HarvardX Capstone Report - MovieLens Project

Neli Tsereteli

9/1/2020

Contents

Deliverables

There are three deliverables for the project:

1. Report in .Rmd format
2. Report in .pdf format knit from the .Rmd file
3. R script in .R format that generates predicted movie ratings and calculates RMSE

This report documents the analysis for the Movielens Capstone project and presents the findings, along with supporting statistics and figures.

Introduction

Project overview - MovieLens

This project HarvardX: PH125.9x, Data Science: Capstone is a part of the Professional Certificate in Data Science course led by HarvardX. This program was supported in part by NIH grant R25GM114818.

Dataset

For this project, I created a movie recommendation system using the MovieLens dataset. You can find the entire latest MovieLens dataset at https://grouplens.org/datasets/movielens/latest/.

GroupLens Research has collected and made available rating data sets from the MovieLens web site. The data sets were collected over various periods of time, depending on the size of the set.

I used the 10M version of the MovieLens dataset to make the computation for the project a little easier. It is considered to be a stable benchmark dataset. 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Released 1/2009. Permalink: https://grouplens.org/datasets/movielens/10m/.

Evaluation criteria

I trained a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set and evaluated the final model using RMSE.

Methods

Setting up the environment

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse) # a set of packages that work in harmony
library(caret) # misc functions for training and plotting classification and regression models
library(data.table) # extension of 'data.frame'
library(stringr) # simple, consistent wrappers for common string operations
library(lubridate) # functions to work with date-times and time-spans
library(knitr) # general-purpose tool for dynamic report generation in R
```

Generating the datasets

Create train, test and validation sets. The train and test sets are from the edx set, and the validation set is the final hold-out test set.

Download the datasets: 1) ratings and 2) movies

```
# Create a temporary file name
dl <- tempfile()

# Download the file
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Read in ratings
ratings <- fread(text = gsub("::", "\t",
                 readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

# Read in movies
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
```

Clean up the datasets and merge them

```
# Assign column names
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
```

```r
#                                           title = as.character(title),
#                                           genres = as.character(genres))

# Merge the tables
movielens <- left_join(ratings, movies, by = "movieId")
```

Make the validation and edx sets

Validation set will be 10% of the MovieLens dataset.

```r
# Set seed for reproducibility
set.seed(1, sample.kind = "Rounding")

# Create train-test partitions.
# Validation set will be 10% of MovieLens data.
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.
# That is, ensure that we do not include users and movies in the test
# set that do not appear in the training 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)
```

Divide edx set into edx_train (80%) and edx_test (20%):

```r
# Set seed for reproducibility
set.seed(42)

# Create train-test partitions
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]

# Make sure that we do not include users and movies in the test
# set that do not appear in the training set
edx_test <- edx_test %>%
      semi_join(edx_train, by = "movieId") %>%
      semi_join(edx_train, by = "userId")
```

Clean up the environment

```r
# Clean up the environment
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Helper functions

get_genres()

Create a function to get genres of each movie.

```r
get_genres <- function(dataset) {
  # Create a logical variable for each genre
  dataset <- dataset %>% mutate(
  is_comedy = ifelse(str_detect(genres, "Comedy"), 1, 0),
  is_romance = ifelse(str_detect(genres, "Romance"), 1, 0),
  is_action = ifelse(str_detect(genres, "Action"), 1, 0),
  is_crime = ifelse(str_detect(genres, "Crime"), 1, 0),
  is_thriller = ifelse(str_detect(genres, "Thriller"), 1, 0),
  is_drama = ifelse(str_detect(genres, "Drama"), 1, 0),
  is_scifi = ifelse(str_detect(genres, "Sci-Fi"), 1, 0),
  is_adventure = ifelse(str_detect(genres, "Adventure"), 1, 0),
  is_children = ifelse(str_detect(genres, "Children"), 1, 0),
  is_fantasy = ifelse(str_detect(genres, "Fantasy"), 1, 0),
  is_war = ifelse(str_detect(genres, "War"), 1, 0),
  is_animation = ifelse(str_detect(genres, "Animation"), 1, 0),
  is_musical = ifelse(str_detect(genres, "Musical"), 1, 0),
  is_western = ifelse(str_detect(genres, "Western"), 1, 0),
  is_mystery = ifelse(str_detect(genres, "Mystery"), 1, 0),
  is_filmnoir = ifelse(str_detect(genres, "Film-Noir"), 1, 0),
  is_horror = ifelse(str_detect(genres, "Horror"), 1, 0),
  is_documentary = ifelse(str_detect(genres, "Documentary"), 1, 0),
  is_imax = ifelse(str_detect(genres, "IMAX"), 1, 0),
  is_no_genre = ifelse(str_detect(genres, "(no genres listed)"), 1, 0))
}
```

genre_effect()

Create a function that calculates genre effect.

```r
# as.symbol() functionality inspired by the post at:
# https://stackoverflow.com/questions/49371260/using-variables-as-arguments-in-summarize
genre_effect <- function(train_set, genre) {
  # Get effect name
  effect <- str_sub(genre, start = 4, end = -1)
  effect <- as.symbol((paste0("effect_", effect)))
  # Calculate average
  average <- mean(train_set$rating)
  # Calculate effect size
  genre_compared_to_average <- train_set %>%
  group_by(!!genre) %>%
```

```r
  summarize(!!effect := mean(rating - average - effect_movie - effect_user))
}
```

get_genre_score()

Calculate genre score by summing up all the genre effects.
This was inspired by the idea behind genetic risk scores.

```r
get_genre_score <- function(dataset) {
dataset <- dataset %>%
  # sum up all the effects
  mutate(genre_score =
           effect_comedy +
           effect_romance +
           effect_action +
           effect_crime +
           effect_thriller +
           effect_drama +
           effect_scifi +
           effect_adventure +
           effect_children +
           effect_fantasy +
           effect_war +
           effect_animation +
           effect_musical +
           effect_western +
           effect_mystery +
           effect_filmnoir +
           effect_horror +
           effect_documentary +
           effect_imax +
           effect_no_genre)
}
```

get_release_year()

Create a function to extract release year for each movie.
For each movie we expect a name with a year in parentheses, like in "Flinstones, The (1994)".

```r
get_release_year <- function(dataset) {
  dataset$release_year <- str_extract(dataset$title, pattern = "\\([0-9]{4}\\)")
  dataset$release_year <- str_replace_all(dataset$release_year,
                                          pattern = "\\(|\\)",
                                          replacement = "")
  dataset$release_year <- as.numeric(dataset$release_year)
  return(dataset)
}
```

Loss function - rmse()

To compare different models or to see how well a specific model is doing compared to a baseline, we need to quantify what it means to do well. We need to decide on a loss function.
For this project, the measure used is residual mean squared error (RMSE) defined on a test set. As explained in the course material, one can think of the RMSE as of standard deviation. That is, it is the typical error the model makes when predicting an outcome, or a movie rating, in this case.

Define a function that will calculate RMSE

```
rmse <- function(true_rating, predicted_rating) {
    sqrt(mean((true_rating - predicted_rating)^2))
}
```

Continuing building the Recommendation System

As per the project requirements, I am basing this section on the lecture materials from HarvardX: PH125.8x - Data Science: Machine Learning.

Recommendation systems use ratings that users have given items to make specific recommendations to users. Basically, the idea is to predict what rating a given user will give to a specific item. Then, items with predicted high rating for a given user will be recommended to that user.

For example, Netflix uses recommendation systems to predict how many stars a user will give to a specific movie.

Data exploration

Let's look at the edx set

Each row represents a rating given by one user to one movie.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
```

```
## 4:      1     316      5 838983392              Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                 Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:        Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:        Children|Comedy|Fantasy
```

We can look at the number of unique users and unique movies: there are 69,878 unique users and 10,677 unique movies.

```
edx %>% summarize(unique_users = n_distinct(userId), unique_movies = n_distinct(movieId))
```

```
##   unique_users unique_movies
## 1        69878         10677
```

If we multiply these numbers by each other, we get 69,878 * 10,677 = 746,087,406, which is much more than the number of rows in the table. As noted in the course videos, this difference implies that not every user rated every movie. As a result, we can think of this dataset as a very large matrix with users on the rows and movies on the columns with many empty cells.

An example of a subset of such matrix can be generated as follows.

Here, you can see the ratings that each user gave to each movie, with unwatched or unrated movies showing as NAs. You can think of the task of the recommendation systems as filling in the NAs in the sparce matrix example presented below.

```
# Choose random users and movies
random3_users <- sample(unique(edx$userId), 10)
random3_movies <- sample(unique(edx$movieId, 3))

# Subset edx and pivot into a sparce matrix
subset <- edx %>% filter(userId %in% random3_users,
               movieId %in% random3_movies) %>%
  select(userId, movieId, rating) %>%
  pivot_wider(names_from = movieId, values_from = rating)

# Show a subset of columns
subset[, 1:5]
```

```
## # A tibble: 10 x 5
##    userId  `5`   `7`   `9`  `12`
##     <int> <dbl> <dbl> <dbl> <dbl>
##  1   3487     3     3     3     3
##  2  15298    NA    NA    NA    NA
##  3  16262    NA     3     3    NA
##  4  36369     3    NA     3    NA
##  5  42521    NA    NA    NA    NA
##  6  51569    NA    NA    NA    NA
```

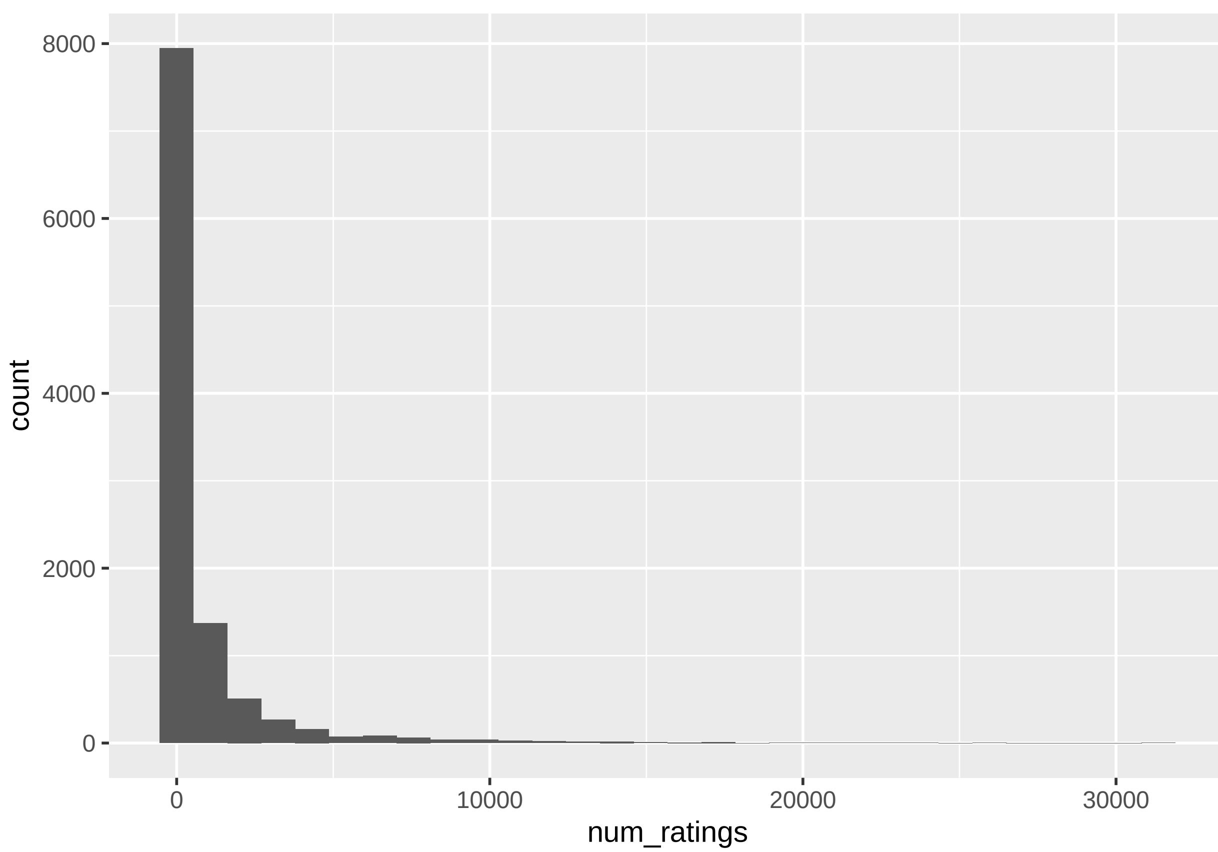
```
## 7   52656    NA    NA    NA    NA
## 8   53186    NA    NA    NA    NA
## 9   61990    NA    NA    NA    NA
## 10  70061    NA     2    NA    NA
```

In this machine learning task, the challenge is a bit more demanding because each outcome y has different set of predictors.

If we are predicting the rating for movie **m** by user **u**, then in principle, all other ratings related to movie m and made by user u can be used as predictors. However, the problem here is that not all users rate all movies and that different users rate not only different number of movies but also different movies. We may aid the recommendation by using information from other movies that have been determined to be similar to movie **m**, as well as by utilizing information from users who are similar to user **u**. This means, that in essence, the entire matrix can be used as predictors for each cell.

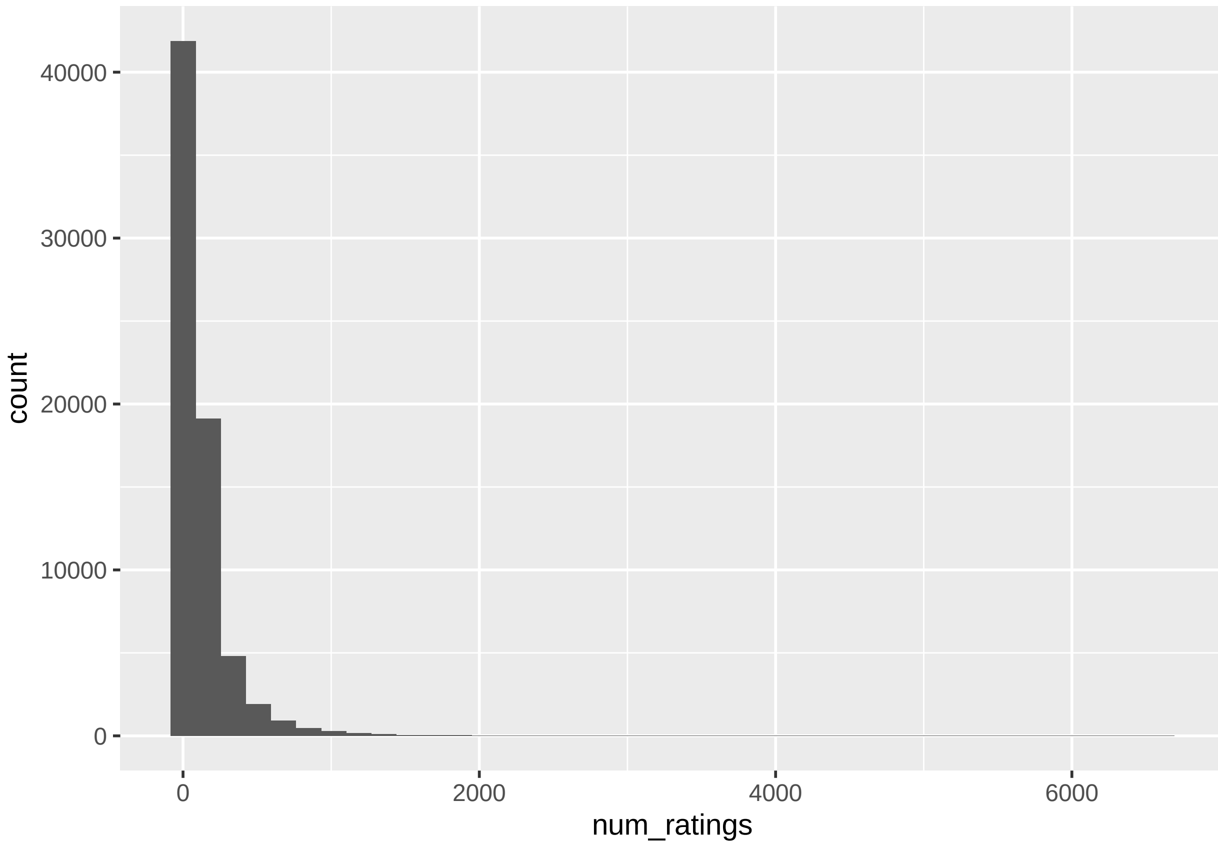
Let's look at the distribution of movie ratings. Some movies get rated more than others.

```
edx %>% group_by(movieId) %>%
  summarise(num_ratings = n()) %>%
  ggplot(aes(x = num_ratings)) + geom_histogram()
```



Similarly, some users rate more movies than other users.

```
edx %>% group_by(userId) %>%
  summarise(num_ratings = n()) %>%
  ggplot(aes(x = num_ratings)) + geom_histogram(bins = 40)
```



Feature engineering

How many unique genres are there?

```
# Get all the genres
# As demonstrated, there are 20 different genres
genre_list <- unlist(str_split(edx$genres, pattern = "\\|"))
(genre_unique <- unique(genre_list))
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

Apply helper function to get genres

```
edx_train <- get_genres(edx_train)
edx_test <- get_genres(edx_test)
```

Can number of genres improve the predictions? Calculate number of unique genres for each movie.

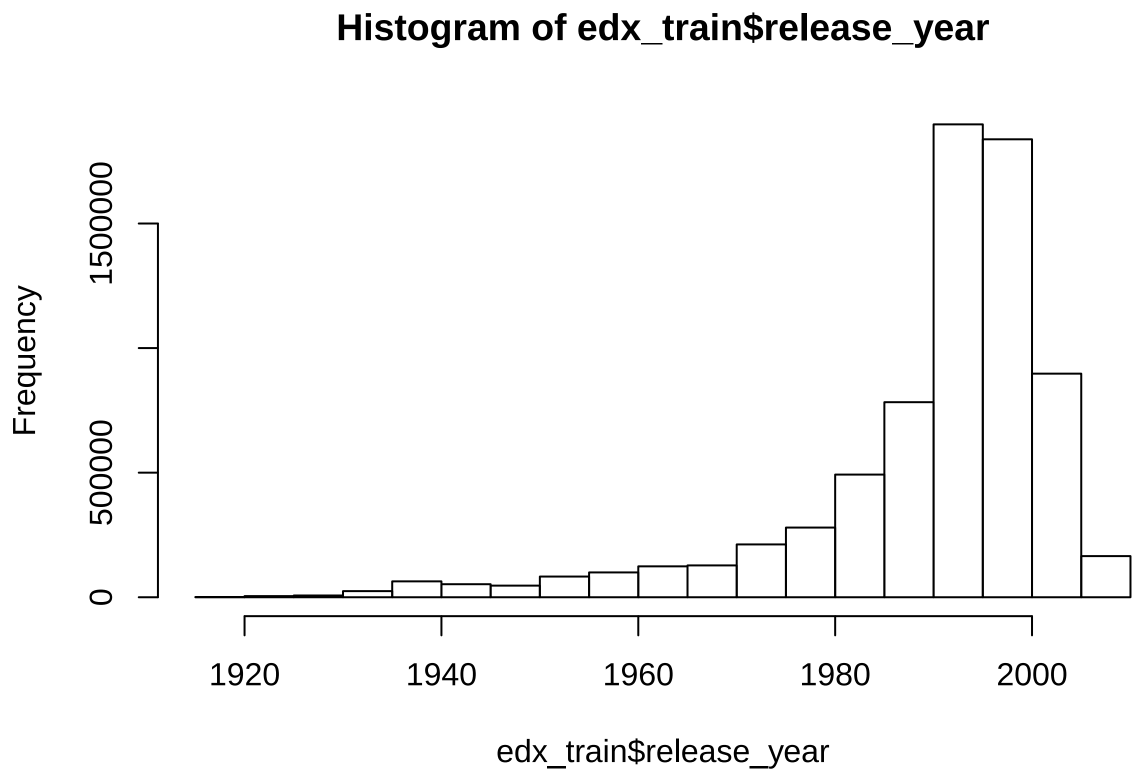
```
# Calculate number of unique genres for each movie
edx_train <- edx_train %>%
  mutate(num_genre = str_count(genres, pattern = "\\|") + 1)
edx_test <- edx_test %>%
  mutate(num_genre = str_count(genres, pattern = "\\|") + 1)
```

Get release year for each movie

```
edx_train <- get_release_year(edx_train)
edx_test <- get_release_year(edx_test)
```

Look at release year distribution

```
hist(edx_train$release_year)
```




```
range(edx_train$release_year)
```

```
## [1] 1915 2008
```

Divide movie age into 5-year intervals and look at them

```
edx_train <- edx_train %>%
  mutate(age_group = (year(now())-edx_train$release_year) - ((year(now())-edx_train$release_year)%%5))
edx_test <- edx_test %>%
  mutate(age_group = (year(now())-edx_test$release_year) - ((year(now())-edx_test$release_year)%%5))

# Unique groups
table(edx_train$age_group)
```

```
##
##      10      15      20      25      30      35      40      45      50
##   82079  736026 1607720 2215609  799596  555119  321055  200861  160468
##      55      60      65      70      75      80      85      90      95
##  117940  107633   78440   50715   46648   79639   21797   12951    4517
##     100     105
##    1015     215
```

Sequel or not?

I thought it might be intersting to see whether a movie is a sequel.
This is a naive approach but since a lot of sequels include a colon in them, I decided to make a naive sequel logical variable.

```
edx_train <- edx_train %>%
  mutate(sequel = str_detect(title, pattern = ":"))
edx_test <- edx_test %>%
  mutate(sequel = str_detect(title, pattern = ":"))

# Let's see if it makes sense
edx_train %>% filter(sequel) %>% head(20) %>% select(title)
```

```
##                                                              title
##  1:                               Robin Hood: Men in Tights (1993)
##  2:                              Terminator 2: Judgment Day (1991)
##  3:  Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
##  4:                                     Mission: Impossible (1996)
##  5:                Star Wars: Episode VI - Return of the Jedi (1983)
##  6:                              Star Trek: First Contact (1996)
##  7:                    Lord of the Rings: The Two Towers, The (2002)
##  8: Pirates of the Caribbean: The Curse of the Black Pearl (2003)
##  9:          Lord of the Rings: The Return of the King, The (2003)
## 10:                             Die Hard: With a Vengeance (1995)
## 11:     Interview with the Vampire: The Vampire Chronicles (1994)
## 12:                                  Star Trek: Generations (1994)
## 13:                              Ace Ventura: Pet Detective (1994)
## 14:                              Terminator 2: Judgment Day (1991)
## 15:     Interview with the Vampire: The Vampire Chronicles (1994)
## 16:          Three Colors: White (Trois couleurs: Blanc) (1994)
## 17:                                     Mission: Impossible (1996)
## 18:                                 Godfather: Part II, The (1974)
## 19:         Star Wars: Episode V - The Empire Strikes Back (1980)
## 20:              Star Wars: Episode I - The Phantom Menace (1999)
```

```
table(edx_train$sequel)
```

```
##
##   FALSE    TRUE
## 6624817  575226
```

Extract timestamp years

```
# Extract timestamp years
edx_train <- edx_train %>%
  mutate(year_stamp = year(as_datetime(timestamp)))

edx_test <- edx_test %>%
  mutate(year_stamp = year(as_datetime(timestamp)))
```

Building and comparing models

Model 1: rating = average

What happens if we look at the performance of a model that simply predicts the average rating? Let's call it `average_guess_rmse`.

```
average <- mean(edx_train$rating)
(rmse_average_guess <- rmse(edx_test$rating, average))
```

```
## [1] 1.059841
```

This means that this naive model would be about 1 star off on average, which seems quite large given a 1-5 star range.

Model 2: rating = average + movie effect

First calculate how good or bad a movie is compared to the average.

```
movie_compared_to_average <- edx_train %>%
  group_by(movieId) %>%
  summarize(effect_movie = mean(rating - average))
```

Look at RMSE:

```
# Merge movie effect with test set
edx_test <- left_join(edx_test, movie_compared_to_average, by = "movieId")
edx_train <- left_join(edx_train, movie_compared_to_average, by = "movieId")

# Make predictions
predicted_rating <- average + edx_test$effect_movie

# RMSE
(rmse_movie_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.9427265
```

Model 3: rating = average + movie effect + user effect

First calculate how "impressed" or "unimpressed" a user is compared to the average user after considering the overall mean and the movie effect.

```
user_compared_to_average <- edx_train %>%
  group_by(userId) %>%
  summarize(effect_user = mean(rating - average - effect_movie))
```

Look at RMSE:

```r
# Merge user effect with test set
edx_test <- left_join(edx_test, user_compared_to_average, by = "userId")
edx_train <- left_join(edx_train, user_compared_to_average, by = "userId")

# Make predictions
predicted_rating <- average + edx_test$effect_movie + edx_test$effect_user

# RMSE
(rmse_user_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8652019
```

Model 4: rating = average + movie + user + genre score

Let's find an effect for each genre and create a "genre score" for each movie because most of the movies are a combination of several genres.

Create genre effects for all the genres

```r
# Create genre effects
# 1. C O M E D Y
comedy_compared_to_average <- genre_effect(edx_train, sym("is_comedy"))
edx_train <- left_join(edx_train, comedy_compared_to_average, by = "is_comedy")
edx_test <- left_join(edx_test, comedy_compared_to_average, by = "is_comedy")

# 2. R O M A N C E
romance_compared_to_average <- genre_effect(edx_train, sym("is_romance"))
edx_train <- left_join(edx_train, romance_compared_to_average, by = "is_romance")
edx_test <- left_join(edx_test, romance_compared_to_average, by = "is_romance")

# 3. A C T I O N
action_compared_to_average <- genre_effect(edx_train, sym("is_action"))
edx_train <- left_join(edx_train, action_compared_to_average, by = "is_action")
edx_test <- left_join(edx_test, action_compared_to_average, by = "is_action")

# 4. C R I M E
crime_compared_to_average <- genre_effect(edx_train, sym("is_crime"))
edx_train <- left_join(edx_train, crime_compared_to_average, by = "is_crime")
edx_test <- left_join(edx_test, crime_compared_to_average, by = "is_crime")

# 5. T H R I L L E R
thriller_compared_to_average <- genre_effect(edx_train, sym("is_thriller"))
edx_train <- left_join(edx_train, thriller_compared_to_average, by = "is_thriller")
edx_test <- left_join(edx_test, thriller_compared_to_average, by = "is_thriller")

# 6. D R A M A
drama_compared_to_average <- genre_effect(edx_train, sym("is_drama"))
edx_train <- left_join(edx_train, drama_compared_to_average, by = "is_drama")
edx_test <- left_join(edx_test, drama_compared_to_average, by = "is_drama")
```

```r
# 7. S C I - F I
scifi_compared_to_average <- genre_effect(edx_train, sym("is_scifi"))
edx_train <- left_join(edx_train, scifi_compared_to_average, by = "is_scifi")
edx_test <- left_join(edx_test, scifi_compared_to_average, by = "is_scifi")

# 8. A D V E N T U R E
adventure_compared_to_average <- genre_effect(edx_train, sym("is_adventure"))
edx_train <- left_join(edx_train, adventure_compared_to_average, by = "is_adventure")
edx_test <- left_join(edx_test, adventure_compared_to_average, by = "is_adventure")

# 9. C H I L D R E N
children_compared_to_average <- genre_effect(edx_train, sym("is_children"))
edx_train <- left_join(edx_train, children_compared_to_average, by = "is_children")
edx_test <- left_join(edx_test, children_compared_to_average, by = "is_children")

# 10. F A N T A S Y
fantasy_compared_to_average <- genre_effect(edx_train, sym("is_fantasy"))
edx_train <- left_join(edx_train, fantasy_compared_to_average, by = "is_fantasy")
edx_test <- left_join(edx_test, fantasy_compared_to_average, by = "is_fantasy")

# 11. W A R
war_compared_to_average <- genre_effect(edx_train, sym("is_war"))
edx_train <- left_join(edx_train, war_compared_to_average, by = "is_war")
edx_test <- left_join(edx_test, war_compared_to_average, by = "is_war")

# 12. A N I M A T I O N
animation_compared_to_average <- genre_effect(edx_train, sym("is_animation"))
edx_train <- left_join(edx_train, animation_compared_to_average, by = "is_animation")
edx_test <- left_join(edx_test, animation_compared_to_average, by = "is_animation")

# 13. M U S I C A L
musical_compared_to_average <- genre_effect(edx_train, sym("is_musical"))
edx_train <- left_join(edx_train, musical_compared_to_average, by = "is_musical")
edx_test <- left_join(edx_test, musical_compared_to_average, by = "is_musical")

# 14. W E S T E R N
western_compared_to_average <- genre_effect(edx_train, sym("is_western"))
edx_train <- left_join(edx_train, western_compared_to_average, by = "is_western")
edx_test <- left_join(edx_test, western_compared_to_average, by = "is_western")

# 15. M Y S T E R Y
mystery_compared_to_average <- genre_effect(edx_train, sym("is_mystery"))
edx_train <- left_join(edx_train, mystery_compared_to_average, by = "is_mystery")
edx_test <- left_join(edx_test, mystery_compared_to_average, by = "is_mystery")

# 16. F I L M - N O I R
filmnoir_compared_to_average <- genre_effect(edx_train, sym("is_filmnoir"))
edx_train <- left_join(edx_train, filmnoir_compared_to_average, by = "is_filmnoir")
edx_test <- left_join(edx_test, filmnoir_compared_to_average, by = "is_filmnoir")

# 17. H O R R O R
horror_compared_to_average <- genre_effect(edx_train, sym("is_horror"))
edx_train <- left_join(edx_train, horror_compared_to_average, by = "is_horror")
```

```
edx_test <- left_join(edx_test, horror_compared_to_average, by = "is_horror")

# 18. D O C U M E N T A R Y
documentary_compared_to_average <- genre_effect(edx_train, sym("is_documentary"))
edx_train <- left_join(edx_train, documentary_compared_to_average, by = "is_documentary")
edx_test <- left_join(edx_test, documentary_compared_to_average, by = "is_documentary")

# 19. I M A X
imax_compared_to_average <- genre_effect(edx_train, sym("is_imax"))
edx_train <- left_join(edx_train, imax_compared_to_average, by = "is_imax")
edx_test <- left_join(edx_test, imax_compared_to_average, by = "is_imax")

# 20. N O   G E N R E
nogenre_compared_to_average <- genre_effect(edx_train, sym("is_no_genre"))
edx_train <- left_join(edx_train, nogenre_compared_to_average, by = "is_no_genre")
edx_test <- left_join(edx_test, nogenre_compared_to_average, by = "is_no_genre")
```

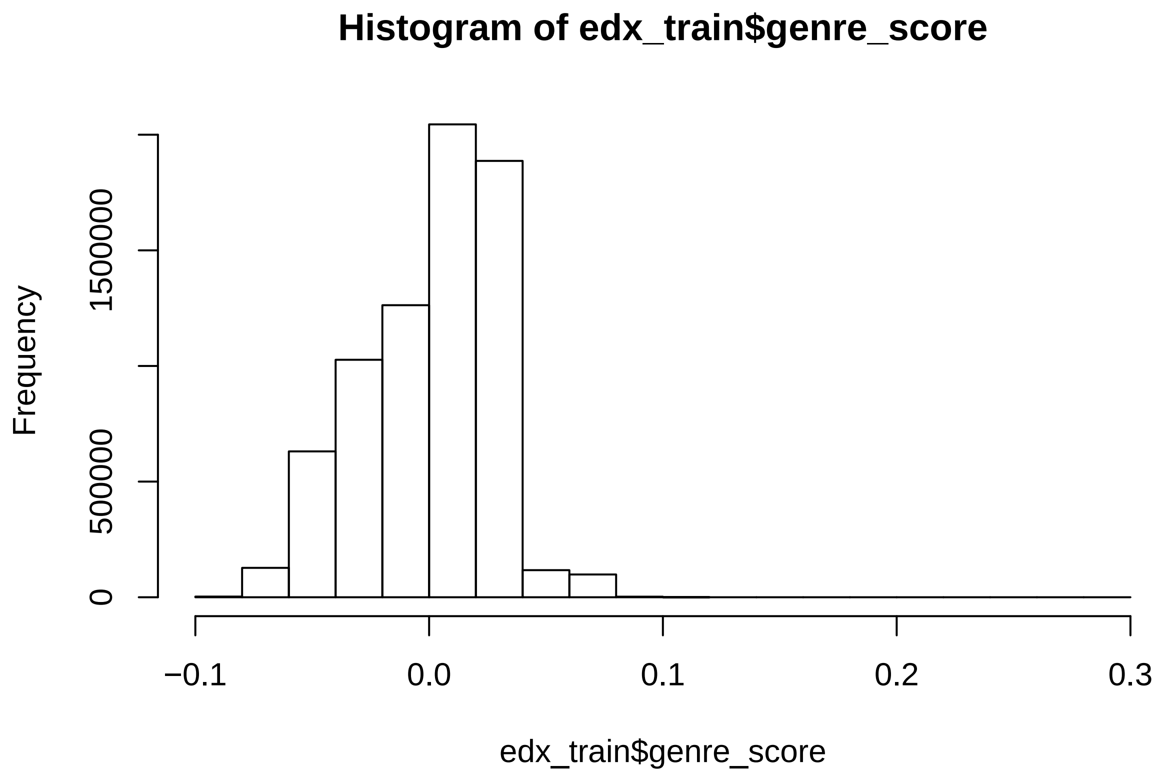
Clean up the environment

```
# Clean up the environment
rm(action_compared_to_average, adventure_compared_to_average,
   animation_compared_to_average, children_compared_to_average,
   comedy_compared_to_average, crime_compared_to_average,
   documentary_compared_to_average, drama_compared_to_average,
   fantasy_compared_to_average, filmnoir_compared_to_average,
   horror_compared_to_average, imax_compared_to_average,
   musical_compared_to_average, mystery_compared_to_average,
   nogenre_compared_to_average, romance_compared_to_average,
   scifi_compared_to_average, thriller_compared_to_average,
   user_compared_to_average, war_compared_to_average,
   western_compared_to_average,
   genre_list, genre_unique, random3_movies, random3_users,
   movie_compared_to_average
   )
```

Calculate genre scores and look at their distribution

```
edx_train <- get_genre_score(edx_train)
edx_test <- get_genre_score(edx_test)

# Histogram
hist(edx_train$genre_score)
```



Divide genre_score into 10 equally-distanced groups

```
edx_train$group <- as.numeric(cut_number(edx_train$genre_score, 10))
edx_test$group <- as.numeric(cut_number(edx_test$genre_score, 10))

# Tabulate
table(edx_train$group)
```

```
##
##      1      2      3      4      5      6      7      8      9     10
## 722701 755156 698532 864517 796240 503355 701885 742666 774033 640958
```

Find genre group effect

```
group_compared_to_average <- edx_train %>%
  group_by(group) %>%
  summarize(effect_group = mean(rating - average - effect_movie - effect_user))
```

Look at RMSE

```r
# Merge group effect with the train and test sets
edx_train <- left_join(edx_train, group_compared_to_average, by = "group")
edx_test <- left_join(edx_test, group_compared_to_average, by = "group")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group)

# RMSE
(rmse_genre_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8650607
```

Model 5: rating = average + movie + user + genre score + number of genres

Get genre number effect

```r
number_compared_to_average <- edx_train %>%
  group_by(num_genre) %>%
  summarize(effect_num = mean(rating - average - effect_movie - effect_user - effect_group))
```

Look at RMSE

```r
# Merge the group effect with the train and test sets
edx_train <- left_join(edx_train, number_compared_to_average, by = "num_genre")
edx_test <- left_join(edx_test, number_compared_to_average, by = "num_genre")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group +
                     edx_test$effect_num)

# RMSE
(rmse_genre_num_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8650527
```

Model 6a: rating = average + movie + user + genre score + number of genres + movie age group

Get movie age group effect

```
# Get grouped means
age_compared_to_average <- edx_train %>%
  group_by(age_group) %>%
  summarize(effect_age = mean(rating - average - effect_movie - effect_user - effect_group - effect_num
```

Look at RMSE

```
# Merge movie age effect with the train and test sets
edx_train <- left_join(edx_train, age_compared_to_average, by = "age_group")
edx_test <- left_join(edx_test, age_compared_to_average, by = "age_group")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group +
                     edx_test$effect_num +
                     edx_test$effect_age)

# RMSE
(rmse_age_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8647623
```

Model 6b: rating = average + movie + user + genre score + number of genres + release year

Can we improve the result by considering release year instead of age group?

```
# Get grouped means
release_compared_to_average <- edx_train %>%
  group_by(release_year) %>%
  summarize(effect_release = mean(rating - average - effect_movie - effect_user - effect_group - effect_
```

Look at RMSE - it's better than that of model using age group

```
# Merge movie age effect with the train and test sets
edx_train <- left_join(edx_train, release_compared_to_average, by = "release_year")
edx_test <- left_join(edx_test, release_compared_to_average, by = "release_year")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group +
                     edx_test$effect_num +
```

```
                     edx_test$effect_release)

#RMSE
(rmse_release_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8647295
```

Model 7: rating = average + movie + user + genre score + number of genres + release year + sequel

Add sequel as an effect

```
# Get grouped means
sequel_compared_to_average <- edx_train %>%
  group_by(sequel) %>%
  summarize(effect_sequel = mean(rating - average - effect_movie - effect_user - effect_group - effect_r
```

Look at RMSE

```
# Merge movie sequel effect with the train and test sets
edx_train <- left_join(edx_train, sequel_compared_to_average, by = "sequel")
edx_test <- left_join(edx_test, sequel_compared_to_average, by = "sequel")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group +
                     edx_test$effect_num +
                     edx_test$effect_release +
                     edx_test$effect_sequel)

#RMSE
(rmse_sequel_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8647286
```

Model8: rating = average + movie + user + genre score + number of genres + release year + sequel + rating year

Get year_stamp effect

```
# Now get the year_stamp effect to see if it can improve the model
# Get grouped means
stamp_year_compared_to_average <- edx_train %>%
  group_by(year_stamp) %>%
```

```
  summarize(effect_year_stamp = mean(rating - average - effect_movie - effect_user
                                     - effect_group - effect_num -
                                       effect_release - effect_sequel))
```

```
# Look at RMSE
# Merge stamp year effect with the train and test sets
edx_train <- left_join(edx_train, stamp_year_compared_to_average, by = "year_stamp")
edx_test <- left_join(edx_test, stamp_year_compared_to_average, by = "year_stamp")

# Make predictions
predicted_rating <- (average +
                     edx_test$effect_movie +
                     edx_test$effect_user +
                     edx_test$effect_group +
                     edx_test$effect_num +
                     edx_test$effect_release +
                     edx_test$effect_sequel +
                     edx_test$effect_year_stamp)

# RMSE
(rmse_year_stamp_effect <- rmse(edx_test$rating, predicted_rating))
```

```
## [1] 0.8646657
```

RMSE comparison table

This section presents a table with the RMSEs for various models.

model	RMSE
Average star rating	1.059841
Average + movie	0.942727
Average + movie + user	0.865202
Average + movie + user + genre	0.865061
Average + movie + user + genre + num genre	0.865053
Average + movie + user + genre + num genre + year	0.864730
Average + movie + user + genre + num genre + year + sequel	0.864729
Average + movie + user + genre + num genre + year + sequel + year stamp	0.864666

Final model

Based on the test set RMSE, the lowest RMSE is achieved using the following model:
Predicted rating = average + movie + user + genre + num of genres + release year + sequel + year stamp
Before running the model on the validation set, I will retrain it using the whole edx dataset, instead of edx_train only.

Clean up the environment

```
# Clean up the environment
rm(age_compared_to_average, group_compared_to_average,
   number_compared_to_average, release_compared_to_average,
   sequel_compared_to_average, stamp_year_compared_to_average,
   edx_train, edx_test, subset)
```

Conclusion

Re-training the final model

This entails repeating all steps from above but using edx instead of edx_train.

Final RMSE of the re-trained model

The model here achieves an RMSE of 0.8647926.

```
predicted_rating <- (average +
                     validation$effect_movie +
                     validation$effect_user +
                     validation$effect_group +
                     validation$effect_num +
                     validation$effect_release +
                     validation$effect_sequel +
                     validation$effect_year_stamp)
#RMSE
(rmse_final <- rmse(validation$rating, predicted_rating))
```

```
## [1] 0.8647926
```

Limitations and future work

It is important to note that nowadays deep neural networks are also used for recommendation systems so the approach presented here could be considered a naive algorithm. In addition, this dataset does not have information on users. It would have been interesting to build a model that would take users' age and sex into account. If given user demographics, it would also have been possible to look at interactions - for example age-sex-genre interactions.

Interesting resources

For an interesting resource about recommendation systems, listen to SuperDataScience podcast episode SDS 265: Data Science in the World of Big Data, where big data expert and educator Frank Kane talks about recommender systems.