PH125.9X - Capstone: MovieLens Project

Mohammed Al-Areqi

21-Nov-23

Contents

1 Introduction

Movie recommendation systems is a machine learning model generated to predict the user behavior. Based on the user ratings, a list of movies that will be enjoyed by the user will be recommended.

2 Overview

We will be using 10M version of the MovieLens data set (https://grouplens.org/datasets/movielens/10m/). The data set contains `10,000,054` different ratings for `10,677` movies. The ratings are based on `10` different ratings starting from `0.5` as lowest rating and ends with `5` as the excellent movie rating.

The method that will be used to evaluate the performance of the model is Root Means Square Error (`RMSE`). It measures the accuracy of the model by comparing the forecast error of a model with values of a specific database. The closer the value of `RMSE` to zero the better.

The following is the `RMSE` function formula:

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

The goal of the project is to get an `RMSE` value for the model which is lower than **0.86490**.

The Capstone project provides the code which generates the data sets and separates them into training set called edx (90% of the data), used to develop the algorithm, and the test (Validation) set called final_holdout_test (10% of the data), used to test the final model developed. Since we are not allowed to build our algorithm and test them using the test set, we will further split the edx data into a train and test set.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

# MovieLens 10M data set:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file),
```

```
                                    fixed("::"), simplify = TRUE),
                           stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file),
                                  fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

3 Data Structure

A quick display of the data and summary shows the following:

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

```
str(edx, vec.len = 2)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 ...
##  $ rating   : num  5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" ...
```

The data set has 6 columns (userId, movieId, rating, timestamp, title, and genre) and 9,000,055 rows. Each row represents a rating for a movie by a user on a specific date and time.

Edx has `69878` different users, `10677` different movieId, `10676` different movie titles, `10` different ratings starting from `0.5` to `5`, and `20` different genres that produced `797` combinations of one or more genres.

```
edx %>% summarize(user_unique    = n_distinct(userId),
                  movieId_unique = n_distinct(movieId),
                  title_unique   = n_distinct(title),
                  genre_unique   = n_distinct(genres),
                  rating_unique  = n_distinct(rating)) %>%
        kable( digits = 4, format = "pipe")
```

user_unique	movieId_unique	title_unique	genre_unique	rating_unique
69878	10677	10676	797	10

MovieId and titles are supposed to be exactly the same. However, the analysis above shows that there is one more movieId than titles. To find why, we will run the following code:

```
edx %>% group_by(movieId, title) %>%
  summarize(n = n(), rating_avg = mean(rating)) %>%
  find_duplicates(title) %>%
  kable( digits = 4, format = "pipe")
```

movieId	title	n	rating_avg
34048	War of the Worlds (2005)	2460	3.1801
64997	War of the Worlds (2005)	28	2.9107

The table shows that movie title "War of the Worlds (2005)" has two movieId. Each movieId has different average rating but they are not far off and wouldn't make much significance. To fix it we would change movieId `64997` to `34048`.

```
edx %>% mutate(movieId = ifelse(movieId == 64997, 34048, movieId)) %>%
        summarize(user_unique    = n_distinct(userId),
                  movieId_unique = n_distinct(movieId),
                  title_unique   = n_distinct(title),
                  genre_unique   = n_distinct(genres),
                  rating_unique  = n_distinct(rating)) %>%
        kable( digits = 4, format = "pipe")
```

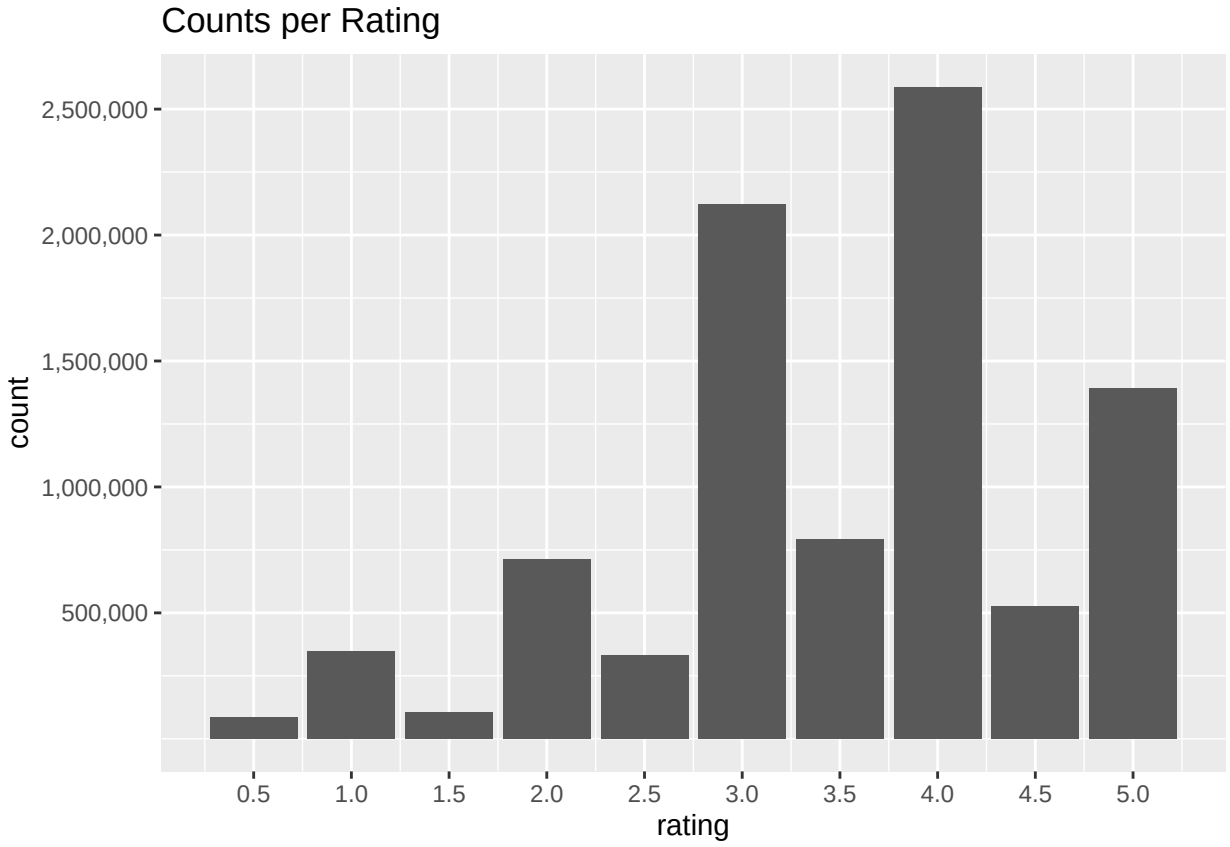
user_unique	movieId_unique	title_unique	genre_unique	rating_unique
69878	10676	10676	797	10

However, for this project we are not allowed to change any rows for the final_holdout_test data set, so we wouldn't be able to run the above code.

4 Analysis

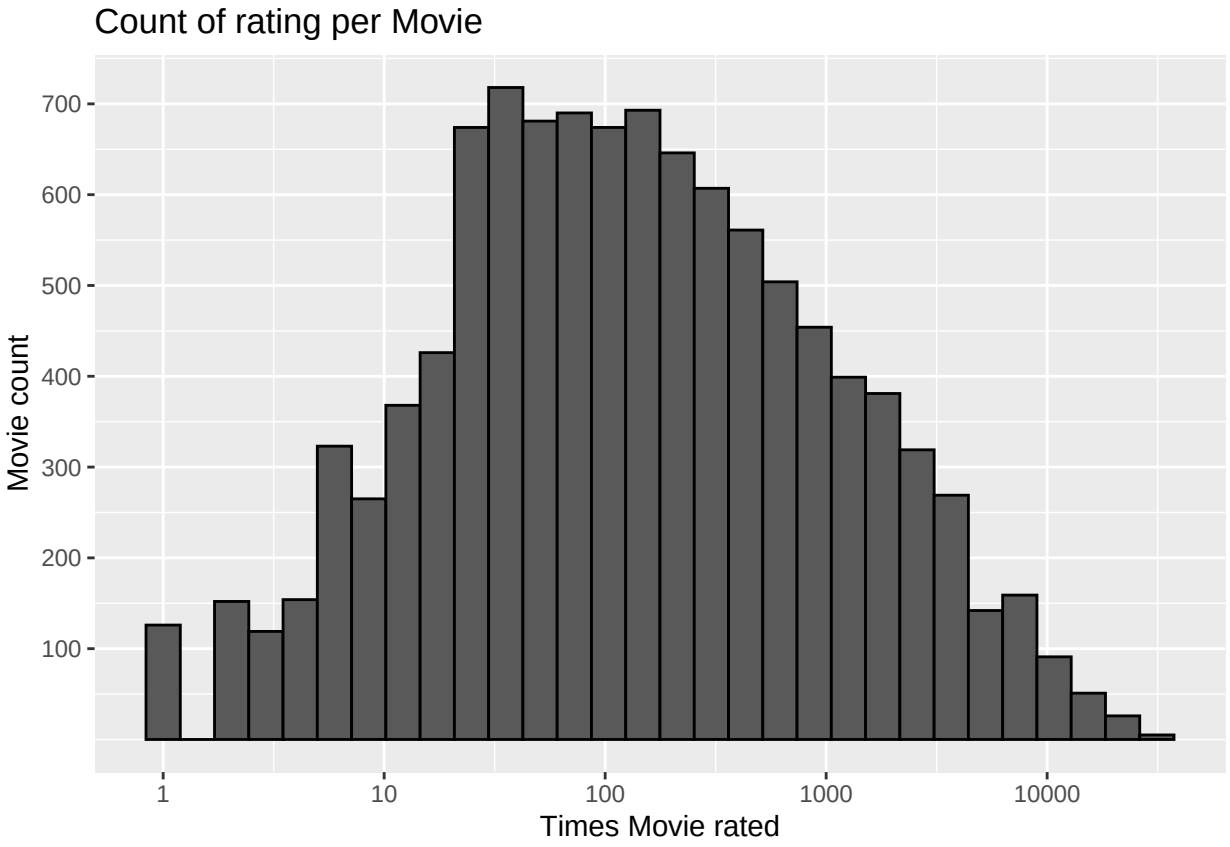
4.1 Rating Analysis

It seems that the half stars ratings (like 0.5, 1.5, 2.5, 3.5, and 4.5) are not much of a choice for users rating movies. Also you can notice from the graph below that most of the people are rating movies that they enjoy. You can see that because most of the ratings are either 3, 4 or 5 star.



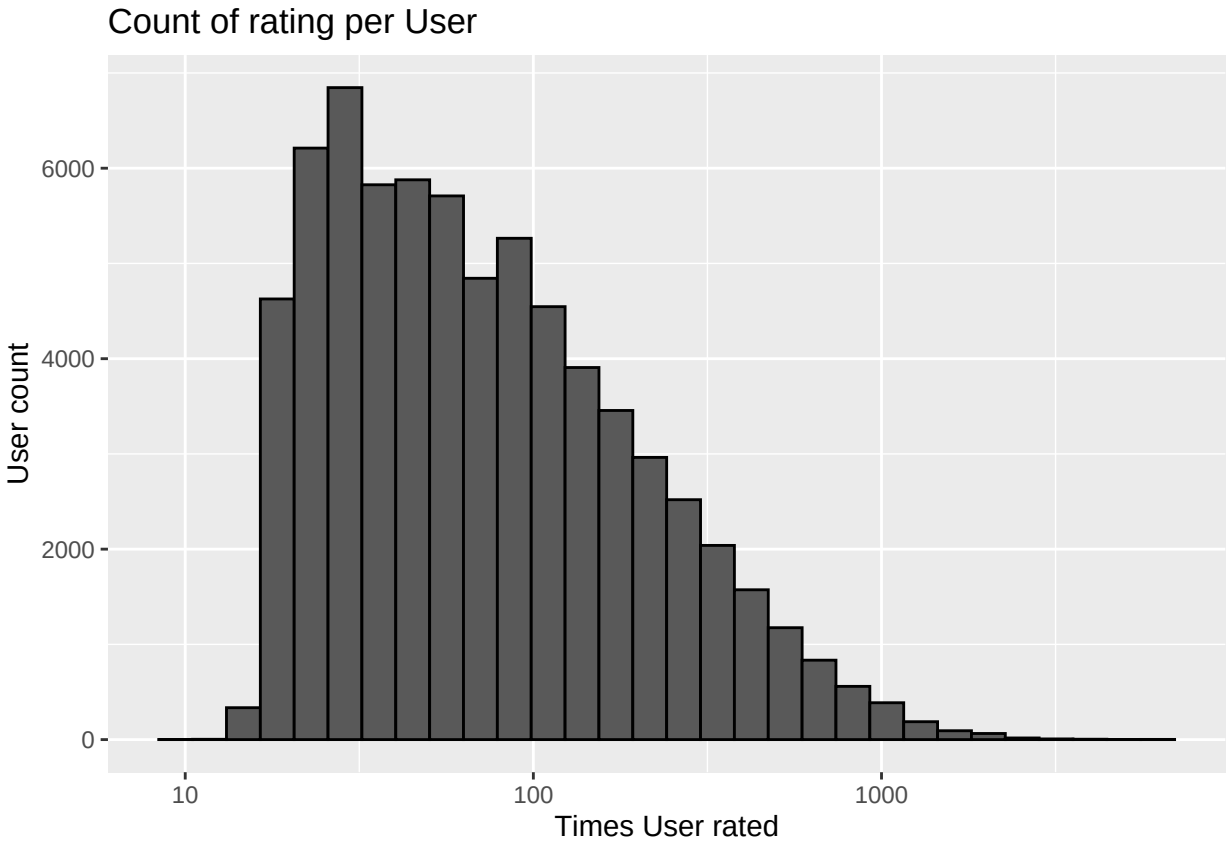
4.2 Movie Analysis

The plot below shows that all movies are not rated equally. Some movies are rated more frequently than others. Even in some circumstances some movies are rated only once.



4.3 User Analysis

Also users are not equally rating movies. Some users rate more often than other users.



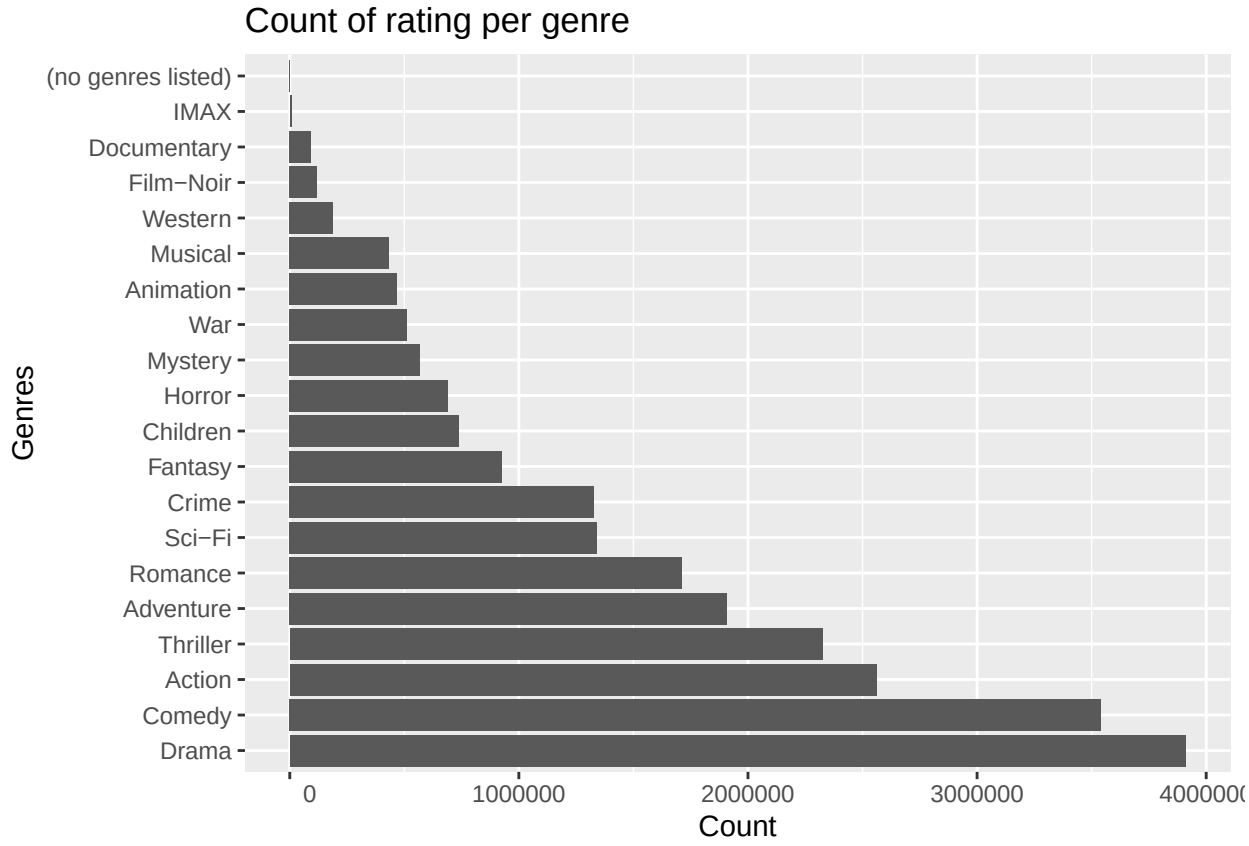
4.4 Genre Analysis

Since there are 797 genre combinations we will split the genres into single genre. This will give us 20 different genres. One movie with 7 ratings didn't have a genre. However the percentage it represents is almost zero percent of the data, so we will ignore it.

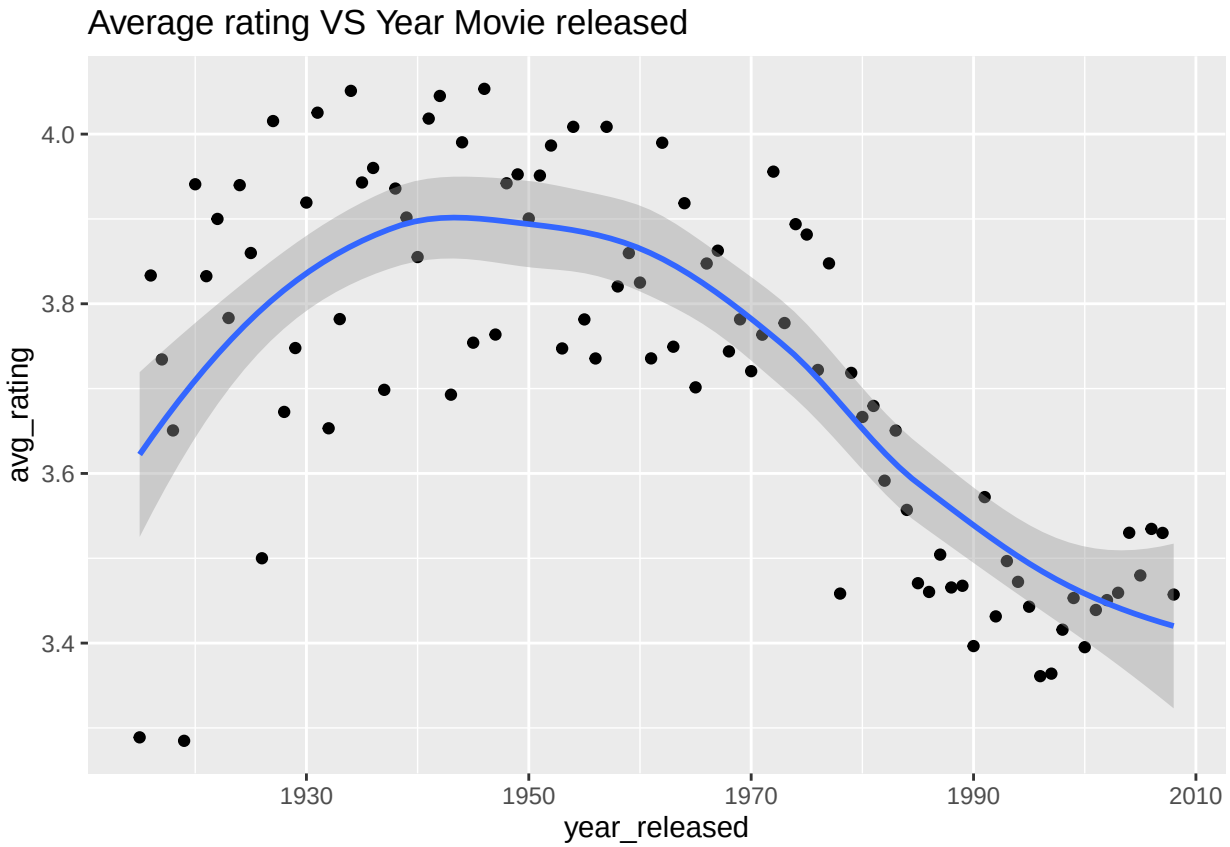
genres	count	percent
Drama	3910127	16.73%
Comedy	3540930	15.15%
Action	2560545	10.96%
Thriller	2325899	9.95%
Adventure	1908892	8.17%
Romance	1712100	7.33%
Sci-Fi	1341183	5.74%
Crime	1327715	5.68%
Fantasy	925637	3.96%
Children	737994	3.16%
Horror	691485	2.96%
Mystery	568332	2.43%
War	511147	2.19%
Animation	467168	2.00%

genres	count	percent
Musical	433080	1.85%
Western	189394	0.81%
Film-Noir	118541	0.51%
Documentary	93066	0.40%
IMAX	8181	0.04%
(no genres listed)	7	0.00%

There is a genre effect since more than `50%` of the ratings are for the top `4` genres: Drama, Comedy, Action, and Thriller.



4.5 Time Analysis



The graph shows the older the movie after 1940, the higher the rating they get. This makes sense since older movies have longer time to be watched and rated.

5 Data Separation

In order to create our model we will be using `edx` data. We are not allowed to use `final_holdout_test` to train our model. So we will separate `edx` data into training set (`edx_train_set`) which will hold about `90%` of the data and a test set (`edx_test_set`) which will have the rest of the `10%`.

```
# split edx into a training set 90% of the data and temp set 10% of the data
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
                                  list = FALSE)
edx_train_set <- edx[-test_index,]
edx_temp <- edx[test_index,]

# Make sure movieId and userId in edx_test_set are also in edx_train_set
edx_test_set <- edx_temp %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId")

# Add rows removed from edx_test_set back into edx_train_set
removed <- anti_join(edx_temp, edx_test_set)
edx_train_set <- rbind(edx_train_set, removed)

# Remove unnecessary names
rm(test_index, edx_temp,removed)
```

6 Data Modeling

Before we start our model, we will create the function that will calculate our Residual Mean Squared Error (RMSE) to measure our accuracy.

```
#RMSE function to test the prediction with the test set
RMSE <- function(test_ratings , predict_ratings){
  sqrt(mean((test_ratings - predict_ratings)^2))
}
```

6.1 Average Rating Model

We will start our model by making a naive prediction by taking the average of rating. Assuming that all movies are rated the same by all users. Then we will build on our model by adding movie, user, release year, and genre effects.

The average rating formula used is the following:

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

where $Y_{u,i}$ represent rating Y for movie i by user u and $\epsilon_{u,i}$ represent sampling independent errors.

```
# First model overall mean rating using the edx training set
mu <- mean(edx_train_set$rating, na.rm = TRUE)
mu
```

```
## [1] 3.512456
```

The average of the rating ($\mu = 3.5124556$) is used in making a naive prediction.

```
# Predict rating in edx test set with overall mean only and run the RMSE
first_rmse <- RMSE(edx_test_set$rating, mu)
results <- tibble(Method = "Just the average(mu only)" ,
                  RMSE = first_rmse,
                  Target = ifelse(first_rmse < 0.86490, "Hit", "Away"))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away

We get an RMSE = 1.0600537 which is higher from the target by a lot.

6.2 Movie Effect Model

Now we will start with the second model movie effect by adding the term b_i which represent the average rating of movie i.

We will use the following formula:

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

```r
#Second model movie effect b_i. Average rating per movie
movie_effect <-  edx_train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-mu))
movie_effect
```

```
## # A tibble: 10,677 x 2
##    movieId     b_i
##      <int>   <dbl>
##  1       1  0.415
##  2       2 -0.306
##  3       3 -0.361
##  4       4 -0.637
##  5       5 -0.442
##  6       6  0.302
##  7       7 -0.152
##  8       8 -0.397
##  9       9 -0.514
## 10      10 -0.0856
## # i 10,667 more rows
```

```r
# Predict rating in edx test set by adding the movie effect b_i and run the RMSE
y_hat_movie <- mu+edx_test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  pull(b_i)

second_rmse <- RMSE(y_hat_movie, edx_test_set$rating)

#Adding second model to the results
results <- bind_rows (results, tibble(Method = "movie effect" ,
                                      RMSE = second_rmse,
                                      Target = ifelse(second_rmse < 0.86490,
                                                      "Hit", "Away")))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away
movie effect	0.9430	Away

The movie effect gives us `RMSE =` 0.9429615 which is better than the naive prediction (average). However, it is still not close to the target.

6.3 User Effect Model

Next we will implement the user effect by adding the term b_u which represents the average rating of user u.

We will use the following formula:

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

```r
#Third model user effect b_u. Average rating per user
user_effect <- edx_train_set %>%
  left_join(movie_effect, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu-b_i))
user_effect
```

```
## # A tibble: 69,878 x 2
##    userId       b_u
##     <int>     <dbl>
##  1      1  1.67
##  2      2 -0.434
##  3      3  0.268
##  4      4  0.698
##  5      5  0.100
##  6      6  0.336
##  7      7  0.000383
##  8      8  0.203
##  9      9  0.180
## 10     10  0.102
## # i 69,868 more rows
```

```r
# Predict rating in edx test set by adding the movie b_i and user b_u effect and run the RMSE
y_hat_movie_user <- edx_test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
third_rmse <- RMSE( y_hat_movie_user, edx_test_set$rating)

#Adding third model to the results
results <- bind_rows (results, tibble(Method = "movie and user effect" ,
                                      RMSE = third_rmse,
                                      Target = ifelse(third_rmse < 0.86490,
                                                      "Hit", "Away")))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away
movie effect	0.9430	Away
movie and user effect	0.8647	Hit

The movie and user effect gives us `RMSE` = 0.8646843 which is better than the movie effect and also hit the target of having an `RMSE` which is lower than `0.86490`. However, this will not be a strong model to try and testing it on the final_holdout_test data. We need to get even a lower `RMSE` to be in the safe side.

6.4 Time Effect Model

Next we will implement the time effect by adding the term y_i which represents the average rating of the year when the movie i was released.

We will use the following formula:

$$Y_{u,i} = \hat{\mu} + b_i + b_u + y_i + \epsilon_{u,i}$$

```
#Before starting the model we would need to extract the year the movie was
#released from the train set.
edx_train_set <- edx_train_set %>%
  mutate(year_released = str_extract(title,"[(]\\d{4}[)]")) %>%
  mutate(year_released = as.numeric(str_replace_all(year_released, "[//(//)]", "")))

#Forth model time effect (y_i). Average rating per year when the movie was released
time_effect <- edx_train_set %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  group_by(year_released) %>%
  summarize(y_i = mean(rating-mu-b_i-b_u))
time_effect
```

```
## # A tibble: 94 x 2
##    year_released    y_i
##            <dbl>  <dbl>
##  1          1915 0.0883
##  2          1916 0.0353
##  3          1917 0.173
##  4          1918 0.0441
##  5          1919 0.156
##  6          1920 0.123
##  7          1921 0.0876
##  8          1922 0.0755
##  9          1923 0.132
## 10          1924 0.109
## # i 84 more rows
```

```
# add movie year_released column to the edx test set to match the edx training set.
edx_test_set <- edx_test_set %>%
  mutate(year_released = str_extract(title,"[(]\\d{4}[)]")) %>%
  mutate(year_released = as.numeric(str_replace_all(year_released, "[//(//)]", "")))

# Predict rating in edx test set by adding the movie b_i, user b_u, and
#year released y_i effect and run the RMSE
y_hat_movie_user_time <- edx_test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  left_join(time_effect, by = "year_released") %>%
  mutate(pred = mu + b_i + b_u + y_i) %>%
  .$pred
forth_rmse <- RMSE( y_hat_movie_user_time, edx_test_set$rating)

#Adding forth model to the results
results <- bind_rows (results,
                      tibble(Method = "movie, user, and time effect" ,
                             RMSE = forth_rmse,
                             Target = ifelse(forth_rmse < 0.86490,
                                             "Hit","Away")))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away
movie effect	0.9430	Away
movie and user effect	0.8647	Hit
movie, user, and time effect	0.8643	Hit

The movie, user, and time effect gives us `RMSE =` 0.8643301 which is better than the movie and user effect by 3.5422412×10^{-4}. Still the improvement is not significant to use in the test set.

6.5 Genre Effect Model

Next we will implement the genre effect by adding the term $g_{u,i}$ which represents the average rating of genres g.

We will use the following formula:

$$Y_{u,i} = \hat{\mu} + b_i + b_u + y_i + g_{u,i} + \epsilon_{u,i}$$

```
# Fifth model genre effect (g). Average rating per distinct genre
# Very slow but needed to separate the genres
genres_effect <- edx_train_set %>%
  left_join(movie_effect, by ="movieId") %>%
  left_join(user_effect, by = "userId") %>%
  left_join(time_effect, by = "year_released") %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(g = mean(rating- mu - b_i - b_u - y_i))
genres_effect
```

```
## # A tibble: 20 x 2
##    genres                    g
##    <chr>                 <dbl>
##  1 (no genres listed)  0.224
##  2 Action             -0.0120
##  3 Adventure          -0.0158
##  4 Animation          -0.0171
##  5 Children           -0.0239
##  6 Comedy             -0.00108
##  7 Crime               0.00847
##  8 Documentary         0.0551
##  9 Drama               0.0106
## 10 Fantasy            -0.00867
## 11 Film-Noir           0.0153
## 12 Horror              0.00214
## 13 IMAX               -0.0137
## 14 Musical            -0.0155
## 15 Mystery             0.0103
## 16 Romance            -0.00206
## 17 Sci-Fi             -0.0140
## 18 Thriller           -0.00315
## 19 War                 0.00210
## 20 Western            -0.00883
```

```r
# Separating edx test set genres to match the edx training set.
# It will be slow but we can't run the model without modifying test set
edx_test_set_date_genres <- edx_test_set %>%
  separate_rows(genres, sep = "\\|")

# Predict rating in edx test set by adding the movie b_i, user b_u,
#year released y_i, and genre g effect and run the RMSE
y_hat_movie_user_time_genres <- edx_test_set_date_genres %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  left_join(time_effect, by = "year_released") %>%
  left_join(genres_effect, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + y_i + g) %>%
  .$pred
fifth_rmse <- RMSE( y_hat_movie_user_time_genres, edx_test_set_date_genres$rating)

#Adding fifth model to the results
results <- bind_rows (results,
                      tibble(Method = "movie, user, time effect, and genres" ,
                             RMSE = fifth_rmse,
                             Target = ifelse(fifth_rmse < 0.86490,
                                             "Hit", "Away")))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away
movie effect	0.9430	Away
movie and user effect	0.8647	Hit
movie, user, and time effect	0.8643	Hit
movie, user, time effect, and genres	0.8624	Hit

The movie, user, time, and genre effect gives us `RMSE =` 0.8623594 which is the lowest of all our models. It is 0.0025406 lower than the target `0.86490`.

7 Results

We used the edx data set to create our model. Our final model which consists of (movie, user, time, and genre effect) gives us the lowest `RMSE` of 0.8623594, we will now use it on the Validation (final_holdout_test) data set.

Before testing the model on the Validation data set, we need to extract the release year of movies and separate the genres so we can match the data on the training set.

```
# Add a movie release date column to Final holdout data to match the edx training set.
final_holdout_test_date <- final_holdout_test %>%
  mutate(year_released = str_extract(title,"[(]\\d{4}[)]")) %>%
  mutate(year_released = as.numeric(str_replace_all(year_released, "[//(//)]", "")))

# Separate genres in Final holdout data to match the edx training set.
# Note: This will take some time.
final_holdout_test_date_genres <- final_holdout_test_date %>%
  separate_rows(genres, sep = "\\|")
```

Now we can apply our best model on the Validation data set.

```
# Predict rating in Final holdout data set by adding the movie b_i, user b_u,
#year released y_i, and genre g effect and run the RMSE
y_hat_movie_user_time_genres_final <- final_holdout_test_date_genres %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(user_effect, by = "userId") %>%
  left_join(time_effect, by = "year_released") %>%
  left_join(genres_effect, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + y_i + g) %>%
  .$pred
final_rmse <- RMSE( y_hat_movie_user_time_genres_final,
                    final_holdout_test_date_genres$rating)

#Running final model to the results
results <- bind_rows (results,
                      tibble(Method = "final validation" ,
                             RMSE = final_rmse ,
                             Target = ifelse(final_rmse < 0.86490,
                                             "Hit", "Away")))
```

Method	RMSE	Target
Just the average(mu only)	1.0601	Away
movie effect	0.9430	Away
movie and user effect	0.8647	Hit
movie, user, and time effect	0.8643	Hit
movie, user, time effect, and genres	0.8624	Hit
final validation	0.8633	Hit

After testing our best model on the Validation set we get `RMSE =` 0.8632876 which is lower than the goal `0.86490` by 0.0016124.

8 Conclusion

We first explained the project, the purpose of the MovieLens data, and the goal that needs to be achieved in the project. Then, we analyzed the data which gave us an insight on how to approach our Machine Learning model. At the end, we generated a final recommendation system that takes into consideration the movie, user, release year, and genre effects. The final model gave us `RMSE =` 0.8632876 when applied on the Validation set.

There is more room for improvement in the model if we use the Regularization. Unfortunately, it wasn't applied on this model because the model was very slow after the genres were separated.

Also, we didn't use the timestamp column in the model. It could've given us a better predication if applied to the model.

9 References

Irizarry, R.A. (2019) Introduction to data science, rafalab. Available at: https://rafalab.github.io/dsbook/.