Movielens Rating Prediction Project

Pradeep Kumar

16/06/2020

Contents

1 Executive Summary

The main goal of this project is to predict movie ratings using the **MovieLens(10M)** dataset, which contains the ratings of several movies given by various users. In this project, we start with importing, followed by cleaning and preparing the data for analysis. Later, we explore the dataset to find any valuable patterns. Next, we test models to try and predict the ratings given by a specific user to a specific movie. The train set (**`edx`**) is divided into a training set (**`training`**) and a testing set (`testing`). The `training` set is used to build the model and the `testing` set is used to perform intermediate evaluations of the model. The `validation` set is not used to train the algorithm. The `validation` set is used only for the final RMSE evaluation.

The metrics used to evaluate the models is **Root Mean Square Error(RMSE)** as per project guidelines. We have obtained rmse of approximately 0.864 for the best model.

2 Introduction

Recommendation systems are ubiquitous nowadays. From recommending restaurants to diagnoses, used in startups to google, recommender systems seem to be virtually everywhere. These systems often work using user-generated data. Large companies like Amazon that sell many products to many customers and permit these customers to rate their products can collect massive datasets. These datasets power their algorithms which use online learning to predict what rating a particular user will give to a specific item. Products are then suggested to the user based on the predicted ratings. **Recommender systems** are a useful alternative to search algorithms since they help users discover items they might not have found otherwise. Of note, recommender systems are often implemented using search engines indexing non-traditional data.

The vast success and implementation of recommender systems comes from the fact that these systems can be applied to movies, news, shops, restaurants, E-Commerce sites and so on.

For this project, we will create a movie recommendation system using the 10M version of MovieLens dataset, collected by **GroupLens Research.**

2.1 Aim of the project

The goal of this project is to train a machine learning algorithm that predicts user ratings (on a scale of 0.5 to 5 stars) using the MovieLens dataset split into training and validation sets to train on and predict movie ratings the validation set.

The measure used to evaluate the algorithm's performance is the Root Mean Square Error or RMSE. RMSE is one of the most used measures of the differences between values predicted by a model and the observed values. RMSE is a measure of correctness; to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; Hence, more substantial errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers.

The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

2.2 Dataset

The dataset used is the MovieLens dataset of 10,000,054 ratings applied to 10,681 movies by 71,567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users had rated at least 20 movies.

The Script to download and create the training(edx) and validation sets: - MovieLens 10M dataset - MovieLens 10M dataset - zip file

```
################################
# Create edx set, validation set
################################

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

# Note: this process could take a couple of minutes
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

We split the dataset into train(edx) and validation sets. Model training is carried out on the “edx” subset only, “validation” subset will be used only to test the trained model.

```
# 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)` instead
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)

# Delete the unnecessary dataframes created to free up memory
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

3 Exploratory Data Analysis

In this section we explore the data in `edx`, try to find patterns and insights to inform further steps of the data science cycle.

3.1 Basic Information

First, To get familiar with the dataset, we look at the head of the dataset. The subset contains the six variables: `userID`, `movieID`, `rating`, `timestamp`, `title`, and `genres`. We also note that the data is in tidy format–Each row represents a single rating of a user for a single movie.

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

A summary of the subset confirms that there are **no missing values**(NAs). Hence, we neend not bother ourselves with removing or imputing missing values.

```
     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
 Length:9000055     Length:9000055
 Class :character   Class :character
 Mode  :character   Mode  :character
```

- **Unique users and movies:** The number of unique users in the edx dataframe is about 70,000 and it contains ratings for about 10,700 unique movies.

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

```
  n_users n_movies
1   69878    10677
```

- **Highest rated Movies:**

```
# Highest Rated movies
edx %>% group_by(title) %>%
        summarize(numberOfRatings = n(), averageRating = mean(rating)) %>%
        arrange(desc(averageRating)) %>%
        top_n(10, wt=averageRating)
```

```
# A tibble: 10 x 3
   title                                          numberOfRatings averageRating
   <chr>                                                    <int>         <dbl>
 1 Blue Light, The (Das Blaue Licht) (1932)                     1          5
 2 Fighting Elegy (Kenka erejii) (1966)                         1          5
 3 Hellhounds on My Trail (1999)                                1          5
 4 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                          2          5
 5 Shadows of Forgotten Ancestors (1964)                        1          5
 6 Sun Alley (Sonnenallee) (1999)                               1          5
 7 Constantine's Sword (2007)                                   2          4.75
 8 Human Condition II, The (Ningen no joken II) (~              4          4.75
 9 Human Condition III, The (Ningen no joken III)~              4          4.75
10 Who's Singin' Over There? (a.k.a. Who Sings Ov~              4          4.75
```

These highest rated movies are very obscure movies. It can be noted that the number of ratings for these movies is meagre, in some cases only a single rating. To find a more fair list of highest rated movies, we need to take into amount the number of ratings. The following shows a list of the highest-rated movies that at least 100 ratings.

```
# Highest Rated movies with atleast 100 ratings
edx %>% group_by(title) %>%
        summarize(numberOfRatings = n(), averageRating = mean(rating)) %>%
        filter(numberOfRatings > 100) %>%
        arrange(desc(averageRating)) %>%
        top_n(10, wt=averageRating)
```

```
# A tibble: 10 x 3
   title                                        numberOfRatings averageRating
   <chr>                                                  <int>         <dbl>
 1 Shawshank Redemption, The (1994)                       28015          4.46
 2 Godfather, The (1972)                                  17747          4.42
 3 Usual Suspects, The (1995)                             21648          4.37
 4 Schindler's List (1993)                                23193          4.36
 5 Casablanca (1942)                                      11232          4.32
 6 Rear Window (1954)                                      7935          4.32
 7 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)           2922          4.32
 8 Third Man, The (1949)                                   2967          4.31
 9 Double Indemnity (1944)                                 2154          4.31
10 Paths of Glory (1957)                                   1571          4.31
```

Each movie is assigned one or more genres, and the genres are encoded into one field genres. The different genres and the number of movies they are assigned to are as folows.

```
# Extract unique genres with seperate_rows and arrange them in descending order
genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(n = n()) %>%
  arrange(desc(n))
```

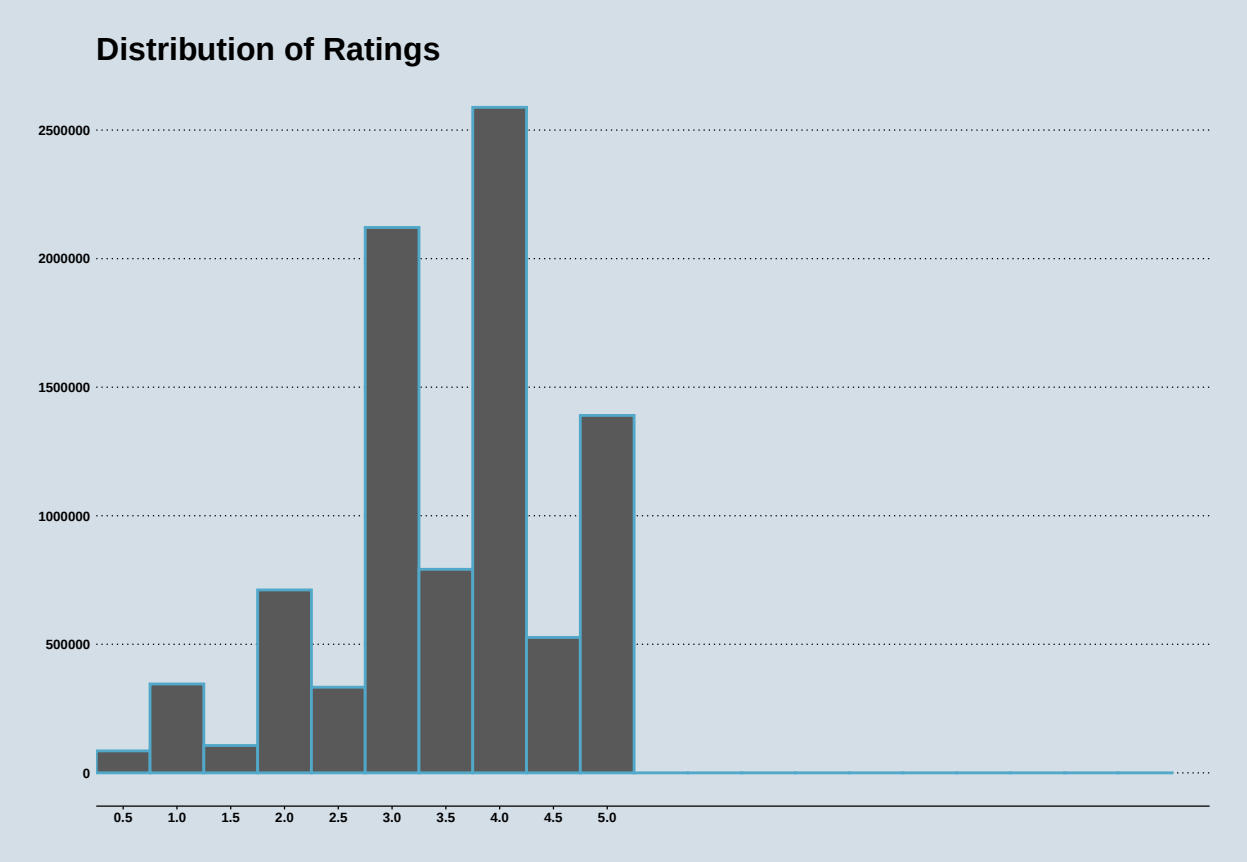
```
genres %>% print.data.frame()
```

```
               genres       n
1               Drama 3910127
2              Comedy 3540930
3              Action 2560545
4            Thriller 2325899
5           Adventure 1908892
6             Romance 1712100
7              Sci-Fi 1341183
8               Crime 1327715
9             Fantasy  925637
10           Children  737994
11             Horror  691485
12            Mystery  568332
13                War  511147
14          Animation  467168
15            Musical  433080
16            Western  189394
17          Film-Noir  118541
18        Documentary   93066
19               IMAX    8181
20 (no genres listed)       7
```

We note that the dataset contains **20** different genres and a pseudo-genre called (no genres listed) indicating that the movie has not been assigned any genres.

3.2 Visuals

We note that there are higher compared to lower ones as seen by the distribution of ratings shown below. This discrepancy may be because people tend to rate movies they love or hate and do not bother to rate the movies they feel are average. Further, We observe that 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half-a-star ratings are less common than whole star ratings.

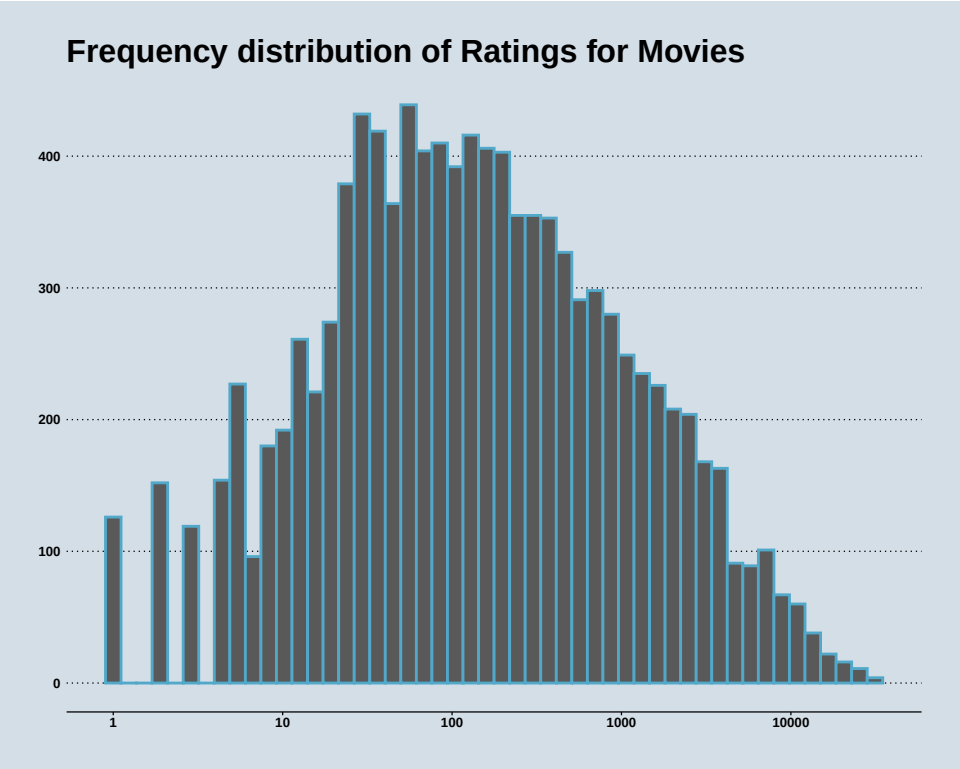


We observe that some movies are rated by more users compared to others, while some have very few ratings, and some have only one rating. One hundred twenty-five movies have ratings from a single user. This fact is important to note as deficient ratings might result in our model not generalizing well, leading to inaccurate estimates for our predictions.

So, we consider introducing **Regularization** later. Regularizations are techniques used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting (the production of an analysis that corresponds too closely or exactly to a particular set of data and may, therefore, fail to fit additional data or predict future observations reliably). A penalty term is added to the cost function. The additional term controls the excessively fluctuating function such that the coefficients do not take extreme values. For this reason, Adding Regularisation and a penalty term to the models is expected to lead to better models as it penalizes features.

```
# Generate log scaled frequency distribution of movies and ratings
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "#51A8C9") +
  scale_x_log10() +
  ggtitle("Frequency distribution of Ratings for Movies") +
  scale_colour_wsj("colors6", "") +
  theme_wsj(base_size = 5, color = "blue",
```

```
        base_family = "sans", title_family = "sans")
```



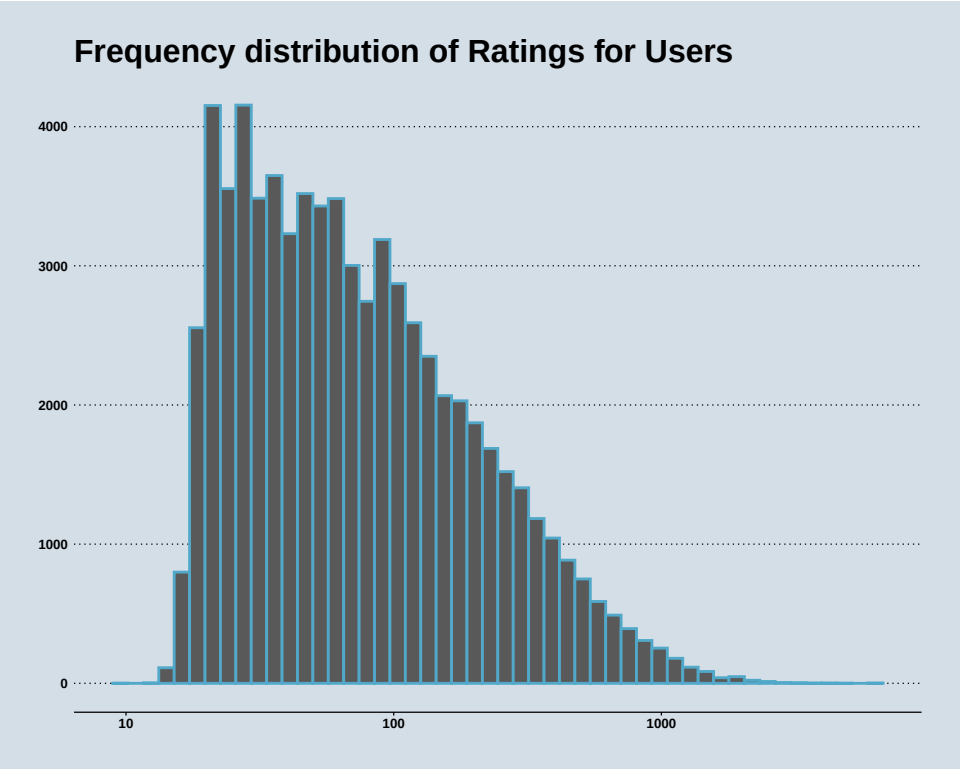
These obscure movies are outliers which have been rated only once by user, Predictions of future ratings for them might prove to be difficult.

```
# Find the movies with only a single user rating (Outliers)
edx %>% group_by(movieId) %>%
  summarize(ratings = n()) %>%
  filter(ratings == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = ratings) %>%
  slice(1:20) %>%
  knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

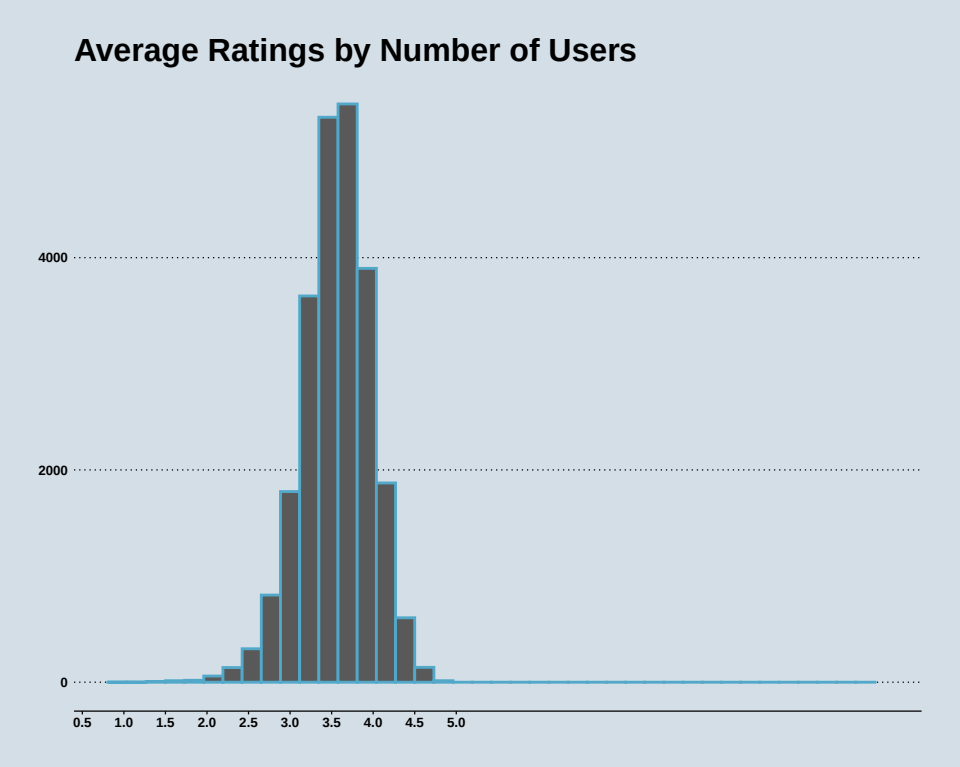
We observe that the majority of users have rated only between 30 and 100 movies. So, a **user penalty term** needs to be included later in our models to account for this.

```
# Generate the log scaled frequency distribution of users and ratings
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "#51A8C9") +
  scale_x_log10() +
  ggtitle("Frequency distribution of Ratings for Users") +
  scale_colour_wsj("colors6", "") +
  theme_wsj(base_size = 5, color = "blue",
            base_family = "sans", title_family = "sans")
```



Also, Users differ vastly in how critical they are with their ratings. Some users tend to give much lower ratings, and some users tend to give higher ratings than average. The graph below only includes users that have rated at least 100 movies.

```
# Generate mean ratings of users who have rated atleast 100 movies
edx %>% group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins = 40, color = "#51A8C9") +
  xlab("Average rating") +
  ylab("Count of users") +
  ggtitle("Average Ratings by Number of Users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_colour_wsj("colors6", "") +
  theme_wsj(base_size = 5, color = "blue",
            base_family = "sans", title_family = "sans")
```


Average Ratings by Number of Users
4000
2000
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

4 Models and Analysis

We write now the loss-function, previously explained, that computes the RMSE:

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

The RMSE is our evaluation metric for the models. The lower the RMSE, the better our model is. The function we use to compute the RMSE for vectors of ratings, and their corresponding predictions is as follows. Where N is the number of user/movie combinations, and the sum of squared errors is calculated over all these combinations.

```
# Root of mean of squared errors
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

4.1 Naïve Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4. This model simply always predicts the average of all the ratings.

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

where $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the average rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings:

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

```
[1] 3.512465
```

If we predict all unknown ratings with μ or mu, we obtain the first naive RMSE:

```
# Rmse for average rating model
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
[1] 1.061202
```

Tabulated as follows:

```
rmse_results <- tibble(Model = "Naïve Average movie rating model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Model	RMSE
Naïve Average movie rating model	1.061202

This give us our baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

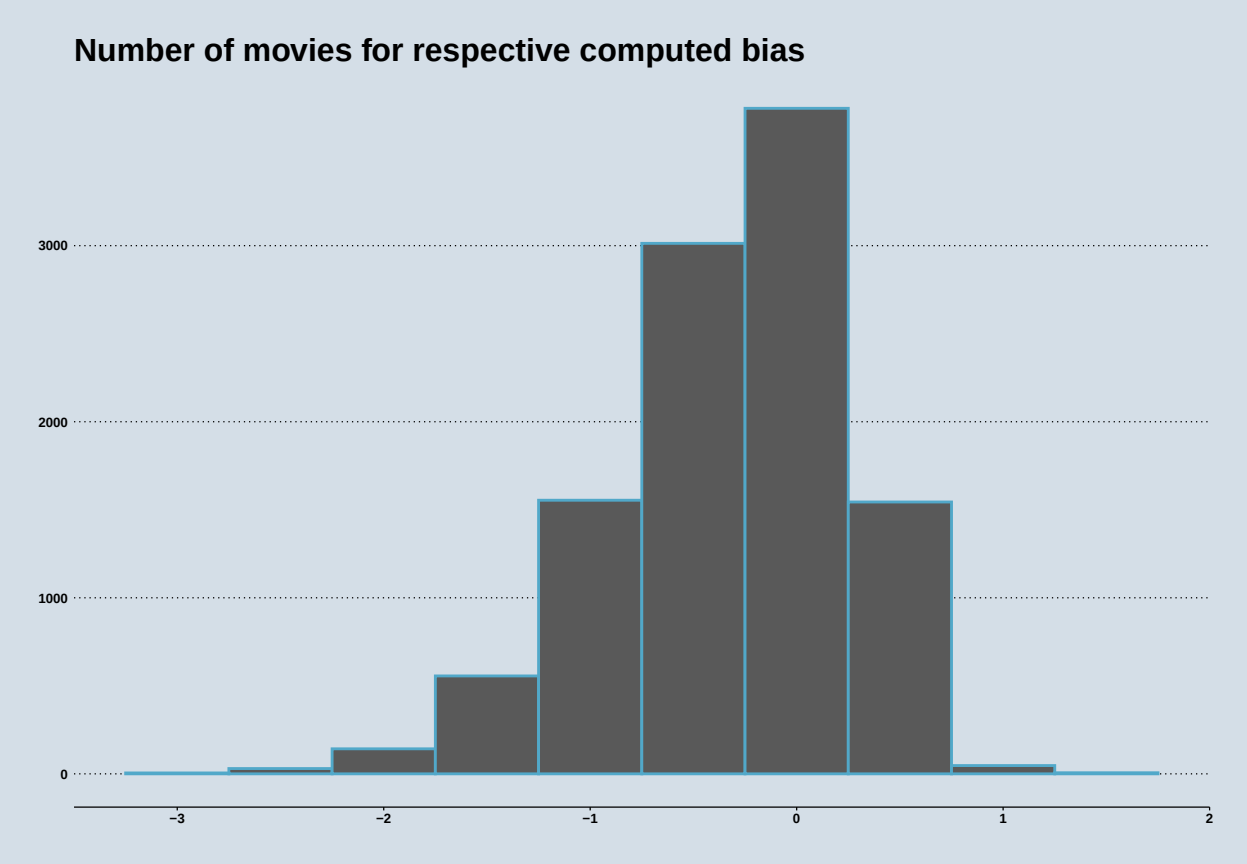
4.2 Movie effect model

To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movie's mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

The distribution is skewed, implying that more movies have negative effects

```
# Compute the bias terms of movie ratings
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# plot distribution
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("#51A8C9"),
                     main = "Number of movies for respective computed bias") +
               scale_colour_wsj("colors6", "") +
               theme_wsj(base_size = 5, color = "blue",
                         base_family = "sans", title_family = "sans")
```



We incorporate the movie effect term to our prediction:

```
# Left join moive_avgs on key movieId
predictions <- mu +  validation %>%
                     left_join(movie_avgs, by='movieId') %>%
                     pull(b_i)

movie_effect_rmse <- RMSE(predictions, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Movie effect model",
                                 RMSE = movie_effect_rmse ))
rmse_results %>% knitr::kable()
```

Model	RMSE
Naïve Average movie rating model	1.0612018
Movie effect model	0.9439087

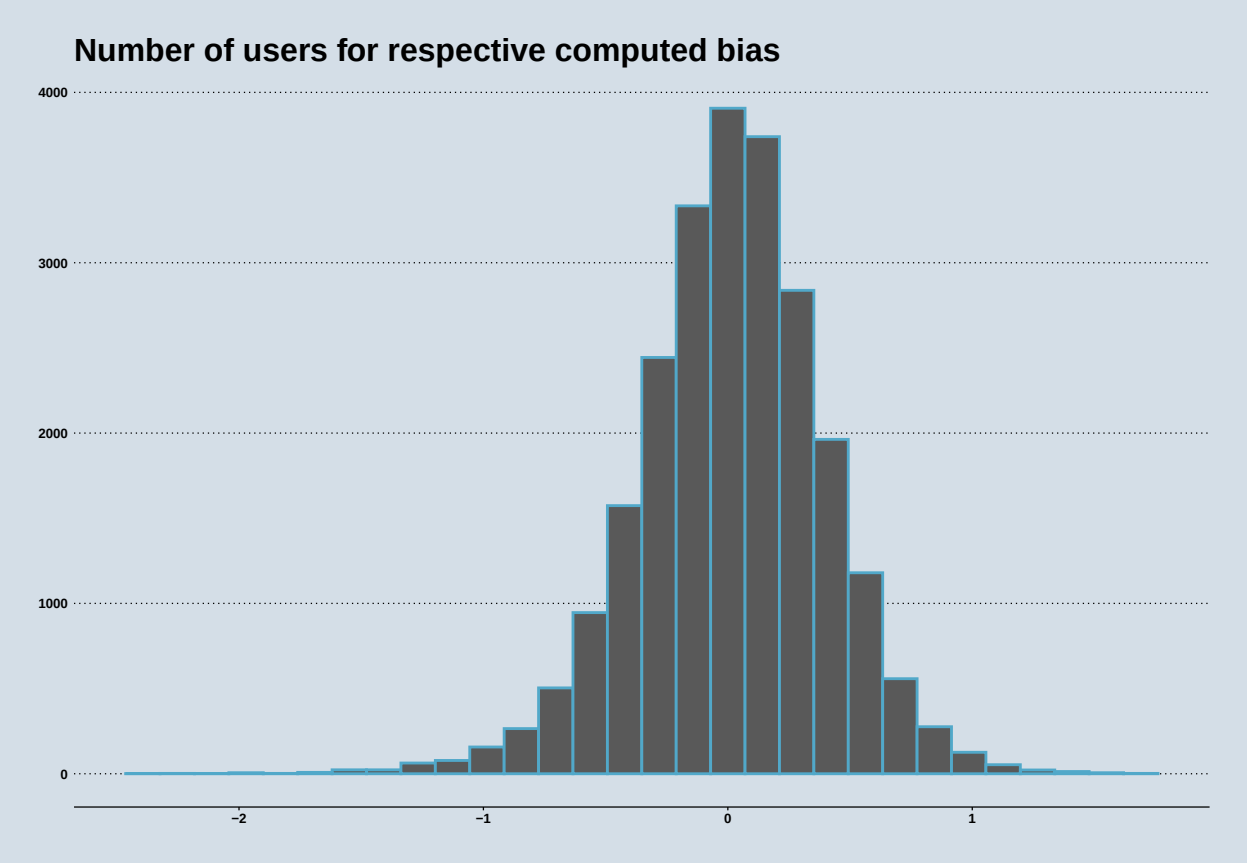
So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse than the average rating of all movies μ, we predict that it will rate lower than μ by b_i, the difference of the individual movie average from the total average.

We can see an improvement, but this can be further improved by consider the individual user rating effect.

4.3 Movie and User effect model

We compute the average rating for user μ, for those that have rated atleast 100 movies. In fact users affect the ratings positively or negatively. So, We incorporate the user effect term to our prediction.

```
# Compute the bias terms of user ratings
user_avgs<- edx %>% left_join(movie_avgs, by='movieId') %>%
                    group_by(userId) %>%
                    filter(n() >= 100) %>%
                    summarize(b_u = mean(rating - mu - b_i))
# plot distribution
user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("#51A8C9"),
                   main = "Number of users for respective computed bias")  +
              scale_colour_wsj("colors6", "") +
              theme_wsj(base_size = 5, color = "blue",
                        base_family = "sans", title_family = "sans")
```



We can see an improvement, but this model does not consider the individual user rating effect. So we can improve our model by incorporating user rating effect:

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

We can see an improvement, but this model does not consider the individual user rating effect. So we can improve our model by incorporating user rating effect: where b_u is a user-specific effect. If a choosy user (negative b_u rates a great movie (positive b_i), the effects counter each other, and we may be able to correctly predict that this user gave this great movie a three rather than a five.

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

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

We can now construct predictors and see that the RMSE improves:

```
# predictions for this model
predictions <- validation%>%
               left_join(movie_avgs, by='movieId') %>%
               left_join(user_avgs, by='userId') %>%
               mutate(pred = mu + b_i + b_u) %>%
               pull(pred)

user_effect_rmse <- RMSE(predictions, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Movie & User effect model",
                                 RMSE = user_effect_rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
Naïve Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie & User effect model	0.8653488

Our rating predictions further reduced the RMSE. However, the purported best and worst movie were rated by a few users, in most cases, just one user. These movies were mostly obscure ones. This fact is because, with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this, we introduce the concept of regularization, that permits to penalize high estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the cost function(RMSE) that we want to minimize. So having many large b_i, makes it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

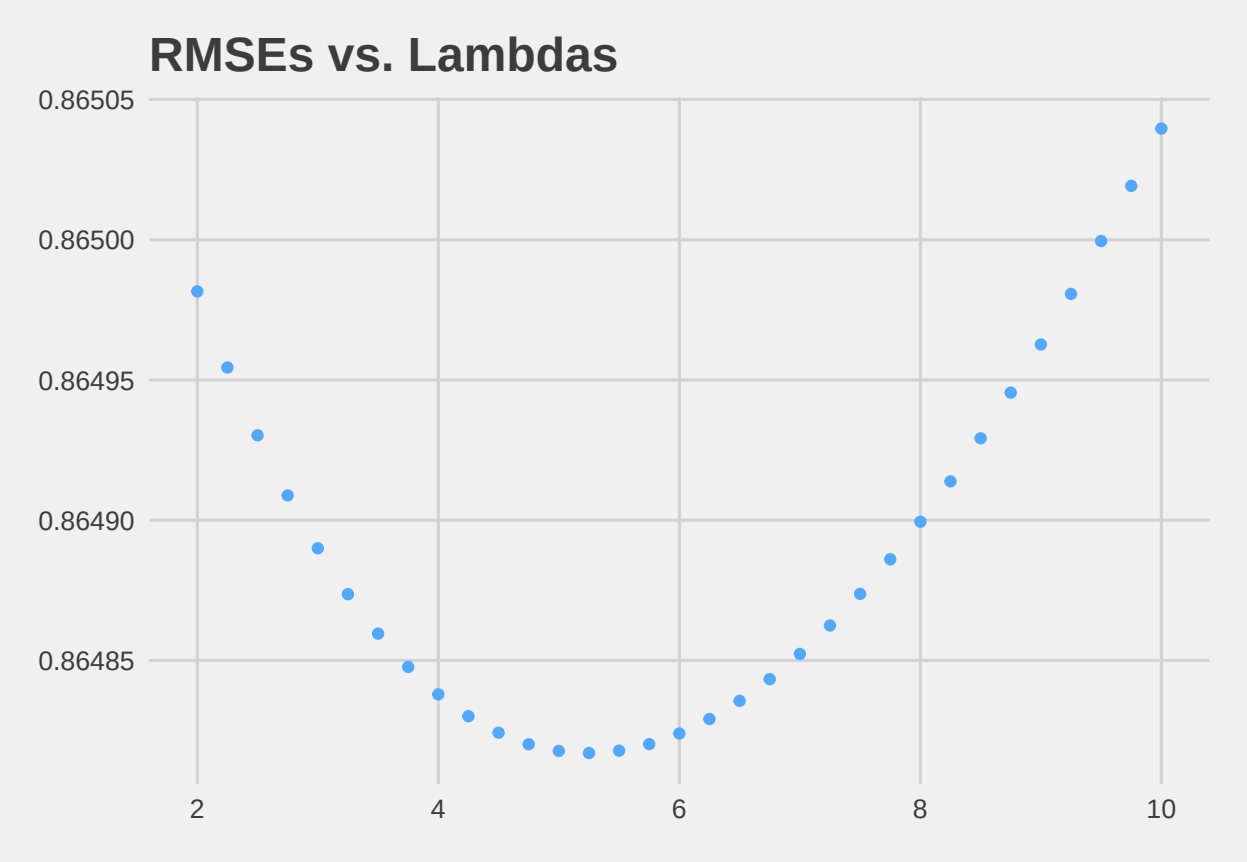
4.4 Regularized Movie and User effect model

So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a tiny number of movies. Hence this can strongly influence the prediction—the use of regularization permits to penalize these aspects. We should find the value of lambda (regularization term) that will minimize the RMSE. This shrinks the b_i and b_u in case of a small number of ratings.

```
# Trying different values for the regularization term
lambdas <- seq(2, 10, 0.25)
rmses <- sapply(lambdas, function(lambda){

  movie_avg <- mean(edx$rating)

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

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

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

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

We plot RMSE vs lambdas to select the optimal lambda:

```
# Plot rmses vs lambdas to select the optimal omega
# Plot rmses vs lambdas to select the optimal omega
qplot(lambdas, rmses,  color = I("#51A8FF"),
      main = "RMSEs vs. Lambdas") +
      theme_fivethirtyeight()
```



For the final model, the optimal lambda is:

```
# get the lamba for minimum value of rmse
lambda <- lambdas[which.min(rmses)]
lambda
```

```
[1] 5.25
```

For the final model, the **optimal lambda(λ) is 5.25**.

The final results are:

```
# Test and save results
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Regularisation & Movie & User effect model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Model	RMSE
Naïve Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie & User effect model	0.8653488
Regularisation & Movie & User effect model	0.8648170

5 Results

The RMSE values of all the represented models: ## The Result

Machine Learning Model	RMSE
Naïve Average movie rating model	1.061
Movie effect model	0.943
Movie & User effect model	0.865
Regularisation & Movie & User effect model	0.864

We find the lowest value of RMSE to be at 0.8648170

6 Conclusion

The final model for this project is:

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

This model works well if the average user doesn't rate a particularly good/popular movie with a large positive b_i, and vice versa.

We have built multiple machine learning algorithms to predict movie ratings using the MovieLens dataset with increasing complexity to lower the RMSE. The regularized model, including the effect of the users, is characterized by the lower RMSE value and is hence the optimal model to use for the present project. With this model, we have achieved our goal of creating an algorithm with RMSE(0.8648170) lower than the allowed RMSE(0.8649). Further, RMSE could be improved by adding other effects (genre, year, age,..). Applying different machine learning models could also improve the results further. The models build on the assumptions that also the movie popularity, the user opinion and genre popularity are constant over time. Further work could be done to investigate if this is true, and if not, the changes over time could be built into the models.