Movielens Project

Melissa Mayer

12/26/2021

Introduction

This project is based on the 2006 Netflix Challenge, which was a competition for the data science community to create a winning algorithm that would improve upon Netflix's movie recommendation system by ten percent. Netflix uses a 5-star recommendation system to predict how a movie will be rated by a user. This system is used to generate movie recommendations for users steering them towards movies that they will most likely rate highly. Many companies use such recommendation systems based on collected data to recommend products to users. Products that are predicted to be rated highly by a user will be recommended to that particular user.

Although the Netflix data is not publicly available, Grouplens generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. This project utilizes the MovieLens 10M data set which consists of 10 million ratings given to 10,000 movies by 37,000 users. For this analysis, we will use machine learning techniques to create and test our algorithm. We partitioned the MovieLens data into the edx set which contains 9 million observations of 6 variables. The validation set contains 10% of the MovieLens data and will be used to test the final algorithm. The Residual Mean Squared Error (RMSE) will be the loss function used to assess the performance of the models.

Create edx set, validation set (final hold-out test set)

We are going to load the following libraries:

```
library(tidyverse)
library(caret)
library(data.table)
```

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) %>% mutate(movieId = as.numeric(movieId),
                    title = as.character(title), genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Validation set will be 10% of MovieLens data

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

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

Make sure userId and movieId in validation set are also in edx set

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

Add rows removed from validation set back into edx set

```
removed <- anti_join(temp, validation)
```

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

```
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Methods/Analysis

The MovieLens data was partitioned into a training set (edx) and test set (validation) which contains ten percent of the data. Using the semi_join function, we ensure that we do not include movies or users in the test set that are not in the train set. Our first step is to inspect the data by subsetting it and to observe that it is in tidy format. Each row represents a movie rating given by a user. It also consists of NA's where there is no rating by a user for a particular movie. The data can be seen as a matrix with missing information that our analysis will seek to predict. By creating a matrix of a sample of 100 movies and 100 users, we can show the movie/user combinations within the data and see how much data is missing. What can also be observed in the data is that more popular movies receive a lot more ratings than less popular movies and

that some users are much more active in rating movies than others. Plotting these distributions shows the wide range of the data.

In order to build and test our models, the next step is to partition the edx dataset into training and test sets where we can evaluate our models. The RMSE is the loss function used to compare the true ratings with the ratings generated by our predictions. RMSE is defined as the typical error that is made when predicting a movie rating. If it is greater than 1, it means the error is greater than one star which is not a good prediction. The RMSE will be calculated for each model to identify the best performing model which will finally be tested on the validation set to compare our predictions to the true values.

Building the Recommendation System

Inspect the data

```
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
```

Find out how many movies and users are in edx dataset

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

Create a small subset of the data to see the ratings for a group of movies/users

```
keep <- edx %>%
  dplyr::count(movieId) %>%
  top_n(5) %>%
  pull(movieId)
```

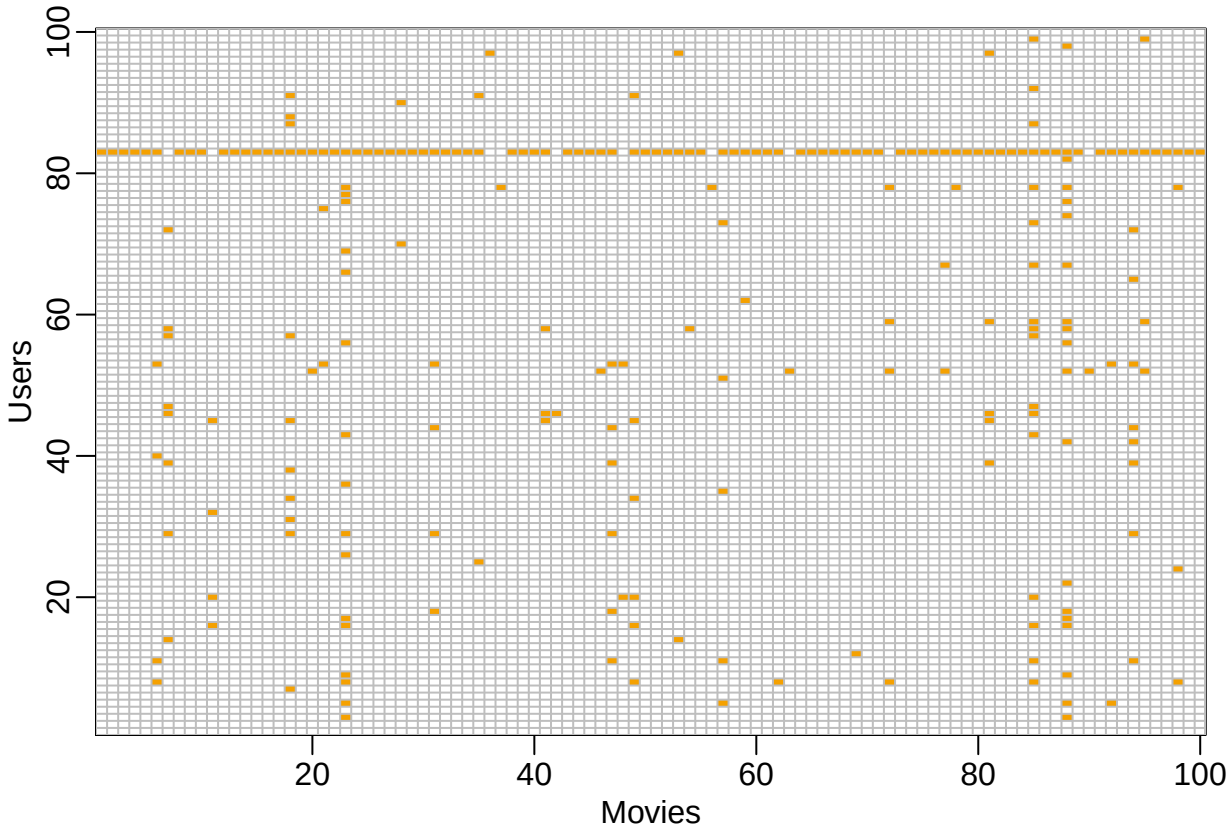
```
## Selecting by n
```

```
tab <- edx %>%
  filter(userId %in% c(13:20)) %>%
  filter(movieId %in% keep) %>%
  select(userId, title, rating) %>%
  spread(title, rating)
tab %>% knitr::kable()
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	5	4.5	5
19	4	1	NA	4.0	NA

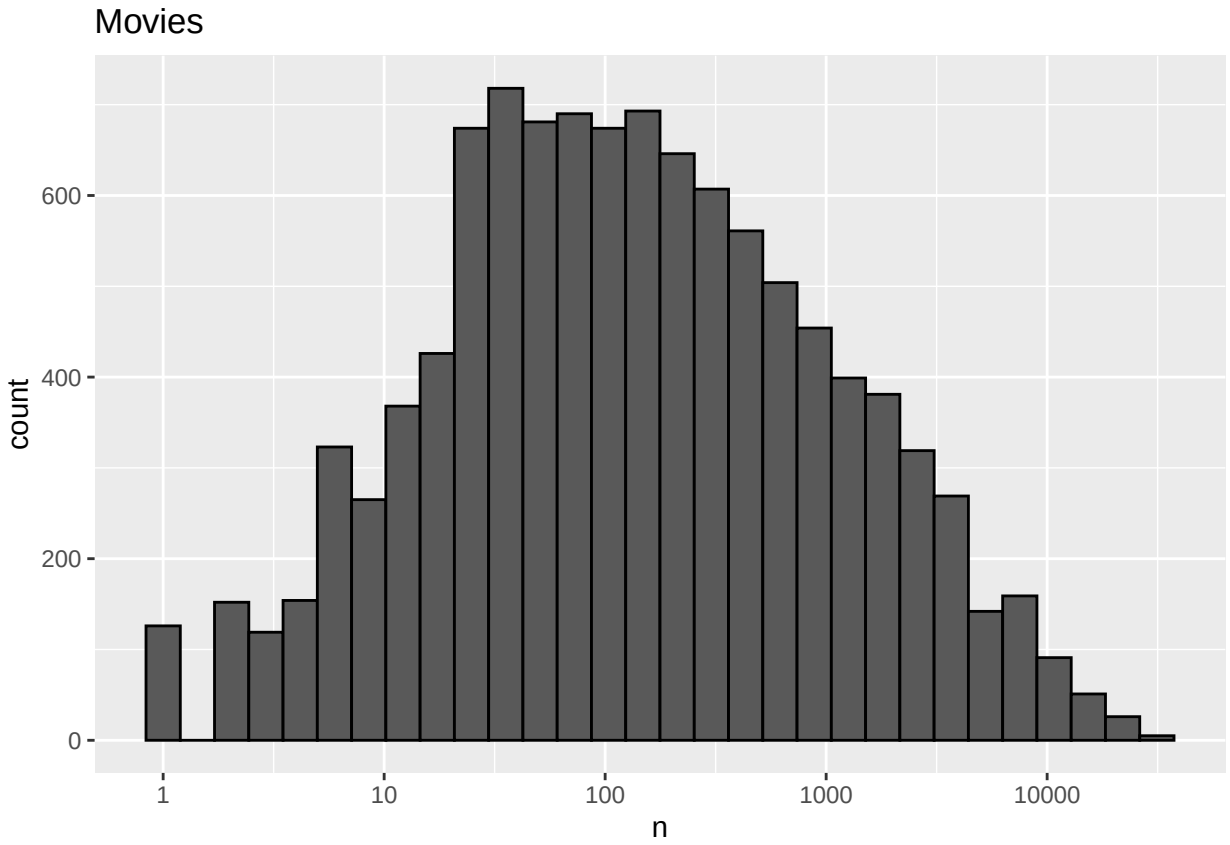
Matrix of combinations of ratings for a random sample of 100 movies and 100 users

```
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



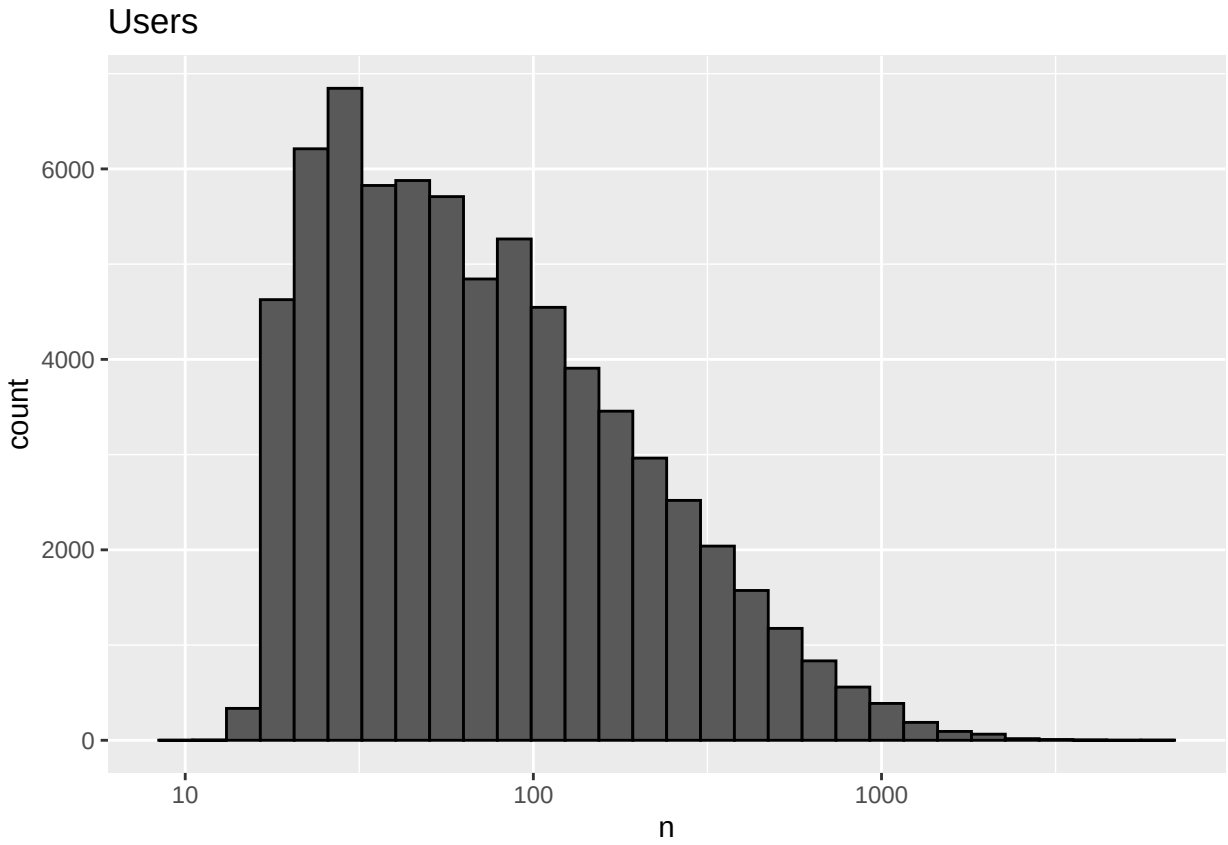
Plot the distribution of Movie ratings

```
edx %>%
    dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



Plot the distribution of User ratings

```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



Create a Test Set to Assess Accuracy of Models

```
set.seed(100, sample.kind = "Rounding")
```

```
## Warning in set.seed(100, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Make sure userId and movieId in test_ set are also in train_set

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Calculate RMSE

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

Results

This analysis incorporates four models that are used to evaluate the accuracy of our predictions which are summarized in a table.

1. A basic model assumes the same rating for all movies and users with differences explained by random variation. Use the average movie rating for all movies, 3.5, to predict all ratings. This approach does not yield a good error result (RMSE>1). As a comparison, we can choose another rating that differs from the average and test how it performs. This test yields an even bigger RMSE (>1).
2. The next model incorporates the movie effect. The average rating for a particular movie can be calculated using the least squares estimate. Adding movie effects to our prediction improves our error (RMSE<1).
3. Building on the previous model which incorporated the average rating for an individual movie, the next step was to add the average rating given by a user. Users were restricted to those who rated at least 100 movies. The user effect further improves our model to an RMSE of 0.8654051.
4. Each movie in the dataset is categorized by at least 1 movie genre with many movies falling into more than category. To further improve the model, the next step was to find the average rating for a particular genre of movie for genres that had at least 1000 ratings. This model produces and RMSE of 0.8650564, the best performing model.

When this model is utilized on the true ratings in the validation set, an RMSE of 0.8649469 is obtained demonstrating the ability of this model to accurately predict movie ratings.

Building the Models

"Just the Average"

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.51234
```

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059899
```

```
predictions <- rep(3, nrow(test_set))
RMSE(test_set$rating, predictions)
```

```
## [1] 1.17751
```

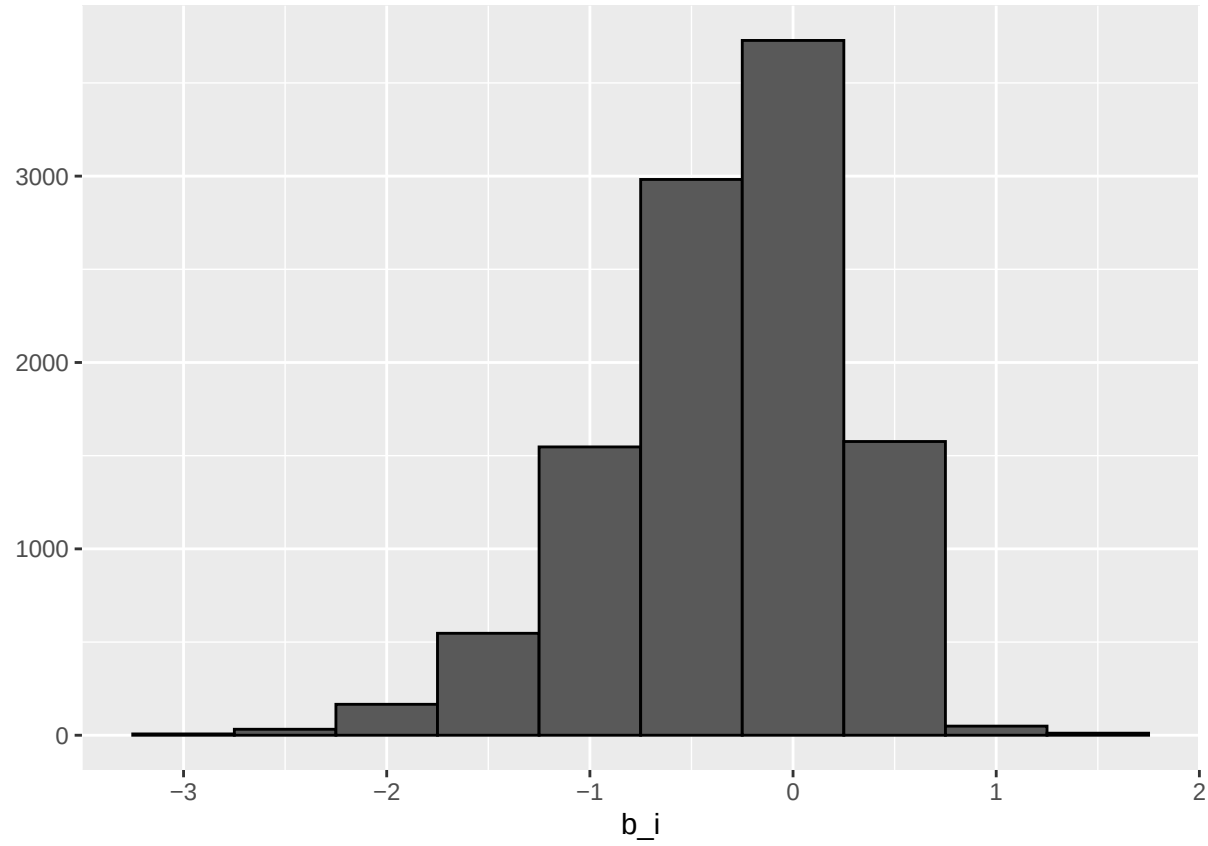
Create a table to store the results

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

Model 1 - fit <- lm(rating ~ as.factor(movieId), data = edx)

```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
    group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



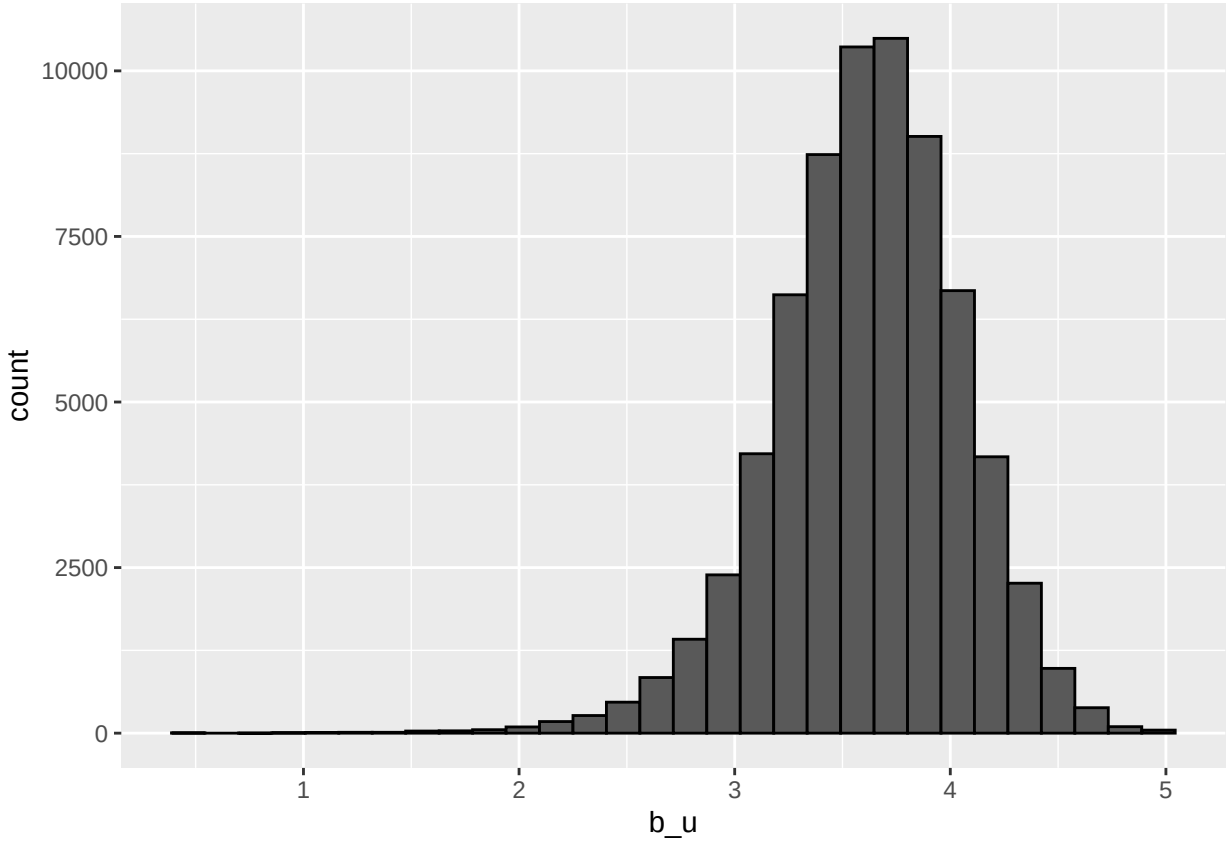

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Movie Effect Model", RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0598986
Movie Effect Model	0.9435824

Explore the data using UserId

```
train_set %>%
    group_by(userId) %>%
   summarize(b_u = mean(rating)) %>%
    filter(n()>=100) %>%
    ggplot(aes(b_u)) +
    geom_histogram(bins = 30, color = "black")
```



Model 2 - lm(rating ~ as.factor(movieId) + as.factor(userId))

```
user_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
```

```
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Movie + User Effects Model", RMSE = model_2_rmse

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0598986
Movie Effect Model	0.9435824
Movie + User Effects Model	0.8654051

Model 3 – lm(rating ~ as.factor(movieId) + as.factor(userId) + as.factor(genres))

```
train_set %>% group_by(genres) %>%
  summarize(g_u_i = mean(rating)) %>%
  filter(n()>=1000)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: genres <chr>, g_u_i <dbl>
```

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(g_u_i = mean(rating - mu - b_i - b_u))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + g_u_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Movie + User + Genre Effects Model", RMSE = model
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0598986
Movie Effect Model	0.9435824
Movie + User Effects Model	0.8654051
Movie + User + Genre Effects Model	0.8650564

Testing the Model with the Validation Set (Movie + User + Genre Effects)

```
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100)
```

```
## # A tibble: 69,878 x 2
##    userId   b_u
##     <int> <dbl>
##  1      1  5
##  2      2  3.29
##  3      3  3.94
##  4      4  4.06
##  5      5  3.92
##  6      6  3.95
##  7      7  3.86
##  8      8  3.39
##  9      9  4.05
## 10     10  3.83
## # ... with 69,868 more rows
```

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

edx %>% group_by(genres) %>%
  summarize(g_u_i = mean(rating)) %>%
  filter(n()>=1000)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: genres <chr>, g_u_i <dbl>
```

```
genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(g_u_i = mean(rating - mu - b_i - b_u))
```

Testing the Final Model

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + g_u_i) %>%
  .$pred

final_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Final Model", RMSE = final_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0598986
Movie Effect Model	0.9435824
Movie + User Effects Model	0.8654051
Movie + User + Genre Effects Model	0.8650564
Final Model	0.8649469

Conclusion

This report attempted to explore the Netflix Challenge to improve upon Netflix's movie recommendation system. The Netflix data contained millions of ratings with thousands of movies and users. In order to refine the recommendation system, the average rating across different variables was found and shown to have a strong effect on ratings thus enabling a model that incorporated movie, user, and genre averages. The model faces some limitations in that it does not take into account the effect of small sample sizes of movies and users. Some movies have very few ratings producing unreliable averages that can lead to large errors in prediction. In addition, there may be other signficiant groupings of the data that reflect patterns among some movies and users, such as sequels whose ratings may be influenced by the preceding movies. There may be other factors that affect how a movie will be rated. Besides genre classifications, movies can be categorized in different ways such as whether they were critically acclaimed or were box office hits. These groupings may have a signficant impact on ratings and thus be used to improve predictions. Finally, it may be impossible for any machine learning algorithm to truly predict human behavior.