Capstone Project

Rei Romero

2022-11-22

Executive Summary

The popularization of streaming services such as Netflix and Hulu has made the consumption and reviewing of films for the general public much more accessible due to the relatively cost effective nature of such subscription services. Furthermore, the widespread use of aforementioned services has made it easier to collect and analyze data regarding the movie scores from millions of people online.

Hence, it would be possible and fruitful to predict the rating a movie will receive from any reviewer. This is the objective of this project: to predict the movie rating any person will give to any movie.

First, we obtained the dataset to be used, the r object edx derived from the MovieLens 10M dataset, which has the following format:

```
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 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)"
"Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller"
"Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Note that the edx dataset has six variables, namely: userId, movieId, rating, timestamp, title, and genres.

The rating variable is the target variable to be predicted using the other variables or some transformation of them. As can be seen, movies can belong to multiple genres, and reviewers can have multiple reviews for differing movies but only one review per movie.

To facilitate the prediction of movie ratings, we made the following steps:

1. Examined the edx dataset.
2. Derived the average rating per movie.

3. Derived the average movie rating per user.
4. Transformed each genre into a binary variable . (0 = the movie is not of an indicated genre, 1 = the movie is of an indicated genre)
5. Fit an XGBoost model using the variables obtained from steps 2-4, with the use of 10-fold cross validation.
6. Computed the RMSE with respect to the validation set.

An XGBoost model was chosen due to its capability to make the most out of hardware thereby speeding up the modelling process which can be quite long especially with other techniques such as linear regression or K nearest neighbors. XGBoost is also easy to use and implement for data in the millions of observations.

Analysis

We start with installing and loading the necessary packages using these lines of code:

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

## Loading required package: class

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

## Loading required package: xgboost

## Warning: package 'xgboost' was built under R version 4.1.3

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice

library(dplyr)
library(caret)
library(class)
library(tidyverse)
library(data.table)
library(xgboost)
```

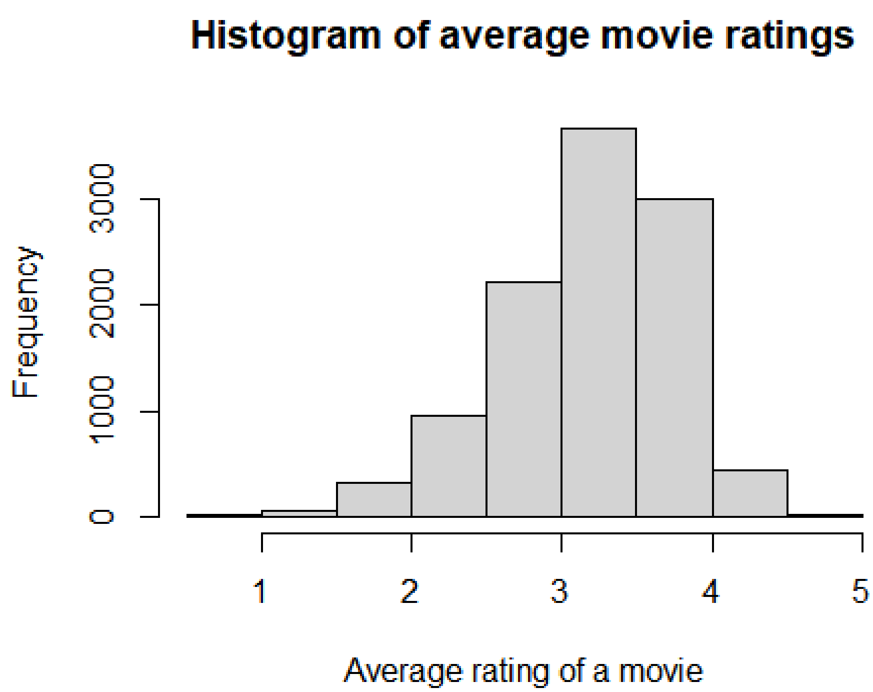
Moving on, the variable to be predicted is the rating variable. However, it seems that the edx dataset is not in any format that could be useful for predicting the target variable, rating. Thus, we will derive some variables from the edx dataset using data wrangling.

An obvious first choice for predicting movie ratings would be the average rating per movie. The following is the code to obtain such a variable:

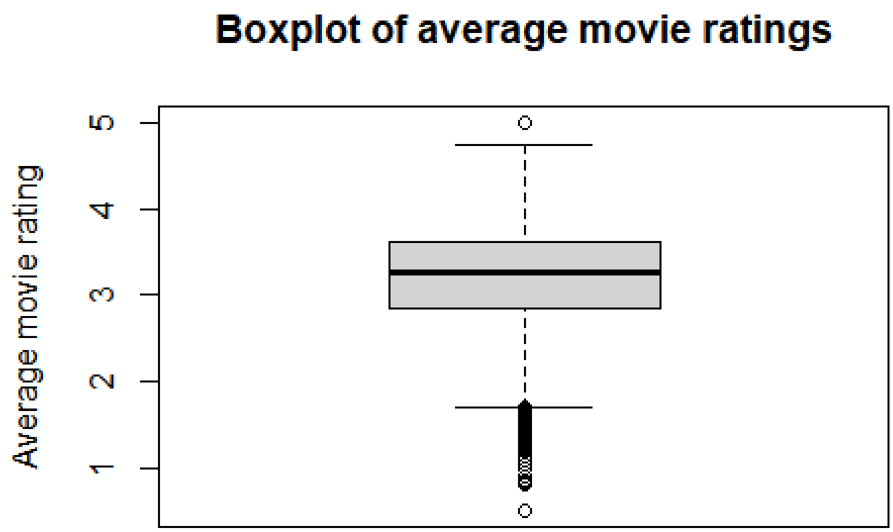
```
# Deriving the average rating per movie
avg_rating_per_movie <- edx %>% group_by(movieId) %>%
summarize(avg_movie_rating = mean(rating))
str(avg_rating_per_movie)

## tibble [10,677 x 2] (S3: tbl_df/tbl/data.frame)
##  $ movieId         : num [1:10677] 1 2 3 4 5 6 7 8 9 10 ...
##  $ avg_movie_rating: num [1:10677] 3.93 3.21 3.15 2.86 3.07 ...
```

Then, we visualize the distribution of the average rating per movie using both a histogram and a boxplot:

```
# Visualizing the distribution of average movie ratings
hist(avg_rating_per_movie$avg_movie_rating,
     xlab = "Average rating of a movie",
     main = "Histogram of average movie ratings")
```




```
boxplot(avg_rating_per_movie$avg_movie_rating,
        main = "Boxplot of average movie ratings",
        ylab = "Average movie rating")
```

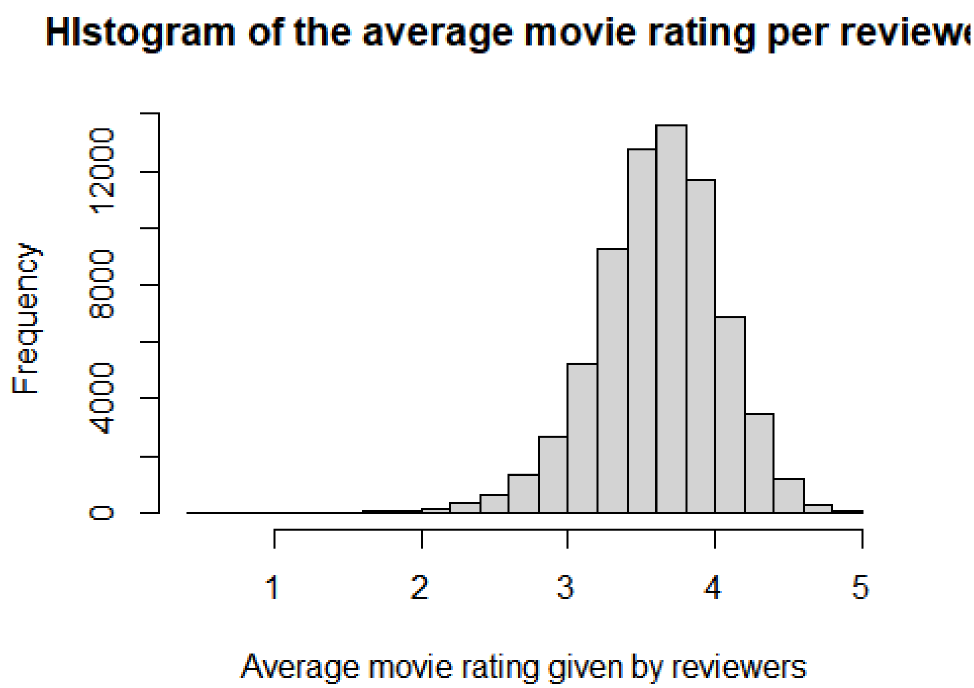


As can be seen from both the histogram and boxplot, most of the averages of movie ratings range from 2 to 4. We then combine these averages with the edx dataset, and create an intermediate dataset called "data", with the following piece of code:

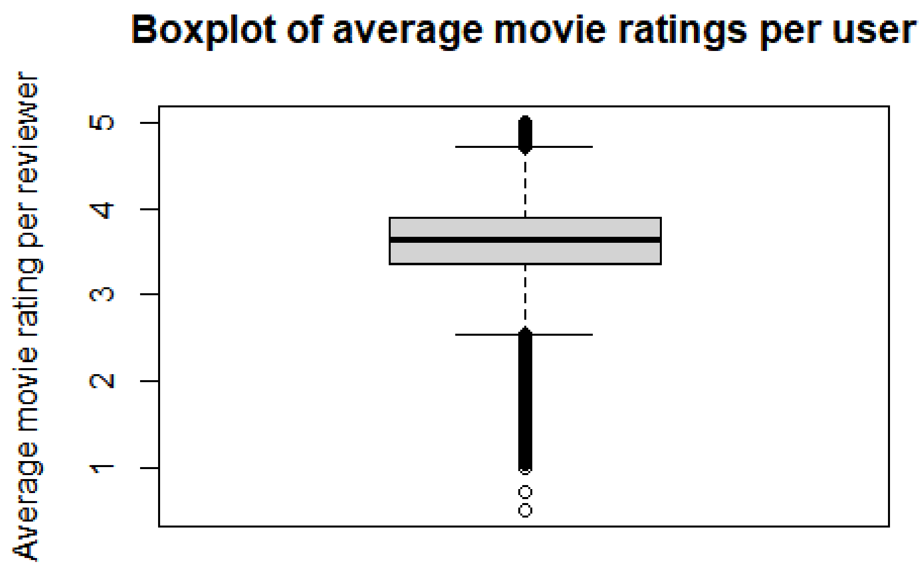
```
# Inserting the average rating per movie for each observation in the edx
dataset
data <- inner_join(edx, avg_rating_per_movie)

## Joining, by = "movieId"

str(data)

## Classes 'data.table' and 'data.frame':   9000055 obs. of  7 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 838983707 838984596 ...
##  $ title           : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak
(1995)" "Stargate (1994)" ...
##  $ genres          : chr  "Comedy|Romance" "Action|Crime|Thriller"
"Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  $ avg_movie_rating: num  2.86 3.13 3.42 3.35 3.34 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Next, we perform the same process that was just done, but now with respect to the average movie rating per user, with the following lines of code:

```r
# Deriving the average rating per user
avgratings_per_user <- edx %>% group_by(userId) %>%
summarize(avg_rating_per_user = mean(rating))

# Visualizing the distribution of average movie rating given by each reviewer
hist(avgratings_per_user$avg_rating_per_user,
     xlab = "Average movie rating given by reviewers",
     main = "HIstogram of the average movie rating per reviewer")
```




```r
boxplot(avgratings_per_user$avg_rating_per_user,
        ylab = "Average movie rating per reviewer",
        main = "Boxplot of average movie ratings per user")
```




```r
# Inserting the average rating per user for each observation in the edx
dataset
data <- inner_join(data, avgratings_per_user)

## Joining, by = "userId"

str(data)

## Classes 'data.table' and 'data.frame':   9000055 obs. of  8 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 838983707 838984596 ...
##  $ title              : chr  "Boomerang (1992)" "Net, The (1995)"
"Outbreak (1995)" "Stargate (1994)" ...
##  $ genres             : chr  "Comedy|Romance" "Action|Crime|Thriller"
"Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  $ avg_movie_rating   : num  2.86 3.13 3.42 3.35 3.34 ...
##  $ avg_rating_per_user: num  5 5 5 5 5 5 5 5 5 5 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

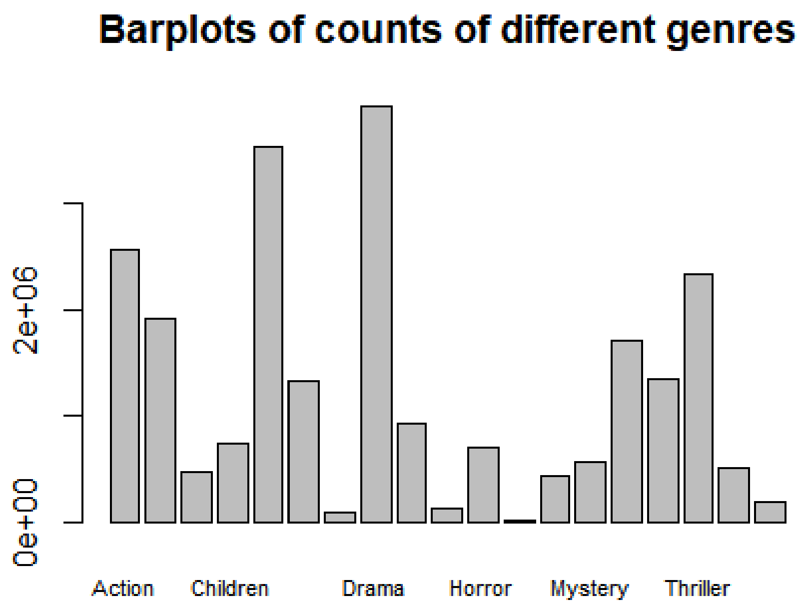
Now, while the average movie rating per reviewer and the average ratings per movie seem to be quite similar, it is a reasonable belief that there is both a reviewer effect and a movie effect wherein reviewers will have tendencies and patterns when it comes to reviewing movies, and movies will most likely be rated similarly by most of their reviewers.

Hence, we kept the average movie rating per reviewer and the average ratings per movie and will be using them for the XGBoost model to be used later.

It is also reasonable to believe that genres will have an effect on how a movie is rated since people have preferences when it comes to movie genres and will thus most likely rate movies higher when they belong to a certain or certain genres.

The following are the lines of code used to discover, list, count, and visualize the genres in the dataset thus far:

```
# Discovering all possible genres listed in the edx dataset
data_genres <- str_split(data$genres, "\\|", simplify = TRUE)
genre_vec <- data_genres %>% factor()
list_genres <- genre_vec %>% levels()

# Counts of genres
genre_vec <- as.data.frame(genre_vec)
genre_count <- genre_vec %>% group_by(genre_vec) %>% count()
genre_count

## # A tibble: 21 x 2
## # Groups:   genre_vec [21]
##    genre_vec                   n
##    <fct>                   <int>
##  1 ""                   48629017
##  2 "(no genres listed)"        7
##  3 "Action"              2560545
##  4 "Adventure"           1908892
##  5 "Animation"            467168
##  6 "Children"             737994
##  7 "Comedy"              3540930
##  8 "Crime"               1327715
##  9 "Documentary"           93066
## 10 "Drama"               3910127
## # ... with 11 more rows

# Visualizing counts of genres

# We take out the empty character strings and "(no genres listed)" strings
genre_vec <- genre_vec[genre_vec != "" ]
genre_vec <- genre_vec[genre_vec != "(no genres listed)"]

# We visualize the counts of the different genres by giving each its own
barplot
barplot(table(genre_vec),
        main = "Barplots of counts of different genres",
        width = 1,
        cex.names = 0.7
        )
```




```
list_genres

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

As we can see, there were seven movies which had no genres listed and there were many blank spaces left by the regex str_split command. However, the barplot shows that the movies included in the edx dataset are of varying genres and that there are some genres such as drama and action which appear much more often than other genres.

Now, we transform each genre into a binary variable for each observation in the data dataset, and finalize the dataset to be used in the cross- validated XGBoost model:

```
data <- as.data.table(data)

data[ , `:=` (ng = data$genres %>%
                str_detect("(no genres listed)") %>%
                as.numeric(),
              action = data$genres %>%
                str_detect("Action") %>%
                as.numeric(),
              adventure = data$genres %>%
```

```
    str_detect("Adventure") %>%
    as.numeric(),
  animation = data$genres %>%
    str_detect("Animation") %>%
    as.numeric(),
  children = data$genres %>%
    str_detect("Children") %>%
    as.numeric(),
  comedy = data$genres %>%
    str_detect("Comedy") %>%
    as.numeric(),
  crime = data$genres %>%
    str_detect("Crime") %>%
    as.numeric(),
  documentary = data$genres %>%
    str_detect("Documentary") %>%
    as.numeric(),
  drama = data$genres %>%
    str_detect("Drama") %>%
    as.numeric(),
  fantasy = data$genres %>%
    str_detect("Fantasy") %>%
    as.numeric(),
  film_noir = data$genres %>%
    str_detect("Film-Noir") %>%
    as.numeric(),
  horror = data$genres %>%
    str_detect("Horror") %>%
    as.numeric(),
  imax = data$genres %>%
    str_detect("IMAX") %>%
    as.numeric(),
  musical = data$genres %>%
    str_detect("Musical") %>%
    as.numeric(),
  mystery = data$genres %>%
    str_detect("Mystery") %>%
    as.numeric(),
  romance = data$genres %>%
    str_detect("Romance") %>%
    as.numeric(),
  scifi = data$genres %>%
    str_detect("Sci-Fi") %>%
    as.numeric(),
  thriller = data$genres %>%
    str_detect("Thriller") %>%
    as.numeric(),
  war = data$genres %>%
    str_detect("War") %>%
    as.numeric(),
```

```r
              western = data$genres %>%
                str_detect("Western") %>%
                as.numeric()
              )
      ]

# Finalizing dataset to be used in the cross-validated XGBoost model
final_data <- data[ , 7:28]
str(final_data)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  22 variables:
##  $ avg_movie_rating   : num  2.86 3.13 3.42 3.35 3.34 ...
##  $ avg_rating_per_user: num  5 5 5 5 5 5 5 5 5 5 ...
##  $ ng                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ action             : num  0 1 1 1 1 0 0 0 0 1 ...
##  $ adventure          : num  0 0 0 1 1 0 0 1 1 0 ...
##  $ animation          : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ children           : num  0 0 0 0 0 1 0 1 1 0 ...
##  $ comedy             : num  1 0 0 0 0 1 1 0 0 1 ...
##  $ crime              : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ documentary        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ drama              : num  0 0 1 0 1 0 1 0 1 0 ...
##  $ fantasy            : num  0 0 0 0 0 1 0 0 0 0 ...
##  $ film_noir          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ horror             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ imax               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ musical            : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ mystery            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ romance            : num  1 0 0 0 0 0 1 1 0 0 ...
##  $ scifi              : num  0 0 1 1 1 0 0 0 0 0 ...
##  $ thriller           : num  0 1 1 0 0 0 0 0 0 0 ...
##  $ war                : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ western            : num  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Moving on, we now perform the same data wrangling steps that were done to the edx dataset, to the validation dataset:

```r
# Validation data wrangling

    # Deriving the average rating per movie
val_avg_rating_per_movie <- validation %>% group_by(movieId) %>%
  summarize(avg_movie_rating = mean(rating))

    # Inserting the average rating per movie for each observation in the
validation dataset
val_data <- inner_join(validation, val_avg_rating_per_movie)

## Joining, by = "movieId"
```

```r
    # Deriving the average ratings per user
val_avgratings_per_user <- validation %>% group_by(userId) %>%
  summarize(avg_rating_per_user = mean(rating))

    #Inserting the average rating per user in the validation dataset
val_data <- inner_join(val_data, val_avgratings_per_user)

## Joining, by = "userId"

    # Making validation genres into binary variables
val_data <- as.data.table(val_data)

val_data[ , `:=` (ng = val_data$genres %>%
                    str_detect("(no genres listed)") %>%
                    as.numeric(),
                  action = val_data$genres %>%
                    str_detect("Action") %>%
                    as.numeric(),
                  adventure = val_data$genres %>%
                    str_detect("Adventure") %>%
                    as.numeric(),
                  animation = val_data$genres %>%
                    str_detect("Animation") %>%
                    as.numeric(),
                  children = val_data$genres %>%
                    str_detect("Children") %>%
                    as.numeric(),
                  comedy = val_data$genres %>%
                    str_detect("Comedy") %>%
                    as.numeric(),
                  crime = val_data$genres %>%
                    str_detect("Crime") %>%
                    as.numeric(),
                  documentary = val_data$genres %>%
                    str_detect("Documentary") %>%
                    as.numeric(),
                  drama = val_data$genres %>%
                    str_detect("Drama") %>%
                    as.numeric(),
                  fantasy = val_data$genres %>%
                    str_detect("Fantasy") %>%
                    as.numeric(),
                  film_noir = val_data$genres %>%
                    str_detect("Film-Noir") %>%
                    as.numeric(),
                  horror = val_data$genres %>%
                    str_detect("Horror") %>%
                    as.numeric(),
                  imax = val_data$genres %>%
                    str_detect("IMAX") %>%
```

```
                  as.numeric(),
                musical = val_data$genres %>%
                  str_detect("Musical") %>%
                  as.numeric(),
                mystery = val_data$genres %>%
                  str_detect("Mystery") %>%
                  as.numeric(),
                romance = val_data$genres %>%
                  str_detect("Romance") %>%
                  as.numeric(),
                scifi = val_data$genres %>%
                  str_detect("Sci-Fi") %>%
                  as.numeric(),
                thriller = val_data$genres %>%
                  str_detect("Thriller") %>%
                  as.numeric(),
                war = val_data$genres %>%
                  str_detect("War") %>%
                  as.numeric(),
                western = val_data$genres %>%
                  str_detect("Western") %>%
                  as.numeric()
                )
        ]

    # Finalizing validation dataset
fin_val_data <- val_data[ , 7:28]
str(fin_val_data)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  22 variables:
##  $ avg_movie_rating   : num  2.95 3.64 3.07 3.57 4.41 ...
##  $ avg_rating_per_user: num  5 5 5 2.67 2.67 ...
##  $ ng                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ action             : num  0 1 0 1 0 1 0 0 0 0 ...
##  $ adventure          : num  0 1 0 0 0 1 1 0 0 1 ...
##  $ animation          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ children           : num  0 0 1 0 0 0 0 0 1 0 ...
##  $ comedy             : num  1 0 1 0 0 0 0 0 1 1 ...
##  $ crime              : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ documentary        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ drama              : num  0 0 0 1 1 0 1 1 1 0 ...
##  $ fantasy            : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ film_noir          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ horror             : num  0 0 0 0 0 1 0 0 0 0 ...
##  $ imax               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ musical            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ mystery            : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ romance            : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ scifi              : num  0 1 0 0 0 1 0 0 0 0 ...
##  $ thriller           : num  0 1 0 0 0 1 0 0 0 0 ...
```

```
##  $ war                 : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ western             : num  0 0 0 0 0 0 1 0 0 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Results

In this section, we will be applying multiple 10-fold cross-validated XGBoost model to the dataset "final_data" which was originally derived from the edx dataset. The hyperparameters are tuned slightly for the sake of optimization while also keeping time considerations in mind and the model which gives the lowest RMSE was selected. These are the following lines of code to achieve this end:

```
# Setting parameters for XGBoost
grid_tune <- expand.grid(nrounds = c(100, 200, 300),
                         max_depth = c(2, 6, 8),
                         eta = c(0.1, 0.3, 0.5),
                         gamma = 0,
                         colsample_bytree = 1,
                         min_child_weight = 1,
                         subsample = 1
                         )

# Setting parameters for cross-validation
train_control <- trainControl(method = "cv",
                              number= 10,
                              verboseIter = FALSE,
                              allowParallel = TRUE)

# Running the XGBoost model on the final_data dataset
xgb_model <- train(x = final_data,
                   y = data$rating,
                   trControl = train_control,
                   tuneGrid = grid_tune,
                   method= "xgbTree",
                   verbose = FALSE,
                   verbosity = 0)

xgb_model

## eXtreme Gradient Boosting
##
## 9000055 samples
##      22 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8100050, 8100050, 8100050, 8100049, 8100049,
8100048, ...
## Resampling results across tuning parameters:
##
```

```
##   eta  max_depth  nrounds  RMSE       Rsquared   MAE
##   0.1  2          100      0.8708138  0.3255889  0.6749877
##   0.1  2          200      0.8705209  0.3259777  0.6743761
##   0.1  2          300      0.8703918  0.3261770  0.6741605
##   0.1  6          100      0.8697732  0.3271347  0.6734235
##   0.1  6          200      0.8695307  0.3275098  0.6731716
##   0.1  6          300      0.8693883  0.3277301  0.6730240
##   0.1  8          100      0.8694687  0.3276058  0.6731349
##   0.1  8          200      0.8692686  0.3279151  0.6729041
##   0.1  8          300      0.8691992  0.3280224  0.6728081
##   0.3  2          100      0.8710777  0.3251151  0.6750525
##   0.3  2          200      0.8705948  0.3258628  0.6743287
##   0.3  2          300      0.8704029  0.3261599  0.6740635
##   0.3  6          100      0.8695177  0.3275298  0.6731247
##   0.3  6          200      0.8692952  0.3278740  0.6728822
##   0.3  6          300      0.8692350  0.3279674  0.6727836
##   0.3  8          100      0.8693541  0.3277834  0.6729139
##   0.3  8          200      0.8694914  0.3275746  0.6729215
##   0.3  8          300      0.8697567  0.3271703  0.6730790
##   0.5  2          100      0.8710900  0.3250978  0.6744892
##   0.5  2          200      0.8704791  0.3260422  0.6740219
##   0.5  2          300      0.8702915  0.3263325  0.6738593
##   0.5  6          100      0.8695077  0.3275460  0.6730331
##   0.5  6          200      0.8694023  0.3277100  0.6728932
##   0.5  6          300      0.8694570  0.3276275  0.6728944
##   0.5  8          100      0.8697335  0.3272024  0.6731154
##   0.5  8          200      0.8702996  0.3263433  0.6734659
##   0.5  8          300      0.8709378  0.3253793  0.6739199
## 
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
## 
## Tuning parameter 'min_child_weight' was held constant at a value of 1
## 
## Tuning parameter 'subsample' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 300, max_depth = 8, eta
##  = 0.1, gamma = 0, colsample_bytree = 1, min_child_weight = 1 and
subsample = 1.
```

Now, while the RMSE is satisfactory, the Rsquared value is not that high. The first possible explanation to this is that the XGBoost model is not overfit to the final_data dataset and is quite accurate when it comes to predicting movie ratings using the derived variables so far.

Thus, it seems this model is satisfactory and we continue to calculating the RMSE with respect to the validation dataset/finalhold out set with the following lines of code:

```
xgb_pred <- predict(xgb_model, fin_val_data)

mse <- mean((validation$rating - xgb_pred)^2)
```

```
mae <- caret::MAE(validation$rating, xgb_pred)
rmse <- caret::RMSE(validation$rating, xgb_pred)

model_eval <- c(mse, mae, rmse)
model_eval

## [1] 0.7121420 0.6526068 0.8438851
```

As can be seen, the 10-fold cross-validated XGBoost model gives a satisfactory RMSE. Clearly, the chosen XGBoost model works well for accurately predicting movie ratings

Conclusion

To summarize, using the edx dataset as the original source of data, we computed the average movie rating and average reviewer rating per movie, transformed each genre into a binary variable, and developed a 10-fold cross-validated XGBoost model to accurately predict movie ratings.

The work here is limited by the lack of comparison with other models such as KNN, linear regression, etc. Also, the r-squared value for the XGBoost is quite low. The model is also limited with its use of averages, so the model might be inadequate for new movies coming out.

Accordingly, future work regarding movie prediction could lead to making a model which can predict movie ratings for even movies which do not yet have movie ratings. This could be done using a model incorporating a similarity score between movies such as the Jaccard Index. Furthermore, there could be even more tuning with hyperparameters in the shown XGBoost model to remedy the relatively low r-squared value.

Resources:

https://www.youtube.com/watch?v=qjeUhuvkbHY&t=723s