Movie Lens Capstone Project

KSN

30/01/2021

Contents

Introduction

A recommendation system is a subclass of information filtering system that seeks to predict the “rating” or “preference” a user would give to an item. The huge amount of data present online has led to the development of recommendation systems. These have become highly popular over the past few decades and are now used in majority of online platforms that we use. The content of such platforms varies from movies, music and books, to social media platforms and e-commerce websites. These systems are often able to gather information about a user’s choices, and use the information to improve that user’s recommendations in the future. For a system to act as a movie recommendation system, it requires basic information like movie cast, movie genre, movie plot, etc. This information helps a system categorize movies in a more efficient manner. User written movie reviews is one such example. It carries substantial amount of movie related information such as location, time period, genre, lead characters and memorable scene descriptions.

Most of the recommendation systems can be classified into either User based collaborative filtering systems or Item based collaborative filtering systems. In User-Based collaborative filtering, a target users’ choices are compared with other users in the database to identify a group of “similar minded” people. Once identified, highly rated content from the group is then recommended to the target user. Item-based collaborative filtering examines each item on the target users’ list of rated items and finds other items in the choice set that seems similar to the item. Such systems would be useful when a group has to take a decision from a large set of possibilities and the decision affects all the members of the group. The system recommends the same movies to users with similar demographic features.

Since each user is different, this approach is considered to be too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. Both research and applications of recommender systems have traditionally focused on providing recommendations to single users; the idea can however easily be extended to recommendations to groups of people.

For this project we will make a movie recommendation system using the 10M MovieLens dataset and use soft computing techniques to develop this system. The idea here is to develop a model which can effectively predict movie recommendations for a given user. In the algorithm, the prevalence can be suppressed using some clever mathematical tricks. The metric used to evaluate recommendation systems is the Root Mean Square Error, or RMSE. RMSE is a measure of accuracy and one of the most used measure of the differences between values predicted by a model and the values observed.

Importing Dataset

In this section we will take the first look at the loaded data frames. The necessary cleaning and transformations in dataset will be performed so that the data becomes more efficient.

```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(ggplot2)) install.packages("ggplot2",
                                        repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
                                        repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                        repos = "http://cran.us.r-project.org")
library(ggplot2)
library(lubridate)
library(caret)
library(tidyverse)
library(gridExtra)
# 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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                            title = as.character(title),
                                            genres = as.character(genres))
# if using R 4.0 or later:
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")
```

Methodology & Analysis

Data Pre-processing

Evaluation of Predicted Ratings using RMSE

The RMSE method is used for algorithm evaluation. Computing the deviation of the prediction from the true value is referred to as the Mean Average Error(MAE) and represents a typical way to evaluate a prediction.

Another popular measure is the Root Mean Square Error (RMSE). In this report, the RMSE value is used to evaluate each model.

```r
# function to calculate the RMSE values
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = T))}
```

Split Data: Train and Test Sets

The movielens dataset is split into two datasets, the edx training dataset consisting of 90% of the data and the temp dataset consisting of the remaining 10% of the data. Movies that only appear in the temp dataset were removed, creating the validation testing dataset. Those removed movies have been added to the edx dataset.

```r
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind = "Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

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

```r
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")
```

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

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

Modifying the Year & Genre

In order to use dependencies between the release year and rating, the release year will be included in a separate column. Likewise for genres, it is necessary to split the multiple genres for each movie into separate rows.

```r
# Modify the year as a column in the edx & validation datasets
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation_CM <- validation
validation <- validation %>% select(-rating)
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 = "\\|")
```

Procedure

Data Interpretation & Visualization

The raw datasets are directly pulled out of the MovieLens website. From the temporary file, the data was pulled in and coerced into two data frames, the ratings data frame, with columns userId, movieId, rating, and timestamp, and the movies data frame, with columns movieId, title, and genres. The two data frames were merged together by movieId, creating a new movielens data frame with six columns, userId, movieId, rating, timestamp, title, and genres.

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

```
## [1] 7
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##
##     title              genres               year
##  Length:9000055     Length:9000055     Min.   : NA
##  Class :character   Class :character   1st Qu.: NA
##  Mode  :character   Mode  :character   Median : NA
##                                        Mean   :NaN
##                                        3rd Qu.: NA
##                                        Max.   : NA
##                                        NA's   :9000055
```

The code above confirms that there are no missing values.

EDX Quiz

No. of zeros were given as ratings in the edx dataset

```
edx %>% filter(rating == 0) %>% tally()
```

```
##   n
## 1 0
```

No. of threes given as ratings in the edx dataset

```
edx %>% filter(rating == 3) %>% tally()
```

```
##         n
## 1 2121240
```

Five most given ratings in descending order

```
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>%
  arrange(desc(count))
```

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

Shows the total number of unique movies in the edx dataset.

```r
n_distinct(edx$movieId)
```

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

Shows the total number of unique users in the edx dataset.

```r
n_distinct(edx$userId)
```

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

Due to the large size of the dataset, data wrangling (for example, the lm model) was not possible because of memory allocation. To solve this problem we computed the least square estimates manually. As the dataset is very sparse, we have included regularization in the model.

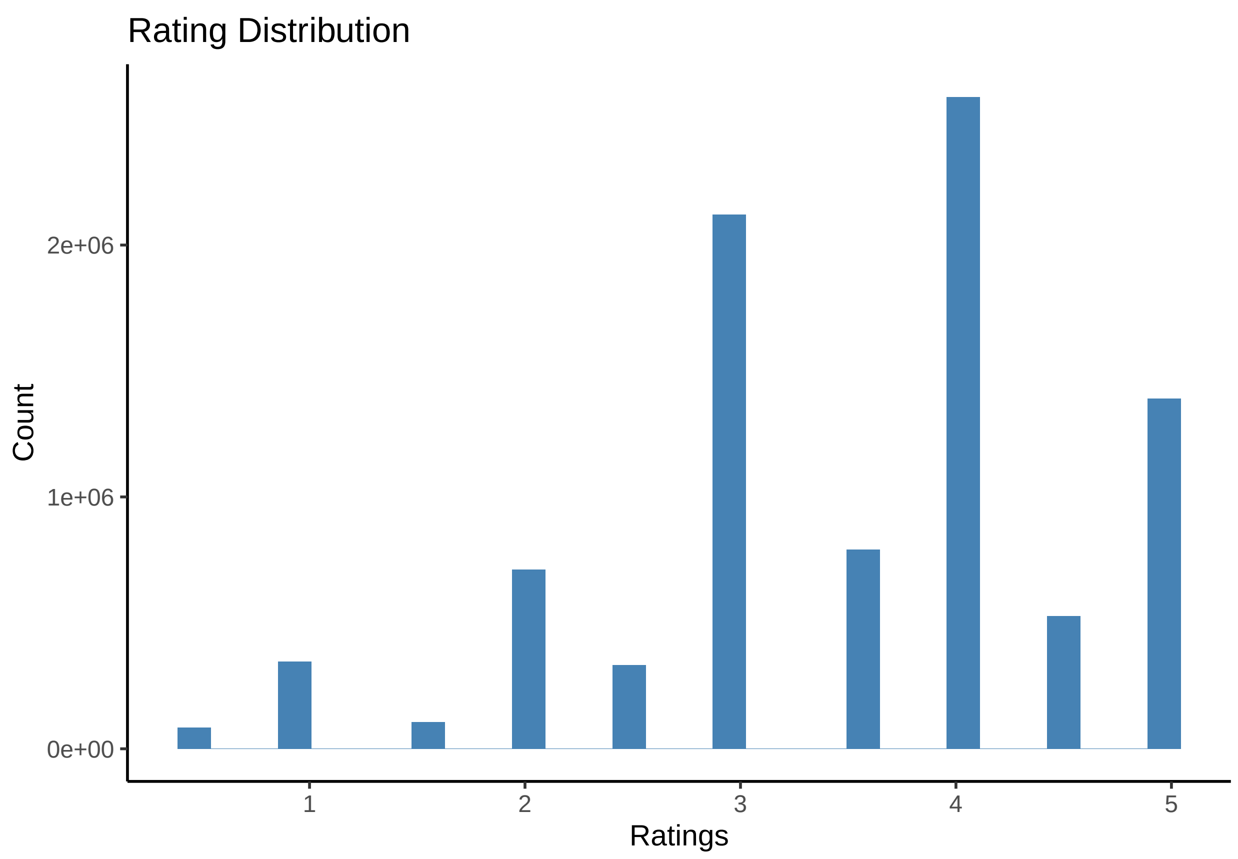
In total, 69878 unique users provided ratings and 10677 unique movies were rated. Considering all the possible combinations between users and movies, we will have around 750 million combinations. Our test set has a little over 9 million rows, which implies that not every user has rated every movie. This number of ratings is only 6% of all possible combinations, which designates a sparse matrix.

Rating Distribution

The level of expectations of the users towards the recommender engine is expected to be medium. Indeed the use of the recommender is not mandatory. This means that users can still use the system if the recommender does not work or performs poorly. However we do expect a shift of the users' expectations as they progressively use the system.

```r
edx %>% ggplot(aes(rating)) +
  geom_histogram(fill = "steelblue") + labs(title = "Rating Distribution",
       x = "Ratings", y = "Count", fill = element_blank()) + theme_classic()
```

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



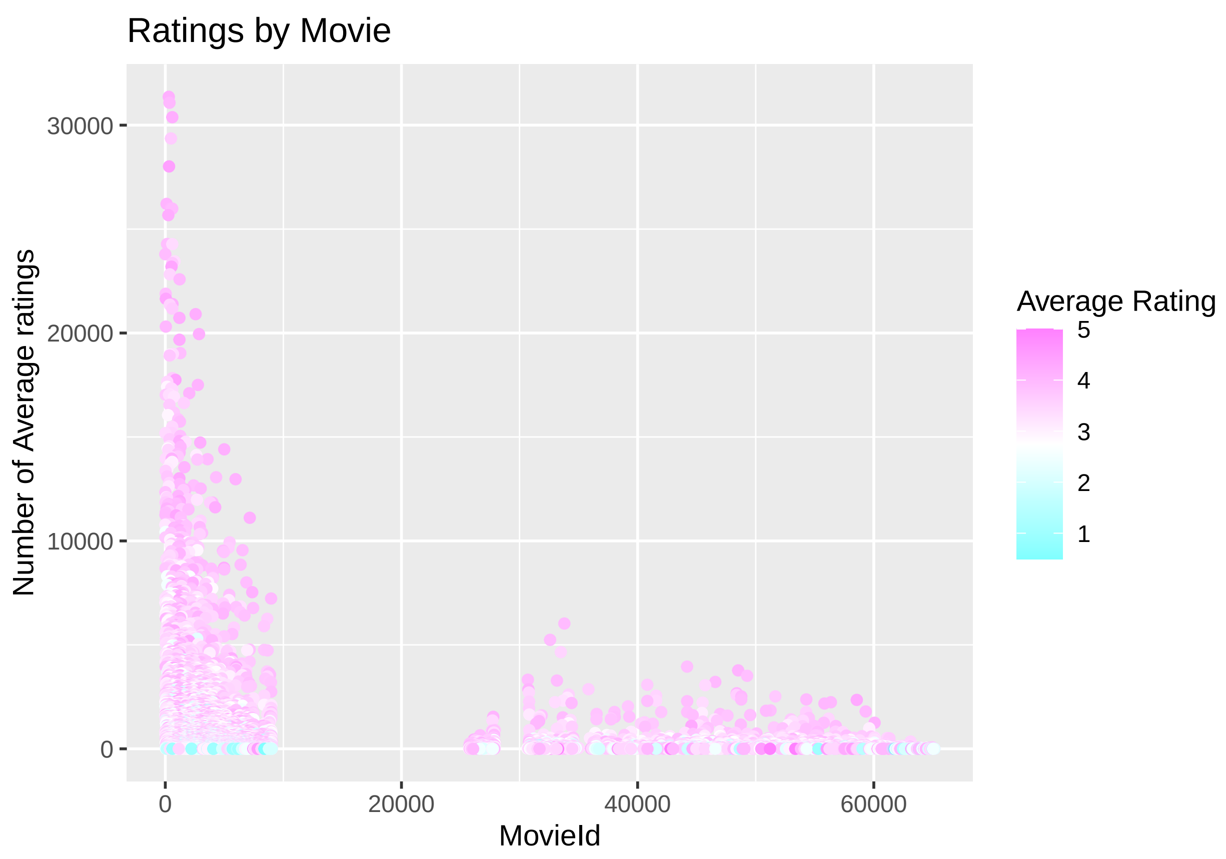
Ratings by Movie

Reviewed Movies often have higher average ratings and greater variation in average ratings than movies which have lesser reviews.

```
edx_movies <- edx %>% group_by(movieId) %>%
  summarize(num_ratings = n(), avg_rating = mean(rating)) %>%
  arrange(desc(num_ratings))
head(edx_movies)
```

```
## # A tibble: 6 x 3
##   movieId num_ratings avg_rating
##     <dbl>       <int>      <dbl>
## 1     296       31362       4.15
## 2     356       31079       4.01
## 3     593       30382       4.20
## 4     480       29360       3.66
## 5     318       28015       4.46
## 6     110       26212       4.08
```

```
edx_movies %>% ggplot(aes(movieId, num_ratings, color = avg_rating)) +
  geom_point() + scale_color_gradientn(colours = cm.colors(5)) +
  labs(x = "MovieId", y = "Number of Average ratings",
       title = "Ratings by Movie", color = "Average Rating")
```

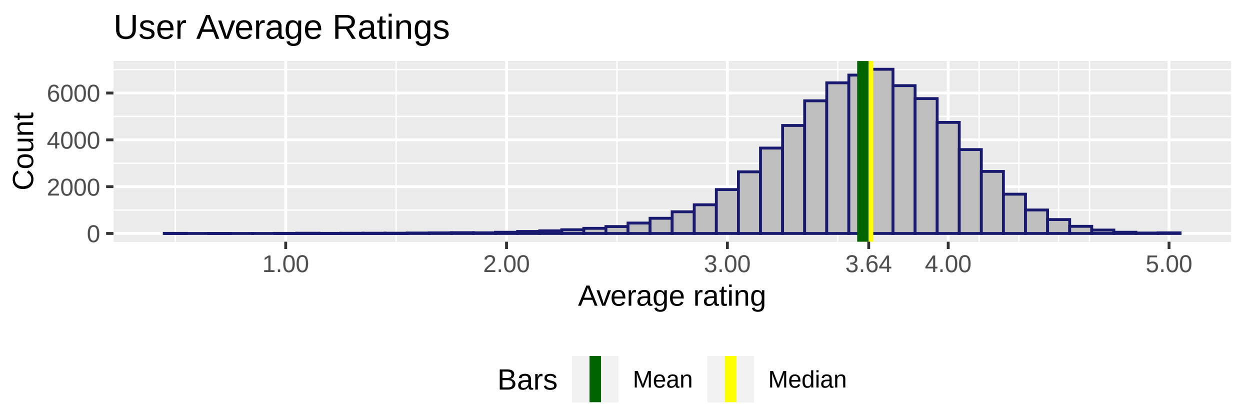


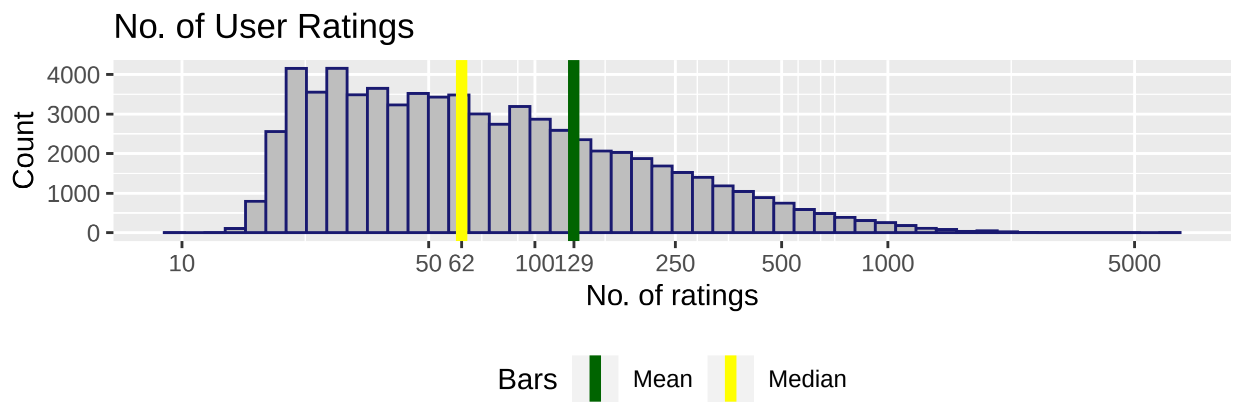
Average Rating & User Rating Distribution

```
cols<- c(Mean = "darkgreen", Median = "yellow")
stats_users <- edx %>% group_by(userId) %>% summarize(countRating=n(),
                      meanRating=mean(rating), medianRating=median(rating))

plot1 <- ggplot(stats_users,aes(x = meanRating)) +
  geom_histogram(binwidth=0.1, colour = "midnightblue", fill = "gray") +
  geom_vline(aes(xintercept=median(`meanRating`), color = "Median"), size=2) +
  geom_vline(aes(xintercept=mean(`meanRating`), color = "Mean"), size=2) +
  scale_colour_manual(name="Bars",values=cols) +
  labs(title = "User Average Ratings",
       x = "Average rating", y = "Count") +
  scale_x_continuous(breaks = c(1:5, round(median(stats_users$meanRating), 2)))  +
  theme(legend.position='bottom')
# plot : User number of ratings
plot2 <- stats_users %>%  ggplot(aes(x = countRating)) +
  geom_histogram(bins=50, colour = "midnightblue", fill = "gray") +
  geom_vline(aes(xintercept = median(`countRating`), color = "Median"), size = 2) +
  geom_vline(aes(xintercept = mean(`countRating`), color = "Mean"), size = 2) +
  scale_colour_manual(name = "Bars",values=cols) +
  labs(title = "No. of User Ratings", x = "No. of ratings", y = "Count") +
  scale_x_log10(breaks = c(10,50,100,250, 500, 1000,5000,
  round(median(stats_users$countRating), 0), round(mean(stats_users$countRating), 0))) +
  theme(legend.position='bottom')
```

```r
# display plots side by side
grid.arrange(plot1, plot2, ncol = 1)
```

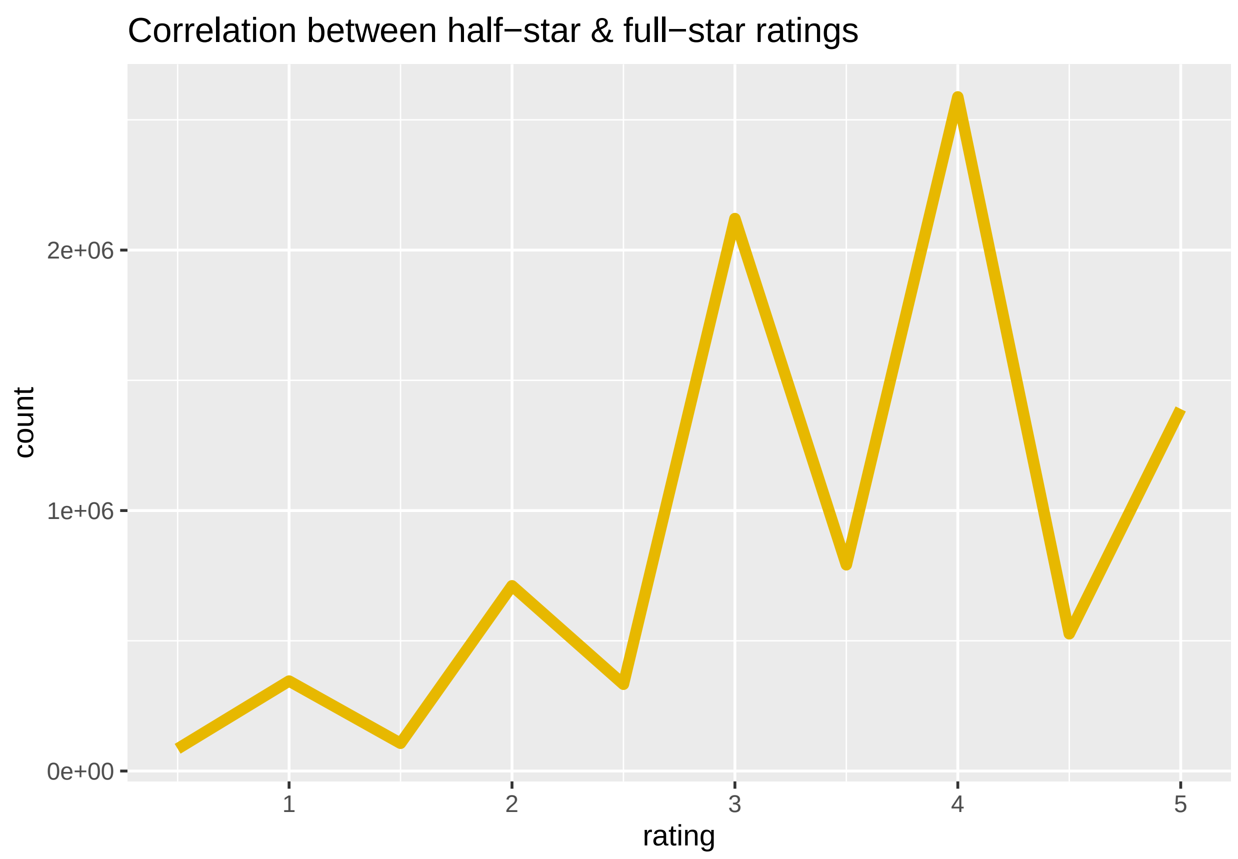




Half star ratings: Less common

Users prefer to give a whole integer rating rather than a decimal rating. Movies are rated higher rather than lower as shown by the ratings distribution below. 4 is the most common rating and 0.5 is the least common rating. Users can rate movies rather higher than lower as demonstrated by the distribution of ratings below.

```r
edx %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count, size = I(2))) +
  geom_line(color = "#E7B800") +
  ggtitle("Correlation between half-star & full-star ratings")
```



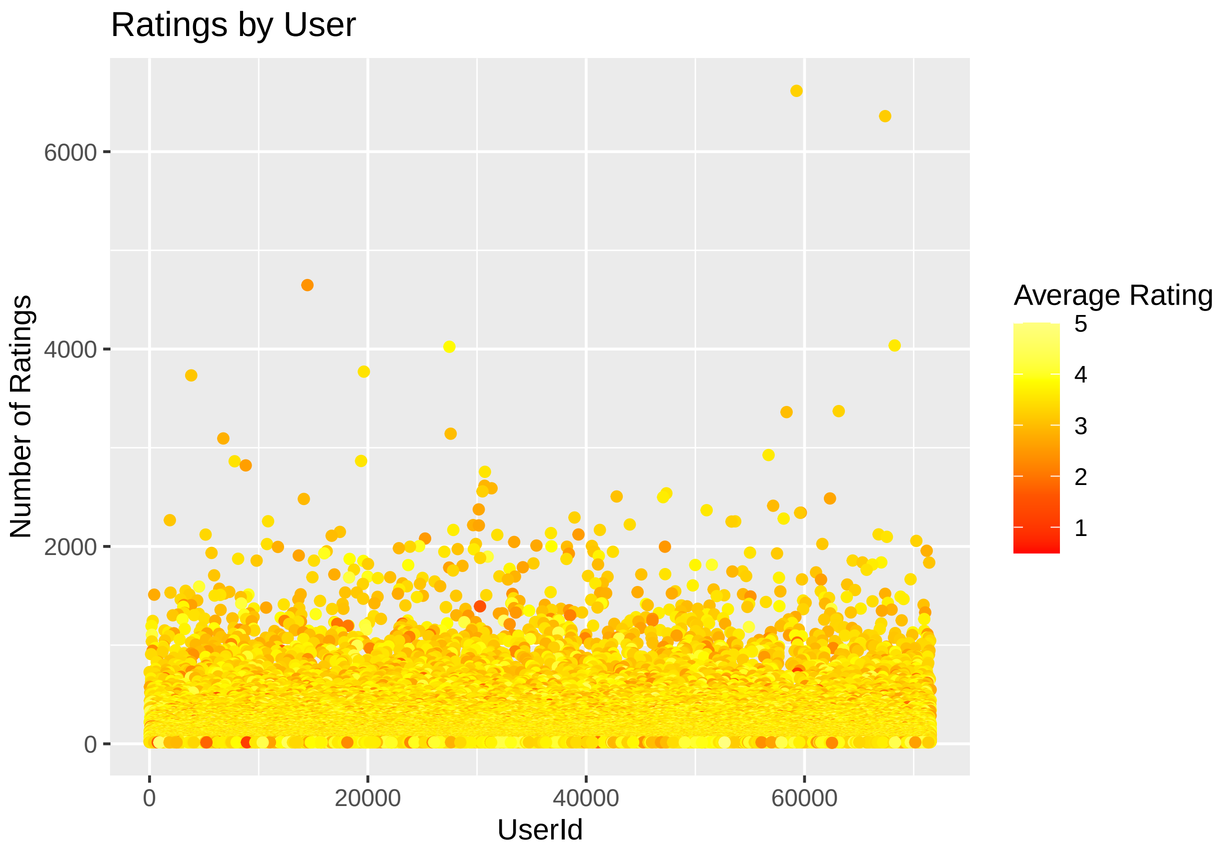
Ratings by User

Users give an average rating near the overall average and there is greater variation in mean ratings for users who given a few ratings, than users who have rated many movies.

```
# average rating
avg_rating <- mean(edx$rating)
# median rating
med_rating <- median(edx$rating)
edx_users <- edx %>% group_by(userId) %>%
  summarize(num_ratings = n(), avg_rating = mean(rating)) %>%
  arrange(desc(num_ratings))
head(edx_users)
```

```
## # A tibble: 6 x 3
##   userId num_ratings avg_rating
##    <int>       <int>      <dbl>
## 1  59269        6616       3.26
## 2  67385        6360       3.20
## 3  14463        4648       2.40
## 4  68259        4036       3.58
## 5  27468        4023       3.83
## 6  19635        3771       3.50
```

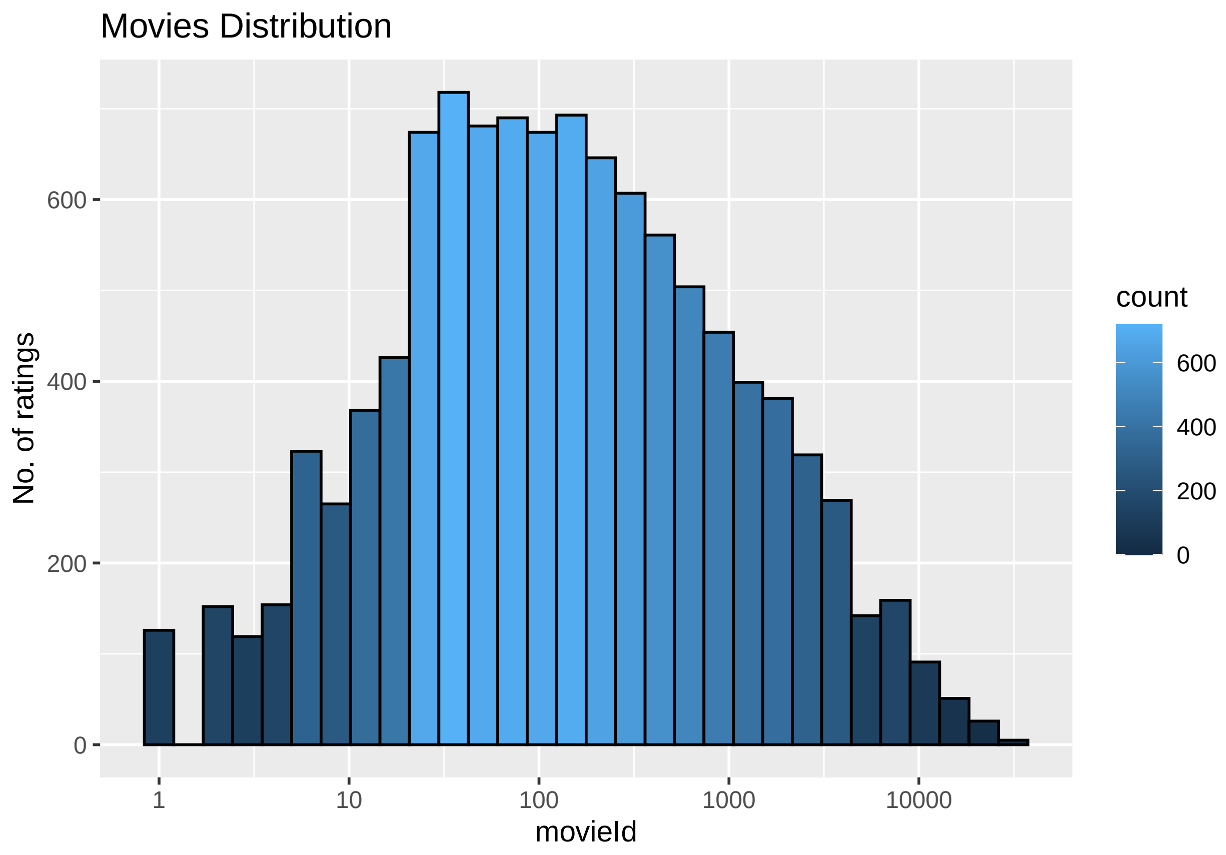
```
edx_users %>% ggplot(aes(userId, num_ratings, color = avg_rating)) +
  geom_point() + scale_color_gradientn(colours = heat.colors(5)) +
  labs(x = "UserId", y = "Number of Ratings",
        title = "Ratings by User", color = "Average Rating")
```



Certain movies are have higher number of user ratings

The histogram below shows that the majority of movies have been reviewed between 50 and 1000 times. A challenge to the ratings prediction is reflected by an interesting finding:around 125 movies have been rated only once. Thus regularization and a penalty term will be applied to the models in this report.

```
edx %>% count(movieId) %>% ggplot(aes(n, fill = ..count..)) +
  geom_histogram(bins = 30, color = "black") + scale_x_log10() +
  ggtitle("Movies Distribution") + labs(x = "movieId", y = "No. of ratings")
```



Model Building

The quality of the model will be assessed by the RMSE (the lower the better). Creating a prediction system that solely utilizes the sample mean represents the initial simplest model. This implies that every prediction is the sample average. The resulting RMSE using this approach is quite high. Due to the size of the dataset, modeling the data using a function like lm is avoided. To solve this problem we computed the least square estimates manually. First, we started with the most simple model to have a baseline: predict the same rating regardless of the user, movie or genre. In this model and all the others tested we have limited the predicted ratings to a minimum value of 0.5 and a maximum of 5.

Sample estimate- mean

The initial step is to compute the datasets' mean rating.

```
rmse_results <- data_frame()
```

```
## 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.
```

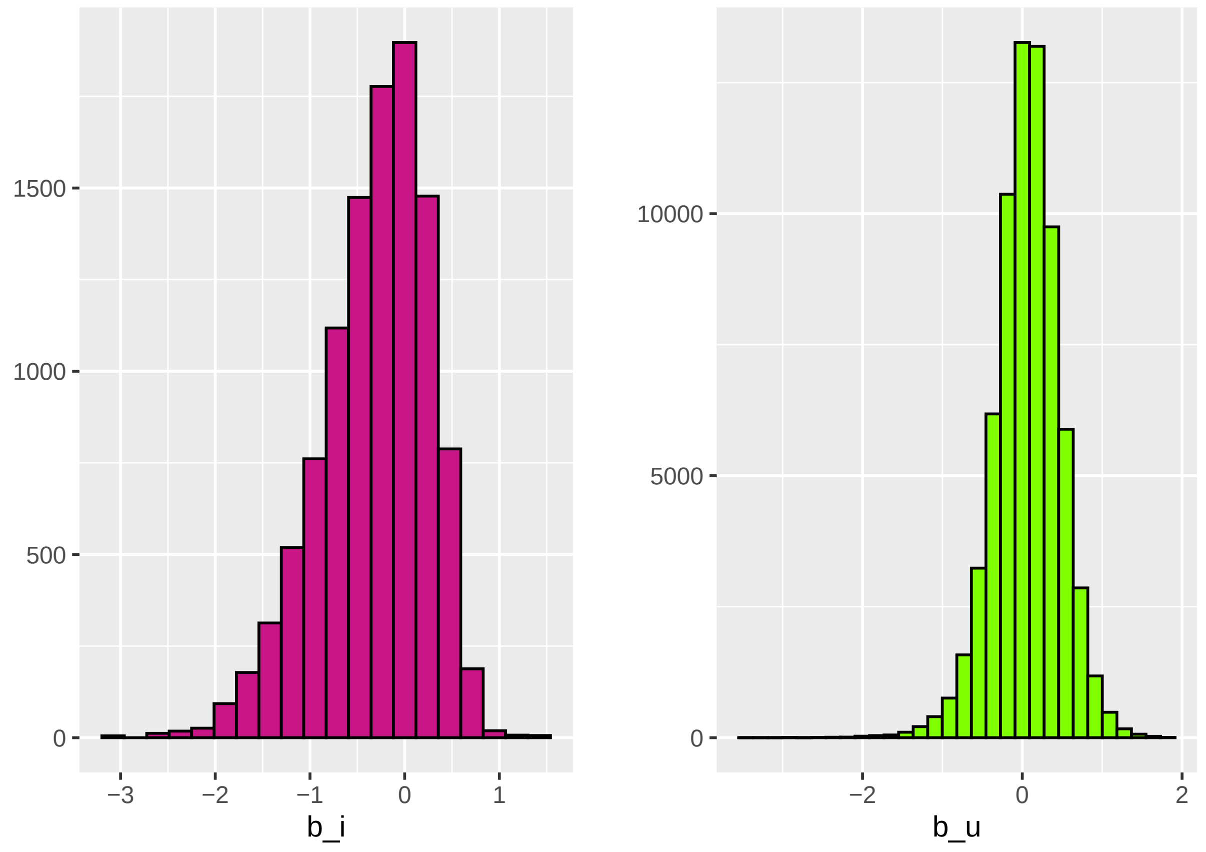
```
mu <- mean(edx$rating)
mu
```

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

Penalty Term-Movie & User Effect

Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. The histogram is left skewed, implying that more movies have negative effects. Similarly users can also affect the ratings either positively or negatively.

```r
movie_avgs_norm <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
plot11 <- movie_avgs_norm %>% qplot(b_i, geom ="histogram", bins = 20, data = .,
                         color = I("black"), fill = I("#C71585"))
user_avgs_norm <- edx %>% left_join(movie_avgs_norm, by = 'movieId') %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu - b_i))
plot21 <- user_avgs_norm %>% qplot(b_u, geom ="histogram", bins = 30, data = .,
                        color = I("black"), fill = I("#7FFF00"))
grid.arrange(plot11, plot21, ncol = 2)
```



Naive Model

The simplest model that someone can build, is a Naive Model that predicts the average of around 3.512. The RMSE on the validation dataset is 1.061. It is very far for the target RMSE (below 0.87) and that indicates poor performance for the model.

```r
naive_rmse <- RMSE(validation_CM$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

```r
rmse_results <- data_frame(method = "Using average only", RMSE = naive_rmse)
rmse_results
```

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

Movie Effect Model

The first Non-Naive Model takes into account the content. In this case the movies that are rated higher or lower with respect to each other.

```r
# 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 average only	1.0612018
Movie Effect Model	0.9439087

```r
rmse_results
```

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

An improvement in the RMSE is achieved by adding the movie effect. Besides the movie effect, we presume that some users rate movies higher than others, so the next model considers both the movie and the user effect. We estimate the user effect as the average of the ratings per user.

Movie and User Effect Model

The second Non-Naive Model consider that the users have different tastes and rate differently.

```r
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)
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 average only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488

```r
rmse_results
```

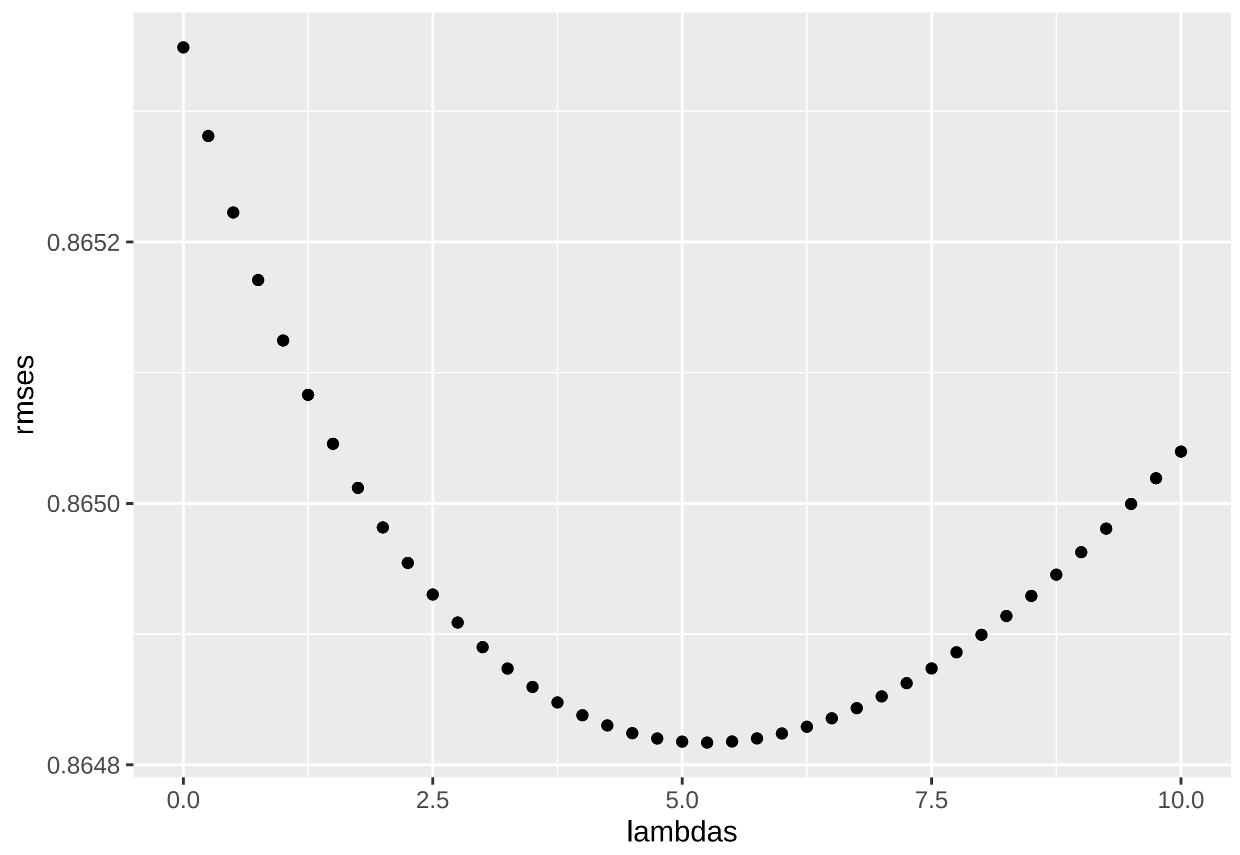
```
## # A tibble: 3 x 2
##   method                       RMSE
##   <chr>                       <dbl>
## 1 Using average only          1.06
## 2 Movie Effect Model          0.944
## 3 Movie and User Effect Model 0.865
```

Regularized Movie and User Effect Model

This model implements the concept of regularization to account for the effect of low ratings' numbers for movies and users. The previous sections demonstrated that few movies were rated only once and that some users only rated few movies.Hence this can strongly influence the prediction. Regularization is a method used to reduce the effect of overfitting. The regularization method allows us to add a penalty lambda to penalize movies with large estimates from a small sample size.

```r
lambdas <- seq(0, 10, 0.25)
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))
})

qplot(lambdas, rmses)
```



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

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

```
# 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())
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 average only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488

method	RMSE
Regularized Movie and User Effect Model	0.8648170

```
rmse_results
```

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

Regularized With All Effects Model

The approach utilized in the above model is implemented below with the added genres and release year effects. The new model has incremented the RMSE. Our final consideration is that the rating estimate for a movie rated many times is more likely to be more accurate than the estimate of a movie rated only a handful of times. Regularization is what allows us to penalize those estimates constructed using small sample sizes. When the sample size is very large, the estimate is more stable and vice versa.

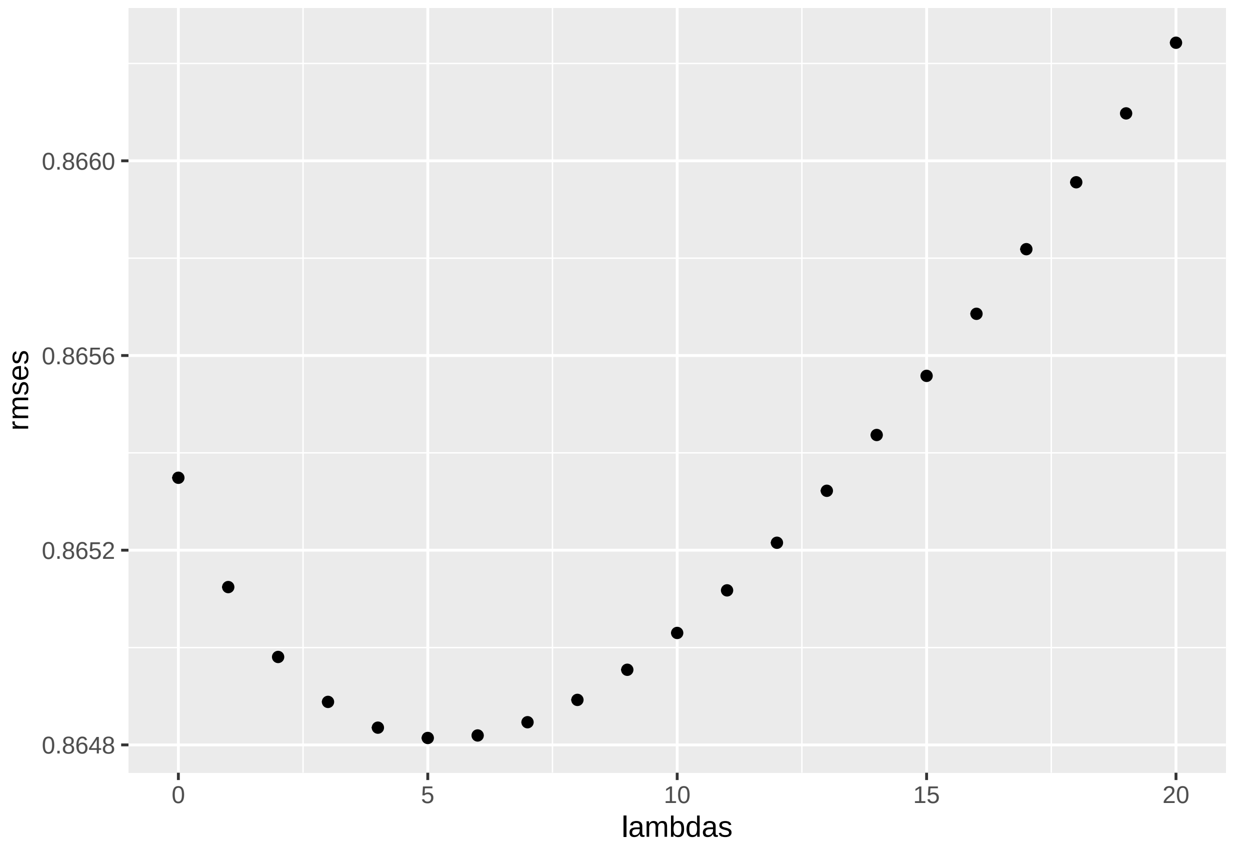
```
# b_y and b_g represent the year & genre effects, respectively
lambdas <- seq(0, 20, 1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- split_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- split_edx %>% left_join(b_i, by="movieId") %>%
    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))
})
```

```r
# Compute new predictions using the optimal lambda
qplot(lambdas, rmses)
```



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

```
## [1] 5
```

```r
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 = "Regularized Movie, User, Year, and Genre Effect Model",
              RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Using average only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170
Regularized Movie, User, Year, and Genre Effect Model	0.8648142

Results

RMSE overview

The RMSE values for the used models are shown below:

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

method	RMSE
Using average only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170
Regularized Movie, User, Year, and Genre Effect Model	0.8648142

Rating Prediction using Model 4

Since, model 4 yielded the best RMSE result, we will consider it as the final prediction model.

```
lambda_3 <- 14
# Redo model 4 analysis
movie_reg_avgs_2 <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_3), n_i = n())
user_reg_avgs_2 <- edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda_3), n_u = n())
year_reg_avgs <- 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_3), n_y = n())
```

```r
genre_reg_avgs <- 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_3), 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) %>%
  group_by(userId,movieId) %>% summarize(pred_2 = mean(pred))
```

```
## `summarise()` has grouped output by 'userId'. You can override using the `.groups` argument.
```

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

method	RMSE
Using average only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170
Regularized Movie, User, Year, and Genre Effect Model	0.8648142

To predict movie ratings we build models that considered the effects of movies, users, genres and interactions of these. The best model considered all, achieving an RMSE of 0.8648. The movie effect decreased the RMSE the most, suggesting that the movie in itself is of utmost significance to determine the rating.

Conclusion

Recommendation Systems are the most popular type of machine learning applications that are used in all sectors. They are an improvement over the traditional classification algorithms as they can take many classes of input and provide similarity ranking based algorithms to provide the user with accurate results. These recommendation systems have evolved over time and have incorporated many advanced machine learning techniques to provide the users with the content that they want. With movie reviews data, movie topics capture the essential movie aspects such as popular genre and mood. The regularized model herein includes the effect of movie, user, genre and year and is characterized by the lowest RMSE value (0.8648), and hence the optimal model to use for the present project.

Future Scope

Recommendation systems has been under research for over a decade. Although recommendation systems have noticed extraordinary improvements starting from very primitive content based and collaborative filtering methods, a lot of research is going on to improve the accuracy of the output.

In this project, exploring different machine learning models (e.g., Item Based Collaborative Filtering) could improve the results further but since this dataset is massive, it would be left for future work. Our approach can be further extended to other domains to recommend songs, books, news, tourism and e-commerce sites, etc. The next technology would be building Recommendation systems using deep learning.

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.3
## year           2020
## month          10
## day            10
## svn rev        79318
## language       R
## version.string R version 4.0.3 (2020-10-10)
## nickname       Bunny-Wunnies Freak Out
```