HarvardX Capstone Project - Movielens Recommendation System

Jamuna Bhaskar

December 31 2019

1 Introduction

Movie Recommendation system provides recommendation of specific movies to the users based on the movie rating predicted by the system. This predicted rating is derived using the ratings given by various users for the movies watched in the past. In order to facilitate this system, users are encouraged to record their ratings once they watch movies in movie streaming services like Netflix, Amazon Prime as well as in movie review-aggregation websites such as Rotten tomatoes, Google reviews etc.

The objective of this project is to train a machine learning algorithm to predict the movie ratings using data input. Let us assume we have the following table with users and their rating for certain movies. There are one or more movies that each user has not rated yet and our project would predict the ratings for those movies.

USERID_MOVIEID	M_231	M_266	M_293
U_294		3	3
U_47613	3.5		
U_35893		1	3

To accomplish our objective, a training dataset (**edx**) and a validation dataset (**validation**) would be established. RMSE shall be used to evaluate the accuracy of the prediction.

- The purpose of **edx** dataset is used to develop the algorithm and train it with data patterns.
- The **validation** dataset would be used to test the algorithm and to validate it's predicted rating outcome.
- Residual Mean Square Error (RMSE) is the measure of prediction errors.

As mentioned in Project Overview, the 10M version of the MovieLens dataset made available by GroupLens research lab has been used.

1.1 Residual Mean Square Error (RMSE)

The quality of a machine learning algorithm can be elevated by minimizing the prediction errors. In order to minimize the error, we need a method of measuring the error in predicted outcome.

In this project, RMSE provides us the measure of typical error while predicting the movie rating. Here RMSE is defined as the sum of squared 'difference between predicted rating and actual rating of the respective movies' divided by the number of samples. If RMSE is greater than 1, then the predictions are far more deviated from the actual, meaning the algorithm needs to be tuned to minimize the error.

RMSE =

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

N - the number of user/movie combinations and the sum occurring over all these combinations

$\hat{y}_{u,i}$ – Predicted rating derived by the system on behalf of user u for movie i

$y_{u,i}$ - Actual rating provided by user u for movie i

2 MovieLens Dataset

The following table has 10 records from MovieLens dataset. Each row displays the rating given by a user for a movie. It also has the timestamp when the movie was rated and genre of the movie.

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
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical

The table below shows the number of unique users who have provided ratings and the number of unique movies that were rated. If every user has rated every movie, then the size of dataset would be exceeding the size of 10M version of the MovieLens dataset. Thus we infer that users has not rated every movie watched/available in the database.

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

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

Using the MovieLens dataset, let’s create train dataset (edx) and test dataset (validation) to evaluate the accuracy of our model.

2.1 Train set

```
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list =
FALSE)
edx <- movielens[-test_index,] #train set
temp <- movielens[test_index,] #test set
str(edx)
```

```
## 'data.frame':    9000047 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392
838984474 838983653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)"
"Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Dr
ama|Sci-Fi|Thriller" ...
```

Now the edx dataset contains 9000047 rows of data.

2.2 Test set

The validation set is created with the same structure as edx and contains 999993 rows.

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

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

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

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

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

str(validation)
```

```
## 'data.frame':    999993 obs. of  6 variables:
##  $ userId   : int  1 2 2 3 3 3 4 4 4 5 ...
##  $ movieId  : num  588 1210 1544 151 1288 ...
##  $ rating   : num  5 4 3 4.5 3 3 3 3 5 3 ...
##  $ timestamp: int  838983339 868245644 868245920 1133571026 1133571035 11648856
17 844416656 844417070 844416834 857912840 ...
##  $ title    : chr  "Aladdin (1992)" "Star Wars: Episode VI - Return of the Jedi
(1983)" "Lost World: Jurassic Park, The (Jurassic Park 2) (1997)" "Rob Roy (1995)"
...
##  $ genres   : chr  "Adventure|Animation|Children|Comedy|Musical" "Action|Advent
ure|Sci-Fi" "Action|Adventure|Horror|Sci-Fi|Thriller" "Action|Drama|Romance|War" .
..
```

userId – Unique Id of user

movieId – Unique Id of movie

rating – Rating provided by user between 0 and 5 for respective movie

timestamp – timestamp when the rating was provided

title – Title of the movie

genres – Genre of the movie

2.3 Data Exploration

2.3.1 Study I

Certain movies are rated by more number of users than certain others. For example, commercially successful movies are rated by thousands of users whereas art/independent movies are rated by few users comparatively.

The following table lists the number of times a movie has been rated and we have taken only the top 10 movies that has been rated by larger number of users.

```
edx %>%
  count(movieId) %>%
  arrange(desc(n)) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##    movieId     n
##      <dbl> <int>
##  1     296 31336
##  2     356 31076
##  3     593 30280
##  4     480 29291
##  5     318 27988
##  6     110 26258
##  7     589 26115
##  8     457 26050
##  9     260 25809
## 10     592 24343
```

From the data, we see that movie id '296' has been rated by largest number of users. Let us see the actual rating provided by different users for this particular movie.

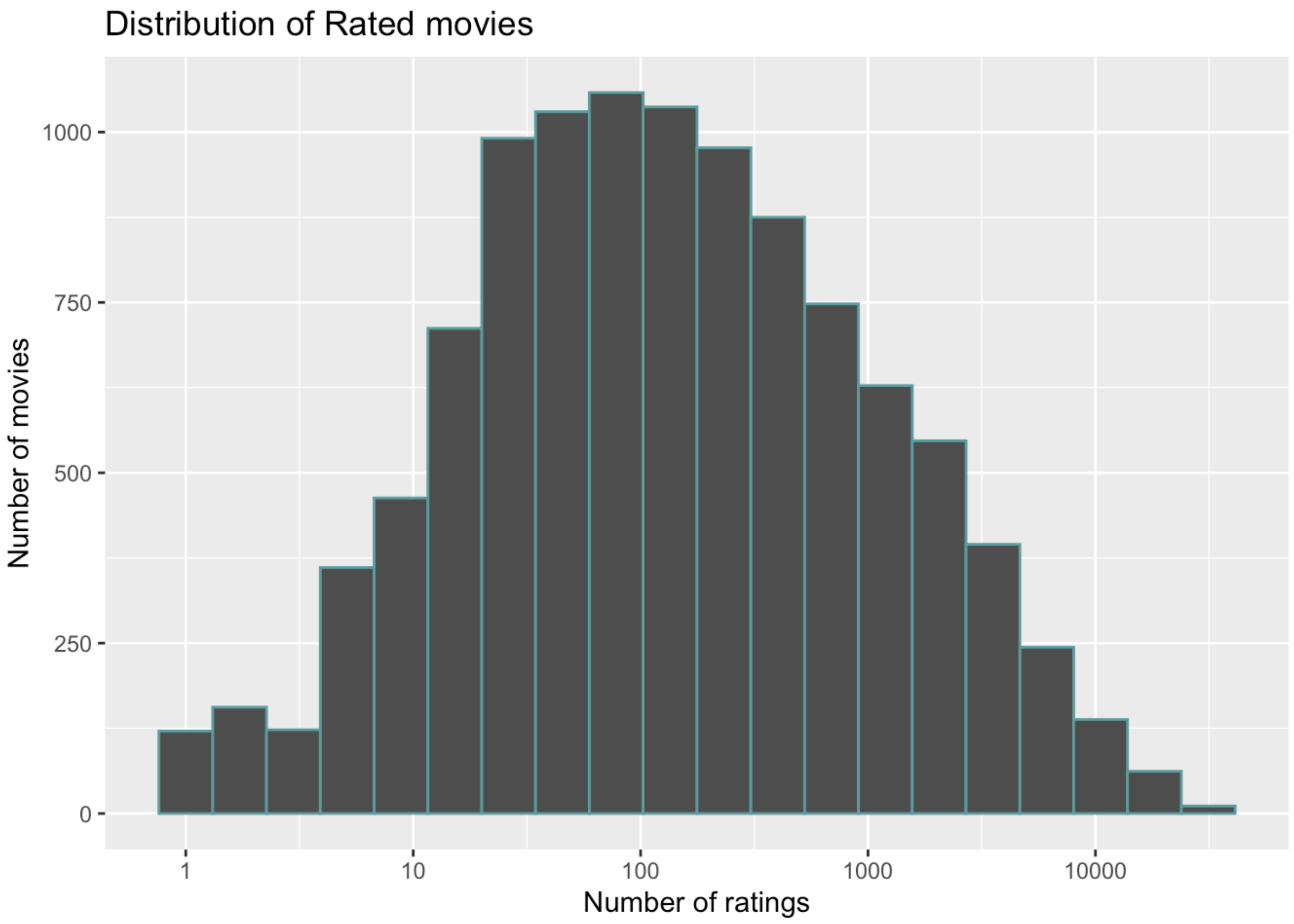
```
edx %>%
 filter(movieId == 296) %>%
 head()
```

```
##   userId movieId rating  timestamp               title             genres
## 1     10     296      2  941529864 Pulp Fiction (1994) Comedy|Crime|Drama
## 2     11     296      3  946012105 Pulp Fiction (1994) Comedy|Crime|Drama
## 3     13     296      4 1035218406 Pulp Fiction (1994) Comedy|Crime|Drama
## 4     18     296      5 1111545895 Pulp Fiction (1994) Comedy|Crime|Drama
## 5     22     296      5  838983277 Pulp Fiction (1994) Comedy|Crime|Drama
## 6     23     296      4  849544083 Pulp Fiction (1994) Comedy|Crime|Drama
```

The above table displays the actual rating provided by 6 random users for the movie id '296' and we observe that this movie has got ratings in all ranges from 2 through 5. Though the movie is rated by largest number of users, the actual rating is distributed across the highest to lowest rating values.

2.3.2 Study II

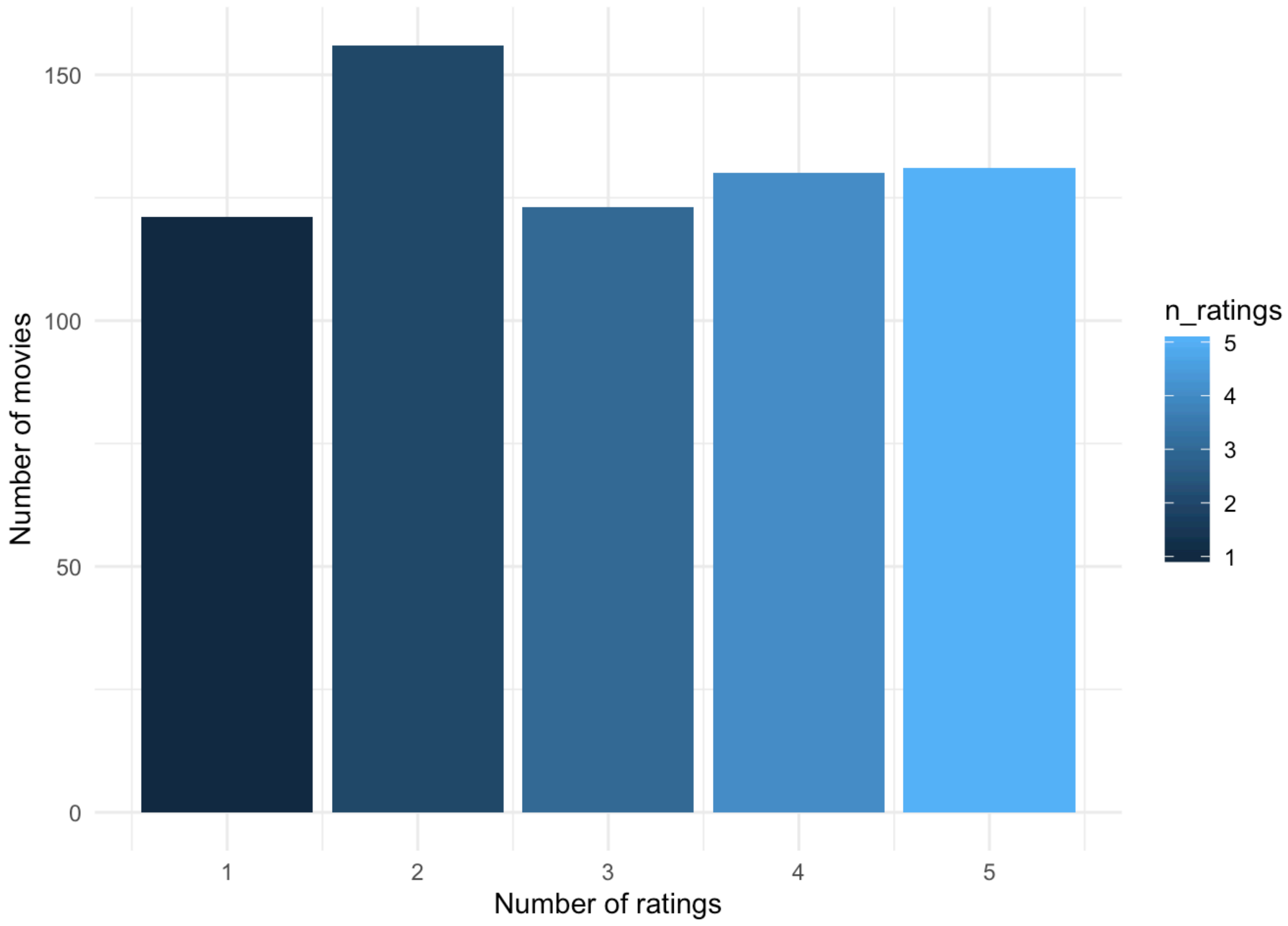
As a next step of our data exploration, let us study the distribution of rated movie through graphical representation.



2.3.2.1 Observation 1

The above graph displays the distribution of rated movies i.e. number of times certain number of movies have been rated. This distribution helps us to understand that there are movies that have been rated by many users and there are movies that have been rated by few users. The number of times movies have been rated is not uniformly distributed.

Let’s take a closer look at the movies that are rated by few number of users.



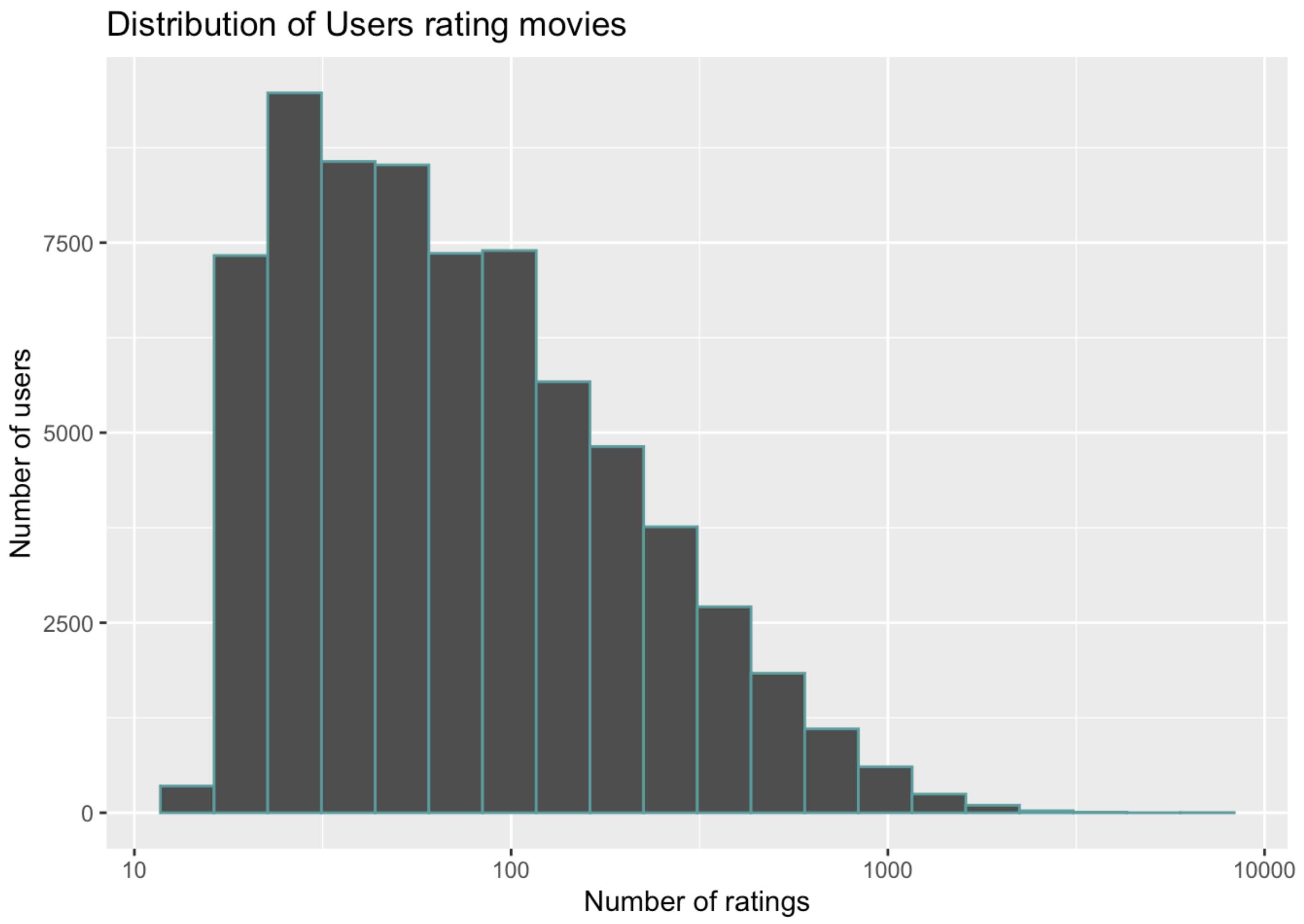
The above graph displays the number of movies that have been rated 5 times or less. For instance there are 121 movies that have been rated just once. This rating information is too little and uncertain to be used to predict future rating for these 121 movies.

2.3.2.2 Inference 1

From the above observations, we infer that we need to incorporate the effect of “variation in the distribution of rated movies” in our model.

2.3.3 Study III

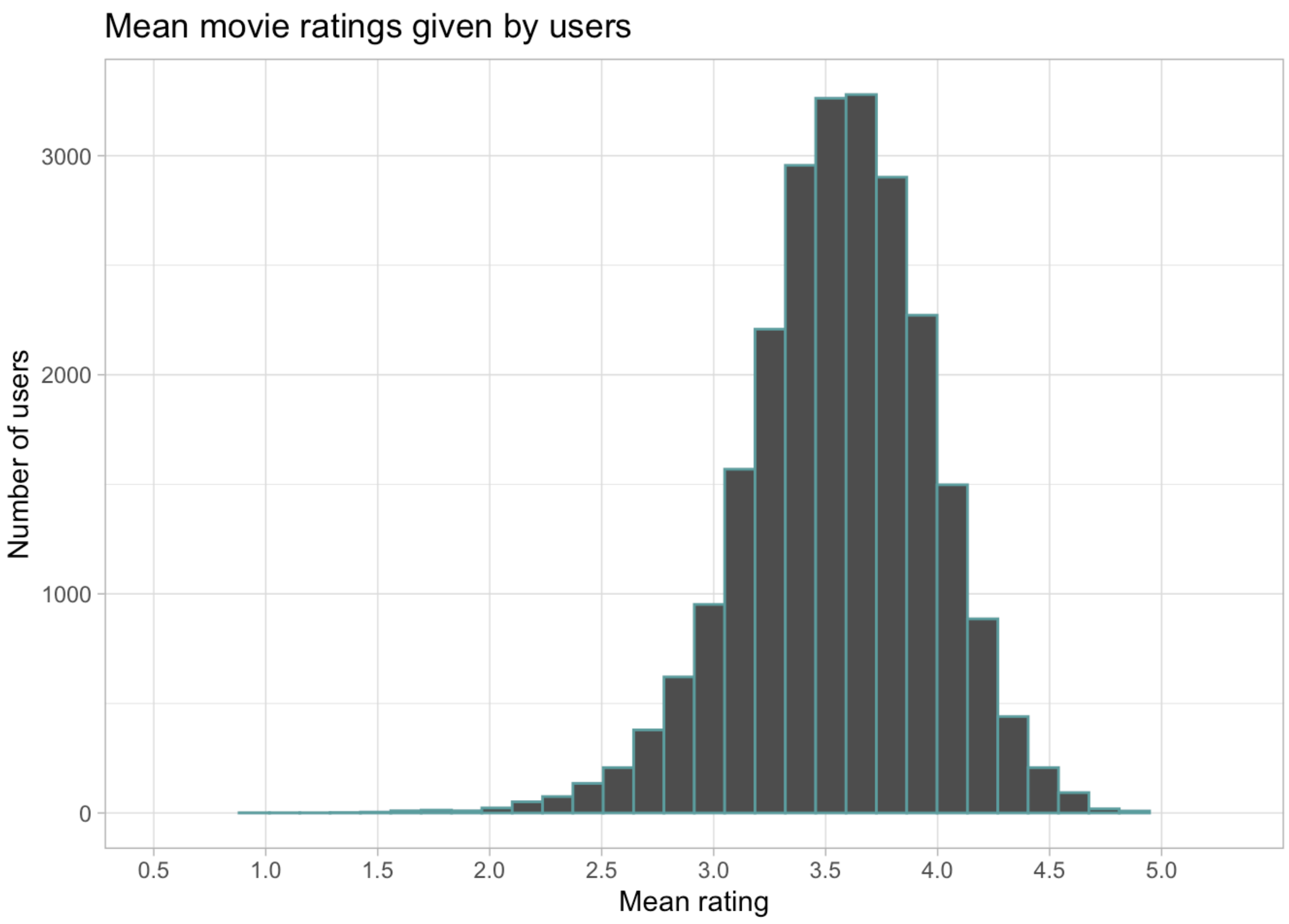
Next, we’ll see the distribution of users rating the movies.



2.3.3.1 Observation 2

The above graph displays the distribution of users rating the movies i.e. number of times movies have been rated by a certain number of users. From this distribution we observe that as the number of times (the movies are rated) increases, the number of users drops gradually. In other words, the majority of the users have rated less than 100 number of movies.

Let’s focus on users who have rated more than 100 movies. In order to understand how these user’s ratings contribute to our model, we need to look at the mean rating of all these users who have rated more than 100 movies.



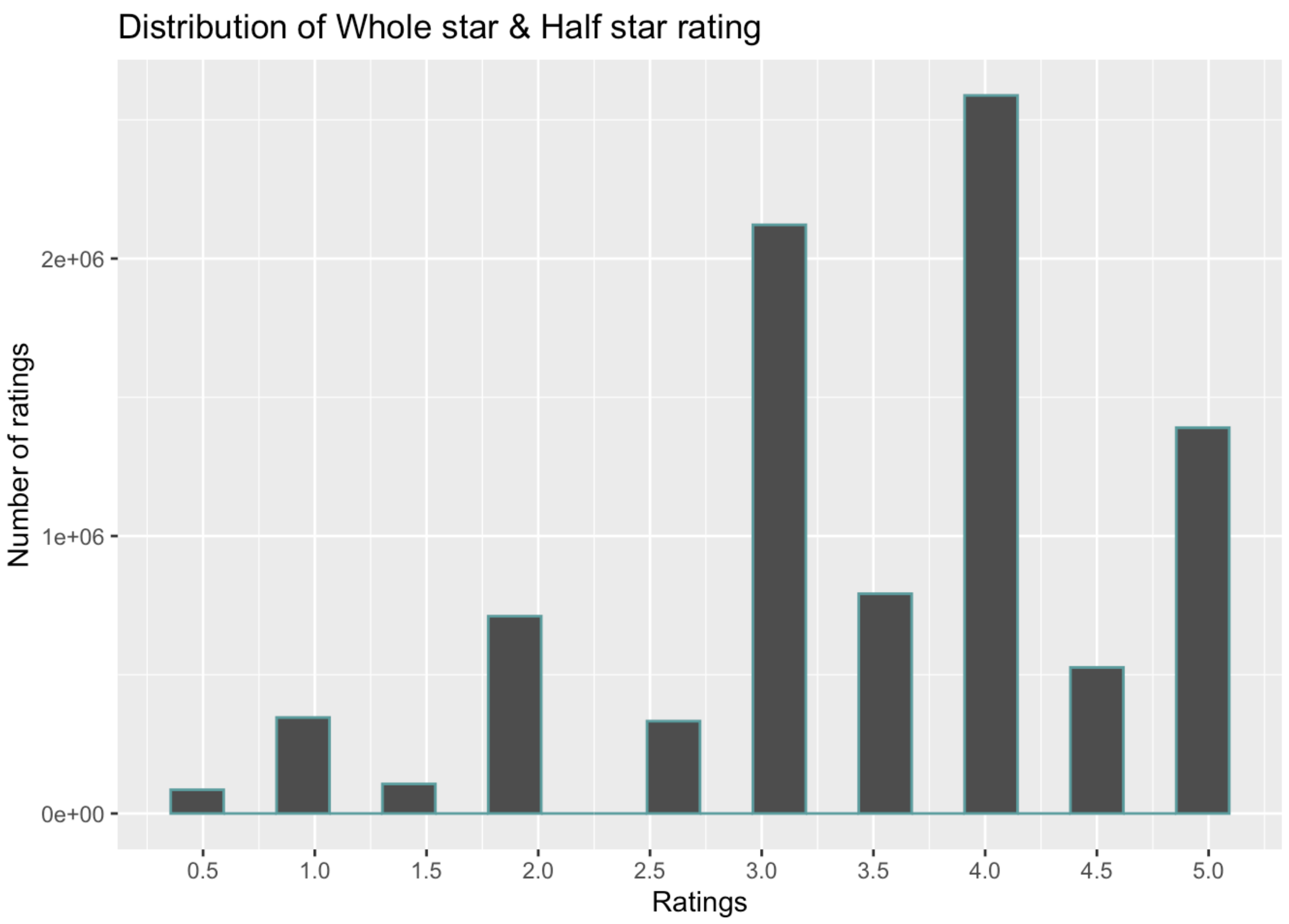
From the above graph, we observe that certain unusual pattern where mean rating is too low or too high, which means those users have rated too low (could be stingy or stringent) or too high (probably generous) for more than 100 movies.

2.3.3.2 Inference 2

From the above observations, we infer that we need to incorporate such user's effect in our model. This would counter the effect of high rating provided by a generous user for a less than average movie and thus we may be able to improve our predictions.

2.3.4 Study IV

As a last step of our data exploration, let us look at the distribution of actual rating values across movies.



2.3.4.1 Observation 3

The above graph shows the distribution of actual movie rating i.e. number of times users have rated using a certain rating value. For better clarity we have considered whole and half star rating values. Following are the observations -

- The top first and second whole star ratings are 4 and 3 respectively
- The top first and second half star ratings are 3.5 and 4.5 respectively
- Half star ratings are less common than whole star ratings

2.3.4.2 Inference 3

In general, users tend to provide good rating for a movie. In order to improve our model, we need to discover appropriate regularization technique.

3 Modeling Approaches

3.1 First Model

The basic approach to start building the recommendation system is to predict the same rating for all movies across all users available in the database. Let's go for model based approach, where the same rating is assumed for all movies and users.

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

where

μ- the mean rating for all movies

$\varepsilon_{u,i}$ -independent errors sampled from the same distribution centered at 0

3.1.1 Rating Prediction

Let’s calculate the average of all movie ratings from train dataset.

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

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

We have got a value of 3.5 (approx). This means we’ll be predicting 3.5 rating for all movies across users as shown in the following table.

USERID_MOVIEID	M_231	M_266	M_293
U_294	3.5	3.0	3.0
U_47613	3.5	3.5	3.5
U_35893	3.5	1.0	3.0

3.1.2 Predicted Rating Validation

Let’s verify this value against Validation dataset for movie id = 231 by finding the actual mean rating of this movie in validation set.

```
validation %>% filter(movieId == 231) %>% group_by(movieId) %>% summarize(m_i = me
an(rating))
```

```
## # A tibble: 1 x 2
##   movieId   m_i
##     <dbl> <dbl>
## 1     231  2.94
```

3.1.2.1 Observation 4

The actual mean rating of this movie is 2.95 from validation dataset. However we have predicted it as 3.5 which is not even close to accurate.

3.1.3 Prediction Error Measurement

Now, let’s find out RMSE.

```
rmse_1 <- RMSE(validation$rating, mu)
rmse_1
```

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

3.1.3.1 Observation 5

Higher the RMSE, lower is the accuracy of the model. We have got RMSE greater than 1 which means the predicted value has deviated a lot from the actual value.

3.1.4 Inference-First Model

Based on Observation 4 and Observation 5, it is evident that we have to improve this model.

3.2 Introducing Movie effects

According to our "Observation 1", certain movies are rated by fewer users than certain other movies. The distribution of rated movies is not uniform. Per "Inference 1", we need to add effect/bias b_i to improve our our first model

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

where

μ- the mean rating for all movies

b_i - Effect introduced to handle the non-uniform distribution of rated movies

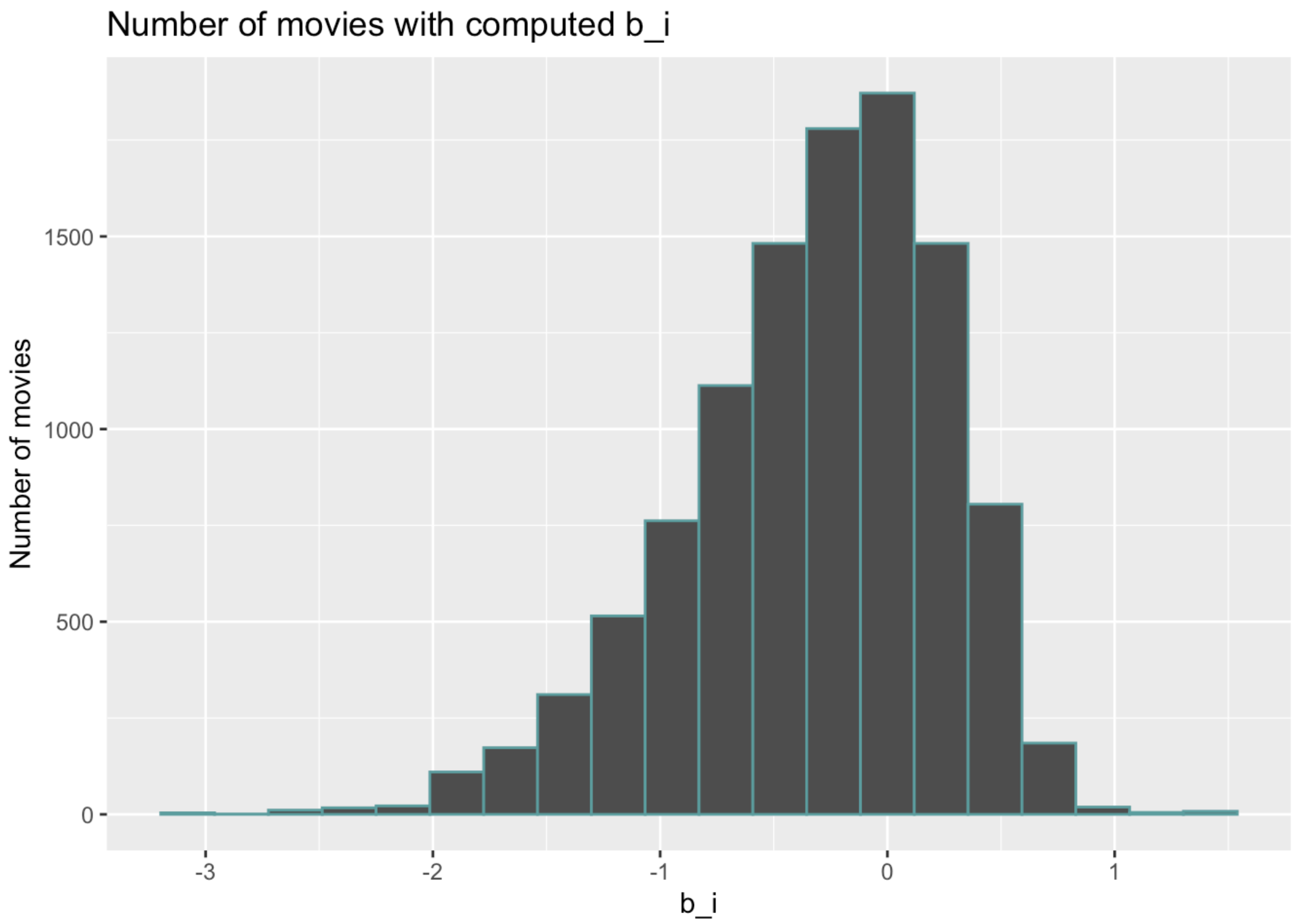
$\varepsilon_{u,i}$ -independent errors sampled from the same distribution centered at 0

3.2.1 Rating Prediction

Let us calculate bi for each movie from train dataset. Here bi is the average of "difference between rating of each movie and mean rating of all movies".

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

movie_avgs %>% ggplot(aes(b_i)) +
  geom_histogram(fill = "grey30",bins = 20, color = "cadetblue") +
  ylab("Number of movies") +
  ggtitle("Number of movies with computed b_i")
```



The above is a graphical presentation of the effect bi for each movie. Note that the value of bi varies from -3.5 through 1.5.

With this model, the value of predicted ratings would be as shown in the following table.

USERID_MOVIEID	M_231	M_266	M_293
U_294	2.935	3.000	3.000
U_476	3.500	3.505	4.012
U_358	2.935	1.000	3.000

3.2.2 Predicted Rating Validation

As a next step, let’s run this bi against validation dataset and gather the predicted rating for movies available in the validation dataset.

```
predicted_ratings <- validation %>%
                     left_join(movie_avgs, by='movieId') %>%
                     mutate(pred = mu + b_i) %>%
                     pull(pred)
```

3.2.2.1 Observation 6

As an intermediate verification, let's take movie id = 231.The actual mean rating of this movie is 2.95 from validation dataset. The bi calculated from train set for this movie is -0.577. Using this b_i value, the predicted rating for this movie is computed as 2.935121 which is closer to the actual mean rating for this movie.

Predicted rating of movie '231' = μ (3.512) + b_{231} (-0.577) = 2.935

3.2.3 Prediction Error Measurement

Finally, let's find out RMSE to understand the overall improvement of this model.

```
rmse_2 <- RMSE(predicted_ratings, validation$rating)
rmse_2
```

```
## [1] 0.9437046
```

3.2.3.1 Observation 7

We observe that RMSE has improved. It has come down from greater than 1 to ~0.944.

3.2.4 Inference-Movie Effects

Based on Observation 6 and Observation 7, it is evident that we have improved the predicted rating in this model compared to our previous model by introducing bi effect.

3.3 Introducing User Effects

According to our "Observation 2", some users are quite active in providing rating for the movies while some show least interest. Also some users are critical of their ratings while some are generous. Per "Inference 2", we need to add an effect bu to counter such user's effect and further improve our model.

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

where

μ- the mean rating for all movies

b_i - Effect introduced to handle the non-uniform distribution of rated movies

b_u - Effect introduced to handle the variation in the distribution of users rating the movies

$\varepsilon_{u,i}$ -independent errors sampled from the same distribution centered at 0

3.3.1 Rating Prediction

Let us calculate bu for each movie from train dataset. Here b_u is the average of "Rating - the mean rating for all movies - b_i effect".

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

With this model, the value of predicted ratings would be as shown in the following table.

USERID_MOVIEID	M_231	M_266	M_293

	U_294	3.018	3.000	3.000
	U_476	3.500	3.627	4.134
	U_358	1.976	1.000	3.000

3.3.2 Predicted Rating Validation

As a next step, let’s run this bu against validation dataset and gather the predicted rating for movies available in the validation dataset.

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

3.3.2.1 Observation 8

As an intermediate verification, let’s take movie id = 231. The actual rating of this movie for user id = 294 is 3 from validation dataset and bi is -0.577. The bu calculated from train set for this movie for user id = 294 is 0.083. Using this bu value, the predicted rating is computed as 3.018 which is closer to the actual rating provided by this particular user for this movie compared to the previous model.

Predicted rating - movie ‘231’ - user ‘1’ = μ (3.512) + b_{231} (-0.577) + b_{294} b(0.083) = 3.018

3.3.3 Prediction Error Measurement

Finally, let’s find out RMSE to measure the overall improvement of this model.

```
rmse_3 <- RMSE(predicted_ratings, validation$rating)
rmse_3
```

```
## [1] 0.8655329
```

3.3.3.1 Observation 9

We observe that RMSE has improved. It has further come down to ~0.866.

3.3.4 Inference-User Effects

Based on Observation 8 and Observation 9, it is evident that we have improved the predicted rating in this model compared to our previous models due to the introduction of b_u effect.

4 Regularization

4.1 Need for Regularization

```
rmse_results <- tibble(method = "Model1 : Average Movie Rating", RMSE = rmse_1)
rmse_results <- bind_rows(rmse_results,tibble(method="Model2 : Movie Effect", RMSE
= rmse_2 ))
rmse_results <- bind_rows(rmse_results,tibble(method="Model3 : Movie and User Effe
ct",RMSE = rmse_3))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                         RMSE
##   <chr>                         <dbl>
## 1 Model1 : Average Movie Rating 1.06 
## 2 Model2 : Movie Effect         0.944
## 3 Model3 : Movie and User Effect 0.866
```

The expected RMSE is less than or equal to 0.8649. Even after adding movie and user effects, we haven't achieved the expected RMSE yet. Let's take a step back, revisit our model and find out where we are falling behind.

```
validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i),mu_plus_bi = mu + b_i) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  select(movieId,rating,mu_plus_bi,b_i) %>%
  distinct()
```

```
##   movieId rating mu_plus_bi       b_i
## 1     318    0.5   4.456928 0.9444640
## 2     858    0.5   4.418851 0.9063873
## 3      50    0.5   4.369967 0.8575031
## 4     527    0.5   4.365628 0.8531640
```

The above table has the first 4 movies with largest difference between actual and predicted rating, where the rating is predicted using movie effect b_i. It is obvious that the prediction has largely deviated from actual rating. Let's find out what has brought down the accuracy of b_i value.

We see that the value b_i varies from -3.5 through 1.5 in GRAPH 3.2.1, which means if b_i = 1.5 for a movie, then the predicted rating will be 5, provided μ = 3.5. In other words, those movies with best predicted ratings will have larger estimate of b_i value. Let's first take a look at the number of users rated the top 5 best predicted movies.

```
edx %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:5) %>%
  select(movieId,n)
```

```
## # A tibble: 5 x 2
##   movieId     n
##     <dbl> <int>
## 1    3226     1
## 2   33264     2
## 3   42783     1
## 4   51209     1
## 5   53355     1
```

The 5 movies that we have predicted with best rating were rated by one or few users. For instance, for movie id = 3226, b_i computed is 1.49 based on the rating of one user, leading to more uncertainty in our prediction. Such large estimates derived from fewer samples cannot be relied upon, thus need for regularization arises. When a model poorly performs, Regularization enables us to create a penalized model.

4.2 Penalized Model

We need to create a model that works well with all variability of data samples. Say for instance, the model should predict accurate rating for a movie rated by 100 users as well as for a movie rated by just one user.

First we shall look at least squares estimation,

$$\frac{1}{N} \sum_{u,i} \left(y_{u,i} - \mu \right)$$

For movies rated by 100 users, N=100 and the estimate is probably close to accurate.

However let's consider movies rated by just one user, the estimate will not be accurate.

$$y_{u,i} - \mu$$

Also note that the estimate makes independent error εu,i as zero in the "$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$" model.

In situations like this, a better approach is penalized regression where a penalty lambda is introduced. This tuning parameter or penalty becomes larger when many b_i values are large. The consequence of introducing penalty lambda is to reduce/shrink the estimate b_i value to zero.

4.2.1 Choosing Tuning Parameter *lambda*

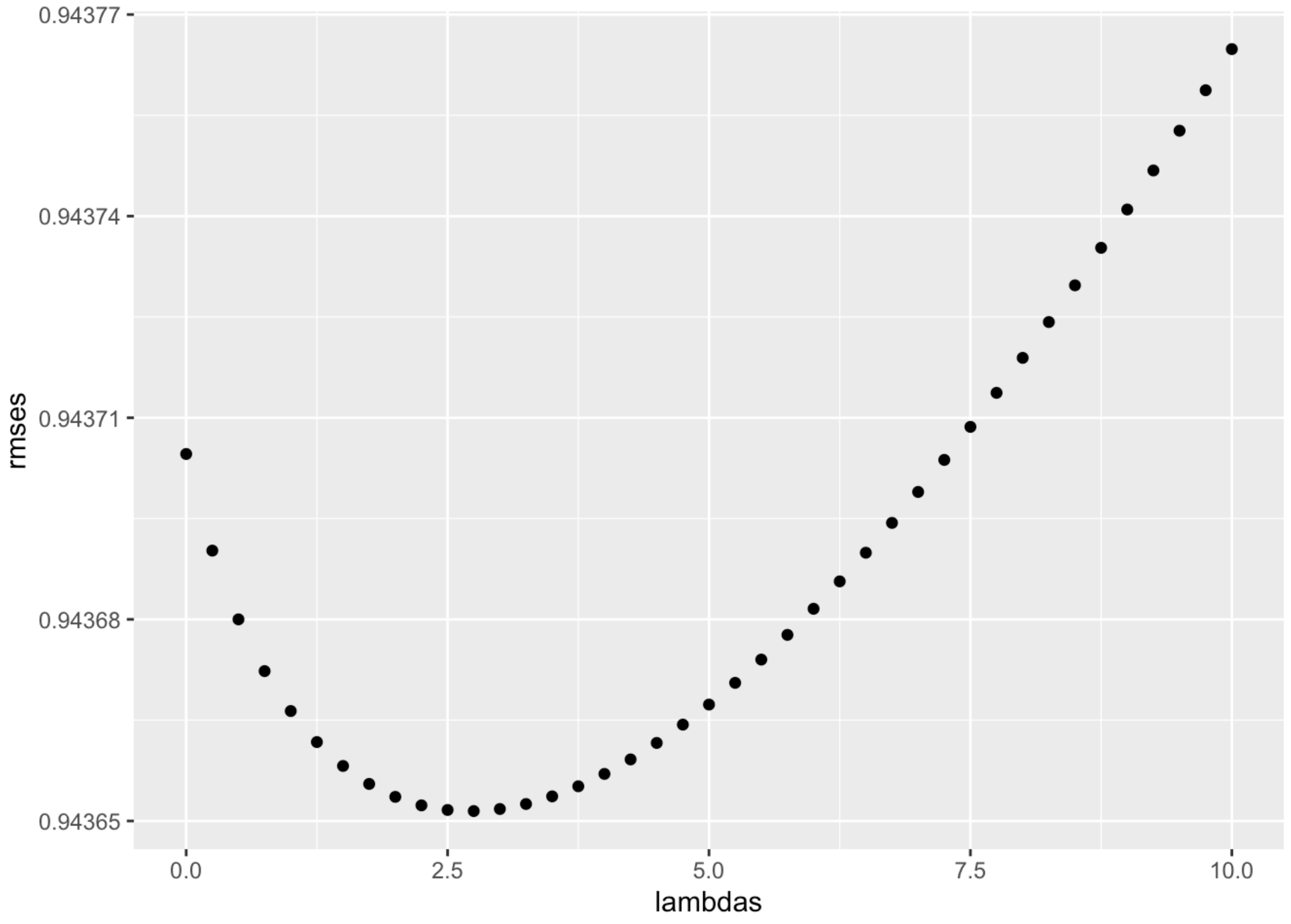
Choosing a right value for lambda is critical as it determines the amount of shrinkage.

Let's use Cross validation, a technique that helps to assess the model to validate if it's generalized enough to apply on an independent data set. In other words, cross validation enables to optimize parameters. Cross validation is done on train set, as the test set should never be used for tuning.

4.2.2 Regularization - Movie Effect

Using values ranging from 0 through 10 for lambda, b_i estimate is calculated. Movie rating is predicted using this b_i estimate and RMSE is measured for the predicted value from train set against rating from validation set.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>% mutate(pred = mu + b_i ) %>% pull(pred)
  return(RMSE(predicted_ratings, validation$rating)) })
qplot(lambdas, rmses)
```



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

```
## [1] 2.75
```

Lambda value for minimum RMSE is calculated. Using this lambda value, b_i estimate for movie id 3226 becomes 0.396675. Thus the contribution of tuning parameter in regularizing the movie effect is apparent.

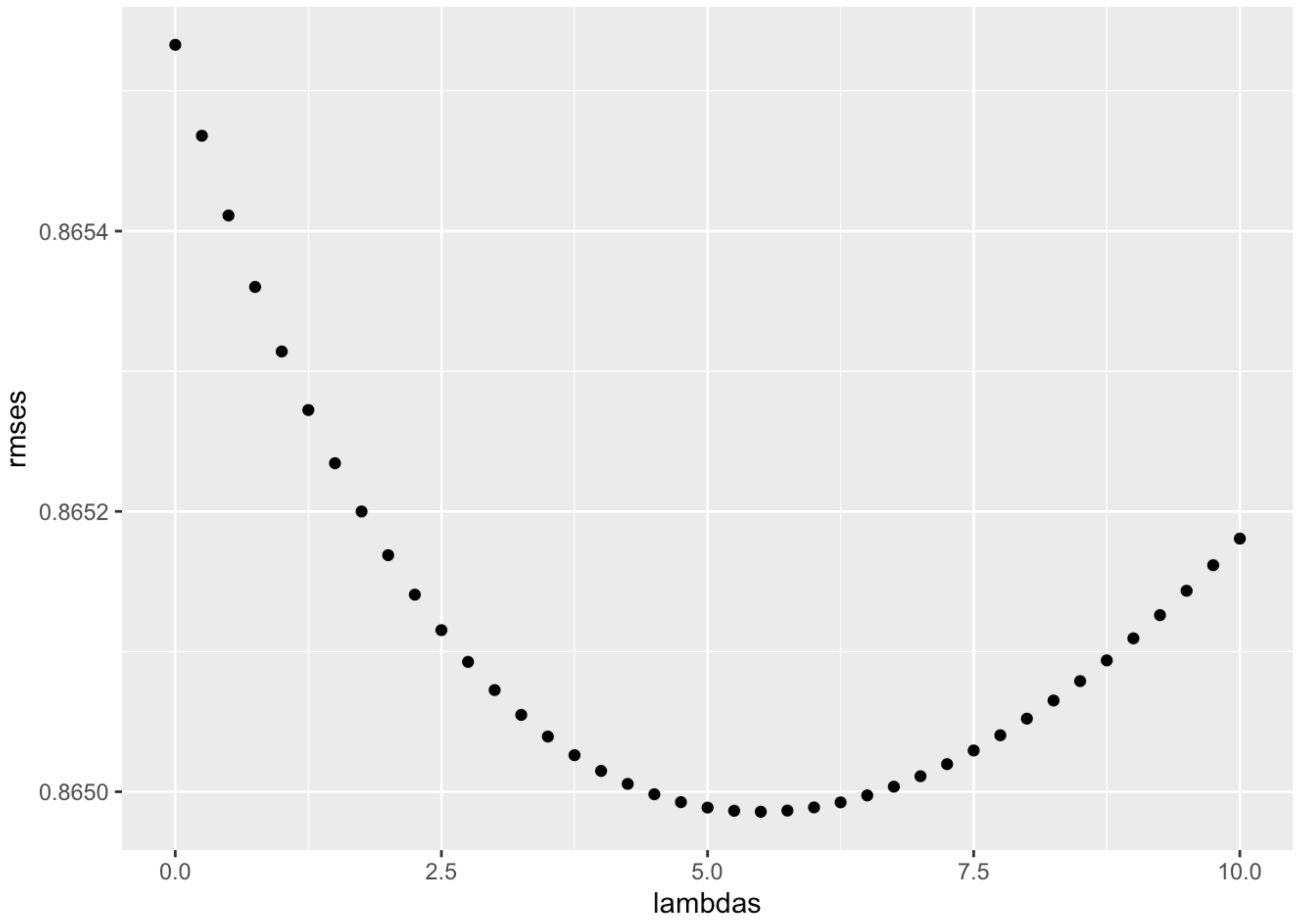
```
rmse_results <- bind_rows(rmse_results,tibble(method="Model4 : Regularized Movie e
ffect model",RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1 : Average Movie Rating	1.0606506
Model2 : Movie Effect	0.9437046
Model3 : Movie and User Effect	0.8655329
Model4 : Regularized Movie effect model	0.9436515

4.2.3 Regularization - Movie and User Effect

Using values ranging from 0 through 10 for lambda, bi and bu estimates are calculated. Movie rating for a user is predicted using these estimates and RMSE is measured for the predicted value from train set against rating from validation set.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pr
ed = mu + b_i + b_u) %>% pull(pred)
  return(RMSE(predicted_ratings, validation$rating)) })
qplot(lambdas, rmses)
```



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

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

Lambda value for minimum RMSE is calculated. Using this lambda value, b_i estimate for movie id 33264 is 0.626 and b_u is 0.168 (for user id 48159). Thus the contribution of tuning parameter in regularizing the movie and user effect is apparent.

```
rmse_results <- bind_rows(rmse_results,tibble(method="Model5 : Regularized Movie a
nd User effect model",RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1 : Average Movie Rating	1.0606506
Model2 : Movie Effect	0.9437046
Model3 : Movie and User Effect	0.8655329
Model4 : Regularized Movie effect model	0.9436515
Model5 : Regularized Movie and User effect model	0.8649857

4.2.3.1 Observation 10

Regularization by introducing penalty in the Movie and User effect model has helped to bring down the RMSE to the expected value.

5 Conclusion

The expected RMSE of value less than or equal to 0.8649 has been attained by using lambda value for the full model.

Thus a regularized Movie and user effect model would predict movie ratings as in the below sample table.

USERID_MOVIEID	M_231	M_266	M_293
U_294	3.012	3.000	3.000
U_476	3.500	3.624	4.131
U_358	2.020	1.000	3.000