HarvardX: PH125.9x Data Science: Movielens Project Submission

Chang Soo Yen

6 January 2020

1 Introduction

This section introduces the dataset to be used in this project, project goals and key steps taken to achieve the project goals.

1.1 Dataset

The dataset used in this project is the MovieLens 10M Dataset. It is initialized for exploratory purposes with the use of the following code:

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

# Note: this process could take a couple of minutes

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")

# 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")
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
```

```r
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)

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

1.2 Project Goals

Recommendation systems use ratings that users have given items to make specific recommendations. In this project, our goal is to create a machine learning algorithm that predicts user ratings (from 0.5 to 5 stars, with a 0.5 interval). We will be utilising the **edx** dataset to predict movie ratings in the **validation** dataset.

1.3 Key Steps

The root mean square error (RMSE) will be used to assess the capabilities of the machine learning algorithms in obtaining as close a prediction to the **validation** dataset.

The RMSE is defined as:

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

For the purposes of this project, we will utilise the following code to calculate the respective RMSE's obtained from each algorithm:

```r
RMSE <- function(actual_rating, predicted_rating){
  round(sqrt(mean((actual_rating - predicted_rating)^2)),10)
}
```

Based on the RMSE of the different models to be used, the model which returns the lowest RMSE will be used to ultimately predict the movie ratings of the **validation** dataset.

2 Methodology

This section analyses the various components of the **edx** dataset which should be taken into consideration when deciding the appropriate modelling approaches suitable for this project. It then discusses the various modelling approaches that were used to obtain the lowest possible RMSE.

To begin off, a familiarisation with the **edx** dataset would be useful in understanding its structure:

```
head(edx)
```

```
##   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
```

It can be seen that the **edx** dataset is in tidy format.

The structure of the **edx** dataset can be checked with the following code:

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

We observe that there are 9000055 rows and 6 columns. It can also be observed that each row represents a rating given by one user to one movie.

It is also crucial to check if the dataset contains any "NA" values which need to taken note of:

```
any(is.na(edx))
```

```
## [1] FALSE
```

Thankfully, the **edx** dataset continues 0 "NA" values as stated FALSE by the above result from the code ran.

Lastly, to have an idea of exactly how many unique users rated movies and unique movies which were rated is crucial in understanding the spread of ratings:

```
edx %>%
  summarize(distinct_users = n_distinct(userId),
            distinct_movies = n_distinct(movieId))
```

```
##   distinct_users distinct_movies
## 1          69878           10677
```

It can be seen that there are 69878 unique users and 10677 unique movies. We can then infer that not every user rated every movie since there are less than 69878 * 10677 rows in the **edx** dataset.

2.1 Introduction to EDX Dataset's Subsets

To find out the best model which would produce the lowest RMSE and **NOT** make use of the **validation** set, we will create subsets in the **edx** dataset, where there would be a **tempedx** dataset likened to be the original **edx** dataset. The **tempvalidation** dataset would then likened to be the original **validation** dataset. The following code produces the above wanted:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
tempedx <- edx[-test_index,]
temporary <- edx[test_index,]

# Make sure userId and movieId in tempvalidation set are also in tempedx set

tempvalidation <- temporary %>%
  semi_join(tempedx, by = "movieId") %>%
  semi_join(tempedx, by = "userId")

# Add rows removed from tempvalidation set back into tempedx set

removed <- anti_join(temporary, tempvalidation)
tempedx <- rbind(tempedx, removed)
```

These subsets will be used to create the various models and calculate the individual RMSEs. Thereafter, the best model will be used to calculate the RMSE for the final predicted rating and the validation set.

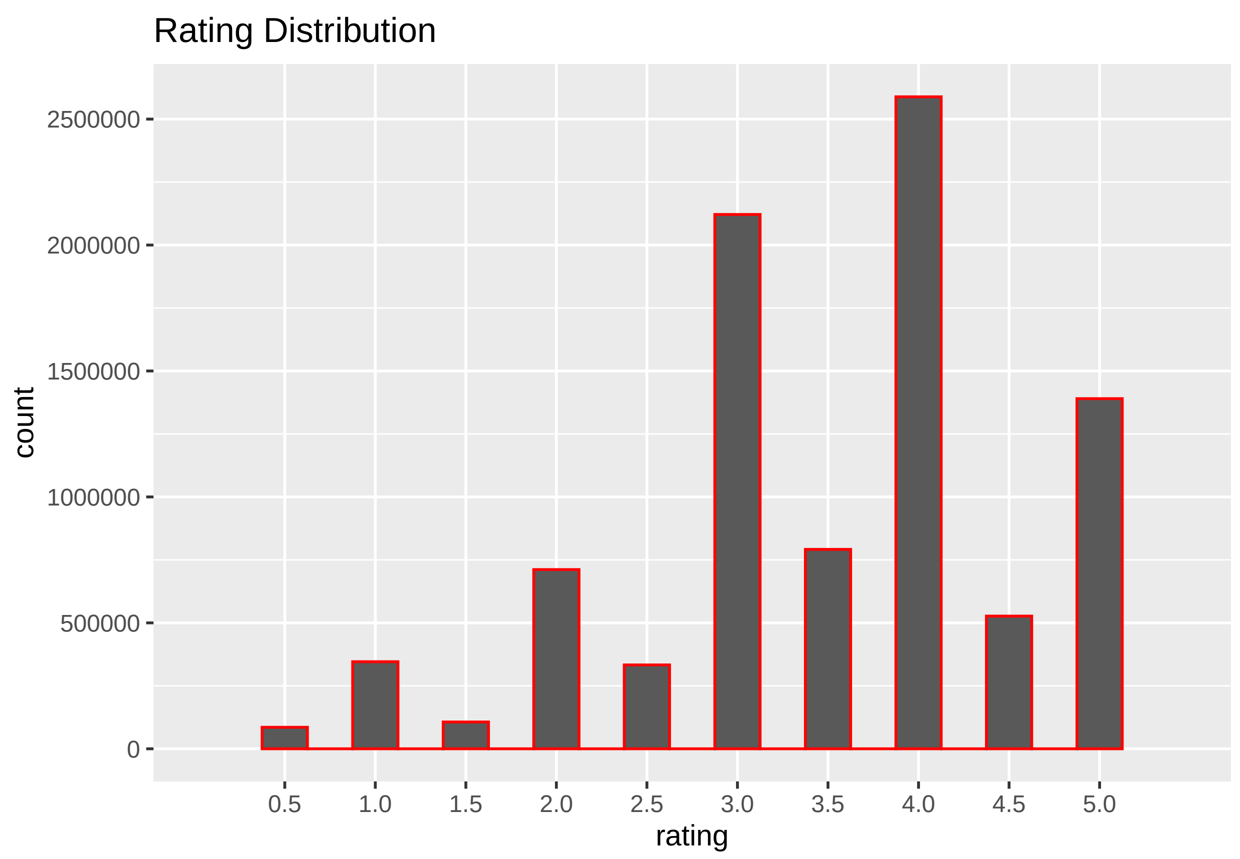
However, analysis will still be done on the entire edx dataset as per normal.

2.2 Average Movie Rating

We now begin with a basic model to have a feel on how big the RMSE is without considering movie and user variabilities.

Before doing so, we will take a look at the distribution of movie ratings with the following code:

```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "red") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating Distribution")
```



It can be seen that 4 is the most common rating followed by 3 then 5. It seems relatively appropriate to assume the same rating for all movies and users with all the differences explained by random variation.

2.2.1 Model

The model would look like this:

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

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies.

We will now utilise **tempedx** dataset and **tempvalidation** dataset to trial on this model's RMSE.

Now, we will find the average of all ratings and name it **mu**:

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

```
## [1] 3.512456
```

We will now use **mu** i.e. μ to predict all the unknown movie ratings:

```
movierating_rmse <- RMSE(tempvalidation$rating, mu)
movierating_rmse
```

```
## [1] 1.060054
```

The RMSE seems to be quite high, considering the nature of this approach which assumes that all variations are random.

For better analysis later, we will now create a results table to compare the RMSEs of the different approaches.

```
options(pillar.sigfig = 5)
rmse_results <- tibble(method = "Temp Average Movie Rating", RMSE = movierating_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Temp Average Movie Rating 1.0601
```

2.3 Movie Effect

As naive as it was, assuming that all variations are random in the previous model would not produce a desirable RMSE.

From the rating distribution, it can be observed that movies have a plethora of ratings, hence including the consideration that some movies are just more popular than others would better our accuracy.

Hence, it would be more precise to augment our previous model and include a term to represent the average ranking for a movie.

2.3.1 Model

The model would look like this:

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

where b_i represents the average ranking for movie i.

We will now utilise **tempedx** dataset and **tempvalidation** dataset to trial on this model's RMSE.

We now calculate the movie averages with the following code:

```
movie_averages <- tempedx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Then, we will proceed to predict our ratings and calculate the RMSE for this model:

```
predicted_ratings <- mu + tempvalidation %>%
  left_join(movie_averages, by="movieId") %>%
  pull(b_i)

movieeffect_rmse <- RMSE(predicted_ratings, tempvalidation$rating)
movieeffect_rmse
```

```
## [1] 0.9429615
```

The RMSE has decreased, but not substantially. We will put it into the results table for further analysis:

```
options(pillar.sigfig = 5)
rmse_results <- bind_rows(rmse_results, tibble(method = "Temp Movie Effect",
                                               RMSE = movieeffect_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method                        RMSE
##   <chr>                        <dbl>
## 1 Temp Average Movie Rating 1.0601
## 2 Temp Movie Effect         0.94296
```

2.4 Movie & User Effects

Since we have considered the movie effects, it would be ideal to also consider user effects.

Some users are generally more lenient than other users in terms of rating, so we should also take into consideration the plethora of ways users rate.

Hence, it would be even more precise to augment our previous model and include a term to represent the average rating for a user.

2.4.1 Model

The model would look like this:

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

where b_i represents the average ranking for movie i and b_u is a user-specific effect.

We will now utilise **tempedx** dataset and **tempvalidation** dataset to trial on this model's RMSE.

We now calculate the user rating averages with the following code:

```
user_averages <- tempedx %>%
  left_join(movie_averages, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Then, we will proceed to predict our ratings and calculate the RMSE for this model:

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

movieusereffect_rmse <- RMSE(predicted_ratings, tempvalidation$rating)
movieusereffect_rmse
```

```
## [1] 0.8646843
```

The RMSE has decreased, but not substantially. We will put it into the results table for further analysis:

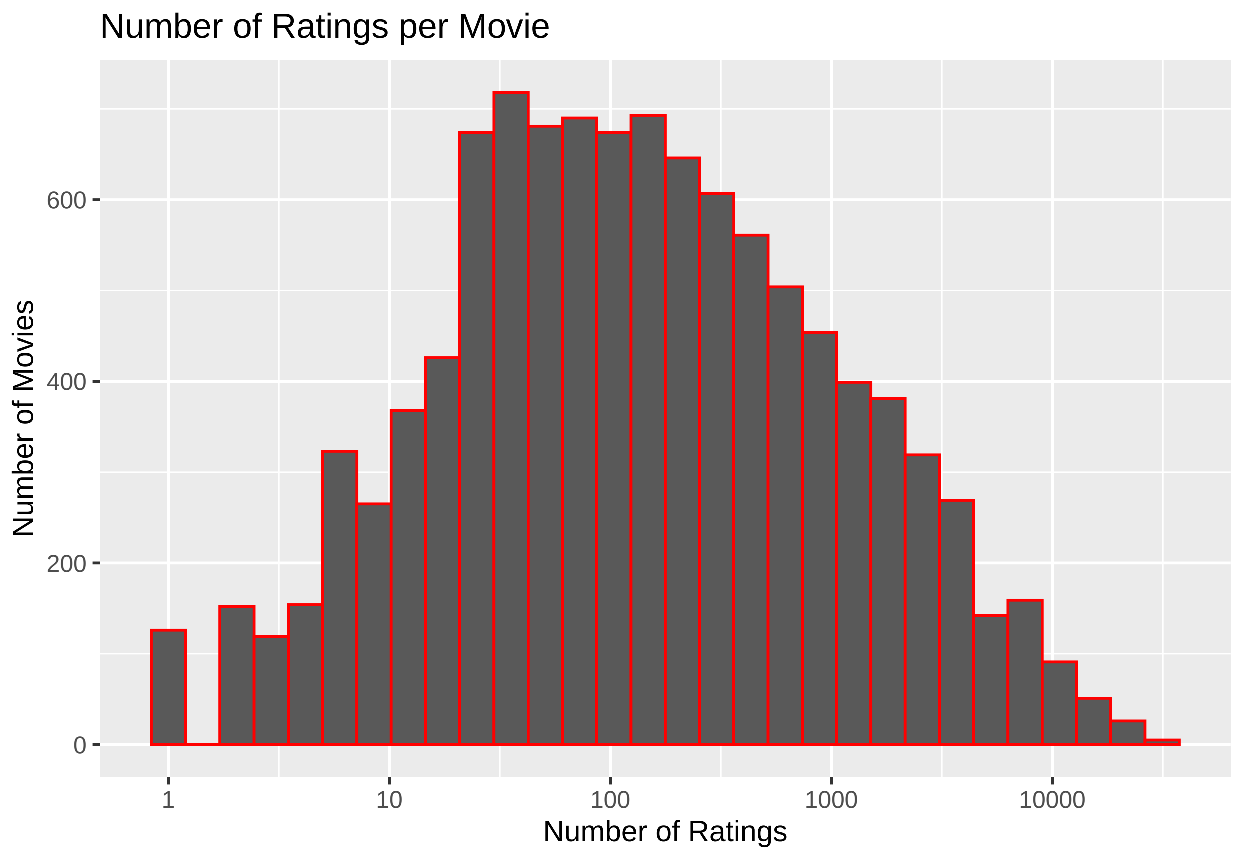
```r
options(pillar.sigfig = 5)
rmse_results <- bind_rows(rmse_results, tibble(method = "Temp Movie & User Effects",
                                               RMSE = movieusereffect_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                       RMSE
##   <chr>                       <dbl>
## 1 Temp Average Movie Rating 1.0601
## 2 Temp Movie Effect         0.94296
## 3 Temp Movie & User Effects 0.86468
```

2.5 Regularized Movie & User Effects

As our RMSE is still relatively high, we move on to analyse the number of ratings per movie to spot any noticeable effect:

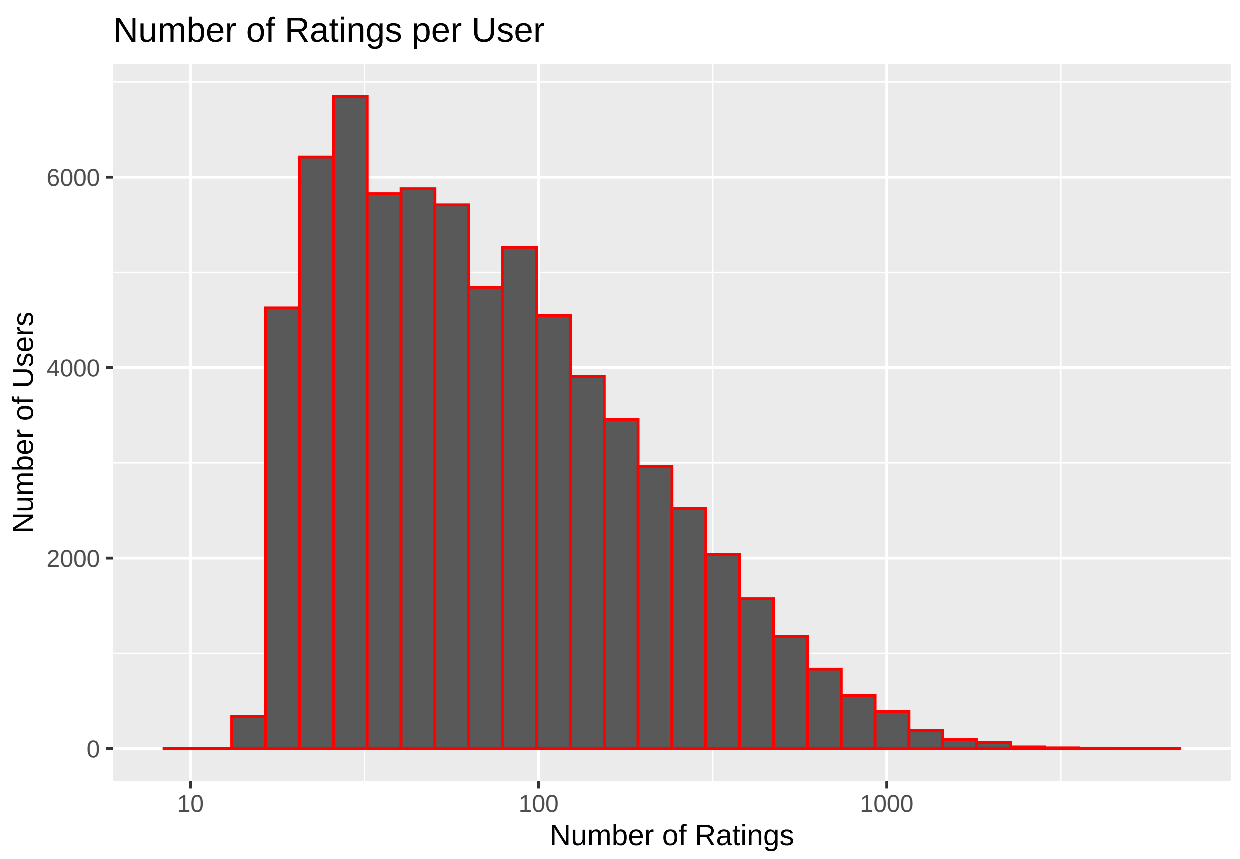
```r
edx %>%
  group_by(movieId) %>%
  summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "red") +
  scale_x_log10() +
  ggtitle("Number of Ratings per Movie") +
  xlab("Number of Ratings") +
  ylab("Number of Movies")
```



We can see that some movies get rated less than others, where 126 movies are only rated once.

To consider if there is huge variability in the number of ratings per user, we will use the following code:

```
edx %>%
  group_by(userId) %>%
  summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "red") +
  scale_x_log10() +
  ggtitle("Number of Ratings per User") +
  xlab("Number of Ratings") +
  ylab("Number of Users")
```



We can see that some users are way more active than others at rating movies, where some users only rate a few times.

From these 2 observations, it should be noted that regularization is required in ensuring that we are able to also penalize large estimates that are formed using small sample sizes.

2.5.1 Model

The model would look like this:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

where

$$\lambda$$

represents the tuning parameter.

We will now utilise **tempedx** dataset and **tempvalidation** dataset to trial on this model's RMSE.

We will now use the partitioned **edx** dataset to choose the tuning parameter, through sampling a list of suitable tuning parameters, finding out which one produces the lowest RMSE. We will then use the lowest found RMSE to compare with the other models:

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(tempedx$rating)

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

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

  predicted_ratings <-
    tempvalidation %>%
    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, tempvalidation$rating))
})

regmovieusereffect_rmse <- min(rmses)
regmovieusereffect_rmse
```

```
## [1] 0.8641362
```

We will put it into the results table for further analysis:

```
options(pillar.sigfig = 5)
rmse_results <- bind_rows(rmse_results, tibble(method = "Temp Regularised Movie & User Effects",
                                               RMSE = regmovieusereffect_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                                   RMSE
##   <chr>                                   <dbl>
## 1 Temp Average Movie Rating              1.0601
## 2 Temp Movie Effect                      0.94296
## 3 Temp Movie & User Effects              0.86468
## 4 Temp Regularised Movie & User Effects 0.86414
```

3 Results

This section analyses the results obtained from calculating the RMSE's of the chosen models with the subsets of the **edx** dataset. The best model with the lowest RMSE is picked and utilised to calculate the RMSE with the **validation** dataset.

3.1 Modelling Results & Performance

The following are the results obtained from the models used to calculate the RMSE of the predicted rating of the *tempvalidation* set:

```
options(pillar.sigfig = 5)
rmse_results
```

```
## # A tibble: 4 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Temp Average Movie Rating            1.0601
## 2 Temp Movie Effect                    0.94296
## 3 Temp Movie & User Effects            0.86468
## 4 Temp Regularised Movie & User Effects 0.86414
```

From this table, we can see that the "Temp Regularised Movie & User Effects" yields the lowest RMSE.

It is evident and expected, since this model not only considered both the movie and user effects, but also regularised it to ensure that the large estimates that are formed using small sample sizes are penalised. This allows the prediction of ratings to be more accurate.

3.2 Final RMSE Result of Selected Model

We will now utilise the Regularised Movie & User Effects model to predict the ratings of our **validation** dataset.

Referring back to the model, we have found the tuning parameter which produces the lowest RMSE result to be 5:

```
lambdas[which.min(rmses)]
```

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

Hence we will now plug that back into the the model and predict the ratings for the **validation** dataset.

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

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+5))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
```

```r
  summarize(b_u = sum(rating - b_i - mu)/(n()+5))

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

Here are some of the predicted ratings:

```r
head(predicted_ratings)
```

```
## [1] 4.264512 4.992708 4.385029 3.347321 4.232387 2.762940
```

Now, we will calculate the final RMSE obtained from this set of predicted ratings:

```r
final_rmse <- RMSE(predicted_ratings, validation$rating)
final_rmse
```

```
## [1] 0.8648177
```

The RMSE is finally lesser than 0.8649, achieving its target.

We will now proceed to make our conclusion from our project findings.

4 Conclusion

This section concludes the entire project, noting the appropriateness of key steps conducted, and discusses possible limitations and future improvements.

4.1 Summary

We have analysed the **edx** dataset and created 4 models for our machine learning algorithm. Through improvements on each other model, we have considered movie and user effects, thereafter regularising them.

The RMSEs of each model also gradually decreased as we accounted for more intentional variabilities instead of assuming that the variability of ratings were due to randomness.

4.2 Limitations & Future Work

While the RMSE may have hit the target set for this project, more improvements to the RMSE could be made if we take into consideration other factors of variability like the genre, year and even possibly timestamp.

To develop an even more versatile model would also require more time and available hardware to run such complex algorithms.

5 Operating System

Following is the operating system utilised for this project:

```
version
```

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```