# MovieLens

Mohammad Awwad

12/12/2021

## Introduction

The capacity to offer item suggestions to future users or customers is one of the key families of machine learning applications in the information technology sector. Netflix issued a challenge to the data science community in 2006. The goal was to improve Netflix's in-house software by 10% and earn a $1 million reward. This capstone project is part of the HarvardX:??PH125.9x  is based on the winners team algorithm. Because Netflix data isn't freely available, an open source dataset from movieLens was used instead: '10M version of the MovieLens dataset'?? The goal of this project is to create a machine learning algorithm that can predict movie scores in the validation set using inputs from one subset. Several machine learning algorithms were utilized, and the results were compared to obtain the highest possible prediction accuracy.

The following sections of this report are written in the following order: problem definition, data intake, exploratory analysis, modeling and data analysis, outcomes, and concluding remarks.

## Problem Defnition

This capstone project on 'Movie recommendation system??? predicts a user's movie rating based on their previous movie ratings. The dataset that was utilized for this project may be available at the following locations.

- [MovieLens 10M dataset] https://grouplens.org/datasets/movielens/10m/
- [MovieLens 10M dataset - zip file] http://files.grouplens.org/datasets/movielens/ml-10m.zip

Given the many different types of biases prevalent in movie reviews, the challenge is not easy. It could be a variety of social, psychological, or demographic factors that influence each user's preference for a certain film. However, the problem can still be tailored to address major biases that are simply represented using mathematical formulae. The goal is to create a model that can accurately forecast movie suggestions for a given user without compromising our judgment owing to various biases. The prevalences can be suppressed in the algorithm utilizing some creative mathematical methods. As we progress through this paper, this will become evident.

## Data Ingestion

The code is provided in the edx capstone project module [Create Test and Validation Sets]

```
#Create test and validation sets
# Create edx set, validation set, and submission file
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")
# 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 <- read.table(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(levels(movieId))[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)
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)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The code above creates a dataset partition for training and testing our dataset. It also cleans up the working directory by removing any unneeded files, which is always a good coding practice ('always clean after you cook').

```
# Validation dataset can be further modified by removing rating column
validation_CM <- validation
validation <- validation %>% select(-rating)
```

```
# extra libraries that might be usefull in analysis and visulizations
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

Once a clean dataset has been obtained, it is necessary to investigate the dataset's features and compute the basic summary statistics.

```
## the dataset and its basic summary statistics
# intial 7 rows with header
head(edx)
```

```
##   userId movieId rating timestamp title genres
## 1      1     122      5 838985046  <NA>   <NA>
## 2      1     185      5 838983525  <NA>   <NA>
## 3      1     231      5 838983392  <NA>   <NA>
## 4      1     292      5 838983421  <NA>   <NA>
## 5      1     316      5 838983392  <NA>   <NA>
## 6      1     329      5 838983392  <NA>   <NA>
```

```
# basic summary statistics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   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:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# total number of observations
tot_observation <- length(edx$rating) + length(validation$rating)
```

We can see that the dataset is in a neat format and is ready to be explored and analyzed.

## Exploratory analysis and data pre-processing

```
#  Since RMSE (root mean squre error) is used frequency so lets define a function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2,na.rm=T))
}
# lets modify the columns to suitable formats that can be further used for analysis
# Modify the year as a column in the edx & validation datasets
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation_CM <- validation_CM %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
# Modify the genres variable in the edx & validation dataset (column separated)
split_edx  <- edx  %>% separate_rows(genres, sep = "\\|")
split_valid <- validation   %>% separate_rows(genres, sep = "\\|")
split_valid_CM <- validation_CM  %>% separate_rows(genres, sep = "\\|")
```

##Exploration of data and statistics in general

```
# The 1st rows of the edx & split_edx datasets are presented below:
head(edx)
```

```
##   userId movieId rating timestamp title genres year
## 1      1     122      5 838985046  <NA>   <NA>   NA
## 2      1     185      5 838983525  <NA>   <NA>   NA
## 3      1     231      5 838983392  <NA>   <NA>   NA
## 4      1     292      5 838983421  <NA>   <NA>   NA
## 5      1     316      5 838983392  <NA>   <NA>   NA
## 6      1     329      5 838983392  <NA>   <NA>   NA
```

```
head(split_edx)
```

```
## # A tibble: 6 x 7
##   userId movieId rating timestamp title genres  year
##    <int>   <dbl>  <dbl>     <int> <chr> <chr>  <dbl>
## 1      1     122      5 838985046 <NA>  <NA>      NA
## 2      1     185      5 838983525 <NA>  <NA>      NA
## 3      1     231      5 838983392 <NA>  <NA>      NA
## 4      1     292      5 838983421 <NA>  <NA>      NA
## 5      1     316      5 838983392 <NA>  <NA>      NA
## 6      1     329      5 838983392 <NA>  <NA>      NA
```

```
# edx Summary Statistics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   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              year
##  Length:9000061     Length:9000061     Min.   : NA
##  Class :character   Class :character   1st Qu.: NA
##  Mode  :character   Mode  :character   Median : NA
##                                        Mean   :NaN
##                                        3rd Qu.: NA
##                                        Max.   : NA
##                                        NA's   :9000061
```

```
# Number of unique movies and users in the edx dataset
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

### Total movie ratings per genre

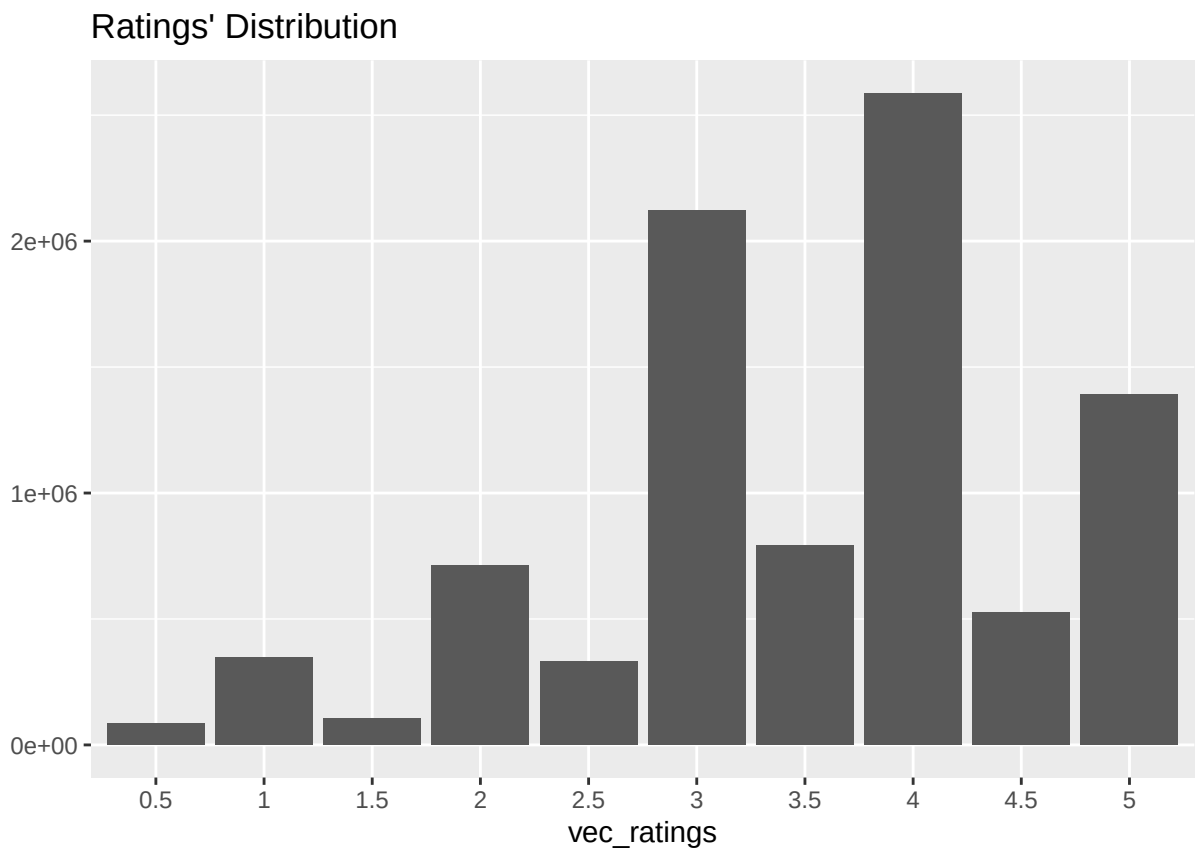
```
genre_rating <- split_edx%>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

### Ratings distribution

```
vec_ratings <- as.vector(edx$rating)
unique(vec_ratings)
```

```
##  [1] 5.0 3.0 2.0 4.5 3.5 4.0 1.0 1.5 2.5 0.5
```

```
vec_ratings <- vec_ratings[vec_ratings != 0]
vec_ratings <- factor(vec_ratings)
qplot(vec_ratings) +
  ggtitle("Ratings' Distribution")
```



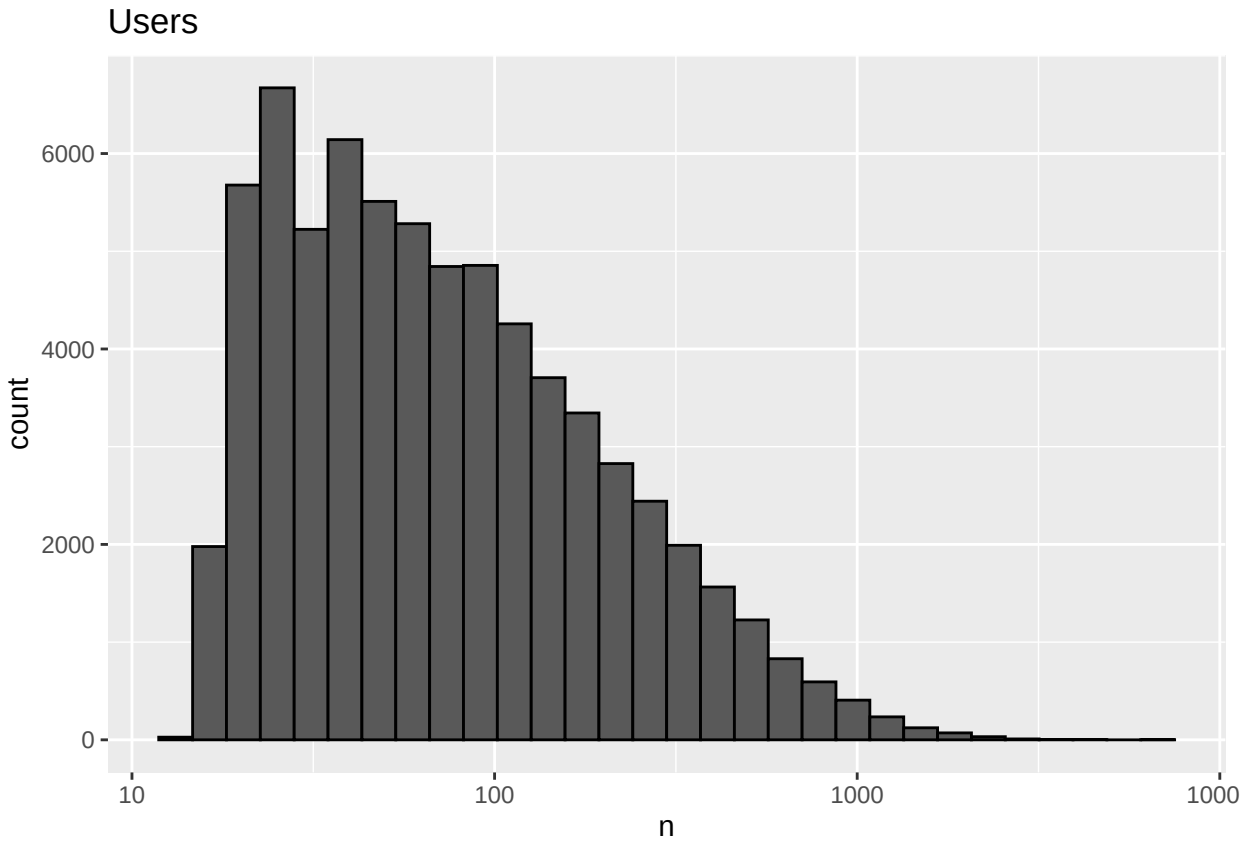

Users have a general inclination to score movies between 3 and 4 stars, as shown by the above rating distribution. This is an overarching conclusion. To develop a strong predictive model, we should investigate the impact of various features further.

## Data Analysis Strategies

- Some films are given higher ratings than others (e.g. blockbusters are rated higher). Find movie bias as a way to incorporate this into our model.
- Some users leave positive ratings, while others leave bad evaluations based on their own personal preferences, regardless of the film. Finding users' bias is one way to address these qualities.
- The popularity of the film genre is heavily influenced by current events. As a result, we should also look into time-dependent analyses. The best way to tackle this concept is to: find the popularity of a genre through time
- Does the user's mindset change over time? This can have an impact on the average movie rating over time. What is the best way to visualize such an effect: storyline rating vs. year of release

**The distribution of movie ratings by each user. This demonstrates the user's partiality.**

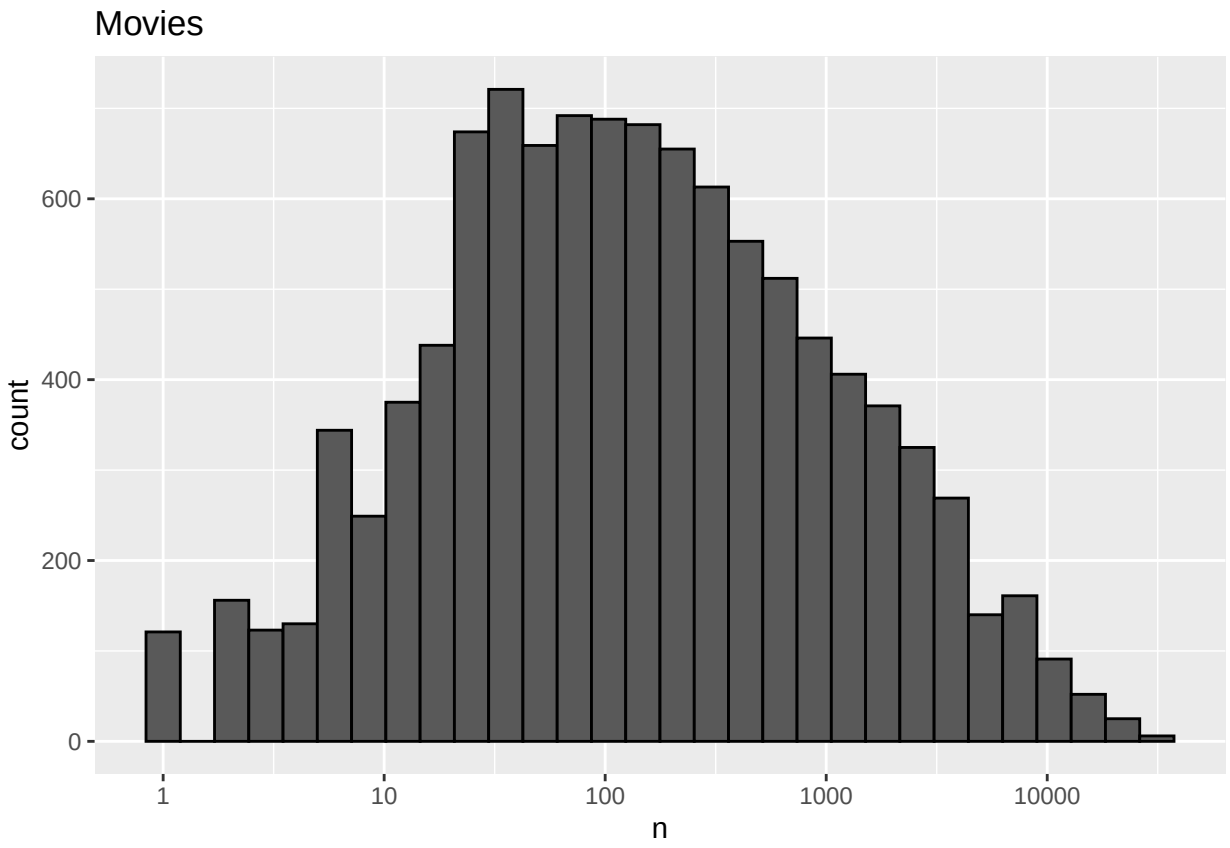
```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```




The graph above indicates that not all users are equally active. Some individuals have only rated a few movies, and their opinions may skew the results.

**Some films receive more ratings than others. Their distribution is seen below. This article looks into movie biases.**

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




The histogram reveals that certain films have only been rated a few times. As a result, they should be given less weight in movie predictions.

## Data Analysis: Model Preparation

```
#Initiate RMSE results to compare various models
rmse_results <- data_frame()
```

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

## Simplest possible model

Regardless of the user or movie, the mean rating of the dataset is utilized to forecast the same rating for all movies.

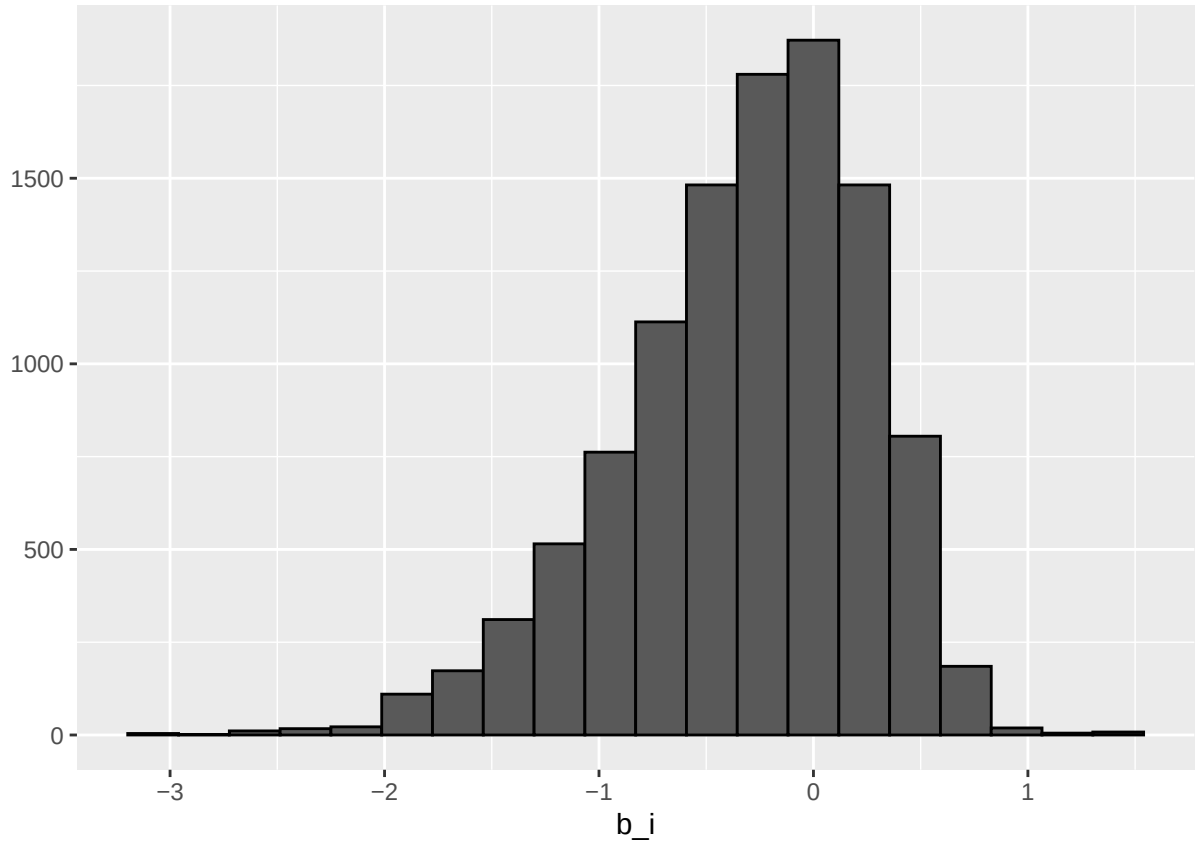
```
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512464
```

## Penalty Term (b_i)- Movie Effect

Films are assessed in a variety of ways. The histogram is not symmetric and skewed towards the negative rating effect, as revealed in the investigation. The movie impact can be accounted for by subtracting the difference from the mean rating, as demonstrated in the code below.

```
movie_avgs_norm <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs_norm %>% qplot(b_i, geom ="histogram", bins = 20, data = ., color = I("black"))
```

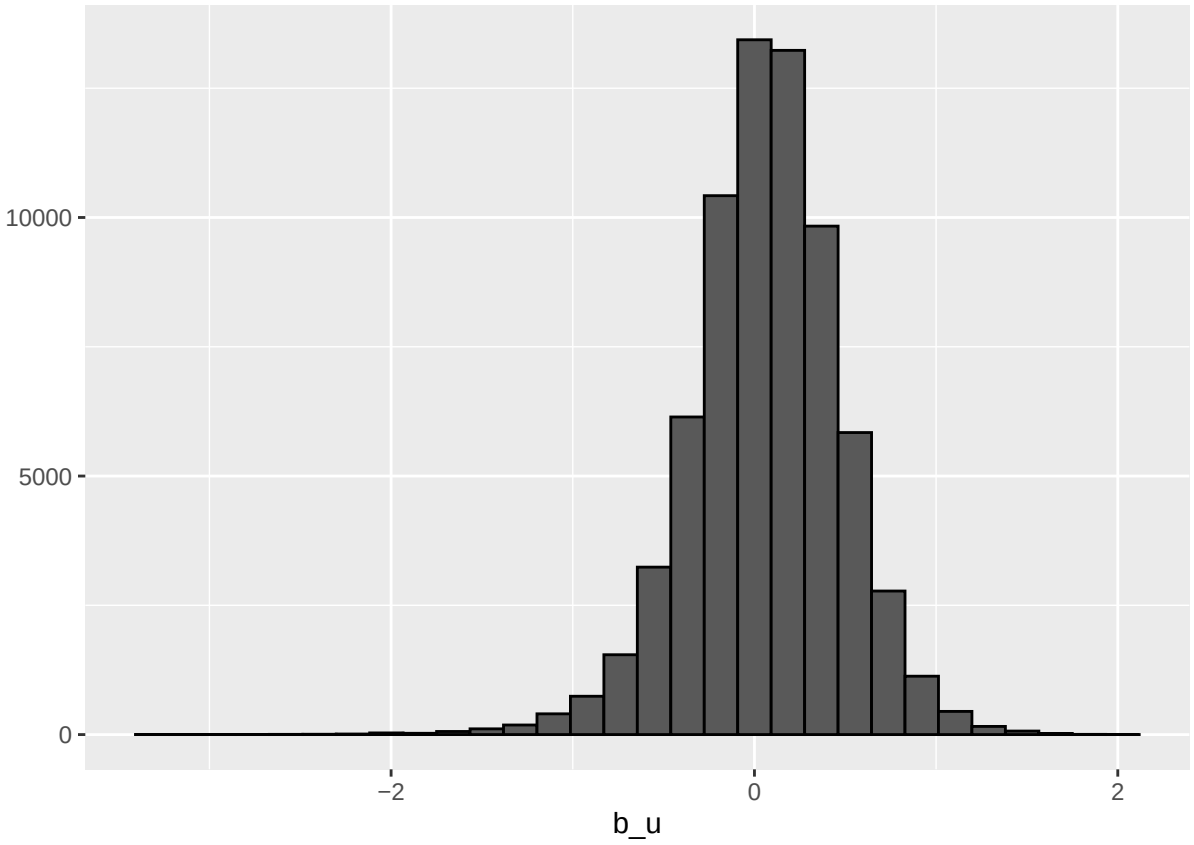



### Penalty Term (b_u)- User Effect

In terms of how people rate movies, different users have varied preferences. Some grumpy people may give an excellent movie a lower rating, while others are simply uninterested in ratings. This pattern was previously

visible in our data exploration plot (user bias). This code can be used to calculate it.

```
user_avgs_norm <- edx %>%
  left_join(movie_avgs_norm, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_avgs_norm %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```




## Model Creation

The RMSE will be used to evaluate the model's quality (the lower the better).

### Baseline Model

It's just a model that ignores all the feathers and calculates the average rating. This model will serve as a benchmark against which we will aim to improve RMSE.

```
# baseline Model: just the mean
baseline_rmse <- RMSE(validation_CM$rating,mu)
## Test results based on simple prediction
baseline_rmse
```

```
## [1] 1.060651
```

```
## Check results
rmse_results <- data_frame(method = "Using mean only", RMSE = baseline_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method           RMSE
##   <chr>           <dbl>
## 1 Using mean only  1.06
```

### Movie Effect Model

By incorporating the movie effect, the RMSE can be improved.

```
# Movie effects only
predicted_ratings_movie_norm <- validation %>%
  left_join(movie_avgs_norm, by='movieId') %>%
  mutate(pred = mu + b_i)
model_1_rmse <- RMSE(validation_CM$rating,predicted_ratings_movie_norm$pred)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Using mean only | 1.0606506 |
| Movie Effect Model | 0.9437046 |

```
rmse_results
```

```
## # A tibble: 2 x 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Using mean only    1.06
## 2 Movie Effect Model 0.944
```

The error has drop by 5% and motivates us to move on this path further.

### Movie and User Effect Model

Because both the movie and user biases confuse the prediction of movie rating, adding the user impact improves the RMSE even more.

```
# Use test set,join movie averages & user averages
# Prediction equals the mean with user effect b_u & movie effect b_i
predicted_ratings_user_norm <- validation %>%
  left_join(movie_avgs_norm, by='movieId') %>%
  left_join(user_avgs_norm, by='userId') %>%
  mutate(pred = mu + b_i + b_u)
# test and save rmse results
```

```
model_2_rmse <- RMSE(validation_CM$rating,predicted_ratings_user_norm$pred)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and User Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Using mean only | 1.0606506 |
| Movie Effect Model | 0.9437046 |
| Movie and User Effect Model | 0.8655329 |

```
rmse_results
```

```
## # A tibble: 3 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Using mean only                1.06
## 2 Movie Effect Model             0.944
## 3 Movie and User Effect Model 0.866
```

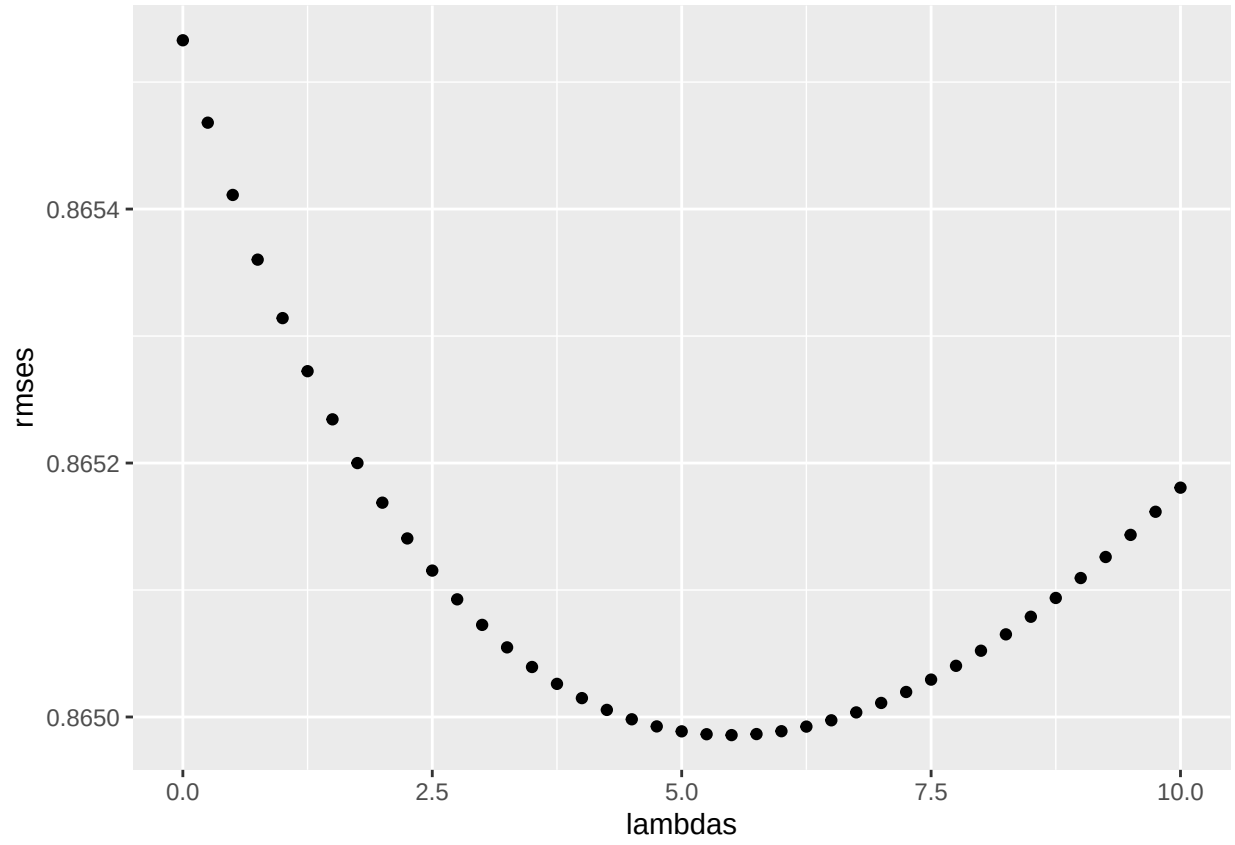
This is a good improvement from our last model.

**An technique based on regularization (motivated by Netflix challenge)**

During our data analysis, we discovered that some people are more engaged in movie reviews than others. There are other users who have rated a small number of films (less than 30 movies). On the other hand, some films receive only a few ratings (say 1 or 2). We should not trust these estimations because they are noisy. Furthermore, RMSE are susceptible to huge mistakes. Our residual mean squared error can be increased by large errors. As a result, we must include a penalty word to devalue such an effect.

```
# lambda is a tuning parameter
# Use cross-validation to choose it.
lambdas <- seq(0, 10, 0.25)
# For each lambda,find b_i & b_u, followed by rating prediction & testing
# note:the below code could take some time
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

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

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

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
```

```
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(validation_CM$rating,predicted_ratings))
})
# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, rmses)
```




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

```
## [1] 5.5
```

```
# Compute regularized estimates of b_i using lambda
movie_avgs_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
# Compute regularized estimates of b_u using lambda
user_avgs_reg <- edx %>%
  left_join(movie_avgs_reg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
# Predict ratings
```

```
predicted_ratings_reg <- validation %>%
  left_join(movie_avgs_reg, by='movieId') %>%
  left_join(user_avgs_reg, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
# Test and save results
model_3_rmse <- RMSE(validation_CM$rating,predicted_ratings_reg)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie and User Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Using mean only | 1.0606506 |
| Movie Effect Model | 0.9437046 |
| Movie and User Effect Model | 0.8655329 |
| Regularized Movie and User Effect Model | 0.8649857 |

```
rmse_results
```

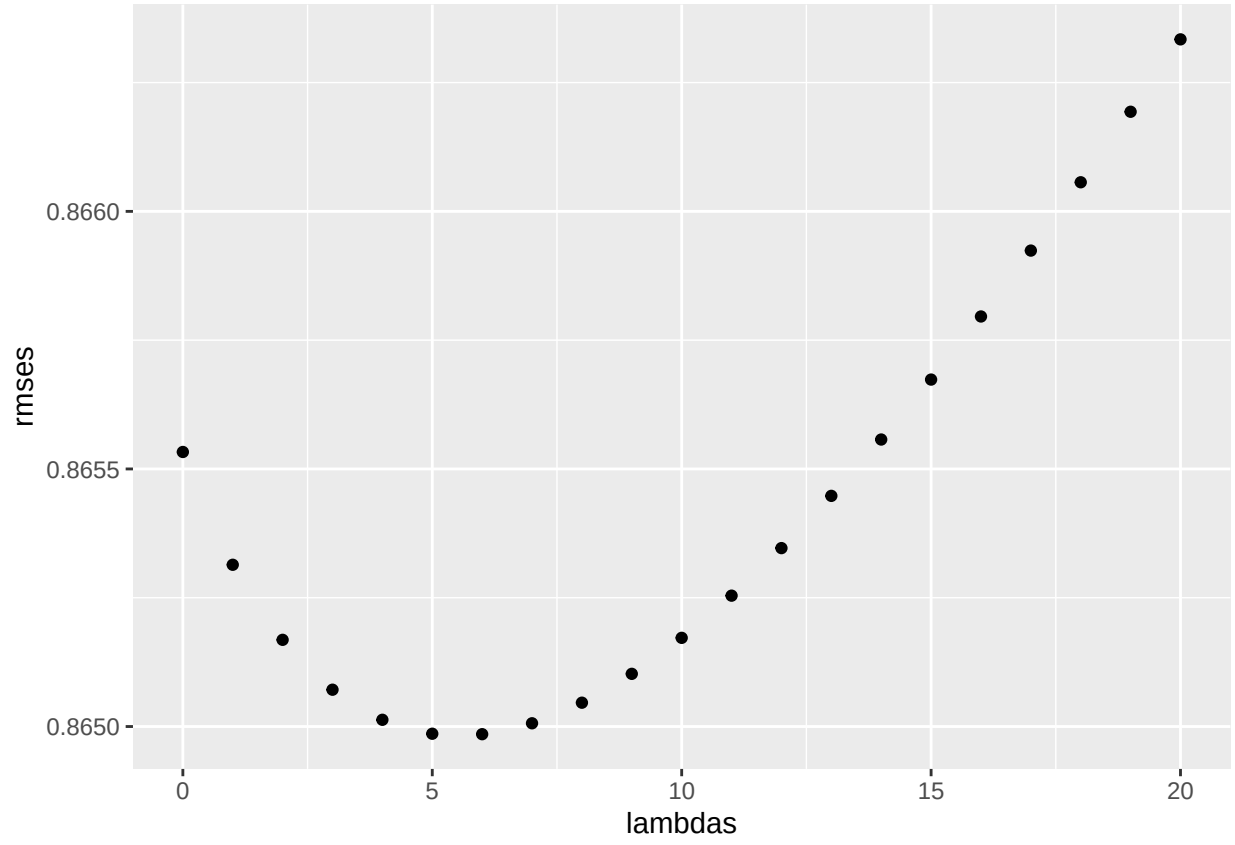
```
## # A tibble: 4 x 2
##   method                                  RMSE
##   <chr>                                  <dbl>
## 1 Using mean only                        1.06
## 2 Movie Effect Model                     0.944
## 3 Movie and User Effect Model            0.866
## 4 Regularized Movie and User Effect Model 0.865
```

**Regularization using movies, users, years and genres.**

The approach utilized in the above model is implemented below with the added genres and release year effects.

```
# b_y and b_g represent the year & genre effects, respectively
lambdas <- seq(0, 20, 1)
# Note: the below code could take some time
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

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

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

  b_y <- split_edx %>%
```

```
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+lambda), n_y = n())

  b_g <- split_edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by = 'year') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+lambda), n_g = n())
    predicted_ratings <- split_valid %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by = 'year') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred

  return(RMSE(split_valid_CM$rating,predicted_ratings))
})
# Compute new predictions using the optimal lambda
# Test and save results
qplot(lambdas, rmses)
```



```
lambda_2 <- lambdas[which.min(rmses)]
lambda_2
```

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

```
movie_reg_avgs_2 <- split_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_2), n_i = n())
user_reg_avgs_2 <- split_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda_2), n_u = n())
year_reg_avgs <- split_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+lambda_2), n_y = n())
genre_reg_avgs <- split_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  left_join(year_reg_avgs, by = 'year') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+lambda_2), n_g = n())
predicted_ratings <- split_valid %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  left_join(year_reg_avgs, by = 'year') %>%
  left_join(genre_reg_avgs, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  .$pred
model_4_rmse <- RMSE(split_valid_CM$rating,predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Reg Movie, User, Year, and Genre Effect Model",
                                     RMSE = model_4_rmse ))
rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Using mean only | 1.0606506 |
| Movie Effect Model | 0.9437046 |
| Movie and User Effect Model | 0.8655329 |
| Regularized Movie and User Effect Model | 0.8649857 |
| Reg Movie, User, Year, and Genre Effect Model | 0.8649851 |

# Results

## RMSE overview

The RMSE values for the used models are shown below:

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

| method | RMSE |
|---|---|
| Using mean only | 1.0606506 |
| Movie Effect Model | 0.9437046 |
| Movie and User Effect Model | 0.8655329 |
| Regularized Movie and User Effect Model | 0.8649857 |
| Reg Movie, User, Year, and Genre Effect Model | 0.8649851 |

## Concluding Remarks

The RMSE table demonstrates how the model improves with varied assumptions. The RMSE of the simplest model, 'Using mean only,' is greater than 1, implying that we may miss the rating by one star (not good!!). After that, adding the 'Movie impact' and 'Movie and user effect' to the model improves it by 5% and 13.5 percent, respectively. Given the model's simplicity, this is a significant improvement. A closer examination of the data indicated that several data points in the features have a significant impact on mistakes. To punish such data points, a regularization technique was applied. The final RMSE is 0.8623, which is a 13.3 percent improvement over the baseline model. This means we may put our faith in our predictions for user-generated movie ratings.

**References**


1. https://github.com/johnfelipe/MovieLens-2
2. https://github.com/cmrad/Updated-MovieLens-Rating-Prediction