MovieLens Recommendation System

Katie Sharp

6/2/2020

Contents

Preface

This data science project is required for the ninth and final course in the HarvardX Professional Certificate in Data Science (HarvardX: PH125.9x). The overall purpose of this course is apply the R data analysis skills we gained throughout the certificate program to a real-world problem.

Introduction

The goal of the project is to create a movie recommendation system using the MovieLens dataset. We are working with a condensed version of the MovieLens dataset that contains more than 10 million ratings for 10,681 movies by 71,567 users of the online movie recommender service MovieLens. Per the dataset documentation, the users were selected at random and all users had rated at least 20 movies.

Recommendation systems are one of the most widely used applications of machine learning technologies. They use ratings by users for a specific item to make product recommendations for a different user. For example, Netflix uses a recommendation system to predict the user rating for a specific movie. The rating is on a scale of one to five stars, whereas one star indicates a bad movie and five stars indicates an excellent movie.

Our objective is to build a movie recommendation system similar to Netflix. We will use the MovieLens dataset to train a machine learning algorithm and build a predictive model that will evaluated by measuring RMSE (Root Mean Squared Error). The ultimate goal is achieve a RMSE lower than 0.8649 with our recommendation system.

The key steps in the project include: data import and preprocessing, data exploration and visualization, methods/analysis, results summary and model performance, and important conclusions.

Data Import and Preprocessing

Here are the required libraries to load for the project.

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

First we download the data from the MovieLens website and prepare it for analysis. We ultimately will create a movielens data file that contains information about the users, movies and ratings.

```r
# Links for MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies, stringsAsFactors = TRUE) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Before splitting the data into training and validation sets, let's get a quick overview of the entire dataset. We see it is a dataframe with more than 10 million rows and six columns.

```r
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

The dataset it is in a tidy format with row representing one observation (a rating given by one user for one movie), and the column names are the features for each observation. For our analysis, the `rating` is the outcome (y) we are trying to predict, and the other variables are the features: user information is contained in the `userId`, the movie information is stored in the `movieId` and `title` columns, the date of the rating is stored in `timestamp`, and the movie category information is provided by `genres`.

```r
movielens %>% summarise(n_movies = n_distinct(movieId),
                        n_users = n_distinct(userId),
                        n_genres = n_distinct(genres))
```

```
##   n_movies n_users n_genres
## 1    10677   69878      797
```

There are 10,677 movies, 69,878 users and 797 genres in our large dataset. Here are some basic summary statistics. Note that ratings can have "half stars", and the timestamp is given in seconds since midnight UTC of January 1, 1970.

```r
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Now we create a `validation` set, which will be 10 percent of the movielens data, and will **only be used for a final test of our algorithm**. The remaining data will be contained in file called `edx` and will be used for training and testing our algorithm.

```
set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
 edx <- movielens[-test_index,]
 temp <- movielens[test_index,]

validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
 edx <- rbind(edx, removed)

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

Data Exploration and Visualization

After a glance at the edx dataset we see that we want to convert the `timestamp` column to a more user-friendly format. Also, the `title` column includes the release year in parentheses, so we want to extract that information to a separate column.

```
head(edx)
```

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

We add a column `year_rated` for when the movie was rated and `year_release` for when the movie was released.

```
edx_new <- edx %>% mutate(timestamp = as_datetime(timestamp), year_rated = year(as_datetime(timestamp))
head(edx_new)
```

```
##   userId movieId rating           timestamp                         title
## 1      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3      1     292      5 1996-08-02 10:57:01              Outbreak (1995)
## 4      1     316      5 1996-08-02 10:56:32              Stargate (1994)
## 5      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
## 6      1     355      5 1996-08-02 11:14:34       Flintstones, The (1994)
##                          genres year_rated year_release
## 1                Comedy|Romance       1996         1992
## 2         Action|Crime|Thriller       1996         1995
## 3  Action|Drama|Sci-Fi|Thriller       1996         1995
## 4       Action|Adventure|Sci-Fi       1996         1994
## 5 Action|Adventure|Drama|Sci-Fi       1996         1994
## 6       Children|Comedy|Fantasy       1996         1994
```

Before we build and train our model, we need to explore our `edx_new` dataset and the distribution of ratings to see if we can find any significant patterns. We now have about 9 million observations and eight variables.

```
str(edx_new)
```

```
## '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   : POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ title       : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres      : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Actio
##  $ year_rated  : num  1996 1996 1996 1996 1996 ...
##  $ year_release: num  1992 1995 1995 1994 1994 ...
```

Ratings

First we'll examine our outcome data, the ratings. They go from 1 to 5, with "half-stars" between each whole number.

```
edx_new %>% group_by(rating) %>%
  summarise(n = n())
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```

Interestingly, the most ratings were "4" and there appears to be more "above average" ratings (greater than 3) than "below average" (lesser than 3). We can confirm this:

```r
mean(edx_new$rating > 3)
```

```
## [1] 0.5885413
```

```r
mean(edx_new$rating < 3)
```

```
## [1] 0.1757668
```

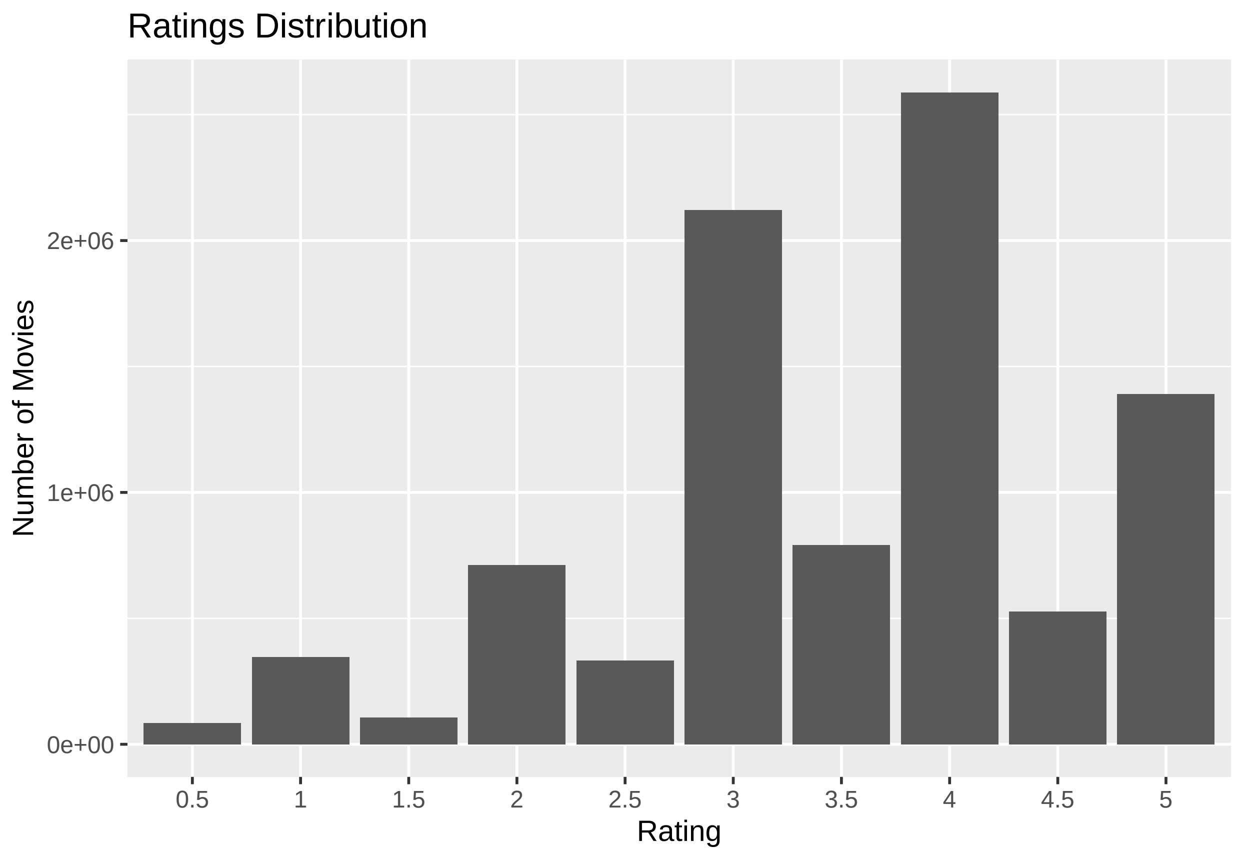
Also, interestingly, more than half of the ratings are either a 3 or a 4.

```r
mean(edx_new$rating ==3 | edx_new$rating ==4)
```

```
## [1] 0.5232935
```

A plot of the ratings distribution shows that users are more likely to rate a movie favorably and there are a higher proportion of "whole" ratings than ratings with "half-stars".

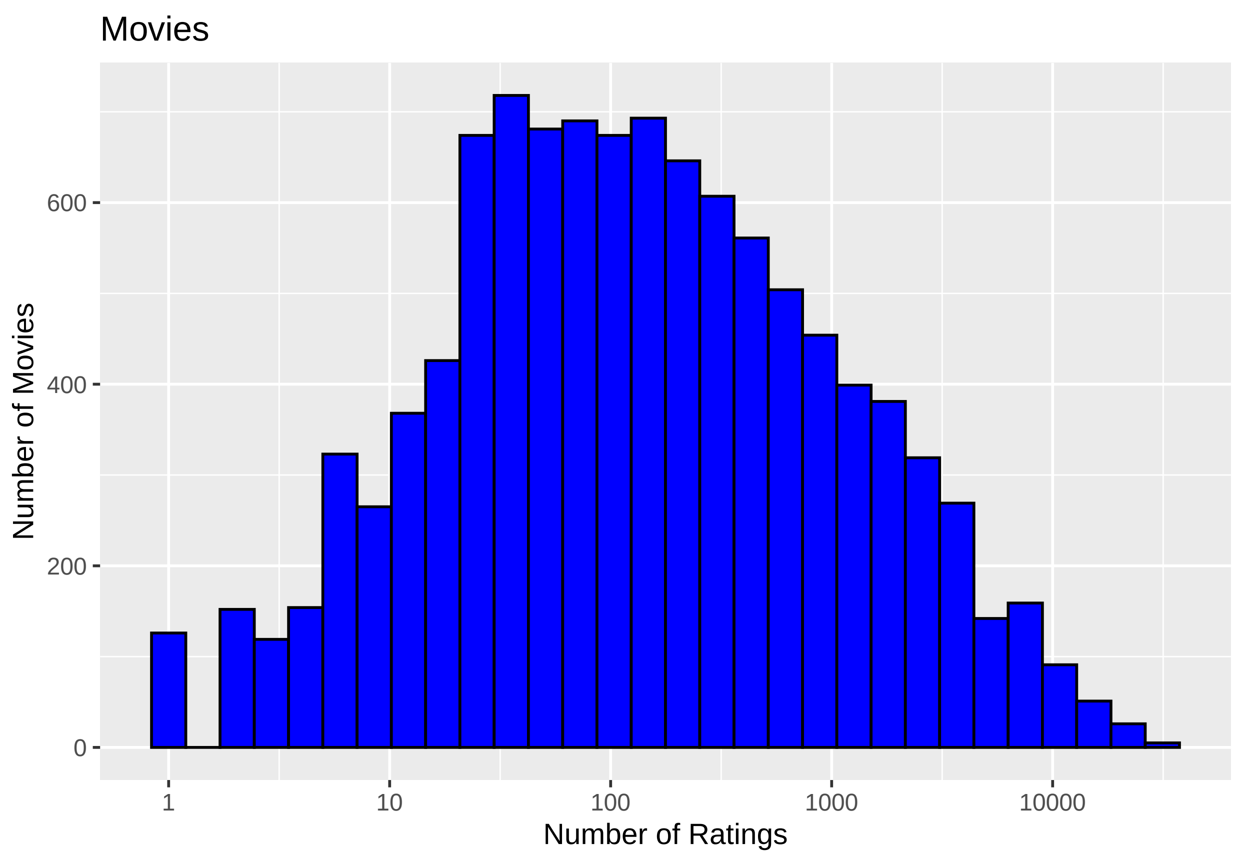
```r
ratings_plot <- as.vector(edx_new$rating)
ratings_plot <- ratings_plot[ratings_plot != 0]
ratings_plot <- factor(ratings_plot)
qplot(ratings_plot) +
  xlab("Rating") + ylab("Number of Movies") +
  ggtitle("Ratings Distribution")
```



Movies

Now lets examine the features, starting with the movies. Our hypothesis is that some movies are rated more than others (i.e. because they are popular blockbusters) and this is a bias we need to model.

```r
edx_new %>% group_by(movieId) %>%
  summarise(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "blue", color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings") + ylab("Number of Movies") +
  ggtitle("Movies")
```



To confirm that blockbusters are rated more often, let's look at the top 10 movies with the most ratings per year since its release (this is better than just looking at the movies with the most total ratings since it will account for the fact that some movies have been in circulation longer).

```r
edx_new %>% group_by(movieId) %>%
  summarize(n = n(), title = title[1], years = 2018 - first(year_release), avg_rating = mean(rating)) %
  mutate(rate = n/years) %>%
  top_n(10, rate) %>%
  arrange(desc(rate))
```

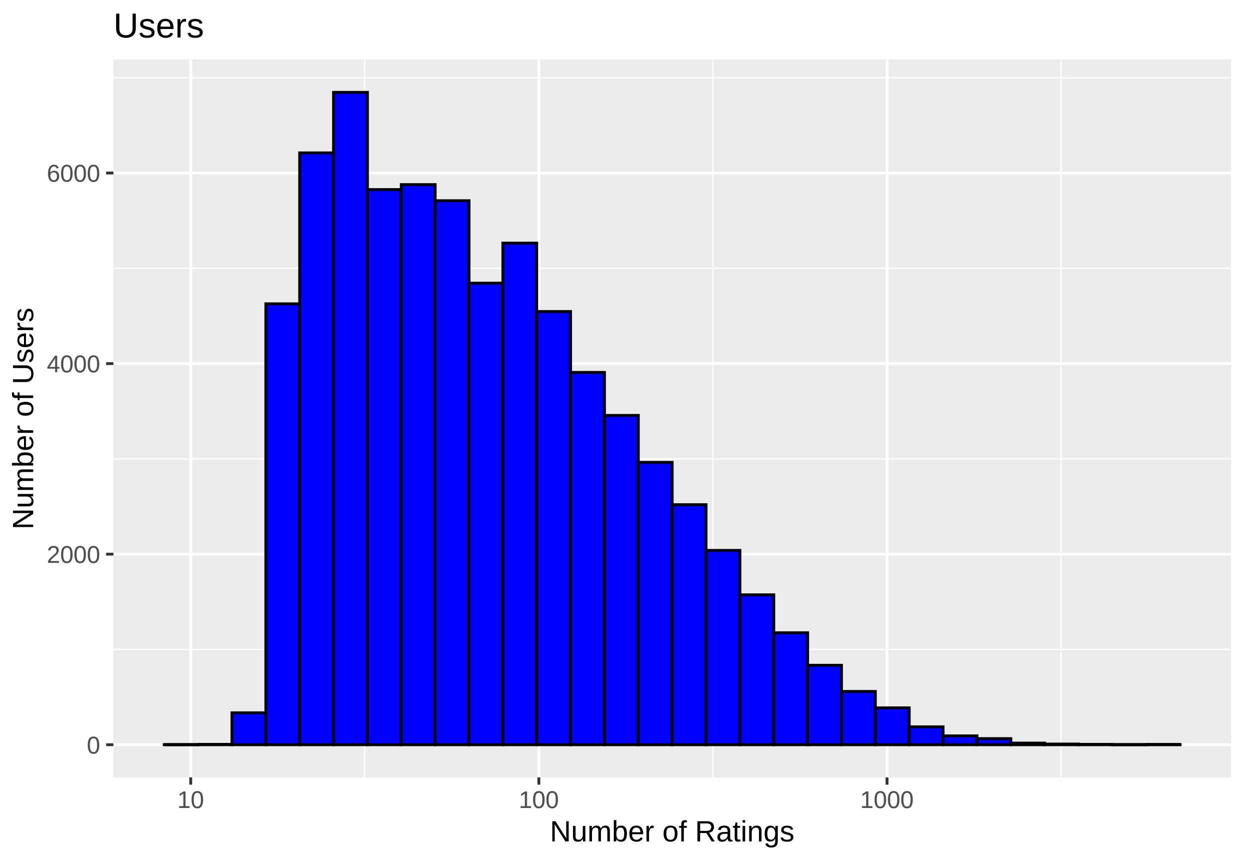
```
## # A tibble: 10 x 6
##    movieId     n title                                          years avg_rating  rate
##      <dbl> <int> <chr>                                          <dbl>      <dbl> <dbl>
##  1     296 31362 Pulp Fiction (1994)                      24        4.15 1307.
##  2     356 31079 Forrest Gump (1994)                      24        4.01 1295.
##  3     480 29360 Jurassic Park (1993)                     25        3.66 1174.
##  4     318 28015 Shawshank Redemption, The (1994)         24        4.46 1167.
##  5     110 26212 Braveheart (1995)                        23        4.08 1140.
##  6     593 30382 Silence of the Lambs, The (1991)         27        4.20 1125.
##  7    2571 20908 Matrix, The (1999)                       19        4.20 1100.
##  8     780 23449 Independence Day (a.k.a. ID4) (1996)     22        3.38 1066.
##  9     150 24284 Apollo 13 (1995)                         23        3.89 1056.
## 10    2858 19950 American Beauty (1999)                   19        4.19 1050
```

We also see that these blockbuster movies that are rated often also seem to have higher (above-average ratings). So it looks like there might also be some user bias.

Users

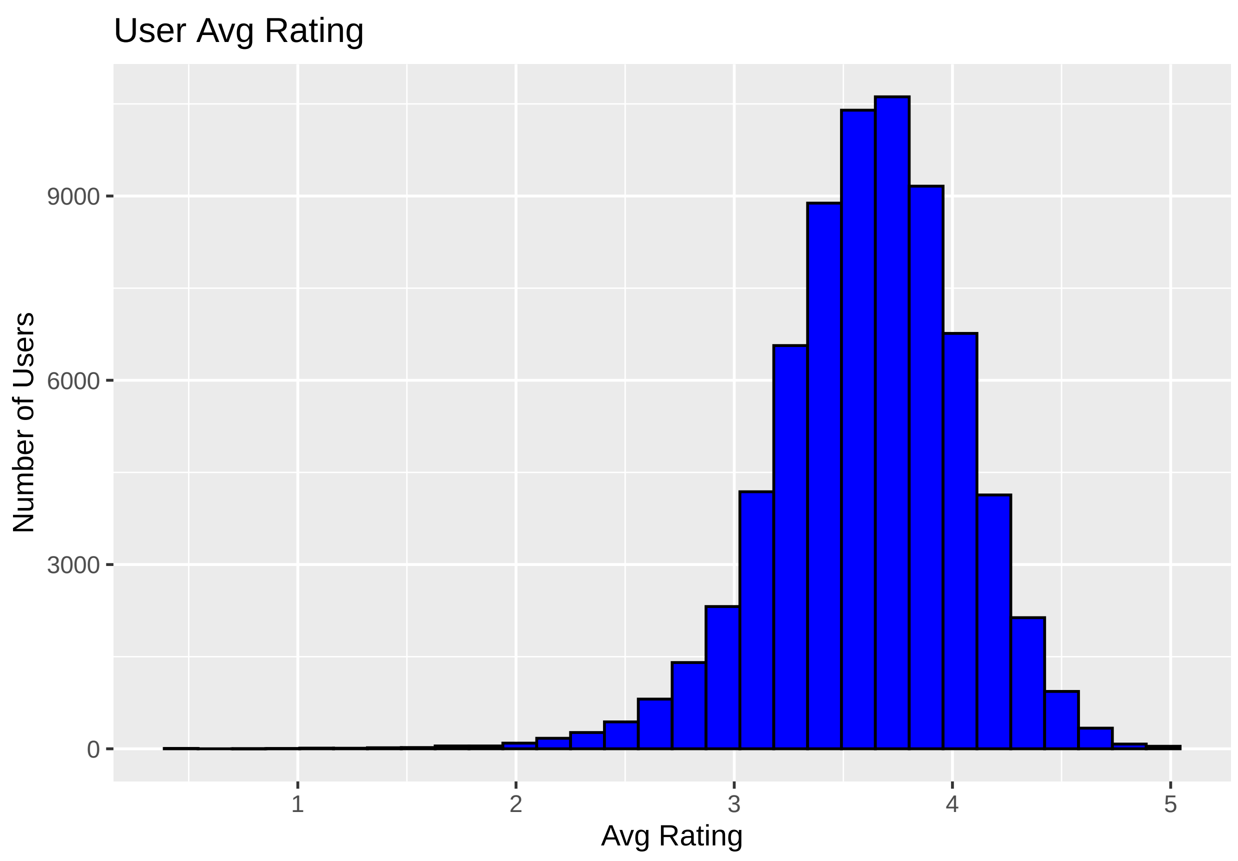
Another observation is that some users are more active at rating movies than others, but the majority of users rated between 50 and 100 movies.

```
edx_new %>% group_by(userId) %>%
  summarise(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "blue", color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings") + ylab("Number of Users") +
  ggtitle("Users")
```



We can also show that the average rating varies across users, further indicating a user bias that we need to model.

```r
edx_new %>%
  group_by(userId) %>%
  summarize(Avg_Rating = mean(rating)) %>%
  ggplot(aes(Avg_Rating)) +
  geom_histogram(color = "black", fill = "blue") +
  xlab("Avg Rating") + ylab("Number of Users") +
  ggtitle("User Avg Rating")
```



Genres

Each movie has a genre tag, and each movie can have multiple genres applied to to it. As we saw above, there are 797 genre combinations. This is a small enough number that the data can be considered categorical like the `userId` and `movieId` variables. However, it is too large to plot all of the genres and look at the average rating for each. Instead we'll filter our data to include only combinations with at least 1000 ratings, then look at the top 10 and bottom 10 genre combinations ranked by average rating.

```
edx_new %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating)) %>%
  filter(n >= 1000) %>%
  top_n(10, avg_rating)
```

```
## # A tibble: 10 x 3
##    genres                                     n avg_rating
##    <chr>                                  <int>      <dbl>
##  1 Action|Adventure|Comedy|Fantasy|Romance 14809       4.20
##  2 Action|Crime|Drama|IMAX                  2353       4.30
##  3 Animation|Children|Comedy|Crime          7167       4.28
##  4 Crime|Film-Noir|Mystery                  4029       4.22
##  5 Crime|Film-Noir|Thriller                 4844       4.21
##  6 Crime|Mystery|Thriller                  26892       4.20
##  7 Crime|Thriller|War                       4595       4.17
##  8 Drama|Film-Noir|Romance                  2989       4.30
##  9 Film-Noir|Mystery                        5988       4.24
## 10 Film-Noir|Romance|Thriller               2453       4.22
```

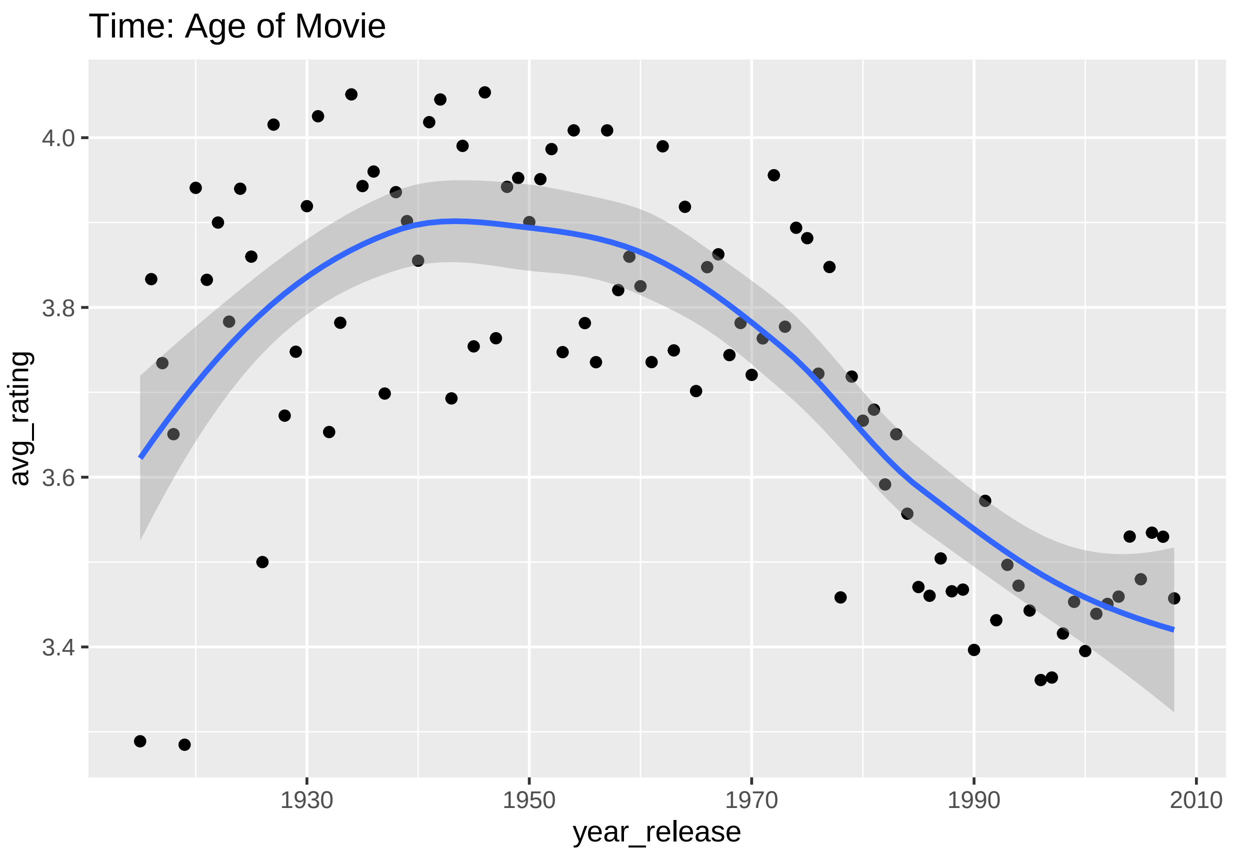
```
edx_new %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating)) %>%
  filter(n >= 1000) %>%
  top_n(-10, avg_rating)
```

```
## # A tibble: 10 x 3
##    genres                                          n avg_rating
##    <chr>                                       <int>      <dbl>
##  1 Action|Adventure|Children|Comedy|Fantasy|Sci-Fi  2832       2.06
##  2 Action|Adventure|Comedy|Fantasy|Sci-Fi|Western  5301       2.26
##  3 Action|Adventure|Fantasy|Thriller              4420       2.21
##  4 Action|Children                                3922       2.04
##  5 Action|Children|Fantasy                        2318       2.28
##  6 Action|Comedy|Musical                          1305       2.43
##  7 Action|Crime|Fantasy                           2693       2.49
##  8 Children|Comedy|Sci-Fi                         2940       2.28
##  9 Crime|Sci-Fi|Thriller                          2351       2.18
## 10 Fantasy|Horror|Thriller                        1427       2.59
```

Time

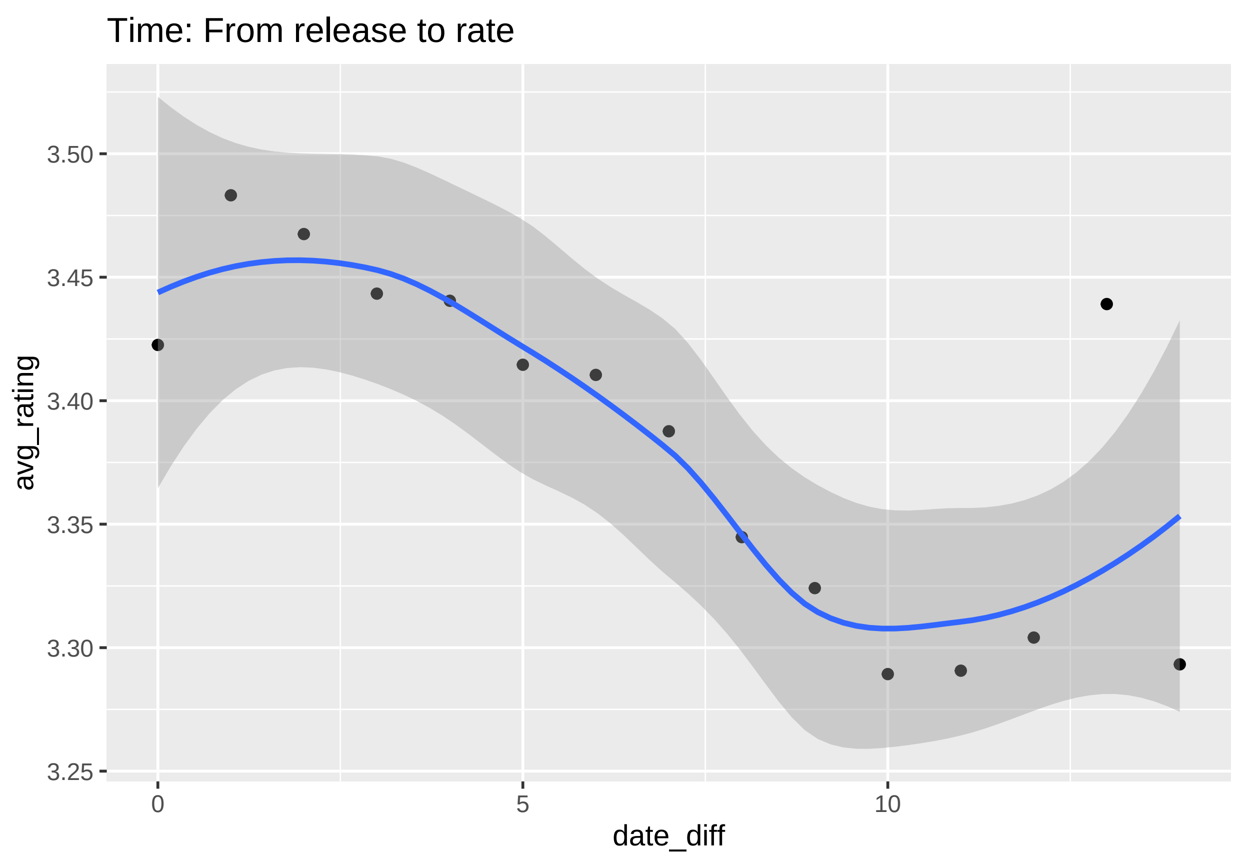
Another hypothesis is that time influences ratings. The time effect might include both the effect of the the release year (i.e. age of the movie) and the review date relative to release year (i.e. newness factor)

```
edx_new %>% group_by(year_release) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(x=year_release, y = avg_rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Time: Age of Movie")
```



Older movies seem to be rated higher than more recent movies. It is possible that present-day users have a bias to rate movies lower in general. Now, let's look at time based on how close the review was to the release year. The first date we have for a rating was in 1995, so we'll only look at movies released since then. Also since we don't have the actual date of the movie release we'll just calculate in years. There seems to be some evidence of a time effect, specifically for the review date relative to the release date: movies that are reviewed closer to the release date are rated higher. See the plot below.

```
edx_new %>% filter(year_release >= 1995) %>%
  mutate(date_diff = year_rated - year_release) %>%
  group_by(date_diff) %>%
  summarise(avg_rating = mean(rating)) %>%
  filter(date_diff >= 0) %>%
  ggplot(aes(x=date_diff, y=avg_rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Time: From release to rate")
```



Methods/Analysis

Before we build our model, we need to define our loss function. This will be used to evaluate our model performance. Our loss function for this project is Root Mean Squared Error (RMSE). RMSE is the error produced when predicting a movie rating; we want to minimize this number. Here is the RMSE function we'll be using for the remainder of this project:

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

Another important step before building our model is to split the `edx` dataset into `train` and `test` datasets so we can test the accuracy of our model as we build it, before applying our final model to the `validation` set we created earlier.

```r
set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(y = edx_new$rating, times = 1, p = 0.2, list = FALSE)
train <- edx_new[-test_index,]
test <- edx_new[test_index,]

test <- test %>%
      semi_join(train, by = "movieId") %>%
      semi_join(train, by = "userId")
```

```
rm(test_index)
```

Average rating model

The simplest recommendation system would predict the same rating for all movies regardless of the user, ignoring any biases and assuming that all the differences are explained by random variation. To minimize RMSE in this case, we simply take the average rating of all movies across all users. Remember we are now using the train dataset and will make our interim predictions on the test dataset.

```
mu <- mean(train$rating)
mu
```

```
## [1] 3.512482
```

So the average rating for all movies across all users is 3.51. Now we'll calculate our RMSE for this simple model.

```
simple_rmse <- RMSE(test$rating, mu)
simple_rmse
```

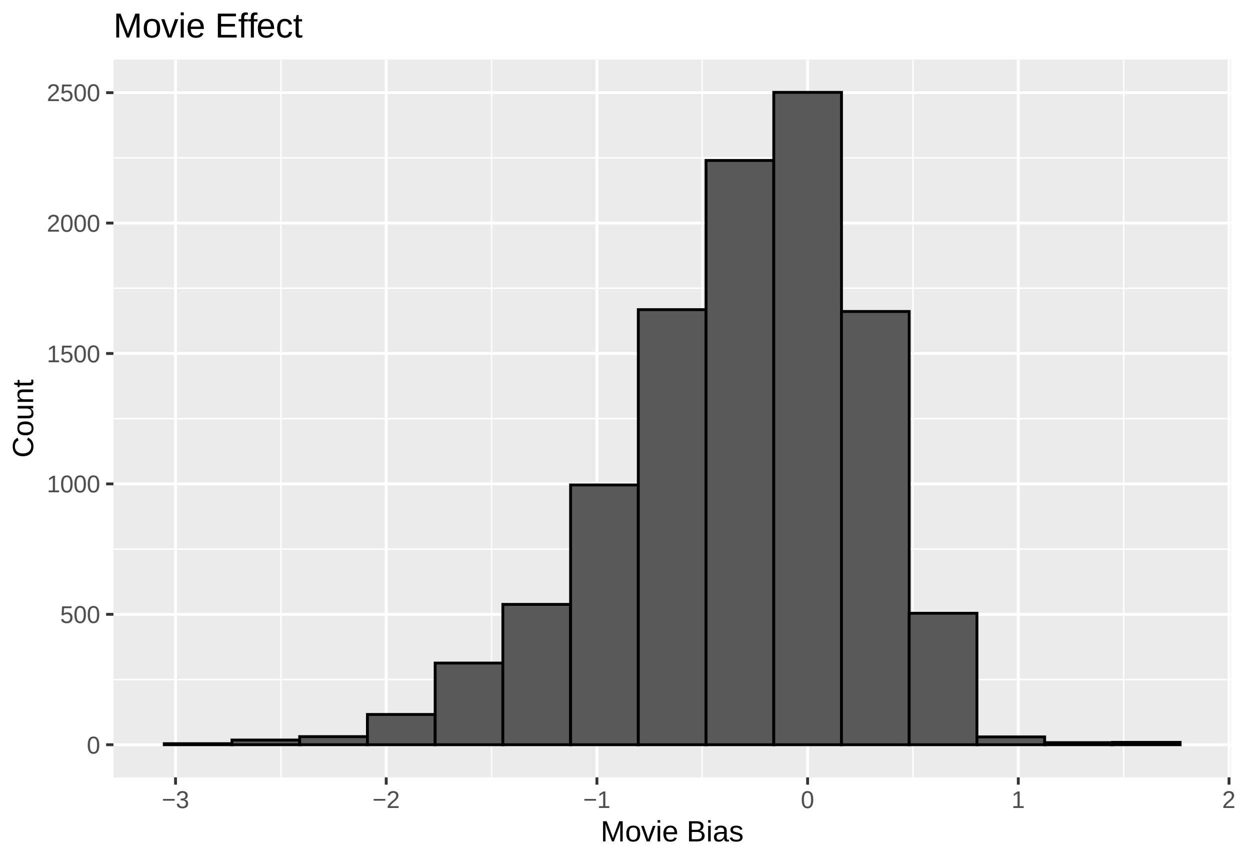
```
## [1] 1.059904
```

Obviously this is not a very robust model, as our error is greater than 1 (i.e. more than one star in our ratings).

Movie Model

We know from our data exploration that some movies are generally rated higher than others (i.e. blockbusters). We'll call this the movie effect (or bias). We can get this bias by finding the average of the difference between the actual rating and overall average rating (mu) for each movie. A plot of the effect shows that it varies alot in our dataset.

```
movie_bias <- train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
movie_bias %>% ggplot(aes(x=b_i)) +
  geom_histogram(bins = 15, color = I("black")) +
  xlab("Movie Bias") + ylab("Count") +
  ggtitle("Movie Effect")
```



We now get our predictions for this new model and calculate the RMSE.

```r
predicted_ratings <- mu + test %>%
  left_join(movie_bias, by='movieId') %>%
  pull(b_i)
movie_rmse <- RMSE(test$rating, predicted_ratings)
movie_rmse
```

```
## [1] 0.9437429
```

Movie + User Model

We have improved our simple model with the movie effect, and it is now below 1, but we can do better. We know from our data observations that users can also impact the predictions, as we showed earlier that the average rating for each user varies substantially. We can find this user effect similarly to what we did with the movie model:

```r
user_bias <- train %>%
  left_join(movie_bias, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Now we can get our predictions for the new model and calculate the RMSE.

```
predicted_ratings <- test %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_rmse <- RMSE(test$rating, predicted_ratings)
user_rmse
```

```
## [1] 0.865932
```

Movie + User + Time Model

The addition of the user bias again improved our model substantially, but we are still not yet at our goal RMSE. Perhaps time has an effect too, as we showed earlier in our data observation. We are looking the release year. Let's add it to our model and calculate the bias.

```
time_bias <- train %>%
  left_join(movie_bias, by = 'movieId') %>%
  left_join(user_bias, by = "userId") %>%
  group_by(year_release) %>%
  summarize(b_y = mean(rating - b_i - b_u - mu))
```

Now we can get our predictions for the new model and calculate the RMSE.

```
predicted_ratings <- test %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  left_join(time_bias, by='year_release') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
time_rmse <- RMSE(test$rating, predicted_ratings)
time_rmse
```

```
## [1] 0.8656117
```

Movie + User + Time + Genre Model

Again we see a small improvement, but the goal RMSE has not yet been achieved. The final feature we explored earlier was the genre (or category) of the movie. We saw this could possibly influence ratings, so we'll add it to our model and calculate the bias.

```
genre_bias <- train %>%
  left_join(movie_bias, by = 'movieId') %>%
  left_join(user_bias, by = "userId") %>%
  left_join(time_bias, by = "year_release") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - b_i - b_u - b_y - mu))
```

Now we can get our predictions for the new model and calculate the RMSE.

```r
predicted_ratings <- test %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  left_join(time_bias, by='year_release') %>%
  left_join(genre_bias, by= "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  pull(pred)
genre_rmse <- RMSE(test$rating, predicted_ratings)
genre_rmse
```

```
## [1] 0.8653661
```

Regularization Model

We were able to reduce our RMSE further, but still not below our target of 0.86490. Based on the results from our Movie Effect model, it's likely that some obscure movies with extreme high and low ratings (but very few users rating the movie) are creating noise in our bias calculations. Let's look at the top 10 best movies according to our estimates.

```r
movie_titles <- train %>%
  select(movieId, title) %>%
  distinct()
movie_bias %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487518
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487518
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487518
Shadows of Forgotten Ancestors (1964)	1.487518
Money (Argent, L') (1983)	1.487518
Fighting Elegy (Kenka erejii) (1966)	1.487518
Sun Alley (Sonnenallee) (1999)	1.487518
Aerial, The (La Antena) (2007)	1.487518
Blue Light, The (Das Blaue Licht) (1932)	1.487518
More (1998)	1.404184

Those are pretty obscure movies. And now the bottom 10 movies:

```r
movie_bias %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012482
Confessions of a Superhero (2007)	-3.012482
War of the Worlds 2: The Next Wave (2008)	-3.012482
SuperBabies: Baby Geniuses 2 (2004)	-2.749982
From Justin to Kelly (2003)	-2.667244
Legion of the Dead (2000)	-2.637482
Disaster Movie (2008)	-2.637482
Hip Hop Witch, Da (2000)	-2.603391
Criminals (1996)	-2.512482
Mountain Eagle, The (1926)	-2.512482

Again, very obscure movies. And here is how often the top 10 movies are rated:

```
train %>% count(movieId) %>%
  left_join(movie_bias) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487518	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487518	3
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487518	2
Shadows of Forgotten Ancestors (1964)	1.487518	1
Money (Argent, L') (1983)	1.487518	1
Fighting Elegy (Kenka erejii) (1966)	1.487518	1
Sun Alley (Sonnenallee) (1999)	1.487518	1
Aerial, The (La Antena) (2007)	1.487518	1
Blue Light, The (Das Blaue Licht) (1932)	1.487518	1
More (1998)	1.404184	6

These movies have not been rated often. Same for the worst movies:

```
train %>% count(movieId) %>%
  left_join(movie_bias) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i	n
Besotted (2001)	-3.012482	1
Confessions of a Superhero (2007)	-3.012482	1
War of the Worlds 2: The Next Wave (2008)	-3.012482	2
SuperBabies: Baby Geniuses 2 (2004)	-2.749982	40

title	b_i	n
From Justin to Kelly (2003)	-2.667244	168
Legion of the Dead (2000)	-2.637482	4
Disaster Movie (2008)	-2.637482	28
Hip Hop Witch, Da (2000)	-2.603391	11
Criminals (1996)	-2.512482	1
Mountain Eagle, The (1926)	-2.512482	1

Regularization allows us to penalize these extreme estimates that are made with very small sample sizes. This penalty term (lambda) is then added to our model. We can find this penalty term using cross-validation , and plot the lambdas to find the optimal parameter. We'll perform regularization on our already-built linear model that includes the movie, user, time and genre effects.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train$rating)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_y <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(year_release) %>%
    summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+l))
  b_g <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year_release") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+l))
  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year_release") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y +b_g) %>%
    .$pred
  return(RMSE(test$rating, predicted_ratings))
})
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.5
```

Now we have our optimal lambda, so we can find the RMSE for that lambda.

```
all_reg_rmse <- min(rmses)
all_reg_rmse
```

```
## [1] 0.8647544
```

Matrix Factorization

While this regularized model produced an RMSE of 0.8647544, which is below our goal of 0.8649, we should still try to improve the model because it is very close to the threshold. Another machine learning technique we can try is matrix factorization. This concept is widely used in predicting ratings in recommendation systems. The train dataset is converted into a matrix such that each row is a user, each column is a movie, and the data in the cell is the rating. The recosystem package allows us to easily perform this matrix factorization. Instructions on how to implement the package can be found here, and this is a quick summary of the process (via the package website):
1. Create a model object by calling Reco().
2. (Optionally) call the $tune() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $train() method. Parameters can be set inside the function.
4. (Optionally) export the model via $output(), i.e. write the factorization matrices P and Q into files or return them as R objects.
5. Use the $predict() method to compute predicted values.

```
set.seed(123, sample.kind = "Rounding")

train_data <- with(train, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
test_data <- with(test, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))

r <- Reco()

opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.01, 0.1),
                                       costp_l1 = 0,
                                       costq_l1 = 0,
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread = 4,
                                       niter = 10))

r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9947   1.0058e+07
##    1       0.8792   8.0970e+06
##    2       0.8471   7.5136e+06
##    3       0.8255   7.1609e+06
##    4       0.8087   6.9123e+06
##    5       0.7954   6.7305e+06
##    6       0.7846   6.5894e+06
##    7       0.7752   6.4763e+06
##    8       0.7672   6.3839e+06
##    9       0.7600   6.3052e+06
##   10       0.7538   6.2380e+06
##   11       0.7483   6.1835e+06
##   12       0.7433   6.1317e+06
##   13       0.7387   6.0886e+06
##   14       0.7345   6.0491e+06
##   15       0.7307   6.0168e+06
##   16       0.7272   5.9826e+06
##   17       0.7241   5.9575e+06
##   18       0.7211   5.9318e+06
##   19       0.7183   5.9080e+06
```

```r
predictions_reco <- r$predict(test_data, out_memory())

matrix_rmse <- RMSE(test$rating, predictions_reco)
matrix_rmse
```

```
## [1] 0.7907564
```

This has substantially reduced our RMSE to 0.791, and we now have gotten well below our goal of 0.8649.

Results

The Matrix Factorization model produced the best results from our modeling process. Here is a table of our results from all the models.

```r
models <- c("Avg Rating",
            "Avg Rating + Movie",
            "Avg Rating + Movie + User",
            "Avg Rating + Movie + User + Year",
            "Avg Rating + Movie + User + Year + Genre",
            "All Effects Regularized",
            "Matrix Factorization")
results <- c(simple_rmse, movie_rmse, user_rmse, time_rmse, genre_rmse, all_reg_rmse, matrix_rmse)
results_table <- data.frame(Model = models, RMSE = results)
knitr::kable(results_table)
```

Model	RMSE
Avg Rating	1.0599043
Avg Rating + Movie	0.9437429
Avg Rating + Movie + User	0.8659320
Avg Rating + Movie + User + Year	0.8656117
Avg Rating + Movie + User + Year + Genre	0.8653661
All Effects Regularized	0.8647544
Matrix Factorization	0.7907564

Now we will apply this Matrix Factorization model to train the entire `edx_new` dataset and then make our final predictions on the `validation` set.

```
set.seed(123, sample.kind = "Rounding")

edx_data <- with(edx_new, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating = rating))
validation_data <- with(validation, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))

r <- Reco()

opts <- r$tune(edx_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.01, 0.1),
                                       costp_l1 = 0,
                                       costq_l1 = 0,
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread = 4,
                                       niter = 10))

r$train(edx_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9731   1.2024e+07
##    1       0.8726   9.8767e+06
##    2       0.8393   9.1705e+06
##    3       0.8170   8.7553e+06
##    4       0.8012   8.4704e+06
##    5       0.7893   8.2726e+06
##    6       0.7797   8.1188e+06
##    7       0.7718   8.0026e+06
##    8       0.7650   7.9076e+06
##    9       0.7592   7.8279e+06
##   10       0.7540   7.7622e+06
##   11       0.7493   7.7021e+06
##   12       0.7451   7.6523e+06
##   13       0.7413   7.6071e+06
##   14       0.7379   7.5694e+06
##   15       0.7347   7.5349e+06
##   16       0.7317   7.5012e+06
##   17       0.7291   7.4769e+06
##   18       0.7266   7.4492e+06
##   19       0.7242   7.4262e+06
```

```
predictions_final_reco <- r$predict(validation_data, out_memory())

matrix_final_rmse <- RMSE(validation$rating, predictions_final_reco)
matrix_final_rmse
```

```
## [1] 0.7826948
```

Our matrix factorization model produced a final **RMSE of 0.783**, which is well below our target level of 0.8649. Our motivation for using this type of model is that allows us to account for differences not captured in a linear model. A linear regression model accounts for movie-to-movie differences through a movie bias variable, user-to-user differences through a user bias variable, etc. However, what about the interaction between variables? The major improvement in RMSE occurs because matrix factorization is able to model the fact that *groups* of movies have similar ratings and *groups* of users have similar rating patterns. It does this by approximating a large movie-user matrix into the product of two smaller dimension matrices.

Conclusion

The goal of this project was to create a robust movie recommendation system using the machine learning techniques and data analysis skills learned in previous courses. We started with a large (~10 million rows) movielens dataset, and initially separated out a random subset of ~1 million rows (validation set) that would be used to make our final predictions and calculate the performance of our final model. The performance metric used was RMSE.

Before building our model, we performed an exploratory analysis on the dataset. We created several tables and graphs to better understand the structure of the dataset and any patterns in the data. Once the data exploration was complete, we split out data into training and test sets, and various algorithms were tested to find the optimal model. The lowest RMSE was achieved with our Matrix Factorization model.

Limitations and Future Work

There are several limitations to our model. First, the model obviously relies on users to be active in rating movies. The interests of the population might change, but without actual ratings by these people, the changes cannot be detected with our model. Second, our model works only for existing users and movies. It must be re-trained and re-run every time a new user or movie or rating is included, and this has a performance impact on the system.

This analysis explored the basic features of a recommendation system: product (movie), user, date and category (genre) informatoin. However, there are several other features we could have tried to incorporate into our model. One area that would be interesting to explore is demographics, such as the age, race, gender, income and geographic region of the users. Also, it would be interesting to further model "popularity". Social media has had a huge impact on trending products and likely is a significant factor in user ratings for these products.

References

https://rafalab.github.io/dsbook
https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html
https://statr.me/2016/07/recommender-system-using-parallel-matrix-factorization/
http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf
https://grouplens.org/datasets/movielens/10m/