Which Movie Should I Watch Tonight?

Movie Recommendation System - HarvardX MovieLens Project

Charles Ajax Hulebak

Class of 2020

Contents

Preface

This project is for HarvardX's Data Science Professional Certificate program, Movielens capstone project. [1]

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. [2]

[1]https://www.edx.org/professional-certificate/harvardx-data-science>
[2]http://rmarkdown.rstudio.com>

Introduction and Project Overview

Many of the most popularized data science methods originate from machine learning, in which a variety of algorithms are built to extract insights from data.[1] Recommendation systems, which are a type of machine learning, have been popularized for a variety of business products to align customer's interests with satisfaction, leading to a higher rate of retention, sales, growth, and revenue.

There are three primary broad groups of recommendation systems, which include:

1. **Content-based systems:** that examine properties of the items recommended
2. **Collaborative filtering systems:** that recommend items based on similarity measures between users and/or items
3. **Hybrid systems:** combine attributes from both

Examples of recommendation systems have been popularized on the internet by using your computer browser's cache and history, and other methods and used by advertisement companies that will use keywords and data to present product recommendations when you access websites. Cloud-based companies such as Amazon will seek to present alternative, associative, or combinations of items during your shopping experience. Lastly, online movie streaming services such as Netflix, HBOGo, and others have popularized recommendation systems based on previous shows you have watched as well as sampled ratings from user groups. Netflix incentivized a reward to improve their recommendation system by 10% in 2006, known as the "Netflix Challenge".[2,3] All of these examples and the use of recommendation systems are critical success factors to drive economic growth for these companies.

The objective for this project is to use the MovieLens (ML-10) dataset[4], while utilizing machine learning tasks to develop a movie recommendation system. This project will use a collaborative filtering recommender system. The initial objective is to build a recommender system that will provide the most accurate predictions, then we would like to use this system to choose a genre, or a combination of genres, then have a user's interest populate with the top 20 list of movies to choose from based on the reviewer ratings. The reviewer ratings are scaled from 1 to 5, with 1 being the lowest and 5 the highest. The process of users watching movies (or previously watched movies) may also encourage individuals to submit their own review from MovieLens ratings by agreeing, or disagreeing and providing a rating of their own.

The MovieLens (ML-10) dataset

The University of Minnesota established the GroupLens research lab, which has collected data and created datasets for thousands for movies, and millions of ratings by users. Although there are larger datasets that are publicly available, we will use the MovieLens (ML-10) dataset. This dataset includes 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users through the online recommender service MovieLens.[5]

Analysis & Methods

The objective of using this recommendation system is to develop the highest accuracy recommender system from an algorithm which will predicts outcomes based on the features that we establish. The algorithms explained within this project will be trained using a dataset in which we do not know the outcome, and apply the algorithm to make future predictions.

There are several loss functions to calculate the regression loss, which include the mean Absolute Error (MAE, also known as the L1 Loss), Mean Square Error (MSE, also known as Quadratic Loss, or L2 Loss), Mean Bias Error (MBE), Root Mean Square Logarithmic Error (RMSLE), Huber Loss/Smooth Mean Absolute Error, Log cosh Loss, and Residual Mean Squared Error (RMSE).[6,7,8] However, due to time constraints of testing the best loss function, we will focus on measuring the typical error loss function to measure the genre winners (for our movie choice) based on the RMSE on the test set. RSME was used for the winner of the Netflix Challenge, and the formula is defined as:

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

$y_{u,i}$ is the rating of movie i by user u and denote our prediction with $\hat{y}_{u,i}$. N is the number of ratings/genre combinations and the the sum occuring over all the combinations. Our initial objective of using RSME is to miminize the loss, which will produce the most accurate predictions.

Machine Learning Process

The first step of this project includes downloading the data and preparing it to be processed and analyzed. Next, we will explore this data to understand the content, features, and how we can use it to develop predictions. The third step will include cleaning the data to remove unnecessary information and speed up the prediction process. Lastly, we will analyze the data and make predictions based upon the content.

Download relevant libraries and the dataset, followed by preparing the data for processing

In this sequence of steps we will install and import appropriate libraries, download the dataset, create the movies dataframe, and join the datafiles.

We will separate the data into the test, training, and validations sets. An 80/20 data partition split is generally used for training and test data, which is based on the Pareto rule, however we will use a 90/10 partition split. The training data is used to build the model, in which our analysis will be conducted. The test data will use a portion of the data to test the training data, and lastly the validation data is used to calculate the final regression loss using RMSE.

```r
# suppress summarize warnings
if(!require(dplyr))
  install.packages("dplyr", repos = "http://cran.us.r-project.org")
library(dplyr, warn.conflicts = FALSE)
options(dplyr.summarise.inform = FALSE)

# Install libraries
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")

# Import Libraries
library(tidyverse)
library(caret)
library(data.table)
```

```r
# Download the MovieLens 10M dataset:
# Reference: https://grouplens.org/datasets/movielens/10m/
# Source: http://files.grouplens.org/datasets/movielens/ml-10m.zip
dl <- tempfile()

download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Define ratings and movies from the dataset
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")

# Create the movies dataframe from the movieID, title, and genres
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

# Join the datafiles and define as the movielens
movielens <- left_join(ratings, movies, by = "movieId")

# Create the test index, which will encompass 10% of the MovieLens data,
# then define training and test data
set.seed(1, sample.kind="Rounding") # if R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating,
                                  times = 1, p = 0.1, list = FALSE)
train_data <- movielens[-test_index,] #renamed from test_data
temp_data <- movielens[test_index,]   #renamed from temp_data

# Name the validation data by creating joins with specified training data
test_data <- temp_data %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")

# Add  removed rows from the validation set into the train data set
removed <- anti_join(temp_data, test_data)

train_data <- rbind(train_data, removed)

rm(test_index, temp_data, removed)

head(movielens)
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Data Exploration

The Dataset

Let's begin with looking at the data that we have downloaded so we have a better idea of how to build our model:

```
str(train_data)
```

```
## 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 838
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)"
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" '
##  - attr(*, ".internal.selfref")=<externalptr>
```

Testing data set structure and features:

```
head(train_data)
```

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

Training data sets rows and columns:

```r
# training data
dim(train_data)
```

```
## [1] 9000055       6
```

Data Characteristics

The movie title and movieId are target variables in which the remainder of the variables describe. Now that we understand the data content, we will explore the following characteristics within the training data in greater detail:

1. **Data characteristics:** number of users, movies, and genres
2. **Timestamp:** the duration of the data collection
3. **Genre:** the labels associated with the movie data
4. **Rating:** distribution of user ratings for the movies
5. **UserId:** which users rated movies more often and their ratings

Number of distinct Users, Movies, and Genres within the dataset (variables between 1 or more genre combinations):

```r
train_data %>% summarize('Number of Users' = n_distinct(userId),
            'Number of Movies' = n_distinct(movieId),
            'Number of Genres (unique and combinations)' =
              n_distinct(genres))
```

Number of Users	Number of Movies	Number of Genres (unique and combinations)
69878	10677	797

Timestamp

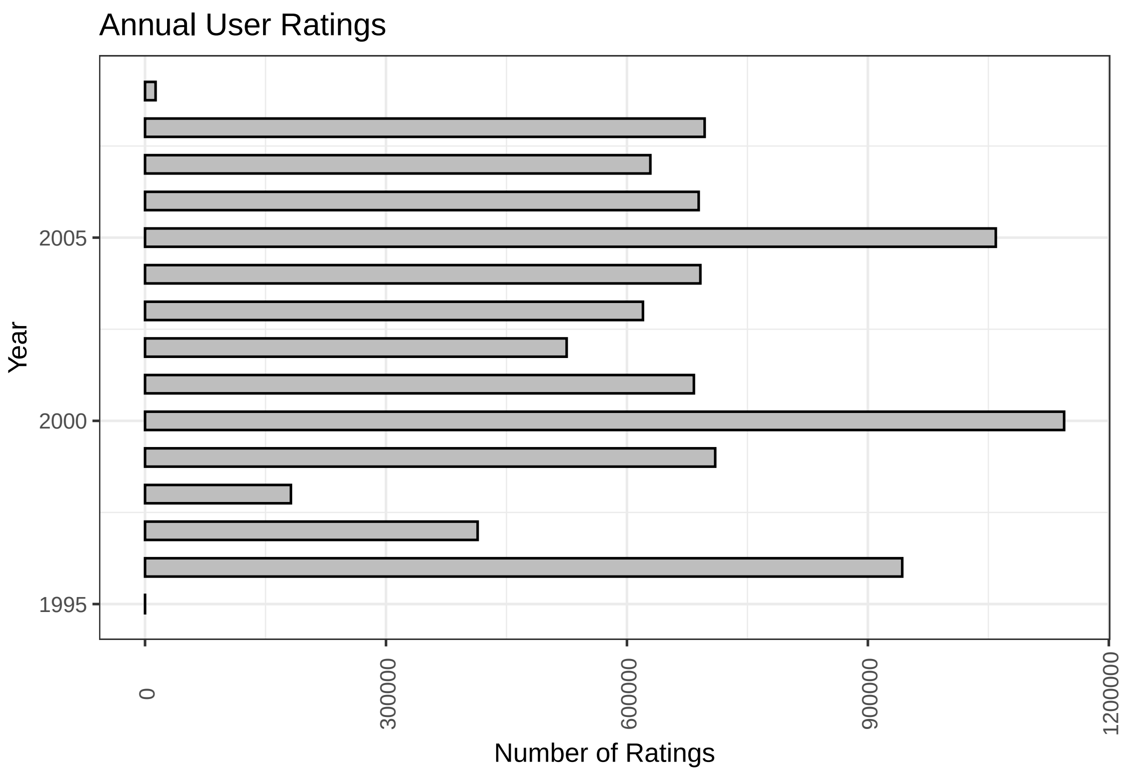
```r
if(!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
library(lubridate)
tibble(`Date of First Movie in Dataset` =
         date(as_datetime(min(train_data$timestamp))),
       `Date of Last Movie in Dataset` =
         date(as_datetime(max(train_data$timestamp)))) %>%
        mutate(Duration =
         duration(max(train_data$timestamp)-min(train_data$timestamp)))
```

Date of First Movie in Dataset	Date of Last Movie in Dataset	Duration
1995-01-09	2009-01-05	441479727s (~13.99 years)

It appears that the first movie in the dataset was rated in 1995, whereas the last was in 2009, spanning nearly 14 years for ratings within the MovieLens (ML-10) dataset (training set).

This visualization shows us the distribution of ratings across the duration of years:

```r
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
library(ggthemes)
train_data %>% mutate(year = year(as_datetime(timestamp))) %>%
  ggplot(aes(x=year), ratings) +
    geom_bar(width = .5, fill="grey", color="black") +
    ggtitle("Annual User Ratings") +
    xlab("Year") +
    ylab("Number of Ratings") +
    theme_bw() +
    coord_flip() +
    theme(axis.text.x = element_text(angle=90, vjust=0.6))
```



Genres

There are 19 unique genres, as well as no genres listed, which are classified independently or in combination with other genres:

```
tibble(count = str_count(train_data$genres, fixed("|")),
  genres = train_data$genres) %>%
  group_by(count, genres) %>%
  summarise('Quantity of Movies by Genre' = n()) %>%
  head(20)
```

count	genres	Quantity of Movies by Genre
0	(no genres listed)	7
0	Action	24482
0	Adventure	2276
0	Animation	329
0	Children	745
0	Comedy	700889
0	Crime	3197
0	Documentary	70041
0	Drama	733296
0	Fantasy	86
0	Film-Noir	1575
0	Horror	68738
0	IMAX	14
0	Musical	3851
0	Mystery	246
0	Romance	8410
0	Sci-Fi	10125
0	Thriller	94662
0	War	2300
0	Western	15300

This data shows that the highest genres (which may be a included in conjunction with others) are drama, followed by comedy, then thrillers. ### Rating Movies may be rated between 1 and 5 stars, which includes half-increments.

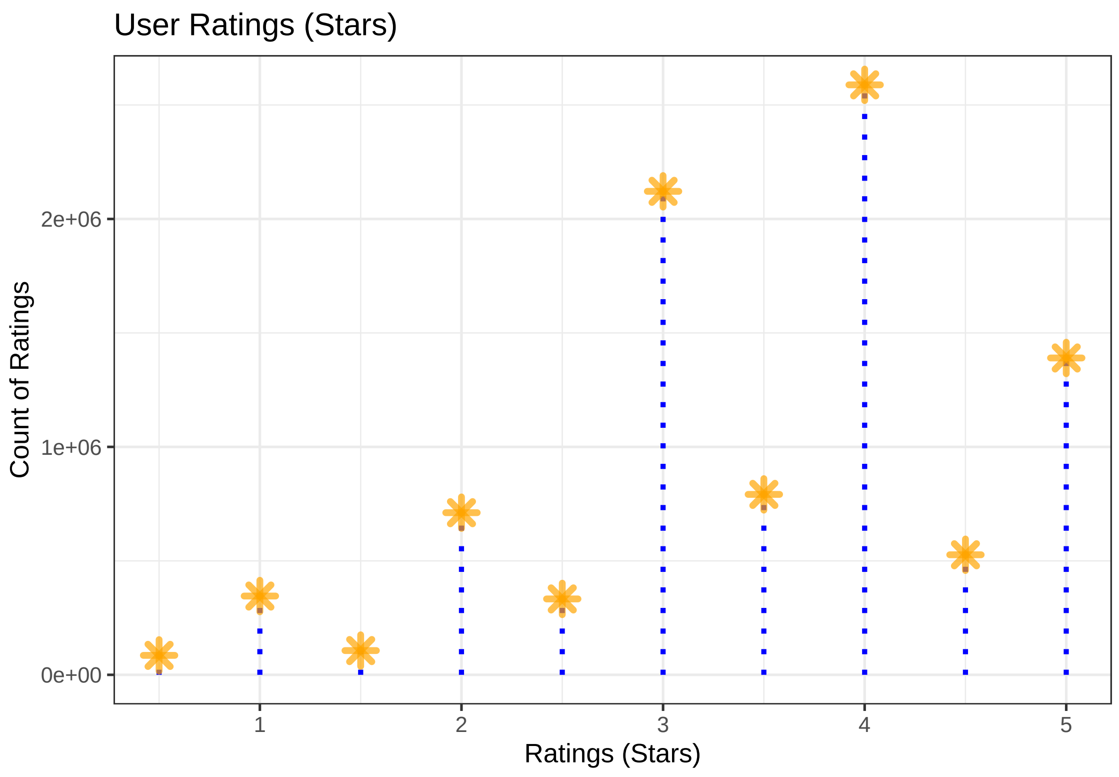
```
train_data %>%
  group_by('User Ratings - Stars' = rating) %>%
  summarize('Number of Ratings' = n())
```

User Ratings - Stars	Number of Ratings
0.5	85374
1.0	345679

User Ratings - Stars	Number of Ratings
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

Rating data visualization:

```r
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
library(ggplot2)
train_data %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x = rating, y = count)) +
  ggtitle("User Ratings (Stars)") +
  xlab("Ratings (Stars)") +
  ylab("Count of Ratings") +
  geom_segment(aes(x=rating, xend=rating, y=0, yend=count),
               size=1, color="blue", linetype = "dotted") +
  geom_point(size=3, color="orange", fill=alpha("orange", 0.3),
             alpha=0.7, shape=8, stroke=2) +
  theme_bw()
```

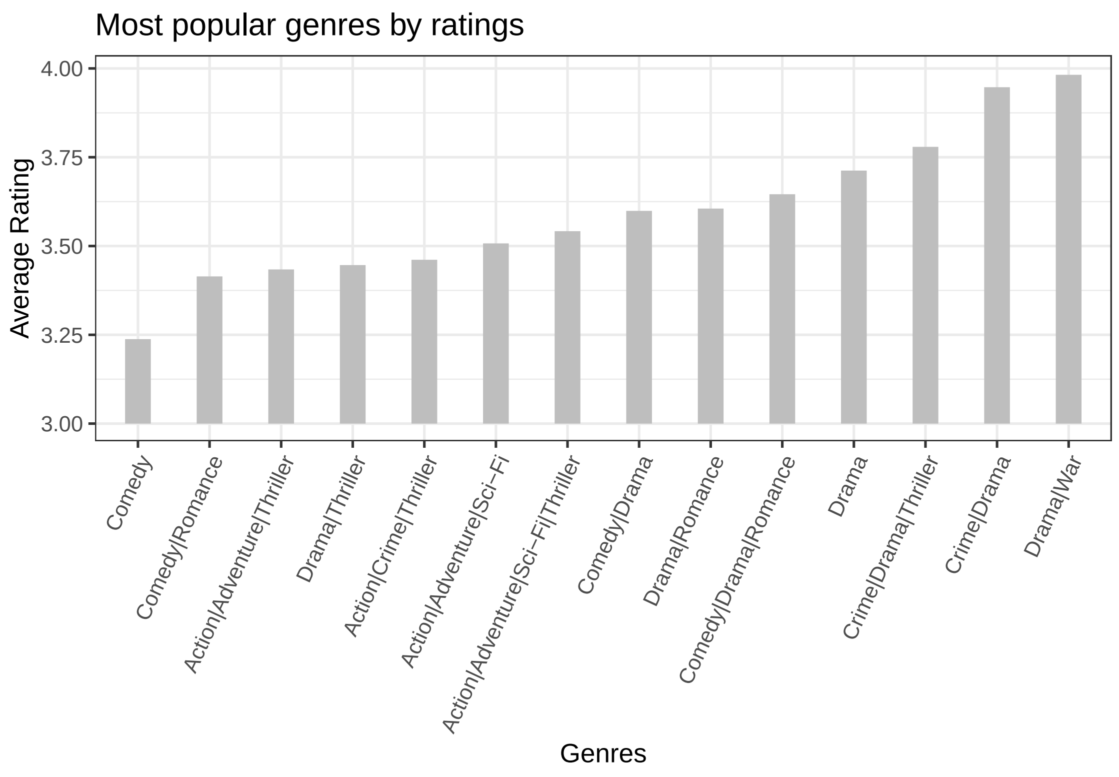


The data shows that 4 stars were given the most often, followed by 3 stars, then 5.

Now, let's correlate the genre data with the associated ratings

These are the most popular genres by ratings:

```
train_data %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_segment(aes(xend=genres, yend=3), size = 5, color="gray") +
  ggtitle("Most popular genres by ratings") +
  xlab("Genres") +
  ylab("Average Rating") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 65, hjust = 1))
```



UserId

We previously explored the user data, we determined that there are over 69,000 users in the training set.

```
length(unique(train_data$userId)) # Obtain quantity of users in training set
```

```
## [1] 69878
```

However, now we should consider which users rated movies more often and how they rated the movies:

Most ratings by userId:

```
train_data %>% group_by(userId) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  head(10)
```

userId	n
59269	6616
67385	6360

userId	n
14463	4648
68259	4036
27468	4023
19635	3771
3817	3733
63134	3371
58357	3361
27584	3142

Least ratings by userId:

```
train_data %>% group_by(userId) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  tail(10)
```

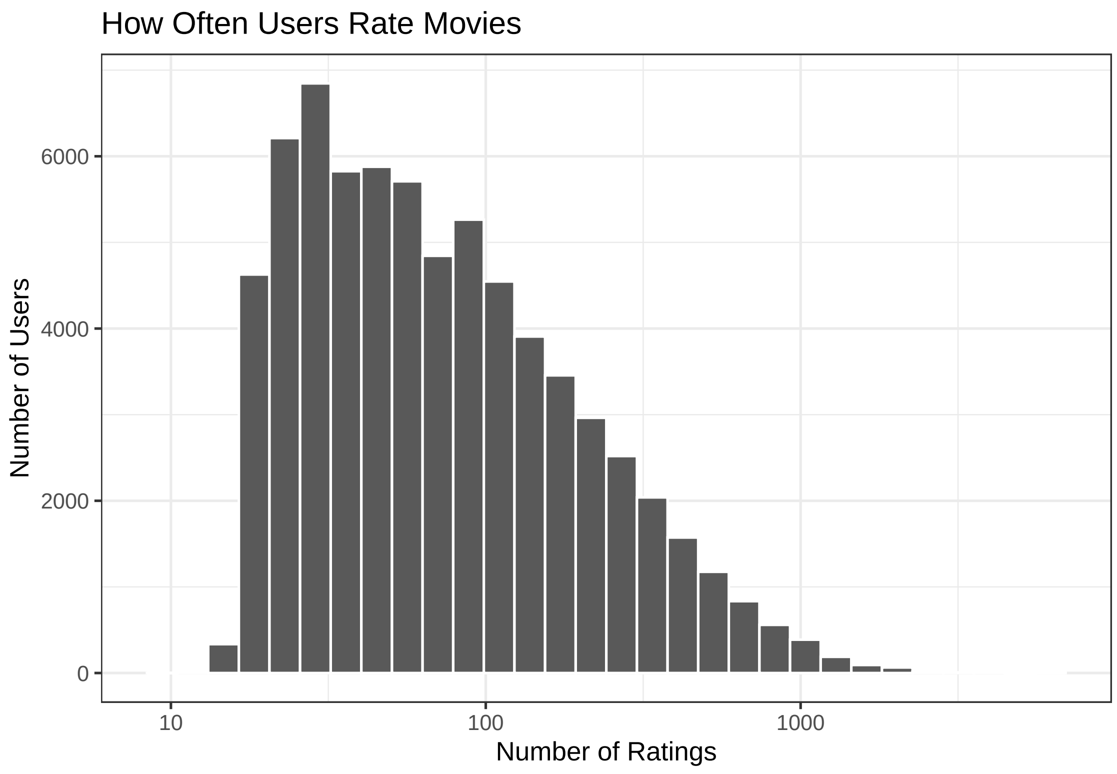
userId	n
57894	14
62317	14
63143	14
68161	14
68293	14
71344	14
15719	13
50608	13
22170	12
62516	10

It appears that some users rated movies more often, whereas other users rated less than ten movies.

This informational visualization indicates the frequency of movie ratings by users, let's look at the distribution with a histogram:

```
train_data %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("How Often Users Rate Movies") +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The user ratings appear to be skewed to the right, which indicates that some users are rating movies more often, and others rated movies less often, which we explored previously.

Data Strategy

Now that we have a better understanding of the dataset, we need to assess the type of data that we are seeking to analyze to increase our efficiency. These measures will speed up the data processing time by decreasing the model complexity.

For this project, we are interested in the ability to select 20 of the top movies from the genres based on user ratings. These selections will provide the ability to choose a movie from the list. If we have watched all of the movies on the list, we can modify the top movie list to include additional movies (i.e. top 50, 100, and so on).

Our initial step is to build the recommendation system while seeking to use the lowest RMSE possible.

Define RSME:

```r
# Define Root Mean Squared Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Creating our Model

Let's begin with creating a simple model for the recommendation system, followed by more complexity.

The objective of our recommendation system is to generate a model that will make the highest recommendations possible. This can be achieved by using various regression loss algorithms and tuning various model parameters to make predictions.

Initially, we will take a sample of movies shown in a matrix, and recognize that some users did not rate movies (shown as "NA"):

```
movie_Id <- train_data %>%
     dplyr::count(movieId) %>%
     top_n(5) %>%
     pull(movieId)
```

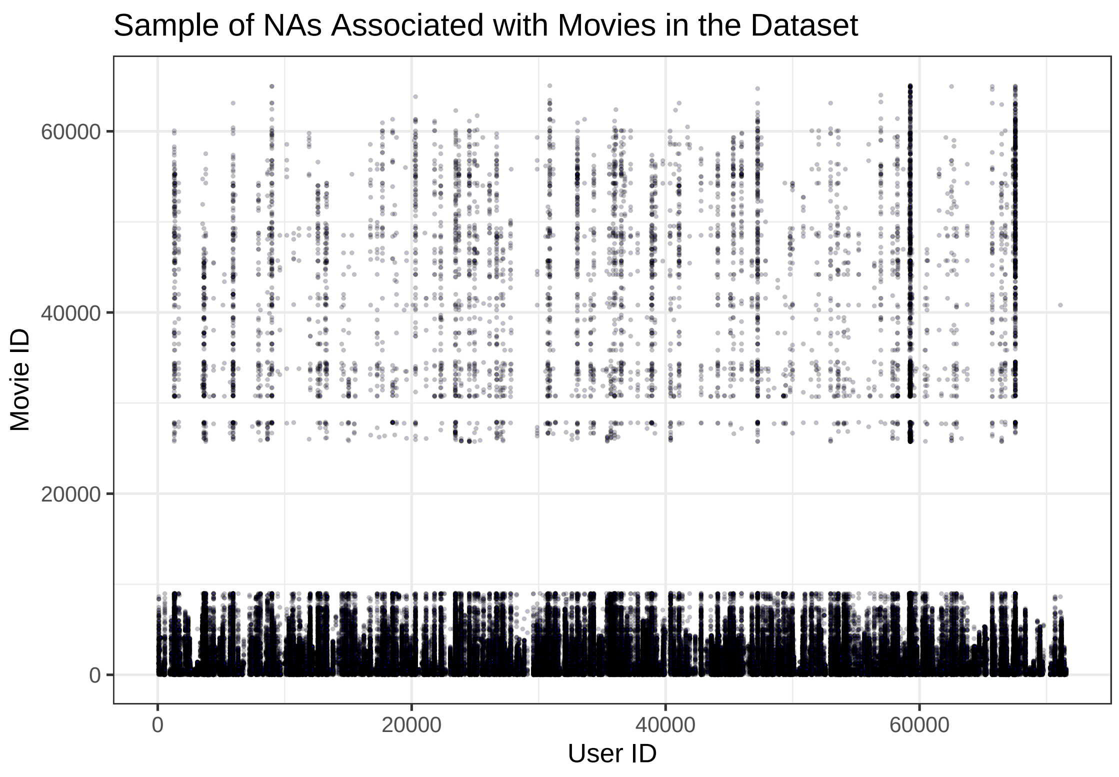
```
## Selecting by n
```

```
table_1 <- train_data  %>%
     filter(userId %in% c(13:20)) %>%
     filter(movieId %in% movie_Id) %>%
     select(userId, title, rating) %>%
     spread(title, rating)
table_1
```

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

Next, let's visualize the entire data set to see that there are many NAs in the table by taking a sample from 1000 users:

```
users <- sample(unique(train_data$userId), 1000)
data <- train_data %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = NA)

ggplot(data, aes(userId, movieId, fill = rating)) +
  geom_point(shape = 21, fill = "blue", size = .25, alpha = .25) +
    ggtitle("Sample of NAs Associated with Movies in the Dataset") +
    xlab("User ID") +
    ylab("Movie ID") +
    theme_bw()
```



It appears that some users provided more NAs than others, and it is difficult to assess why they exist. In addition, some movieIDs were more prone to having NAs than others. Our objective with tuning the recommendation system is seeking to fill in as many of the NAs as possible by optimizing the recommender system.

Recommender System Optimization

Now, let's create our recommendation system to minimize this loss function. We will build a series of models with an objective to build the most accurate recommendation system while minimizing losses. Our objective is to minimize the loss, which is reflected with the lowest possible RMSE as possible; an RMSE > 1 is not a good recommendation system.

First, we'll begin by omitting data that is unnecessary for our analysis, in this case we will not include the timestamp:

```
train_data <- train_data %>% select(userId, movieId, rating, title, genres)
test_data <- test_data %>% select(userId, movieId, rating, title, genres)
```

Define RMSE and determine the average

```
mu <- mean(train_data$rating)
rmse_mu <- RMSE(test_data$rating, mu)
rmses <- data_frame(method = "Average", RMSE = rmse_mu)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
rmses
```

method	RMSE
Average	1.061202

That average RMSE is relatively high, and we will need to improve it for making better recommendations.

Simplest Model

The simplest model that assumes the same rating for all movies and users explained by random variation uses the formula:

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

where $\epsilon_{i,u}$ independent errors sampled from the distribution centered at 0, and mu the "true" rating for the movies. RMSE is increased with values other than the mean.

Create the predictor from the training set ratings:

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

```
## [1] 3.512465
```

Movie Effects Bias

As we know, some movies are favored over others and have a higher rating. To compensate for this, we introduce the variable b_i which is referred to as the bias. This formula is as follows:

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

The following code may be used, which will take the least squares estimate b_i as the average $Y_{u,i}$ for each movie {i} (penalizing least squares):

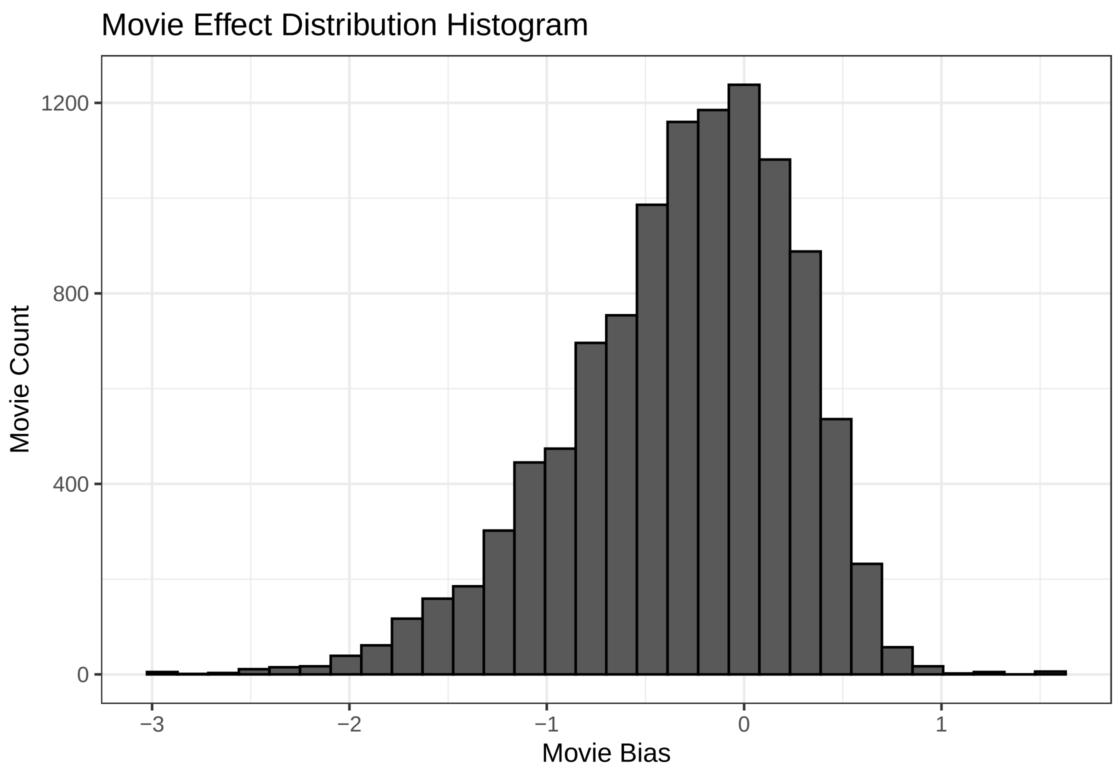
```r
# Movie effects bias
m_e_bias <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
head(m_e_bias)
```

movieId	b_i
1	0.4151725
2	-0.3070658
3	-0.3654817
4	-0.6481659
5	-0.4437933
6	0.3028191

Movie Effect Histogram:

```r
ME_hist <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_i)) +
    geom_histogram(color = "black") +
    ggtitle("Movie Effect Distribution Histogram") +
    xlab("Movie Bias") +
    ylab("Movie Count") +
    theme_bw()
plot(ME_hist)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Let’s see how the movie bias improves the RMSE for the recommendation system predictions:

```
movie_bias_predict <- mu + test_data %>%
  left_join(m_e_bias, by = 'movieId') %>%
  .$b_i
rmse_average <- RMSE(movie_bias_predict, test_data$rating)
rmses <- bind_rows(rmses,
                          tibble(method="Movie Bias Effect",
                                      RMSE = rmse_average))
rmses
```

method	RMSE
Average	1.0612018
Movie Bias Effect	0.9439087

User Effects Bias

Our RMSE is improving, however let’s see if the User Effects Bias will improve recommendations. This is the formula we will use, adding the user effects variable to the previous formula:

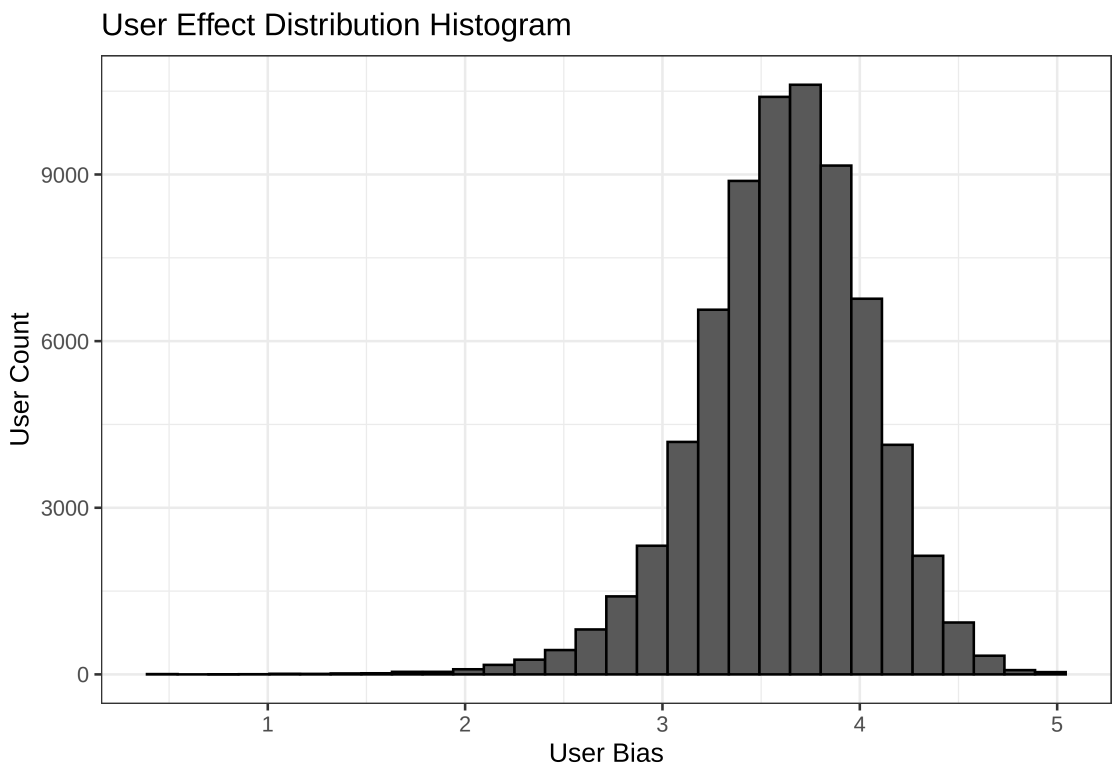
$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

```r
u_e_bias <- train_data %>%
  left_join(m_e_bias, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

u_e_predict <- test_data %>%
  left_join(m_e_bias, by='movieId') %>%
  left_join(u_e_bias, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_results_u_e <- RMSE(u_e_predict, test_data$rating)
rmses <- bind_rows(rmses,
                          tibble(method="User Bias Effect",
                                        RMSE = model_results_u_e))
rmses
```

method	RMSE
Average	1.0612018
Movie Bias Effect	0.9439087
User Bias Effect	0.8653488

The user effect is normally distributed.

```r
UE_hist <- train_data %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
    geom_histogram(color = "black") +
    ggtitle("User Effect Distribution Histogram") +
    xlab("User Bias") +
    ylab("User Count") +
    theme_bw()
plot(UE_hist)
```



There is variability across users, showing that some users rated movies with low scores whereas others rated every movie with a high score. The user effect seeks to more accurately predict approximations by computing both $\hat{u}$ and $\hat{b}_i$, and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{u} - \hat{b}_i$.

Although the RMSE is improving for the recommendation system, we would like to introduce the genres effect, as well as regularization to tune the model by adding the penalty factor $lambda$.

Genre Bias Effect

Let's see how the genres effect our predictions. Comparatively to the previous movie and user effects bias, we will add the genres using the following formula:

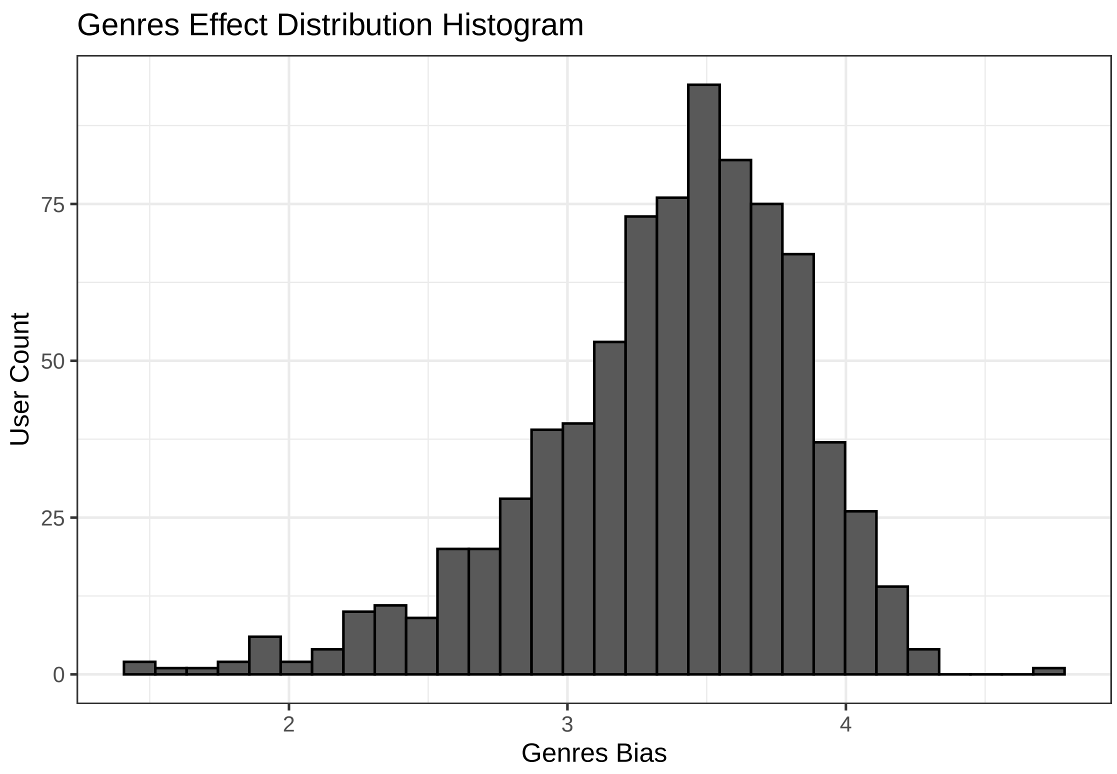
$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

```r
b_g_bias <- train_data %>%
  left_join(m_e_bias, by = 'movieId') %>%
  left_join(u_e_bias, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

b_g_predict <- test_data %>%
  left_join(m_e_bias, by = 'movieId') %>%
  left_join(u_e_bias, by='userId') %>%
  left_join(b_g_bias, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

model_results_g_e <- RMSE(b_g_predict, test_data$rating)
rmses <- bind_rows(rmses,
                          tibble(method="Genres Bias Effect",
                                      RMSE = model_results_g_e))
rmses
```

method	RMSE
Average	1.0612018
Movie Bias Effect	0.9439087
User Bias Effect	0.8653488
Genres Bias Effect	0.8649469

The genres effect is normally distributed, while slightly skewed to the right. It appears that users has a bias to rate some genres higher than others.

```
GE_hist <- train_data %>%
  group_by(genres) %>%
  summarize(g_e = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(g_e)) +
    geom_histogram(color = "black") +
    ggtitle("Genres Effect Distribution Histogram") +
    xlab("Genres Bias") +
    ylab("User Count") +
    theme_bw()
plot(GE_hist)
```



Regularization

Now that we have tuned this model with both the movie bias, user bias, and genre bias, we notice that our RMSE has improved. Let's initially explore the mistakes that were created by adding the bias, then determine how to correct this with regularization.

10 largest mistakes:

```r
# Using movie effects bias only, showing larger error rate
mistakes <- test_data %>%
  left_join(m_e_bias, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual)))

mistakes[1:10]
```

userId	movieId	rating	title	genres	b_i	residual
56965	6371	5.0	PokÃ©mon Heroes (2003)	Animation|Children	-2.4832681	3.970803
14871	318	0.5	Shawshank Redemption, The (1994)	Drama	0.9426660	-3.955131

userId	movieId	rating	title	genres	b_i	residual
39827	318	0.5	Shawshank Redemption, The (1994)	Drama	0.9426660	-3.955131
45000	318	0.5	Shawshank Redemption, The (1994)	Drama	0.9426660	-3.955131
37651	858	0.5	Godfather, The (1972)	Crime|Drama	0.9029008	-3.915366
51578	858	0.5	Godfather, The (1972)	Crime|Drama	0.9029008	-3.915366
59583	858	0.5	Godfather, The (1972)	Crime|Drama	0.9029008	-3.915366
29999	50	0.5	Usual Suspects, The (1995)	Crime|Mystery|Thriller	0.8533885	-3.865854
48793	50	0.5	Usual Suspects, The (1995)	Crime|Mystery|Thriller	0.8533885	-3.865854
53358	50	0.5	Usual Suspects, The (1995)	Crime|Mystery|Thriller	0.8533885	-3.865854

Now lets look at the 10 best and worst ranked movies according to the movie bias:

```
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()
```

Top 10 best movies:

```
# Using movie effects bias only, showing larger error rate
best <- m_e_bias %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  pull(title)
best[1:10]
```

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)"
##  [3] "Shadows of Forgotten Ancestors (1964)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Blue Light, The (Das Blaue Licht) (1932)"
##  [7] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
##  [8] "Human Condition II, The (Ningen no joken II) (1959)"
##  [9] "Human Condition III, The (Ningen no joken III) (1961)"
## [10] "Constantine's Sword (2007)"
```

10 worst movies:

```
# Using movie effects bias only, showing larger error rate
worst <- m_e_bias %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  pull(title)

worst[1:10]
```

```
##  [1] "Besotted (2001)"
##  [2] "Hi-Line, The (1999)"
##  [3] "Accused (Anklaget) (2005)"
##  [4] "Confessions of a Superhero (2007)"
##  [5] "War of the Worlds 2: The Next Wave (2008)"
##  [6] "SuperBabies: Baby Geniuses 2 (2004)"
##  [7] "Hip Hop Witch, Da (2000)"
##  [8] "Disaster Movie (2008)"
##  [9] "From Justin to Kelly (2003)"
## [10] "Criminals (1996)"
```

There are a few of these movies that we may be familiar with, however this doesn't seem right for a recommendation system! Let's explore a little further, then figure out how to performance tune this model.

Frequency of best rated movies:

```
# Using movie effects bias only, showing larger error rate
freq <- train_data %>% count(movieId) %>%
  left_join(m_e_bias, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
    arrange(desc(b_i)) %>%
  pull(n)

freq[1:10]
```

```
##  [1] 1 2 1 1 1 1 4 4 4 2
```

This data doesn't make sense - there are few users associated with ratings for the best movies. If we explored the worst movies, we would see that the same condition holds true.

Using Regularization to Tune the Model

With the use of regularization we can further improve the model by penalizing large estimates that are focused on small sample sizes. It is similar to the Bayesian approach which shrinks predictions. The ability to penalize estimates provides the ability to control the variability of the movie effects, instead of minimizing the least squares equation. We use the training set to tune the model (the test set should never be used).

We minimize an equation that adds a penalty using the following formula, where the first term is least squares and the second is a penalty that gets larger when many {b_i} are large:

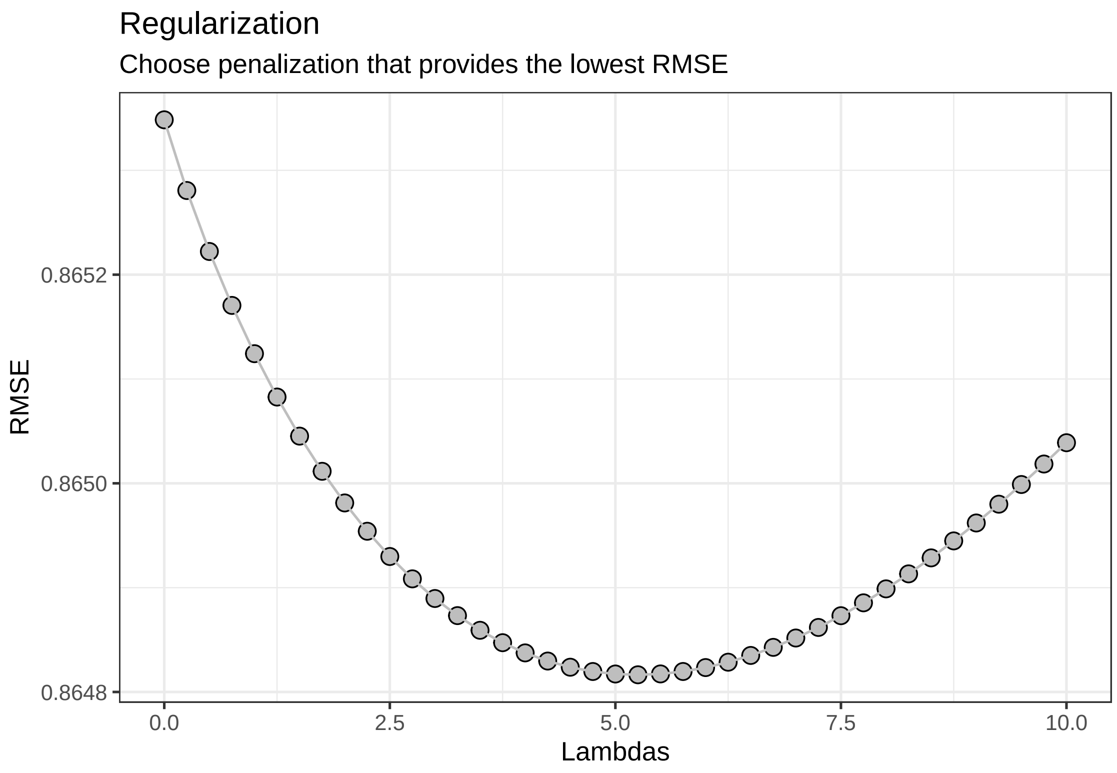
$$\frac{1}{N}\sum_{u,i}(y_{u,i} - - b_i)^2 + \lambda \sum_{i} b_i^2$$

The most effective use of using $lambda$ is to use different values and run several simulations to minimize the RMSE.

We will begin with choosing a set of $lambda$ to tune the model, while using the best performing RMSE model to make our predictions (user bias model - which included the baseline and movie effects). In the subsequent step we will optimize $lambda$:

```r
# Define lambdas penalty to tune the model
lambdas <- seq(0, 10, 0.25)

# Define RMSE as a function of mu, b_i, b_u, and predicted ratings
lambda_tune <- sapply(lambdas, function(l){

  mu <- mean(train_data$rating)

  b_i <- train_data %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_data %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_g <- train_data %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u)/(n()+l))

  predicted_ratings <-
    test_data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_data$rating))
})
```

```r
reg_tune_lambda <- tibble(Lambda = lambdas, RMSE = lambda_tune) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point(shape=21, color="black", fill="grey", size=3) +
  geom_line(color="grey") +
  ggtitle("Regularization",
          subtitle = "Choose penalization that provides the lowest RMSE") +
  xlab("Lambdas") +
  ylab("RMSE") +
  theme_bw()

plot(reg_tune_lambda)
```



Let's choose the $lambda$ that best fits the model using cross-validation, which will provide the lowest RMSE:

```r
lambda <- lambdas[which.min(lambda_tune)]
lambda
```

```
## [1] 5.25
```

Let's use the previous model, this time choosing the smallest $lambda$, then see how the RMSE improves:

```r
rmses <- bind_rows(rmses,
                   tibble(method="Regularization",
                          RMSE = min(lambda_tune)))
rmses
```

method	RMSE
Average	1.0612018
Movie Bias Effect	0.9439087
User Bias Effect	0.8653488
Genres Bias Effect	0.8649469
Regularization	0.8648165

This appears to be a small improvement, however it should correlate to significant improvements to our recommendations.

Tuned Recommender System

Let's apply the best λ to make predictions using regularization:

```r
# Choose the mimimum lambda
lambda <- lambdas[which.min(lambda_tune)]

# Take the mean of mu
mu <- mean(train_data$rating)

# Movie effect bias
b_i <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect bias
b_u <- train_data %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Genre effect bias
b_g <- train_data %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u)/(n()+lambda))

# Prediction
```

```
predict_reg <- train_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
```

Top 10 movies:

```
train_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  distinct(title) %>%
  head(10)
```

title
Shawshank Redemption, The (1994)
Usual Suspects, The (1995)
Schindler's List (1993)
Seven Samurai (Shichinin no samurai) (1954)
Godfather, The (1972)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Star Wars: Episode V - The Empire Strikes Back (1980)
Memento (2000)
Matrix, The (1999)

These seem like more accurate predictions based on the user ratings.

Worst 10 recommended movies:

```
train_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(pred) %>%
  group_by(title) %>%
  distinct(title) %>%
  head(10)
```

title
Superman IV: The Quest for Peace (1987)
Aces: Iron Eagle III (1992)
Dumb and Dumberer: When Harry Met Lloyd (2003)
Police Academy 6: City Under Siege (1989)
Amityville 1992: It's About Time (1992)
Meatballs III (1987)
Turbo: A Power Rangers Movie (1997)
Spice World (1997)
Street Fighter (1994)
Speed 2: Cruise Control (1997)

These selections look much more accurate, although we might be familiar with some of the titles.

And finally…Which Movie Should I Watch Tonight?

Let's use high selectivity with our criteria to ensure that we are making good use of our time. We have the ability to filter by any of the factors within the dataset, such as the timeframe, number of ratings, and others. For tonight's movie we will utilize the entire dataset, while having 20 selections from a few genres of choice:

Action:

```r
movielens %>%
  filter(str_detect(genres,"Action")) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(-pred) %>%
  distinct(title) %>%
  head(20)
```

title
Seven Samurai (Shichinin no samurai) (1954)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Star Wars: Episode V - The Empire Strikes Back (1980)
Matrix, The (1999)
Fight Club (1999)
Saving Private Ryan (1998)
Professional, The (Le Professionnel) (1981)
Butch Cassidy and the Sundance Kid (1969)
Braveheart (1995)

title
Alien (1979)
Glory (1989)
Indiana Jones and the Last Crusade (1989)
Boat, The (Das Boot) (1981)
Star Wars: Episode VI - Return of the Jedi (1983)
French Connection, The (1971)
Snatch (2000)
Aliens (1986)
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
Lord of the Rings: The Fellowship of the Ring, The (2001)

Comedy:

```r
movielens %>%
  filter(str_detect(genres,"Comedy")) %>%
  filter(sum(rating) >= 10000) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(-pred) %>%
  distinct(title) %>%
  head(20)
```

title
Wallace & Gromit: The Wrong Trousers (1993)
Pulp Fiction (1994)
Fargo (1996)
Wallace & Gromit: A Close Shave (1995)
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
Eternal Sunshine of the Spotless Mind (2004)
Butch Cassidy and the Sundance Kid (1969)
Postman, The (Postino, Il) (1994)
Forrest Gump (1994)
Snatch (2000)
Dog Day Afternoon (1975)
Little Miss Sunshine (2006)
Sense and Sensibility (1995)
Incredibles, The (2004)
Office Space (1999)
Lock, Stock & Two Smoking Barrels (1998)
Ferris Bueller's Day Off (1986)
Toy Story (1995)
Amelie (Fabuleux destin d'AmÃ©lie Poulain, Le) (2001)

title
Annie Hall (1977)

Documentary:

```
movielens %>%
  filter(str_detect(genres,"Documentary")) %>%
  filter(sum(rating) >= 10000) %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(-pred) %>%
  distinct(title) %>%
  head(20)
```

title
When We Were Kings (1996)
Hoop Dreams (1994)
Paradise Lost: The Child Murders at Robin Hood Hills (1996)
Celluloid Closet, The (1995)
Microcosmos (Microcosmos: Le peuple de l'herbe) (1996)
Baraka (1992)
Looking for Richard (1996)
Anne Frank Remembered (1995)
Line King: The Al Hirschfeld Story, The (1996)
Thin Blue Line, The (1988)
Shoah (1985)
Hands on a Hard Body (1996)
Crumb (1994)
Stop Making Sense (1984)
Bill Cosby, Himself (1983)
Roger & Me (1989)
Startup.com (2001)
42 Up (1998)
Buena Vista Social Club (1999)
Paris Is Burning (1990)

Results

The results from this movie recommender system project were highly successful. Using several iterations of RMSE analysis, the value was decreased substantially to provide more accurate predictions from the movie data. Movie predictions were evident through the demonstration of movie choices both before and after the RMSE was optimized. The final movie recommender system appeared to have a couple outliers, which were not rated with 5 stars.

```
rmses
```

method	RMSE
Average	1.0612018
Movie Bias Effect	0.9439087
User Bias Effect	0.8653488
Genres Bias Effect	0.8649469
Regularization	0.8648165

Conclusion

The process of building the recommendation system began with exploring the data content, then recognizing how this data could be used to better filter and create a recommendation system.

Limitations

Although this recommendation system was optimized to an initial level of satisfaction for this project, there are many other methods that could be used to increase the effectiveness of this system. Limitations of this system can consider additional measurements of loss functions, uses of regularization, and also considering matrix factorization to reduce the RMSE.

Furthermore, there are other characteristics that could be considered for evaluation, such as filtering outliers, as well as using some of the other larger data sets which include tags, actors, and other detailed information.

Future Work

Although this project was a success, this type of recommender system could be tuned and scaled for use with other types of data sets, as well as the ability to use other types of machine learning and analysis packages that are available.

References

1. Irizarry, R., 2019. Introduction To Data Science. [online] Rafalab.github.io. Available at: <rafalab.github.io/dsbook>.
2. Leskovec, J., Rajaraman, A. and Ullman, J., 2014. Mining of Massive Datasets. 2nd ed. pp.319-350.
3. Koren, Y., 2009. The BellKor Solution to the Netflix Grand Prize. Available at: <www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf>.
4. Lohr, S., 2009. Netflix Awards $1 Million Prize and Starts a New Contest. In: The New York Times. bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest.
5. “MovieLens 10M Dataset.” GroupLens, 2019, Available at: <grouplens.org/datasets/movielens/10m/>.
6. Grover, Prince. “5 Regression Loss Functions All Machine Learners Should Know.” Medium, Heartbeat, 27 May 2020, heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0.
7. Parmar, Ravindra. “Common Loss Functions in Machine Learning.” Medium, Towards Data Science, 2 Sept. 2018, towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.
8. Sobrado, Daniel. “Loss Functions: MAE, MAPE, MSE, RMSE and RMSLE.” Danielsobrado.com, 23 July 2016, <danielsobrado.com/post/loss-functions-mae-mape-mse-rmse-and-rmsle/>.