MOVIE RECOMMENDATION SYSTEM

DENNIS MULUMBI KYALO

2020-11-22

1. INTRODUCTION

A recommendation system is an example of a machine learning algorithm that involves predicting user responses to options. This algorithm uses ratings that users have given to different items. Different companies, such as Amazon, Netflix, Youtube, and Facebook, have been able to use this knowledge to improve their customer satisfaction and increase their revenues. The project will be focusing on the movie recommendation system used by companies such as Netflix. This system aims at predicting the rating of a particular user to a specific movie. Therefore, films with a higher rating based on a specific user or other users with similar interests are recommendable. The analysis will use various machine learning techniques to create a movie recommendation system using the MovieLens dataset found here. From the MovieLens dataset, the 10M version of the MovieLens dataset will be used. The best-chosen algorithm from the set of algorithms used will be the one that gives the minimal error (RMSE).

2. METHOD

We began by downloading the MovieLens dataset then went ahead and explored it. A clear understanding of the dataset's features can be found from the MovieLens Summary found here. We split the single combinations of the genres feature into individual records for further exploration of the dataset.

We then proceeded to split the MovieLens dataset into the *edx* set and *validation* set, each bearing 90% and 10%, respectively. The former will be used to develop the algorithm, while the latter will be used in the *final* test of the best algorithm chosen to predict the movie ratings. In developing the models, we further split the edx set into training and test sets to design and test the various algorithms.

For evaluation, the Root Mean Square Error (RMSE) was used to test how close the predictions were to the real values.

2.1 Load the necessary Packages.

We first load the packages to be used in the analysis.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(digest)) install.packages("digest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: digest
```

```
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```
library(tidyverse)
library(caret)
library(data.table)
library(ggplot2)
library(digest)
library(recosystem)
```

2.2 Download the MovieLens dataset.

The entire latest MovieLens dataset can be found here https://grouplens.org/datasets/movielens/latest/. For the project I shall be using the 10M version of the MovieLens dataset found here https://grouplens.org/datasets/movielens/10m/. A brief summary of the dataset and its features can be found here http://files.grouplens.org/datasets/movielens/ml-10m-README.html.

```
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 now proceed to split the MovieLens dataset into edx set (90%) and validation set (10%).

```
# 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
```

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

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

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

2.3 Data exploration and visualization.

Let's get a general description of the edx dataset.

```
dim(edx) # The dimension of the dataset
```

```
## [1] 9000055       6
```

The edx dataset is comprised of 9000055 rows and 6 columns.

Here is a glance of the dataset.

```
head(edx) %>% knitr::kable(caption = "Edx Dataset") # A glance of the dataset
```

Table 1: Edx Dataset

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	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

```
# unique users and movies
library(dplyr, warn.conflicts = FALSE)
options(dplyr.summarise.inform = FALSE)

edx %>%
  summarize(Unique_Users = n_distinct(userId),
            Unique_Movies = n_distinct(movieId))
```

```
##   Unique_Users Unique_Movies
## 1        69878         10677
```

There are 69878 and 10677 unique users and movies, respectively.

Here we see how different types of ratings are distributed based on their frequency.

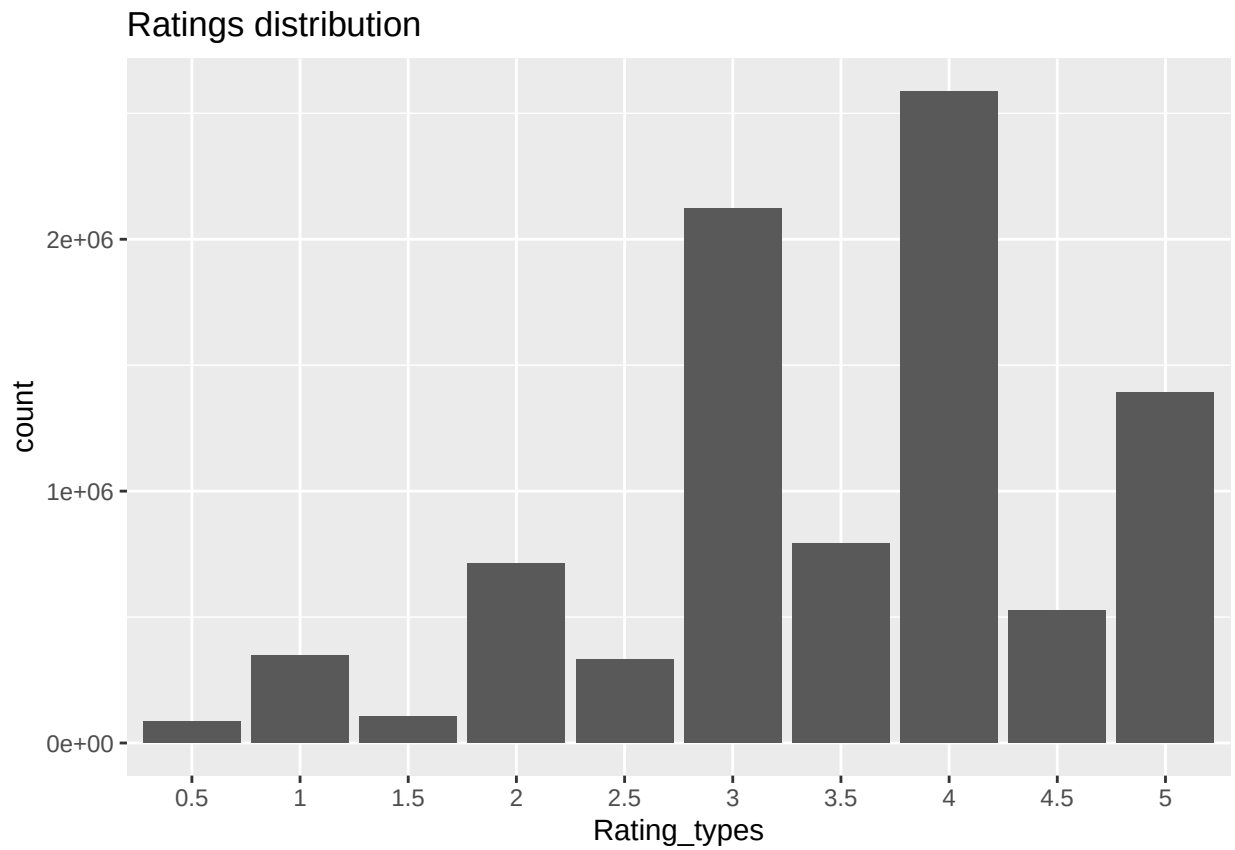
```
#We would like to explore the values of the rating

Rating_types <- as.vector(edx$rating)
table_ratings <- table(Rating_types)
table_ratings %>% knitr::kable(caption = "Rating's Frequency")
```

Table 2: Rating's Frequency

Rating_types	Freq
0.5	85374
1	345679
1.5	106426
2	711422
2.5	333010
3	2121240
3.5	791624
4	2588430
4.5	526736
5	1390114

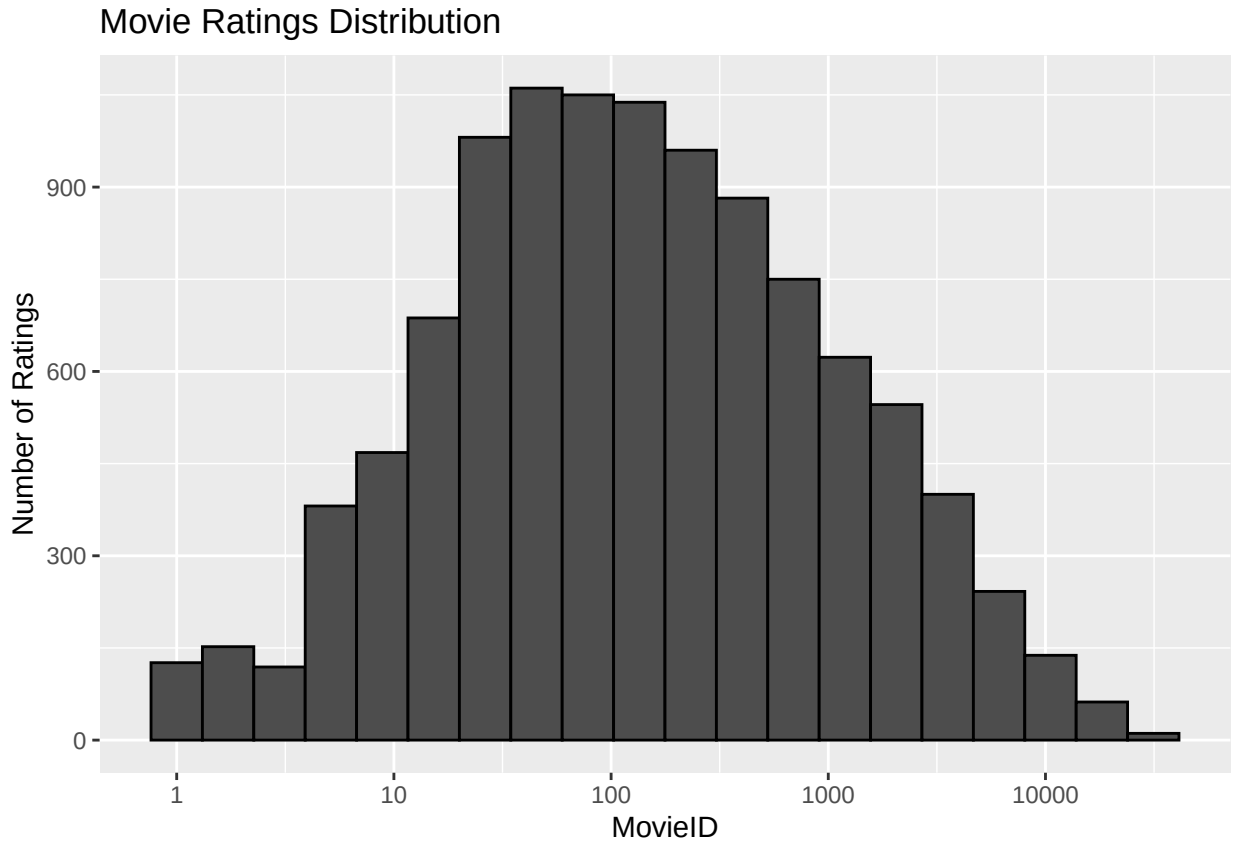
The graph helps us have a clear visualization of the rating distribution.

```
Rating_types <- factor(Rating_types)

edx %>% ggplot(aes(x = Rating_types)) +
  geom_bar(stat = "count") +
  ggtitle("Ratings distribution")
```



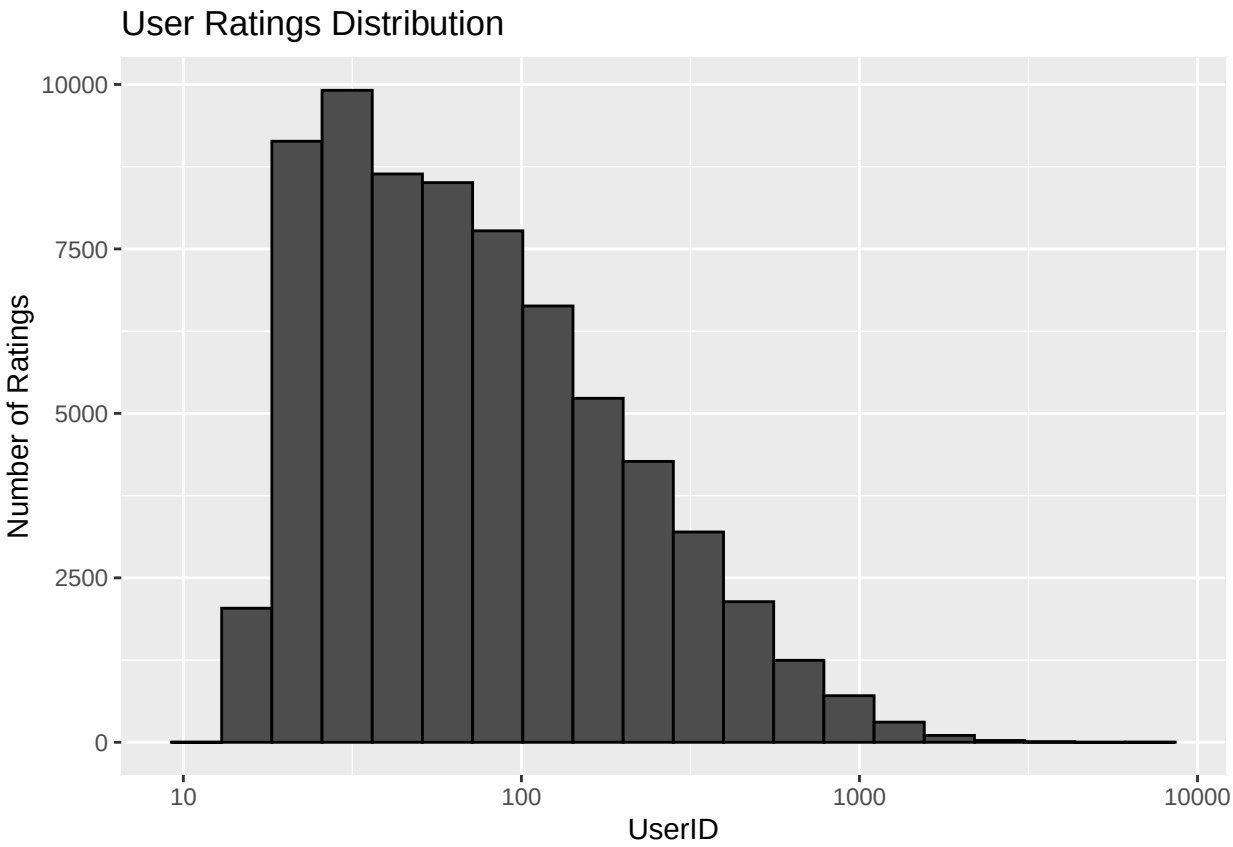
The ratings distribution appears to be negatively skewed (skewed to the left).

```
##We check on the distribution of the movie ratings

edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "grey30", bins = 20, color = "black") +
  scale_x_log10() +
  labs( title = "Movie Ratings Distribution",
        x = "MovieID",
        y = "Number of Ratings")
```



The graph above shows how different movies were rated; as we can see, there are movies with more ratings than others.

```
##We check on the distribution of the movie ratings

edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "grey30", bins = 20, color = "black") +
  scale_x_log10() +
  labs( title = "User Ratings Distribution",
        x = "UserID",
        y = "Number of Ratings")
```



From the graph above, we can see how different users rate different movies; we can also clearly see that some users are more active in rating movies than others.

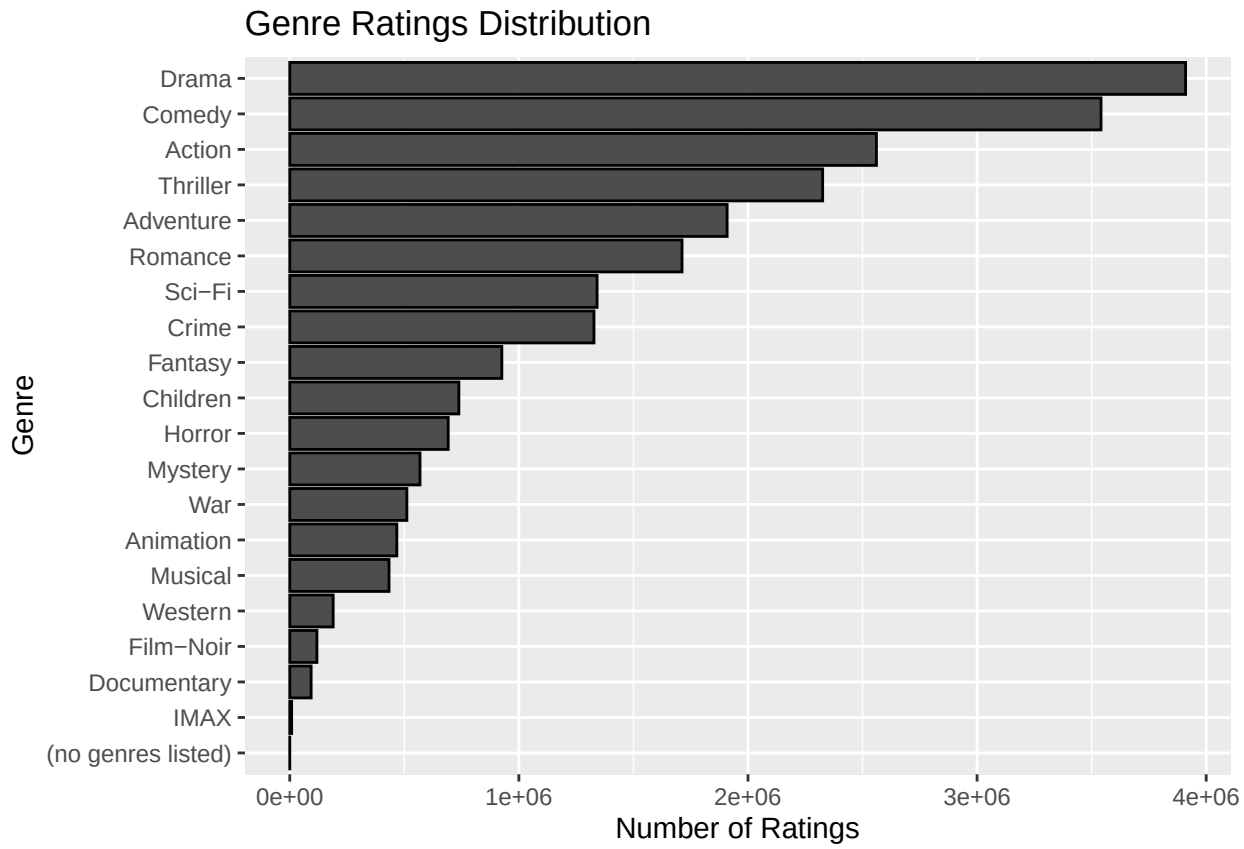
```
##Exploring which movies had the highest rating

edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     count
##      <dbl> <chr>                                                     <int>
##  1     296 Pulp Fiction (1994)                                       31362
##  2     356 Forrest Gump (1994)                                       31079
##  3     593 Silence of the Lambs, The (1991)                          30382
##  4     480 Jurassic Park (1993)                                      29360
##  5     318 Shawshank Redemption, The (1994)                          28015
##  6     110 Braveheart (1995)                                         26212
##  7     457 Fugitive, The (1993)                                      25998
##  8     589 Terminator 2: Judgment Day (1991)                         25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                          24284
## # ... with 10,667 more rows
```

From the list, Pulp Fiction movie(1994), Forrest Gump (1994), and Silence of the Lambs(1991) are in the top three movies that had the highest ratings.

Let us see which genre had the highest number of ratings.

```
## Let's separate the genres and check which genre had the highest number of ratings.
#Note that this code may take some time as it separates the genres.

edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  select(genres, count) %>%
  data.frame() %>%
  ggplot(aes(x = reorder(genres, count), y = count)) +
  geom_bar(stat = "identity", colour = "black", fill = "grey30") +
  coord_flip() +
  labs( title = "Genre Ratings Distribution",
        x = "Genre",
        y = "Number of Ratings")
```



Drama had the highest ratings, while IMAX had the least ratings.

3. RESULTS

To develop a sound movie recommendation system, we'll build different algorithms, test them, and choose the one that gives the minimal error based on the root mean squared error (RMSE). As we have seen from the graphs above, there tends to be biasness on ratings based on the different types of movies, users, and genres. Therefore, the movie effect, user effect, and genre effect will be used to come up with different algorithms. We'll then introduce the regularization factor, which will apply to all three effects. From these models, we'll pick one to calculate the residuals to be used in the matrix factorization technique to develop a more robust algorithm.

We first split the edx dataset into a train and test set to train and test our models.

```
##We split the edx dataset into train and test set.

set.seed(1)

test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)

train_set <- edx[-test_index,]
test_set <- edx[test_index,]

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3.1 Model 1.

Using the same rating (mean) for all movies regardless of the user.

This is a straightforward model based on the assumption that all the movies and users have the same rating. The model is written as Y = mu + error_term.

```
set.seed(1)
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512482
```

```
# We can proceed to predict the ratings which are unknown and
# therefore we get the following RMSE

mu_rmse <- RMSE(test_set$rating, mu)
mu_rmse
```

```
## [1] 1.059904
```

```
rmse_results <- data.frame(Method = "Only average used", RMSE = mu_rmse)
rmse_results
```

```
##              Method     RMSE
## 1 Only average used 1.059904
```

We can see from our first model that the average rating is 3.512482 and the RMSE is 1.059904.

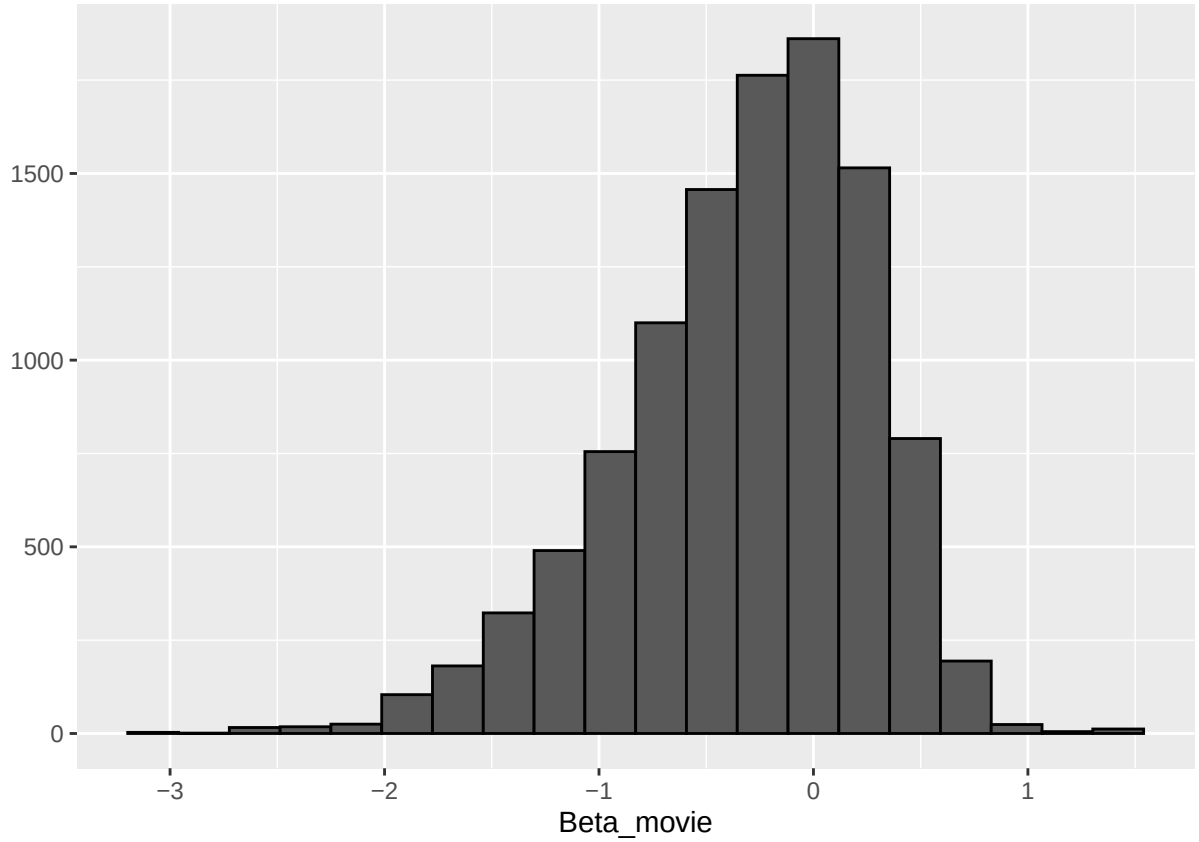
3.2 Model 2.

Modeling movie effects.

Based on the movie's rating distribution graph, we saw the bias in movie ratings. This is evident by how some movies have higher ratings than others. Therefore we shall add the term "Beta_movie" to indicate the average ranking for each movie i. Here we refer to "Beta_movie" as the "bias" term for the movie effect. We write the equation as Y = mu + Beta_movie + error_term, whereby Beta_movie is the average of Y - mu for each movie i.

```
#Beta_movie = mean(Y - mu)

mu <- mean(train_set$rating)
movies_averages <- train_set %>%
  group_by(movieId) %>%
  summarise(Beta_movie = mean(rating - mu))
```

```
# This graph show variation of the estimates.

movies_averages %>% qplot(Beta_movie, geom ="histogram", bins = 20, data = .,
                          color = I("black"))
```



This graph shows how the estimates have a significant variation.

```
#Let's find out if there's an improvement in our prediction.
```

```
set.seed(1)

second_prediction <- mu + test_set %>%
  left_join(movies_averages, by ='movieId') %>%
  pull(Beta_movie)

second_model <- RMSE(second_prediction, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Movie Effect Model",
                                     RMSE = second_model))
rmse_results
```

```
##                Method      RMSE
## 1  Only average used 1.0599043
## 2 Movie Effect Model 0.9437429
```

From this model, we can see a substantial decrease in the RMSE.

3.3 Model 3.

Modeling user effects.

We are now going to further our second model by adding a second term, which encompasses the users' variability in the movie ratings (Beta_user). The model can expressed as Y = mu + Beta_movie + Beta_user + error_term, whereby Beta_user is defined as the average of Y - mu - Beta_movie for each user u and movie i.

```
## Beta_user = mean(Y - mu - Beta_movie)
set.seed(1)

users_averages <- train_set %>%
  left_join(movies_averages, by ='movieId') %>%
  group_by(userId) %>%
  summarize(Beta_user = mean(rating - mu - Beta_movie))

third_prediction <- test_set %>%
  left_join(movies_averages, by = 'movieId') %>%
  left_join(users_averages , by = 'userId') %>%
  mutate(pred = mu + Beta_movie + Beta_user) %>%
  pull(pred)

third_model <- RMSE(third_prediction, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Movie + User Effect Model",
                                     RMSE = third_model))
rmse_results
```

```
##                       Method      RMSE
## 1         Only average used 1.0599043
## 2        Movie Effect Model 0.9437429
## 3 Movie + User Effect Model 0.8659320
```

There seems to be a good improvement when we combine the movie and user bias in the model. Let us now add the genre effect to the model and see how the model performs.

3.4 Model 4.

Modeling Genre effects.

We introduce Beta_genre to indicate the genre bias in our model. The model can be expressed as Y = mu + Beta_movie + Beta_user + Beta_genre + error_term. Beta_genre is the average Y - mu - Beta_movie - Beta_user for each user u, movie i, and genre g.

```
##Note that this code may take some time to complete.
set.seed(1)

genre_averages <- train_set %>% separate_rows(genres, sep = "\\|") %>%
  left_join(movies_averages, by = 'movieId') %>%
  left_join(users_averages, by = 'userId') %>%
  group_by(genres) %>%
  summarize(Beta_genre = mean(rating - mu - Beta_movie - Beta_user))


fourth_prediction <- test_set %>% separate_rows(genres, sep = "\\|") %>%
  left_join(movies_averages, by = 'movieId') %>%
  left_join(users_averages, by = 'userId') %>%
  left_join(genre_averages, by = 'genres') %>%
  mutate(pred = mu + Beta_movie + Beta_user + Beta_genre) %>%
  pull(pred)


fourth_model <- RMSE(fourth_prediction, test_set %>% separate_rows(genres, sep = "\\|")
                     %>% pull(rating))

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie + User + Genre Effect Model",
                                     RMSE = fourth_model))
rmse_results
```

```
##                             Method      RMSE
## 1                 Only average used 1.0599043
## 2                Movie Effect Model 0.9437429
## 3         Movie + User Effect Model 0.8659320
## 4 Movie + User + Genre Effect Model 0.8641894
```

The model shows a slight improvement from the earlier model.

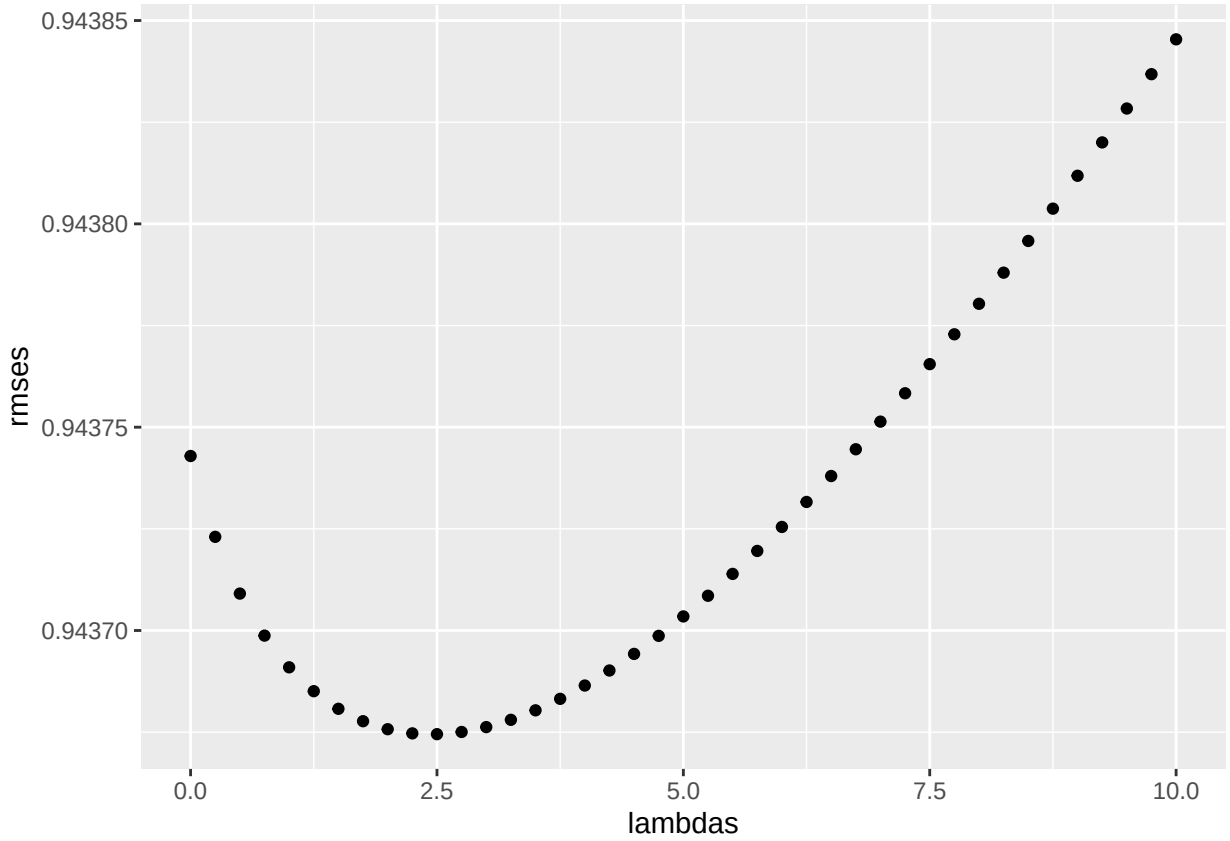
3.5 Regularization.

Regularized Movie Effect Model.

Regularization is used to help prevent overfitting by reducing the beta parameters, making the model simple. This method uses the lambda parameter, which is tuned to find the one that gives the best performance.

Therefore, we will apply this technique to the movie effect model, but first, we have to choose a suitable lambda for this algorithm.

```
##Regularization on Movie Effect
# Let's compute the regularized estimates of Beta_movie.
#We choose the best lambda for our model

##This shows how the estimates shrinks
library(dplyr, warn.conflicts = FALSE)
options(dplyr.summarise.inform = FALSE)

set.seed(1)

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(lam){
  mu <- mean(train_set$rating)

  Beta_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(Beta_movie = sum(rating - mu)/(n()+lam))

  fifth_model <- test_set %>%
    left_join(Beta_movie, by = "movieId") %>%
    mutate(pred = mu + Beta_movie) %>%
    pull(pred)

  return(RMSE(fifth_model, test_set$rating))

})

qplot(lambdas, rmses)
```




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

```
## [1] 2.5
```

The model is optimized when lambda is 2.5.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Regularized Movie Effect Model",
                                     RMSE = min(rmses)))
```

```
## 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.
```

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

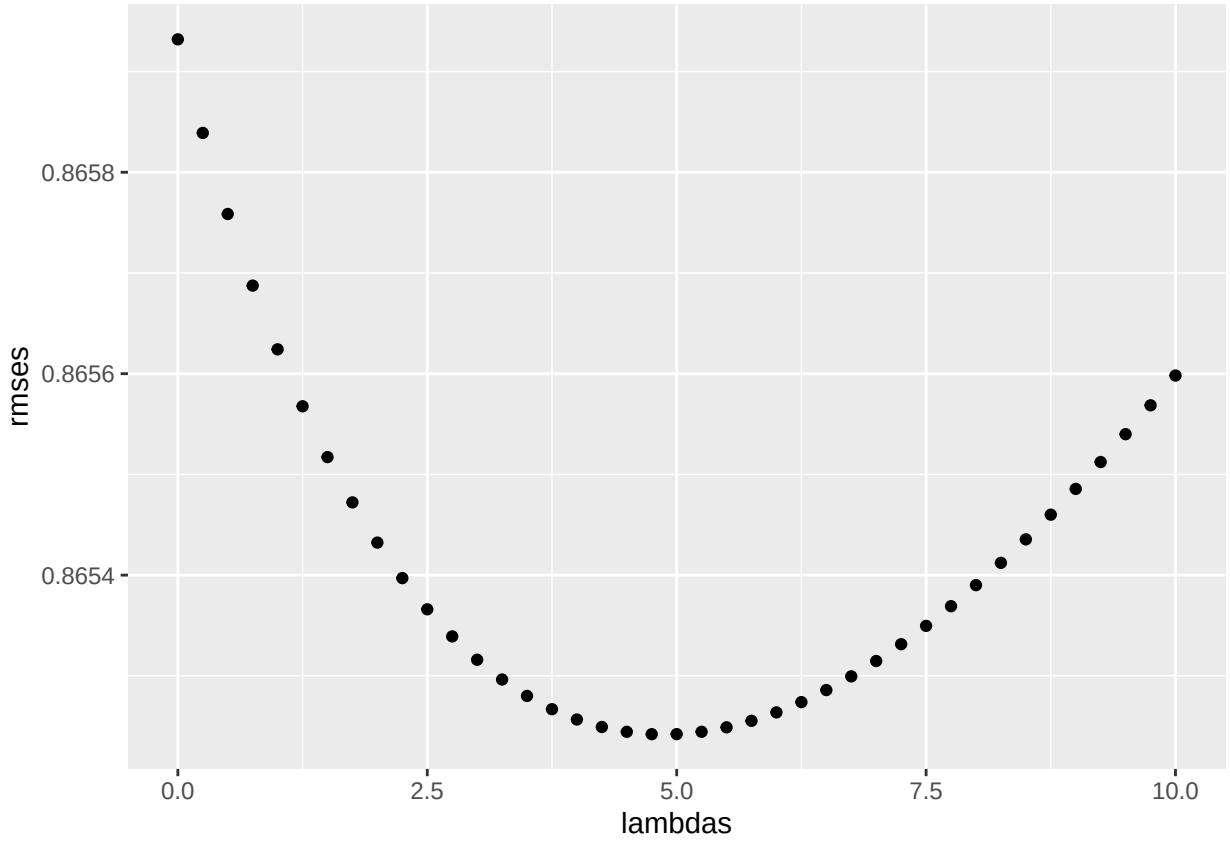
Method	RMSE
Only average used	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genre Effect Model	0.8641894
Regularized Movie Effect Model	0.9436745

There seems to be no much improvement of the RMSE from the model.

Regularized Movie + User Effect Model.

We now proceed to apply regularization on both the movie and user effects. We also have to check the best lambda to suit our model.

```
##Regularization on Movie and User Effect
# Let's compute the regularized estimates of Beta_user.
#We choose the best lambda for our model
set.seed(1)

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(lam){
  mu <- mean(train_set$rating)

  Beta_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(Beta_movie = sum(rating - mu)/(n()+lam)) %>%
    ungroup()

  Beta_user <- train_set %>%
    left_join(Beta_movie, by = "movieId") %>%
    group_by(userId) %>%
    summarize(Beta_user = sum(rating - Beta_movie - mu)/(n()+lam)) %>%
    ungroup()

  sixth_model <-
    test_set %>%
    left_join(Beta_movie, by = "movieId") %>%
    left_join(Beta_user, by = "userId") %>%
    mutate(pred = mu + Beta_movie + Beta_user) %>%
    pull(pred)

  return(RMSE(sixth_model, test_set$rating))
})

## The lambda that the model chooses
qplot(lambdas, rmses)
```




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

```
## [1] 4.75
```

The model is optimized when lambda is 4.75.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

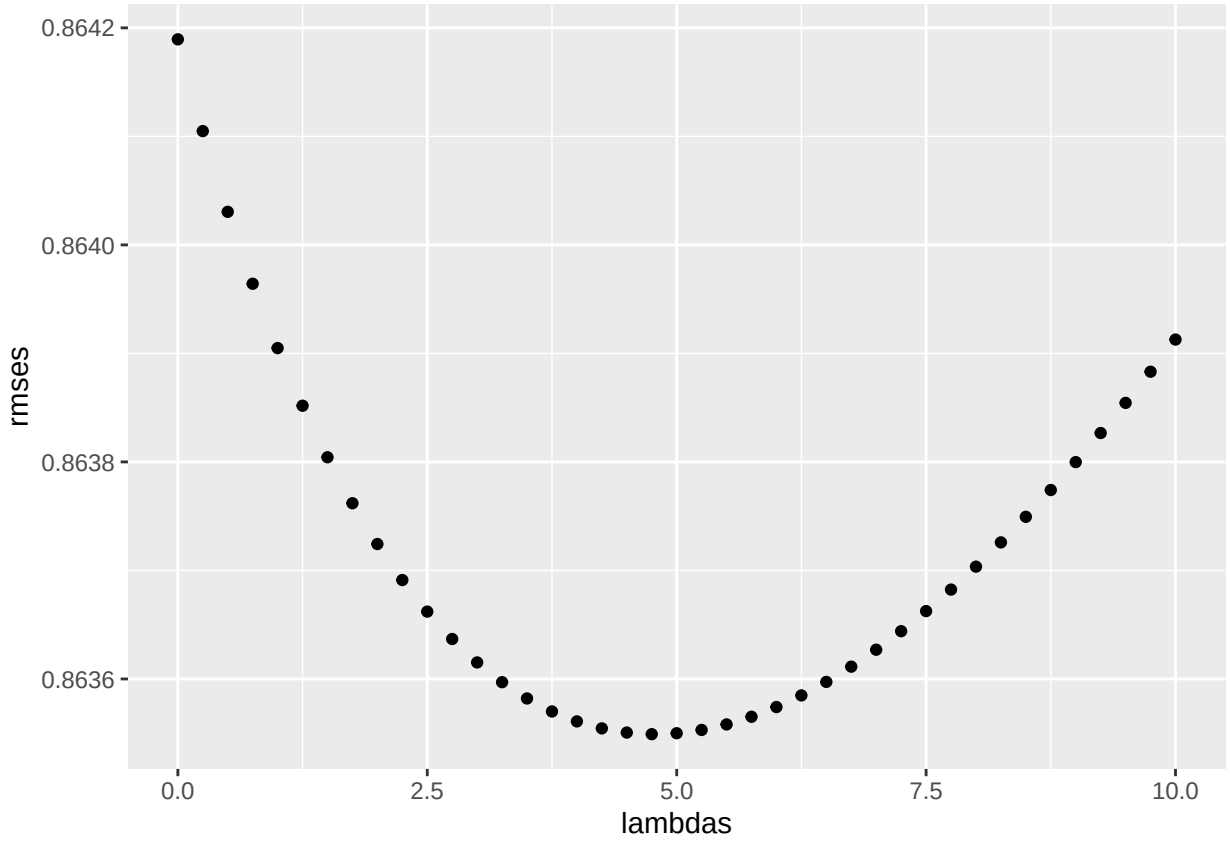
Method	RMSE
Only average used	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genre Effect Model	0.8641894
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421

Still, there is a small improvement in our model.

Regularized Movie + User + Genre Effect Model.

We repeat the previous procedure but now adding the genre effect.

```
#Regularization on Movie, User and Genre Effect
# Let's compute the regularized estimates of Beta_genre.
#We choose the best lambda for our model
set.seed(1)

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(lam){
  mu <- mean(train_set$rating)

  Beta_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(Beta_movie = sum(rating - mu)/(n()+lam)) %>%
    ungroup()

  Beta_user <- train_set %>%
    left_join(Beta_movie, by = "movieId") %>%
    group_by(userId) %>%
    summarize(Beta_user = sum(rating - Beta_movie - mu)/(n()+lam)) %>%
    ungroup()

  Beta_genre <- train_set %>% separate_rows(genres, sep = "\\|") %>%
    left_join(movies_averages, by = 'movieId') %>%
    left_join(users_averages, by = 'userId') %>%
    group_by(genres) %>%
    summarize(Beta_genre = sum(rating - mu - Beta_movie - Beta_user)/(n() + lam))


  seventh_model <- test_set %>% separate_rows(genres, sep = "\\|") %>%
    left_join(Beta_movie, by = "movieId") %>%
    left_join(Beta_user, by = "userId") %>%
    left_join(Beta_genre, by = 'genres') %>%
    mutate(pred = mu + Beta_movie + Beta_user + Beta_genre) %>%
    pull(pred)

  return(RMSE(seventh_model, test_set %>% separate_rows(genres, sep = "\\|") %>% pull(rating)))

})

##The lambda that the model chooses

qplot(lambdas, rmses)
```




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

```
## [1] 4.75
```

The model is optimized when lambda is 4.75.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User + Genre Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Only average used	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genre Effect Model	0.8641894
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genre Effect Model	0.8635491

We can see a minor improvement in the RMSE as well.

From the three models that we apply the regularization technique, we can notice very slight improvements in their RMSEs.

3.6 Matrix Factorization.

The matrix factorization algorithm is used in recommender systems. This technique operates by the decomposition of the user-item interaction matrix into two lower dimensionality rectangular matrices. From our earlier model: Y = mu + The movie to movie variation was accounted for by Beta-movie, while the user to user variation was accounted for by Beta-user. Nonetheless, a significant variation source is left because groups of movies and groups of users tend to have similar rating patterns. We are going to discover these patterns by examining the residuals. We, therefore, calculate the residuals using the formula: Residuals = Y – Beta Next, we then use the "recosystem" package to help us execute the matrix factorization. A better understanding of this package can be found here [https://cran.microsoft.com/web/packages/recosystem/recosystem.pdf] (https://cran.microsoft.com/web/packages/recosystem/recosystem.pdf)

```
##Using User and Movie Effect only

set.seed(1)


lambda <- 4.75

mu <- mean(train_set$rating)

Beta_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(Beta_movie = sum(rating - mu)/(n()+lambda)) %>%
  ungroup()

Beta_user <- train_set %>%
  left_join(Beta_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(Beta_user = sum(rating - Beta_movie - mu)/(n()+lambda)) %>%
  ungroup()


predicted_ratings <- test_set %>%
  left_join(Beta_movie, by = "movieId") %>%
  left_join(Beta_user, by = "userId") %>%
  mutate(pred = mu + Beta_movie + Beta_user) %>%
  pull(pred)

residual <- train_set %>%
  left_join(Beta_movie, by = "movieId") %>%
  left_join(Beta_user, by = "userId") %>%
  mutate(residual = rating - mu - Beta_movie - Beta_user) %>%
  select(userId, movieId, residual)


train_set_residual_matrix <- as.matrix(residual)

test_set_matrix <- test_set %>%
  select(userId, movieId, rating) %>% as.matrix(.)
```

```r
write.table(train_set_residual_matrix , file = "trainset.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

write.table(test_set_matrix, file = "testset.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)


# use data_file() to specify a data set from a file in the hard disk.
set.seed(1)
train_set2 <- data_file("trainset.txt")
test_set2 <- data_file("testset.txt")

# build a recommender object
r <-Reco()

# tuning training set
opts <- r$tune(train_set2, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.8019026
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8174094
## 2   20        0     0.01        0     0.01   0.1 0.8285773
## 3   30        0     0.01        0     0.01   0.1 0.8388422
## 4   10        0     0.10        0     0.01   0.1 0.8139644
## 5   20        0     0.10        0     0.01   0.1 0.8151143
## 6   30        0     0.10        0     0.01   0.1 0.8158853
## 7   10        0     0.01        0     0.10   0.1 0.8088015
```

```
## 8   20        0     0.01        0     0.10   0.1 0.8026818
## 9   30        0     0.01        0     0.10   0.1 0.8019026
## 10  10        0     0.10        0     0.10   0.1 0.8257965
## 11  20        0     0.10        0     0.10   0.1 0.8239280
## 12  30        0     0.10        0     0.10   0.1 0.8238978
## 13  10        0     0.01        0     0.01   0.2 0.8230581
## 14  20        0     0.01        0     0.01   0.2 0.8433125
## 15  30        0     0.01        0     0.01   0.2 0.8600961
## 16  10        0     0.10        0     0.01   0.2 0.8150942
## 17  20        0     0.10        0     0.01   0.2 0.8191425
## 18  30        0     0.10        0     0.01   0.2 0.8229883
## 19  10        0     0.01        0     0.10   0.2 0.8100260
## 20  20        0     0.01        0     0.10   0.2 0.8082710
## 21  30        0     0.01        0     0.10   0.2 0.8099759
## 22  10        0     0.10        0     0.10   0.2 0.8218774
## 23  20        0     0.10        0     0.10   0.2 0.8215379
## 24  30        0     0.10        0     0.10   0.2 0.8216259
```

```
# training the recommender model
r$train(train_set2, opts = c(opts$min, nthread = 1, niter = 50))
```

```
# Making prediction on validation set and calculating RMSE:

predictions_file <- tempfile()
r$predict(test_set2, out_file(predictions_file))
```

```
## prediction output generated at C:\Users\HP\AppData\Local\Temp\RtmpYdDQ3k\file34803e5b2198
```

```
mf_forecasted_residuals <- scan(predictions_file)
mf_forecasted_ratings <- predicted_ratings + mf_forecasted_residuals
rmse_mf <- RMSE(mf_forecasted_ratings, test_set %>% pull(rating))

rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Matrix Factorization",
                                     RMSE = rmse_mf))
rmse_results %>% knitr::kable(caption = "Algorithms' Accuracy")
```

Table 6: Algorithms' Accuracy

Method	RMSE
Only average used	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genre Effect Model	0.8641894
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genre Effect Model	0.8635491
Matrix Factorization	0.8017719

There is a significant improvement in the RMSE.

3.7 Applying Matrix Factorization On The Validation Dataset.

Based on the above models that we have trained and tested using the edx dataset (which we had split into a train and test set), we finally choose the matrix factorization algorithm since it has the least RMSE. Therefore, we proceed with this algorithm and apply it to the validation set, the final hold-out test set.

```
library(recosystem)
set.seed(1)


lambda <- 4.75

mu <- mean(edx$rating)

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

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


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

residual <- edx %>%
  left_join(Beta_movie, by = "movieId") %>%
  left_join(Beta_user, by = "userId") %>%
  mutate(residual = rating - mu - Beta_movie - Beta_user) %>%
  select(userId, movieId, residual)


edx_set_residual_matrix <- as.matrix(residual)

validation_set_matrix <- validation %>%
  select(userId, movieId, rating) %>% as.matrix(.)

write.table(edx_set_residual_matrix , file = "edx.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

write.table(validation_set_matrix, file = "validation.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)


# use data_file() to specify a data set from a file in the hard disk.
set.seed(1)
edx_set <- data_file("edx.txt")
```

```
validation_set <- data_file("validation.txt")

# build a recommender object
r <-Reco()

# tuning training set
opts <- r$tune(edx_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                    costp_l1 = 0, costq_l1 = 0,
                                    nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7936467
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8062166
## 2   20        0     0.01        0     0.01   0.1 0.8117589
## 3   30        0     0.01        0     0.01   0.1 0.8204536
## 4   10        0     0.10        0     0.01   0.1 0.8050304
## 5   20        0     0.10        0     0.01   0.1 0.8020261
## 6   30        0     0.10        0     0.01   0.1 0.8026296
## 7   10        0     0.01        0     0.10   0.1 0.8040986
## 8   20        0     0.01        0     0.10   0.1 0.7964274
## 9   30        0     0.01        0     0.10   0.1 0.7936467
## 10  10        0     0.10        0     0.10   0.1 0.8245663
## 11  20        0     0.10        0     0.10   0.1 0.8232722
## 12  30        0     0.10        0     0.10   0.1 0.8227006
## 13  10        0     0.01        0     0.01   0.2 0.8122205
## 14  20        0     0.01        0     0.01   0.2 0.8237094
## 15  30        0     0.01        0     0.01   0.2 0.8376700
## 16  10        0     0.10        0     0.01   0.2 0.8059328
## 17  20        0     0.10        0     0.01   0.2 0.8063349
```

```
## 18  30        0     0.10        0     0.01   0.2 0.8087048
## 19  10        0     0.01        0     0.10   0.2 0.8023718
## 20  20        0     0.01        0     0.10   0.2 0.7994189
## 21  30        0     0.01        0     0.10   0.2 0.7998097
## 22  10        0     0.10        0     0.10   0.2 0.8220801
## 23  20        0     0.10        0     0.10   0.2 0.8223565
## 24  30        0     0.10        0     0.10   0.2 0.8210572
```

```
# training the recommender model
r$train(edx_set, opts = c(opts$min, nthread = 1, niter = 50))
```

```
# Making prediction on validation set and calculating RMSE:

predictions_file <- tempfile()
r$predict(validation_set, out_file(predictions_file))
```

```
## prediction output generated at C:\Users\HP\AppData\Local\Temp\RtmpYdDQ3k\file34806a84ed
```

```
mf_forecasted_residuals <- scan(predictions_file)
mf_forecasted_ratings <- predicted_ratings + mf_forecasted_residuals
rmse_mf <- RMSE(mf_forecasted_ratings, validation %>% pull(rating))

final_rmse_results <- data.frame(Method = "Matrix Factorization", RMSE = rmse_mf)

final_rmse_results %>% knitr::kable(caption = "Matrix Factorization Results")
```

Table 7: Matrix Factorization Results

Method	RMSE
Matrix Factorization	0.7884114

4. CONCLUSION

From the models used, it can be seen that the "only average used" model had the largest RMSE making it the least suitable model for the recommendation system. We can also see that regularization made only slight changes to the previous models. Therefore, we can confidently conclude that the matrix factorization algorithm achieved the best results by having the least RMSE. Finally, the methods applied to the dataset might not be optimal. Therefore, other recommendation techniques, such as using the "recommenderlab" package, could be tested and compared to the matrix factorization method.

5. REFERENCES

Irizarry, Rafael A. Introduction to Data Science: Data Analysis and Prediction Algorithms with R. CRC Press, 2019.

Gorakala, Suresh K., and Michele Usuelli. Building a recommendation system with R. Packt Publishing Ltd, 2015.

Subramaniyaswamy, V., et al. "A personalised movie recommendation system based on collaborative filtering." International Journal of High Performance Computing and Networking 10.1-2 (2017): 54-63.