Data Science Capstone Project: Movie Recommendations

Catherine Edis

26 January 2022

Contents

---

1 Introduction

1.1 Purpose

This project was undertaken to fulfill the requirements of the "Data Science: Capstone" course, which is part of the Professional Certificate in Data Science program offered by Harvard University through edX. The purpose of the project was to develop a movie recommendation system using the "MovieLens" dataset and the tools explained throughout the courses in the Professional Certificate program.

1.2 MovieLens

MovieLens is a non-commercial service operated by GroupLens Research at the University of Minnesota. It invites its users to rate movies, and uses "collaborative filtering" technology to provide personalized movie recommendations.

The GroupLens Research Group periodically publishes the MovieLens dataset for university students to research personalization, filtering and prediction technologies. The full dataset was last published in September

2018 and contains approximately 27 million ratings. A smaller, stable dataset - the "MovieLens 10M dataset" - was used for this project. It was released in January 2009, and contains approximately 10 million ratings. The MovieLens 10M dataset is available at: https://grouplens.org/datasets/movielens/10m

1.3 Goal of the Project

The aim of this project was to train a machine learning algorithm that used the data in one subset (the "edx" set) of the MovieLens dataset to predict movie ratings in a final hold-out test set (the "validation" set). The movie rating predictions generated by the algorithm were compared to the true ratings in the *validation* set using root mean squared error (RMSE). The goal was to achieve an RMSE of less than 0.86490.

1.4 Key Steps Performed

The following key steps were performed.

1. Load the MovieLens dataset, and create the *edx* and *validation* sets, using the code provided by edX.
2. Examine the structure of the datasets to understand the nature of the data and identify potential predictors that could be used in the modelling.
3. Where required, reformat the data to make it easier to work with.
4. Perform some basic checks on the quality of the data (for example, check for any missing values, view the range of ratings given to ensure they were in the range stated in the documentation).
5. Set up the objects required for modelling (including a training and test set generated from the *edx* set, and a function for calculating the RMSE).
6. Use the training set to develop and train various prediction models, and calculate their RMSE against the test set.
7. Compare the outcome of the models developed and select the one with the lowest RMSE.
8. Run the selected model against the entire *edx* set and use it to predict ratings for the *validation* set.
9. Calculate the RMSE of the predicted ratings compared to the true ratings in the *validation* set, and determine if it met the target of less than 0.86490.

These steps and the results that were produced are described in more detail in the following sections.

2 Analysis

2.1 Preparation

First, the code provided by *edx* was used to load the MovieLens dataset, and create the *edx* and *validation* sets. (For brevity, the code is not displayed in this report.)

Any additional libraries required for the data analysis were then installed and loaded.

```r
# Install any packages required.
if(!require(scales))
  install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")

library(scales)
library(lubridate)
```

2.2 Examination of the Data

Both the *edx* and *validation* sets were then examined to understand their structure, including the:

- class of the set;
- number of observations (rows);
- number of variables (columns); and,
- name and class of each variable.

```
# Examine both of the "edx" and "validation" sets to understand their structure.
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 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
##  - attr(*, ".internal.selfref")=<externalptr>
```

The first few rows in each set were examined to view some examples of content and the format of text fields.

```
# To see some example data, view the first few rows in the "edx" and "validation" sets.
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
```

```
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                  title
## 1:                                Dumb & Dumber (1994)
## 2:                                Jurassic Park (1993)
## 3:                                   Home Alone (1990)
## 4:                                      Rob Roy (1995)
## 5:                               Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                  genres
## 1:                               Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                      Children|Comedy
## 4:                Action|Drama|Romance|War
## 5:                          Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

The examination of the sets confirmed that:

1. Both sets had the same 6 columns.
2. The *edx* set contained 9,000,055 rows and the *validation* set contained 999,999 rows.
3. Each row appears to contain one rating given by one user for one movie.
4. For users, no further information is provided other than a unique identifying number ("userId"). This means that the set contains no user demographic data that could be incorporated into our analysis or modelling.
5. Each row includes the rating given, and the date and time the rating was given ("timestamp"). The timestamp is in a format that is not easily readable by a human.
6. Each row also includes:
   - a unique identifying number for the relevant movie ("movieId");
   - the title of the movie; and,
   - any genres the movie is associated with (eg, "Action", "Comedy").
7. The "title" variable includes the name of the movie followed by the year of release of the movie in parentheses (for example, "Boomerang (1992)").
8. Each movie has one or more "genres", with multiple genres separated by "|".

The sets were then checked for any missing values.

```
##########################################################
# Do some simple checks for missing values.
##########################################################

# Check for any NA values in any column in either set.
apply(edx, 2, function(x) any(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
```

```
apply(validation, 2, function(x) any(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
```

```
# Check columns of type "character" for any empty strings.
edx %>% filter(title =="")
```

```
## Empty data.table (0 rows and 6 cols): userId,movieId,rating,timestamp,title,genres
```

```
edx %>% filter(genres =="")
```

```
## Empty data.table (0 rows and 6 cols): userId,movieId,rating,timestamp,title,genres
```

```
validation %>% filter(title =="")
```

```
## Empty data.table (0 rows and 6 cols): userId,movieId,rating,timestamp,title,genres
```

```
validation %>% filter(genres =="")
```

```
## Empty data.table (0 rows and 6 cols): userId,movieId,rating,timestamp,title,genres
```

No missing values were found.

The *edx* set was explored further, and the number of unique movies and unique users calculated.

```
##########################################################
# Explore the larger of the two sets - "edx" -
# to further understand the nature of the data.
##########################################################

# Calculate the number of unique movies and unique users in the set.
edx %>% summarize(n_movies = n_distinct(movieId),
                  n_users = n_distinct(userId))
```

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

Another data quality check was performed, where the number of movie ids in the *edx* set was compared with the number of distinct combinations of movie id and title. If these two counts were different, this would have meant that one movie id had multiple titles recorded against it in the set, and some data cleaning would therefore be required.

```
# Calculate how many combinations of movieId and title are in the data.
# It should be the same as the number of distinct movieIds; ie, there
# should be only one title for each movieId.
n_distinct(edx$movieId, edx$title)
```

```
## [1] 10677
```

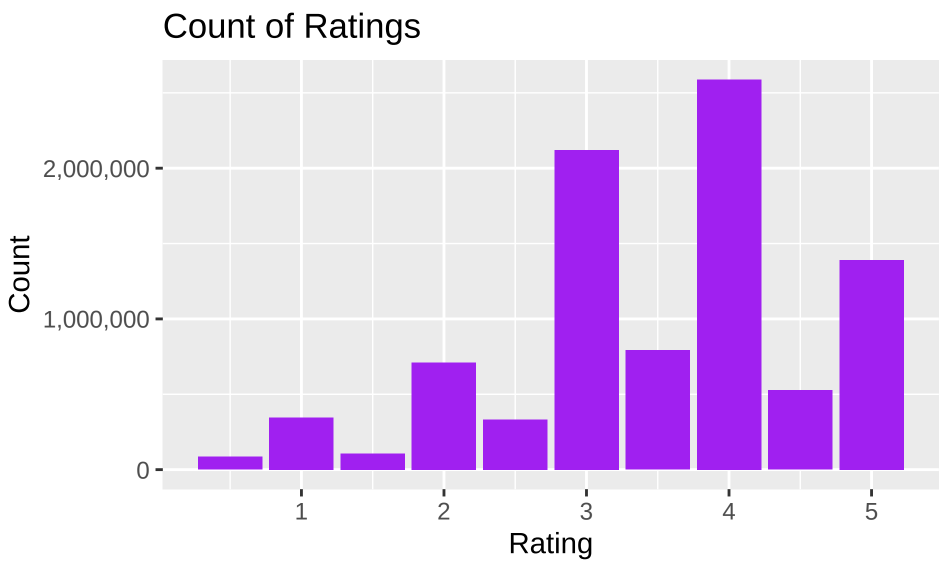
The two counts were found to be the same, so no such data cleaning was required.

Next, the values and distribution of the ratings in the *edx* set was examined.

```
# View the different ratings given and the number of movies for each rating.
edx %>% count(rating)
```

```
##     rating       n
##  1:    0.5   85374
##  2:    1.0  345679
##  3:    1.5  106426
##  4:    2.0  711422
##  5:    2.5  333010
##  6:    3.0 2121240
##  7:    3.5  791624
##  8:    4.0 2588430
##  9:    4.5  526736
## 10:    5.0 1390114
```

```
# Plot the number of times each rating was given.
edx %>% ggplot(aes(rating)) +
  geom_bar(fill = "purple") +
  scale_y_continuous(labels = comma) +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Count of Ratings")
```



This examination revealed the following:

- All ratings in the *edx* set were numeric values, of 0.5 increments between 0.5 and 5.0 inclusive.
- Whole number ratings (1, 2, 3, 4 or 5) were much more common than "half" number ratings (0.5, 1.5, 2.5, 3.5 or 4.5).
- The ratings tended to be in the more positive half of the range (ie, 3 or higher).

Finally, it was noted that, as well as the variables in the other columns, the timestamp and year of release could potentially be used in the modelling. To make this data easier to work with, both of the sets were reformatted to include these variables in individual columns, and in a more easily readable format.

```
##########################################################
# Reformat the data to make it easier to work with.
##########################################################

# Add a column to the edx set that contains the timestamp
# in a human-readable format.
edx <- edx %>% mutate(rating_datetime = as_datetime(timestamp))

# Add a column to the edx set that contains the week the
# rating was given.(This will be used in one of the models.)
edx <- edx %>% mutate(week_of_rating = round_date(rating_datetime, unit = "week"))

# Add a column to the edx set that contains the year of
```

```r
# release of the movie.(This will be used in one of the
# models.)
edx <- edx %>% mutate(year_of_release = as.numeric(str_sub(title,-5,-2)))

# Repeat the above steps for the validation set.
validation <- validation %>% mutate(rating_datetime = as_datetime(timestamp))
validation <- validation %>% mutate(week_of_rating = round_date(rating_datetime, unit = "week"))
validation <- validation %>% mutate(year_of_release = as.numeric(str_sub(title,-5,-2)))

# Examine the first few rows of the modified sets to view the new columns.
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     rating_datetime week_of_rating
## 1:                Comedy|Romance 1996-08-02 11:24:06     1996-08-04
## 2:         Action|Crime|Thriller 1996-08-02 10:58:45     1996-08-04
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01     1996-08-04
## 4:       Action|Adventure|Sci-Fi 1996-08-02 10:56:32     1996-08-04
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32     1996-08-04
## 6:       Children|Comedy|Fantasy 1996-08-02 11:14:34     1996-08-04
##    year_of_release
## 1:            1992
## 2:            1995
## 3:            1995
## 4:            1994
## 5:            1994
## 6:            1994
```

```r
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                       title
## 1:                                       Dumb & Dumber (1994)
## 2:                                       Jurassic Park (1993)
## 3:                                          Home Alone (1990)
## 4:                                             Rob Roy (1995)
## 5:                                      Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                 genres     rating_datetime week_of_rating
## 1:                              Comedy 1996-08-02 10:56:32     1996-08-04
## 2:         Action|Adventure|Sci-Fi|Thriller 1996-08-02 11:00:53     1996-08-04
## 3:                     Children|Comedy 1996-08-02 11:07:48     1996-08-04
## 4:               Action|Drama|Romance|War 1997-07-07 03:34:10     1997-07-06
## 5:                          Crime|Drama 1997-07-07 03:20:45     1997-07-06
## 6: Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-07 03:25:20     1997-07-06
##    year_of_release
## 1:            1994
## 2:            1993
## 3:            1990
## 4:            1995
## 5:            1972
## 6:            1997
```

2.3 Modelling Approach

The modelling approach adopted for this project was based on linear regression. Six models were trained using a training subset of *edx*. For each model, the accuracy of its predictions was measured using RMSE against a test subset of *edx* that contained known ("true") ratings.

The modelling started with three models used in the Machine Learning course:

- a "naive" model, which simply used the average rating of all movies to predict ratings;
- a "movie effect" model, which started with the "naive" model, but modified it to incorporate the average rating for each movie; and,
- a "user effect" model, which started with the "movie effect" model, but modified it to incorporate the average rating given by each user.

This approach was then built upon and expanded by adding the other variables previously identified in the *edx* set, with one model for each variable:

- a "time effect" model, which started with the "user effect" model, but modified it to incorporate the average rating given in each week;
- a "year of release effect" model, which started with the "time effect" model, but modified it to incorporate the average rating for each year of release; and,
- a "genres effect" model, which started with the "year of release effect" model, but modified it to incorporate the average rating given for each genre (or combination of genres).

In preparation for the modelling:

- a training and test set was created from the *edx* set. They comprised approximately 90% and 10% of the *edx* set respectively; and,
- a function was defined to calculate the RMSE.

```r
############################################################
# Set up the objects required for the modelling.
############################################################

# Create Training and Test Sets from the "edx" set.

# The following line of code assumes that R 3.6 or later
# is being used.
set.seed(1, sample.kind = "Rounding")

# Create a training set that is 90% of the edx set,
# and a test set that is 10% of the edx set.
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train_set <- edx %>% slice(-edx_test_index)
temp <- edx %>% slice(edx_test_index)

# For some of the calculations we make later, the users,
# movies, week_of_ratings, year of release, and genres
# that are in the test set must also be in the training set.
# Therefore, at this point we remove any rows in the test
# set with a user, movie, week_of_rating, year of release,
# or genres that are not in the training set.
edx_test_set <- temp %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId") %>%
  semi_join(edx_train_set, by = "week_of_rating") %>%
  semi_join(edx_train_set, by = "year_of_release") %>%
  semi_join(edx_train_set, by = "genres")

# Add rows removed from the test set back into the
# training set.
removed <- anti_join(temp, edx_test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres", "rating_datetime", "w
```

```r
edx_train_set <- rbind(edx_train_set, removed)

# Define a function to calculate the root mean squared
# error (RMSE) of a model.
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The results produced for each model are included below.

3 Results

3.1 Naive Model - Average Rating of All Movies

The first model simply used the average rating of all movies to predict ratings.

```r
##########################################################
# Naive Average Rating Model
# --------------------------
# We start with a very simple model, where every rating is
# predicted to be the same as the average rating of all
# movies in the training set.
##########################################################

# Calculate the average rating for the training set.
mu <- mean(edx_train_set$rating)
mu
```

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

```r
# Use this as the predicted rating for all rows in the test set,
# and calculate the RMSE (distance) between it and the
# actual ratings in the test set.
naive_rmse <- RMSE(edx_test_set$rating, mu)
naive_rmse
```

```
## [1] 1.060054
```

```
rmse_results <- tibble(
  Method = "Naive Model - Average Rating of All Movies",
  RMSE = naive_rmse)

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.060054

As displayed above, when this model was run against the *edx* test set, its predictions were not very accurate - with an RMSE of 1.060054. The effect of other variables was then explored to determine if they improved the accuracy.

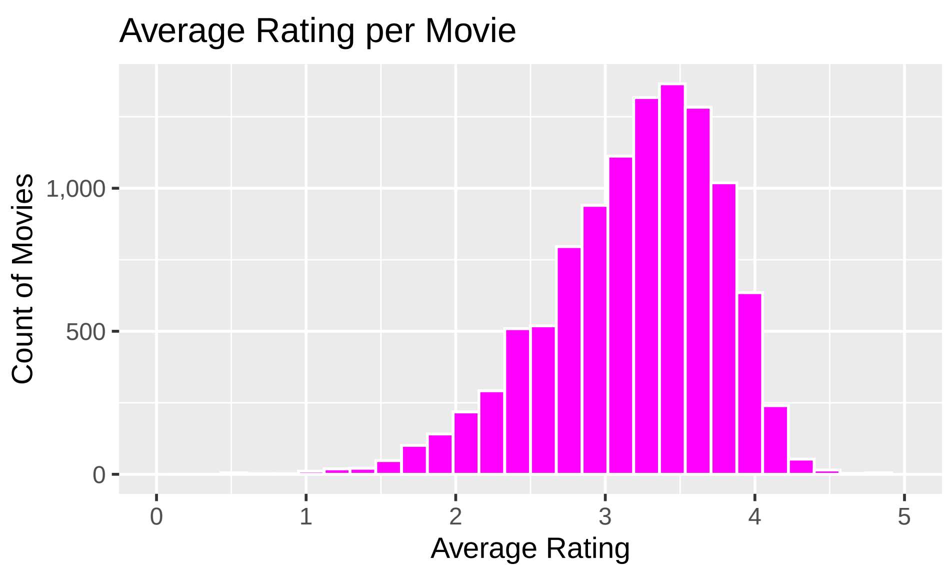
3.2 Movie Effect Model

The second model started with the first, "naive" model, but modified it to incorporate the average rating for each movie - also known as the "movie effect".

The average rating for each movie was calculated, and the distribution plotted.

```
##########################################################
# Movie Effect Model
# -------------------
# Augments the previous model by incorporating the average
# rating for each movie.
##########################################################

# Compute the average rating for each movie in the training
# set, and plot the distribution.
edx_train_set %>%
    group_by(movieId) %>%
    summarize(avg_i = mean(rating)) %>%
    ggplot(aes(avg_i)) +
    geom_histogram(bins = 30, color = "white", fill = "magenta") +
    ggtitle("Average Rating per Movie") + xlab("Average Rating") +
    ylab("Count of Movies") +
    scale_y_continuous(labels = comma) +
    scale_x_continuous(limits = c(0, 5))
```



The plot indicated that the average rating for each movie varied somewhat depending on the individual movie. This suggested that the RMSE of the Movie Effect model would be an improvement on that of the Naive model.

Note that the movie effects could have been calculated using the lm() function. However, because of the relatively large size of this dataset, the lm() function would take a relatively long time to run. Instead, the movie effect was calculated as described in the code comment below.

```
# For each movie in the training set: subtract the overall
# average rating across all movies (mu) from each rating
# that movie was given, and calculate the average. This is
# denoted as "b_i", and gives us the "movie effect" for
# each movie in the training set.
movie_avgs <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Use the movie effects calculated above to predict the
# rating for each row in the test set.
predicted_ratings <- mu + edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Calculate the RMSE using the actual ratings in the test
# set and the predicted ratings calculated above.
movie_effects_rmse <- RMSE(edx_test_set$rating, predicted_ratings)
movie_effects_rmse
```

```
## [1] 0.9429615
```

```
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Movie Effect Model",
         RMSE = movie_effects_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615

As shown above, when the model was run against the *edx* test set, the RMSE improved somewhat.

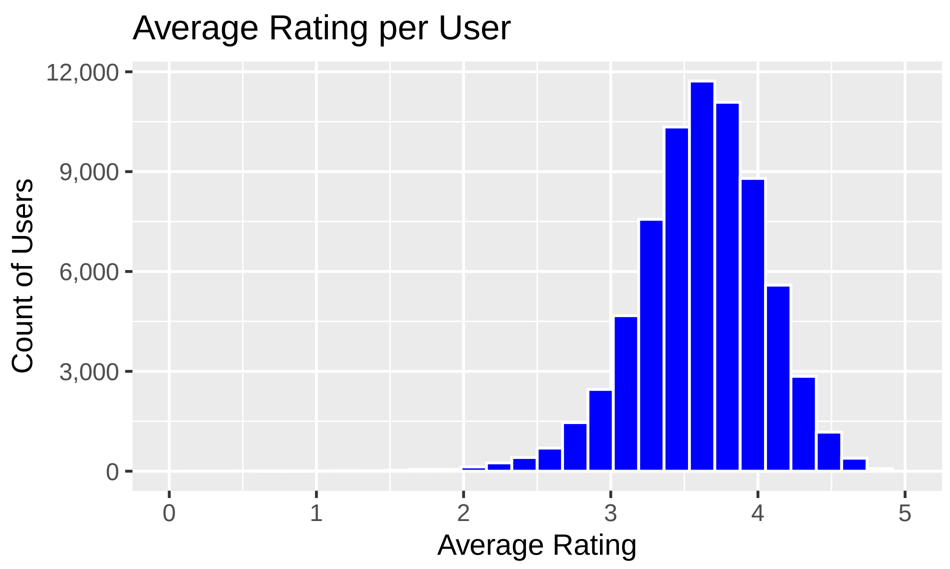
3.3 User Effect Model

The third model expanded on the Movie Effect model by including the average of the ratings given by each user - also known as the “user effect”.

The average rating given by each user was calculated, and the distribution plotted.

```r
##########################################################
# Movie + User Effects Model
# --------------------------
# Augments the previous model by incorporating the average
# rating for each user.
##########################################################

# Compute the average rating for each user in the training
# set, and plot the distribution.
edx_train_set %>%
    group_by(userId) %>%
    summarize(avg_u = mean(rating)) %>%
    ggplot(aes(avg_u)) +
    geom_histogram(bins = 30, color = "white", fill = "blue") +
    ggtitle("Average Rating per User") + xlab("Average Rating") +
    ylab("Count of Users") +
    scale_y_continuous(labels = comma) +
    scale_x_continuous(limits = c(0, 5))
```



The plot above indicated that the average rating given by each user varied somewhat.This suggested that the RMSE of the User Effect model would be an improvement on that of the Movie Effect model.

```
# Apply a similar approach to that used for movie effects
# to calculate the "user effect" for each user ("b_u")
# in the training set.
# Incorporate the user effects into the model we used
# for movie effects. This new model includes both the
# movie effects (b_i), and user effects (b_u).
user_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Use this model to predict ratings for all rows in the
# test set, and calculate the RMSE.
predicted_ratings <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

user_effects_rmse <- RMSE(edx_test_set$rating, predicted_ratings)
user_effects_rmse
```

```
## [1] 0.8646843
```

```
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Movie and User Effects Model",
         RMSE = user_effects_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615
Movie and User Effects Model	0.8646843

As shown above, when the model was run against the *edx* test set, the RMSE improved again. In fact, when this model was used to predict ratings in the *edx* test set, the RMSE was slightly less than the goal of 0.8649. However, the goal of this project was to achieve an RMSE of less than 0.8649 against the **validation** set. There was no guarantee at this point that the User Effect model would achieve this.

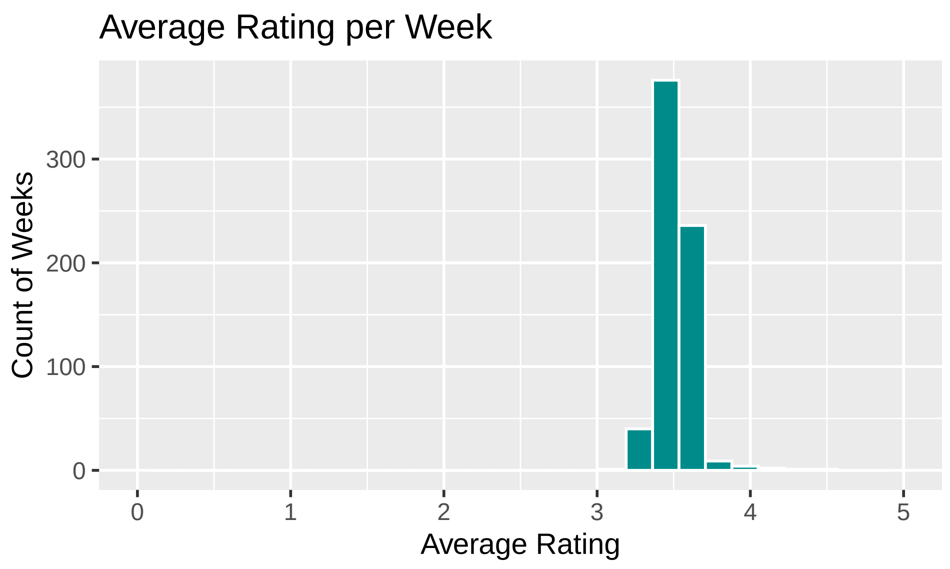
The exploration of the data that was previously undertaken had identified three other variables that could be used in the modelling: the week in which the rating was given, the year of release of the movie, and the genre(s) of the movie. These variables were progressively incorporated into the following three models.

3.4 Time Effect Model

The fourth model expanded on the User Effect model by including the average of the ratings given each week - referred to here as the “time effect”.

```
##########################################################
# Movie + User + Time Effects Model
# --------------------------------
# Augments the previous model by incorporating the average
# rating for each week.
##########################################################

# Compute the average rating per week based on the
# timestamp in the training set, and plot the distribution.
edx_train_set %>%
    group_by(week_of_rating) %>%
    summarize(avg_w = mean(rating)) %>%
    ggplot(aes(avg_w)) +
    geom_histogram(bins = 30, color = "white", fill = "dark cyan") +
    ggtitle("Average Rating per Week") + xlab("Average Rating") +
    ylab("Count of Weeks") +
    scale_y_continuous(labels = comma) +
    scale_x_continuous(limits = c(0, 5))
```



The plot indicated that, in general, the average rating did not differ significantly in different weeks. This suggested that incorporating the time effect into the model would produce little, if any, improvement.

```
# Using the week of the timestamp for each rating in the
# training set, calculate the "time effect" for each week
# ("b_w").
# Incorporate the time effects into the model we used
# above. This new model includes the movie effects (b_i),
# user effects (b_u), and time effects (b_w).
week_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(week_of_rating) %>%
  summarize(b_w = mean(rating - mu - b_i - b_u))

# Use this model to predict ratings for all rows in the
# test set, and calculate the RMSE.
predicted_ratings <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(week_avgs, by='week_of_rating') %>%
  mutate(pred = mu + b_i + b_u + b_w) %>%
  pull(pred)

time_effects_rmse <- RMSE(edx_test_set$rating, predicted_ratings)
time_effects_rmse
```

```
## [1] 0.8645933
```

```
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Movie, User and Time Effects Model",
         RMSE = time_effects_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615
Movie and User Effects Model	0.8646843
Movie, User and Time Effects Model	0.8645933

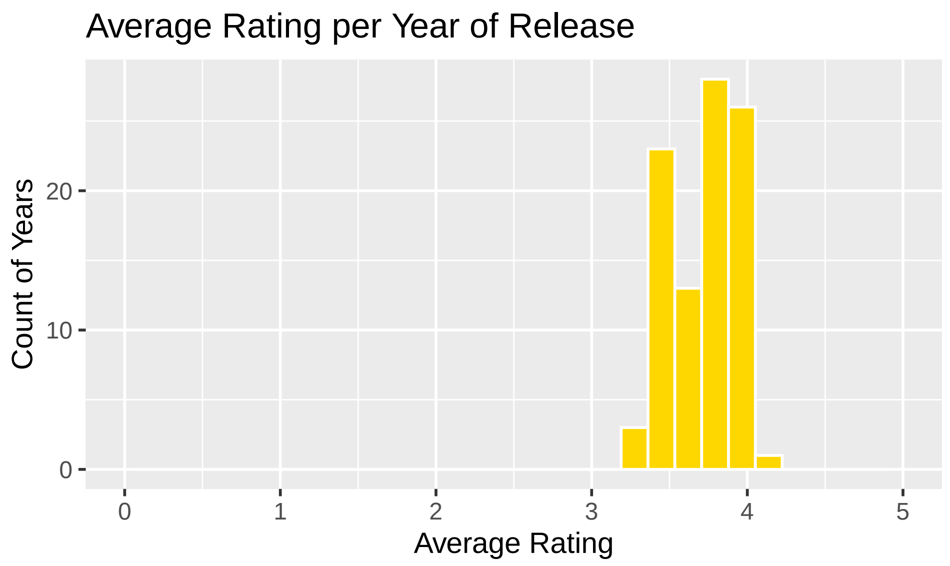
As shown above, when run against the *edx* test set, this model resulted in a slight improvement on the RMSE.

3.5 Year of Release Effect Model

The fifth model expanded on the Time Effect model by including the average rating for each year of release - the "year of release effect".

```
##########################################################
# Movie + User + Time + Year of Release Effects Model
# ---------------------------------------------------
# Augments the previous model by incorporating the average
# rating for the year of release of the movies.
##########################################################

# Compute the average rating for each year of release
# in the training set, and plot the distribution.
edx_train_set %>%
    group_by(year_of_release) %>%
    summarize(avg_y = mean(rating)) %>%
    ggplot(aes(avg_y)) +
    geom_histogram(bins = 30, color = "white", fill = "gold") +
    ggtitle("Average Rating per Year of Release") + xlab("Average Rating") +
    ylab("Count of Years") +
    scale_y_continuous(labels = comma) +
    scale_x_continuous(limits = c(0, 5))
```



It appeared from the plot that the average rating varied slightly between different years of release.

```
# Using the year of release for each movie in the
# training set, calculate the "year effect" for each year
# of release ("b_y").
# Incorporate the year effects into the model we used
# above. This new model includes the movie effects (b_i),
# user effects (b_u), time effects (b_w), and year
# effects (b_y).
year_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(week_avgs, by='week_of_rating') %>%
  group_by(year_of_release) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_w))

# Use this model to predict ratings for all rows in the
# test set, and calculate the RMSE.
predicted_ratings <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(week_avgs, by='week_of_rating') %>%
  left_join(year_avgs, by='year_of_release') %>%
  mutate(pred = mu + b_i + b_u + b_w + b_y) %>%
  pull(pred)

year_effects_rmse <- RMSE(edx_test_set$rating, predicted_ratings)
year_effects_rmse
```

```
## [1] 0.8642348
```

```
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Movie, User, Time and Year of Release Effects Model",
         RMSE = year_effects_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615
Movie and User Effects Model	0.8646843
Movie, User and Time Effects Model	0.8645933
Movie, User, Time and Year of Release Effects Model	0.8642348

When run against the *edx* test set, this model resulted in a slight improvement on the RMSE (although more than that produced by including the week of the rating).

3.6 Genres Effect Model

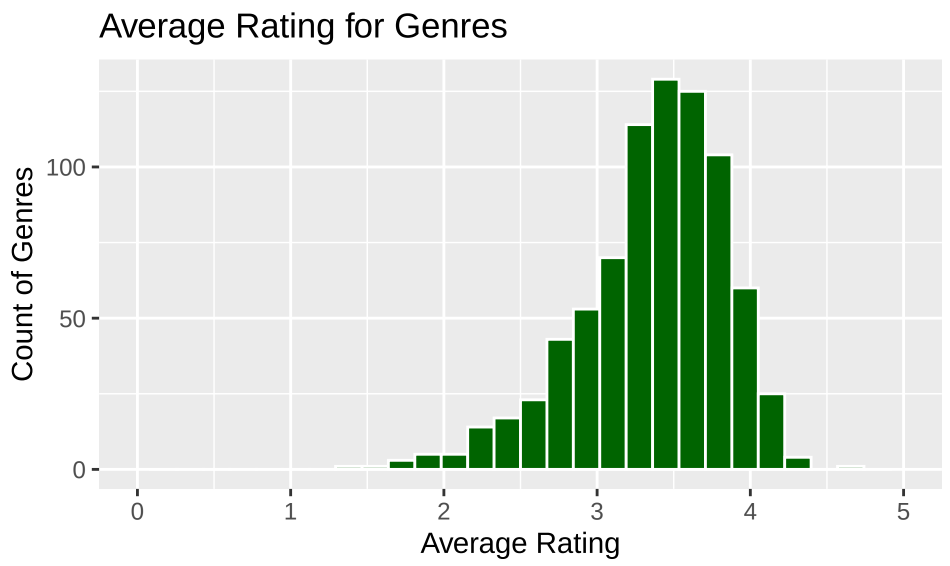
The sixth model expanded on the Year of Release Effect model by including the average of the ratings given to movies in each genre or combination of genres - the "genres effect".

Initially, separate_rows() was used on the "genres" variable in an attempt to calculate the mean rating for each individual genre (but not for combinations of genres). For example, if a movie had a value in the "genres" column of "Action|Comedy", the rating for the movie would be included in the calculation of the average rating for all "Action" movies, as well as the average rating for all "Comedy" movies. There would be no average calculated for the "Action|Comedy" combination of genres.

However, when this approach was attempted, the code ran for a very long period of time and eventually produced an error, reporting insufficient computing resources. The approach was therefore changed, so that values in the "genres" column were used "as is". This meant that the average rating was calculated for each value of "genres" present in the dataset - whether that represented an individual genre (eg, "Action") or a combination of genres (eg, "Action|Comedy").

```
##########################################################
# Movie + User + Time + Year + Genres Effects Model
# -------------------------------------------------
# Augments the previous model by incorporating the average
# rating for each value of "genres".
# The value of "genres" may be a single genre or a
# combination of two or more genres.
# For movies that did not have a genre recorded, the value
# of "genres" was set to "(no genres listed)".
##########################################################

# Compute the average rating for each combination of genres in the training
# set, and plot the distribution.
edx_train_set %>%
    group_by(genres) %>%
    summarize(avg_g = mean(rating)) %>%
    ggplot(aes(avg_g)) +
    geom_histogram(bins = 30, color = "white", fill = "dark green") +
    ggtitle("Average Rating for Genres") + xlab("Average Rating") +
    ylab("Count of Genres") +
    scale_y_continuous(labels = comma) +
    scale_x_continuous(limits = c(0, 5))
```



The plot above indicated that the average rating given to movies in each genre (or combination of genres) varied somewhat.This suggested that the RMSE of the Genres Effect model would be an improvement on that of the Year of Release Effect model.

```
# Using the "genres" for each movie in the training set,
# calculate the "genres effect" for each value of "genres"
# ("b_g").
# Incorporate the genres effects into the model we used
# above. This new model includes the movie effects (b_i),
# user effects (b_u), time effects (b_w), year effects
# (b_y), and genres effect (b_g).
genre_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(week_avgs, by='week_of_rating') %>%
  left_join(year_avgs, by='year_of_release') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_w - b_y))

# Use this model to predict ratings for all rows in the
# test set, and calculate the RMSE.
predicted_ratings <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(week_avgs, by='week_of_rating') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year_of_release') %>%
  mutate(pred = mu + b_i + b_u + b_w + b_y + b_g) %>%
  pull(pred)

genre_effects_rmse <- RMSE(edx_test_set$rating, predicted_ratings)
genre_effects_rmse
```

```
## [1] 0.8639852
```

```r
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Movie, User, Time, Year and Genre Effects Model",
         RMSE = genre_effects_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615
Movie and User Effects Model	0.8646843
Movie, User and Time Effects Model	0.8645933
Movie, User, Time and Year of Release Effects Model	0.8642348
Movie, User, Time, Year and Genre Effects Model	0.8639852

When run against the *edx* test set, this model resulted in another slight improvement on the RMSE, and produced the lowest RMSE of all the models used so far.

The RMSE produced by this model, when run against the *edx* test set, was less than the goal of 0.8649. This did not necessarily mean that it would achieve an RMSE of less than 0.8649 against the **validation** set. However, at this point, the potential predictors available in the MovieLens dataset had been exhausted. Therefore, the genres model was selected as the final model to be used to predict ratings in the *validation* set.

3.7 Predictions for Ratings in the Validation Set

The final model - the genres model - was run against the entire *edx* set, and then used to predict ratings for the *validation* set.

```r
##########################################################
# Predict Ratings in the Validation Set
# ----------------------------------------
# Re-calculates the effects used in the final model based on
# the entire edx set, then uses the final model to produce
# predicted ratings for the validation set.
##########################################################

# Calculate the average rating for the entire edx set.
edx_mu <- mean(edx$rating)

# Calculate the movie effect for the entire edx set.
edx_movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - edx_mu))

# Calculate the user effect for the entire edx set.
edx_user_avgs <- edx %>%
  left_join(edx_movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - edx_mu - b_i))

# Calculate the time effect for the entire edx set.
edx_week_avgs <- edx %>%
  left_join(edx_movie_avgs, by='movieId') %>%
  left_join(edx_user_avgs, by='userId') %>%
  group_by(week_of_rating) %>%
  summarize(b_w = mean(rating - edx_mu - b_i - b_u))

# Calculate the year of release effect for the entire edx set.
edx_year_avgs <- edx %>%
  left_join(edx_movie_avgs, by='movieId') %>%
  left_join(edx_user_avgs, by='userId') %>%
  left_join(edx_week_avgs, by='week_of_rating') %>%
  group_by(year_of_release) %>%
  summarize(b_y = mean(rating - edx_mu - b_i - b_u - b_w))

# Calculate the genre effect for the entire edx set.
edx_genre_avgs <- edx %>%
  left_join(edx_movie_avgs, by='movieId') %>%
  left_join(edx_user_avgs, by='userId') %>%
  left_join(edx_week_avgs, by='week_of_rating') %>%
  left_join(edx_year_avgs, by='year_of_release') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - edx_mu - b_i - b_u - b_w - b_y))

# Use the values calculated above to predict ratings for
# the validation set.
validation_predicted_ratings <- validation %>%
  left_join(edx_movie_avgs, by='movieId') %>%
  left_join(edx_user_avgs, by='userId') %>%
  left_join(edx_week_avgs, by='week_of_rating') %>%
  left_join(edx_year_avgs, by='year_of_release') %>%
  left_join(edx_genre_avgs, by='genres') %>%
  mutate(pred = edx_mu + b_i + b_u + b_w + b_y + b_g) %>%
  pull(pred)

# Calculate the RMSE using the actual ratings in the
# validation set and the predicted ratings calculated
# above.
validation_final_rmse <-RMSE(validation$rating, validation_predicted_ratings)
validation_final_rmse
```

```
## [1] 0.8646133
```

```r
rmse_results <- bind_rows(rmse_results,
  tibble(Method="Final Model on Validation Set",
         RMSE = validation_final_rmse))

knitr::kable(rmse_results)
```

Method	RMSE
Naive Model - Average Rating of All Movies	1.0600537
Movie Effect Model	0.9429615
Movie and User Effects Model	0.8646843
Movie, User and Time Effects Model	0.8645933
Movie, User, Time and Year of Release Effects Model	0.8642348
Movie, User, Time, Year and Genre Effects Model	0.8639852
Final Model on Validation Set	0.8646133

The RMSE calculated when the final model was used to predict ratings in the *validation* set was slightly less than 0.86490. Thus, the goal was achieved.

4 Conclusion

This project explored the use of linear regression to predict movie ratings in the MovieLens 10M dataset, with each variable available in the dataset incorporated into the modelling at some point.

Incorporation of each of the movie and user effects into the prediction model produced reasonable improvements on the RMSE against the *edx* test set. In contrast, incorporation of each of the time, year of release, and genres effects produced only slight improvements. It was sufficient however, to achieve the project goal of an RMSE of less than 0.86490 against the *validation* set.

There were some limitations on the analysis that could be performed, including:

- the inability to explore the effect of individual genres (as opposed to combinations of them) due to computing resource constraints; and,
- the inability to explore the effect of user characteristics on ratings, due to the lack of any user demographic data in the MovieLens 10M dataset.

While the goal of the project was achieved, avenues that could be explored in the future to further improve the accuracy of the predictions include the use of other techniques, such as:

- regularization;
- matrix factorization; and,
- other, non-linear, modelling techniques.