (single value decomposition), PCA (principal component analysis), and deep learning. Moving forward, these could be further explored with a reduced version of the training set, or on a cloud platform.

Finally, the runtime (prediction-time) is also an important consideration. Will the data used in training the model, will also be available (or available in curated form) at runtime in a timely, efficient, and relevant manner?

References:

- GroupLens - MovieLens 10M Dataset: https://grouplens.org/datasets/movielens/10m/
- Rafael A. Irizarry - Introduction to Data Science (2019-04-22): https://rafalab.github.io/dsbook/
- tidyverse: https://tidyverse.tidyverse.org
- Houtao Deng - Recommender Systems in Practice: https://towardsdatascience.com/recommender-systems-in-practice-c
- F Ricci, L Rokach and B Shapira - Introduction to Recommender Systems: Handbook, Springer, 2011: http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf