Movie Recommendation System | Machine Learning Project

Elton Costa

2/22/2022

1. Introduction

Recommendation systems are important for e-commerce and online streaming services. Making the right recommendation for the next product, music or movie increases user activation and engagement, leading to sales and profit growth. Companies competing for customer loyalty invest on systems that capture and analyses the user's preferences, and offer products or services with higher likelihood of purchase.

The goal of this final project is to construct a movie recommendation system using Machine Learning. The steps will include data pre-processing, exploratory analysis, and then to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

This document is structured as follows:

2. Method and Data pre-processing

2.1. MovieLens Dataset

The data was collected and provided by GroupLens. A research lab at the University of Minnesota that specializes in recommendation systems, online communities, mobile and ubiquitous technologies, digital libraries and local geographic information systems. They have collected millions of movie reviews and offer these data sets in a range of sizes. For this project the 10M version will be used. It contains 10 million ratings on 10,000 movies by 72,000 users. It was released in 2009.

https://grouplens.org/datasets/movielens/10m/

http://files.grouplens.org/datasets/movielens/ml-10m.zip

2.2. Data Loading

In this section we download and prepare the dataset to be used in the analysis. We split the dataset in two parts, the training set called edx and the evaluation set called validation with 90% and 10% of the original dataset respectively.

Then, we split the edx set in two parts, the train set and test set with 90% and 10% of edx set respectively. The model is created and trained in the train set and tested in the test set until the RMSE target is achieved, then finally we train the model again in the entire edx set and validate in the validation set. The name of this method is cross-validation.

```r
# 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")

library(tidyverse)
library(caret)
library(data.table)

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

dl <- tempfile()
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

# Validation set will be 10% of MovieLens data
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. Training and Testing of Algorithm

The edx set is used for training and testing, and the validation set is used for final validation to simulate the new data.

Here, we split the edx set in 2 parts: the training set and the test set.

The model building is done in the training set, and the test set is used to test the model. When the model is complete, we use the validation set to calculate the final RMSE. We use the same procedure used to create edx and validation sets.

The training set will be 90% of edx data and the test set will be the remaining 10%.

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

#matches userId and movieId in both train and test sets
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

#adding back rows to the train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

3. Exploratory Analysis

Before start building the model, we need to understand the structure of the data, the distribution of ratings and the relationship of the predictors. This information will help build a better model.

Below is a snapshot of the data structure of edx database. There are 9,000,055 observations and 6 columns. Each observation represents a rating given by one user for one movie. Columns include userId, movieId, rating, timestamp, title and genres. Timestamps represent seconds since midnight UTC January 1, 1970.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000061 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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  - attr(*, ".internal.selfref")=<externalptr>
```

The summary function provides a statistical summary of the data.

```
edx %>% select(-genres) %>% summary()
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title
##  Length:9000061
##  Class :character
```

```
##  Mode  :character
##
##
##
```

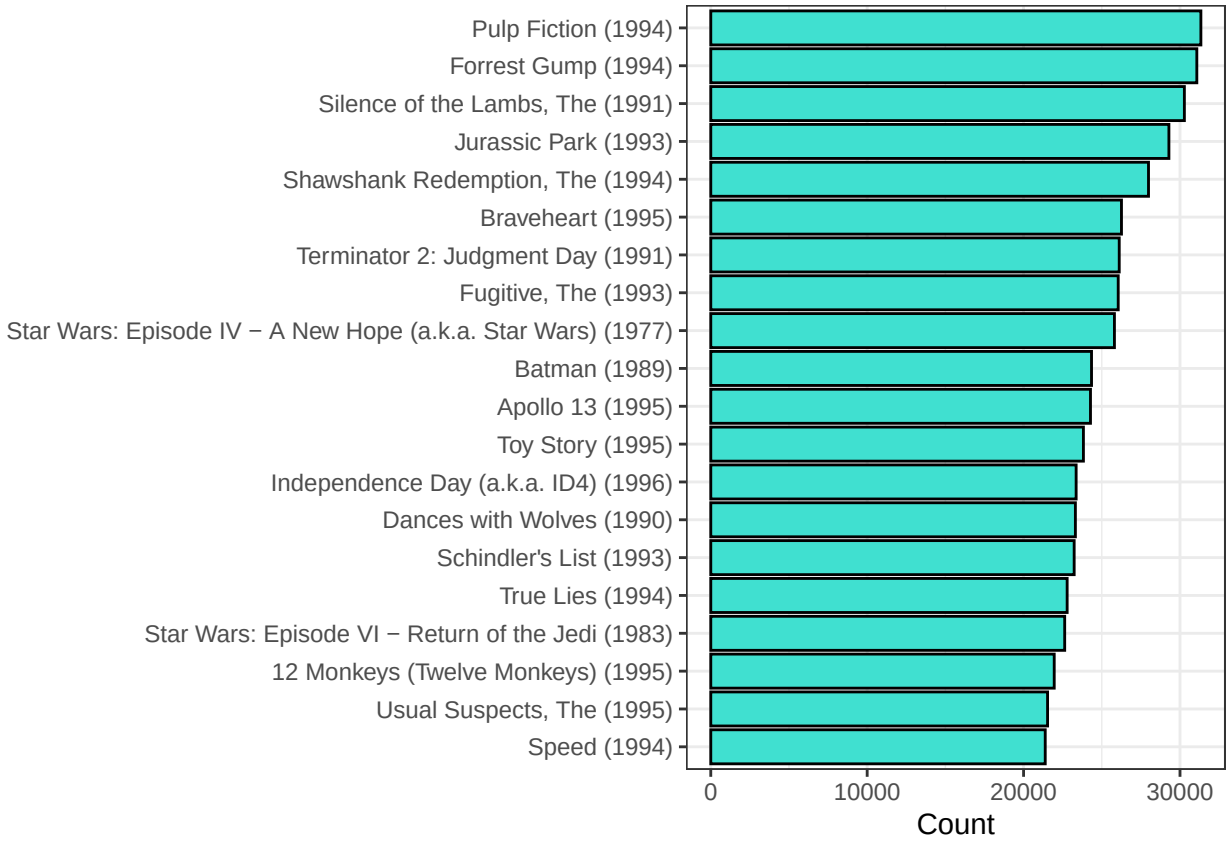
There are 69,878 unique users and 10,677 movies.

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

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

The most popular movies in this dataset.

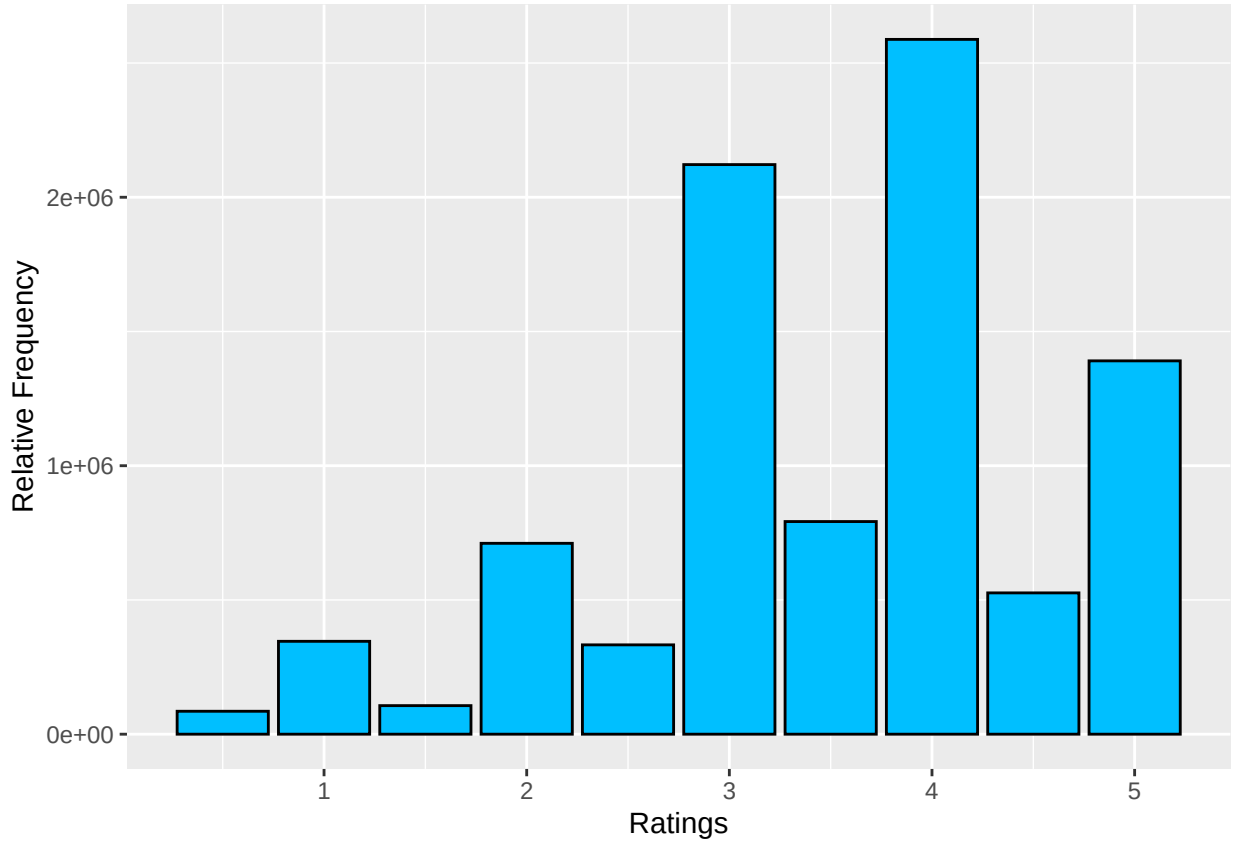
```
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(20, count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(color="black", fill = "turquoise", stat = "identity")+
  xlab("Count")+
  ylab(NULL)+
  theme_bw()
```



The distribution of the movie ratings shows a range of 0.5 to 5 with whole numbers used more often.

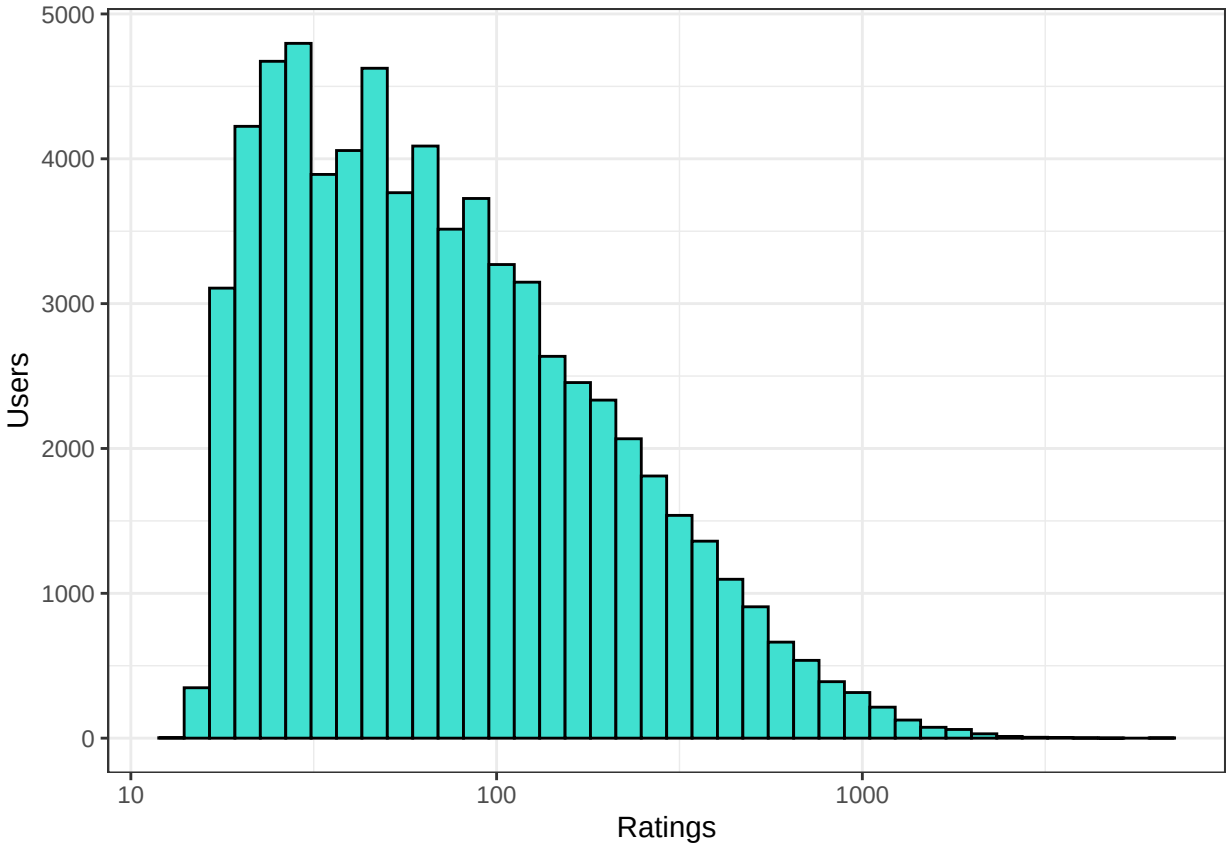
```
edx %>%
  group_by(rating) %>%
```

```
  summarize(count = n())%>%
  ggplot(aes(rating,count))+
  geom_bar(color="black", fill = "deepskyblue", stat = "identity")+
  xlab("Ratings")+
  ylab("Relative Frequency")
```



The distribution of numbers of users by numbers of ratings show right skew in the distribution

```
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  ggplot(aes(count))+
  geom_histogram(color="black", fill = "turquoise", bins = 40)+
  xlab("Ratings")+
  ylab("Users")+
  scale_x_log10()+
  theme_bw()
```



4. Data Cleaning

Several features can be used to predict the rating for a given user. However, many predictors increases the model complexity and requires more computer resources, so in this research the estimated rating uses only movie and user information.

```
train_set <- train_set %>% select(userId, movieId, rating, title)
test_set  <- test_set  %>% select(userId, movieId, rating, title)
```

5. Model Evaluation

In this project we will assess several models and the accuracy of each will be evaluated using the residual mean squared error (RMSE):

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

N is defined as the number of user/movie combinations, $y_{u,i}$ as the rating for movie i by user u with the prediction as $\hat{y}_{u,i}$. The RMSE is a commonly used loss function that simply measures the differences between predicted and observed values. It can be interpreted similarly to a standard deviation. For this exercise if the number is larger than 1 it means our typical error is larger than one star. The goal is to reduce this error below 0.8649. Accuracies will be compared across all models using the code below.

```
RMSE <- function(actual_ratings, predicted_ratings){
  sqrt(mean((actual_ratings - predicted_ratings)^2))
}
```

6. Model Results

This section presents the code and results of the models.

6.1 Model 1

The first model is the simplest approach we could take in making a prediction by only using the average of all movie ratings. With differences explained by random variation.

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

$\epsilon_{u,i}$ is defined as the independent sample errors and μ the true rating for all movies. Statistical theory tells us that the estimate that minimizes the RMSE is the least squares estimate of μ. For this exercise it is the average of all ratings.

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512354
```

If we base our prediction using this mean we obtain the following RMSE:

```
mean_rmse <- RMSE(test_set$rating, mu_hat)
results <- tibble(Method = "Model (1) Simply Mean", RMSE = mean_rmse)
results %>% knitr::kable()
```

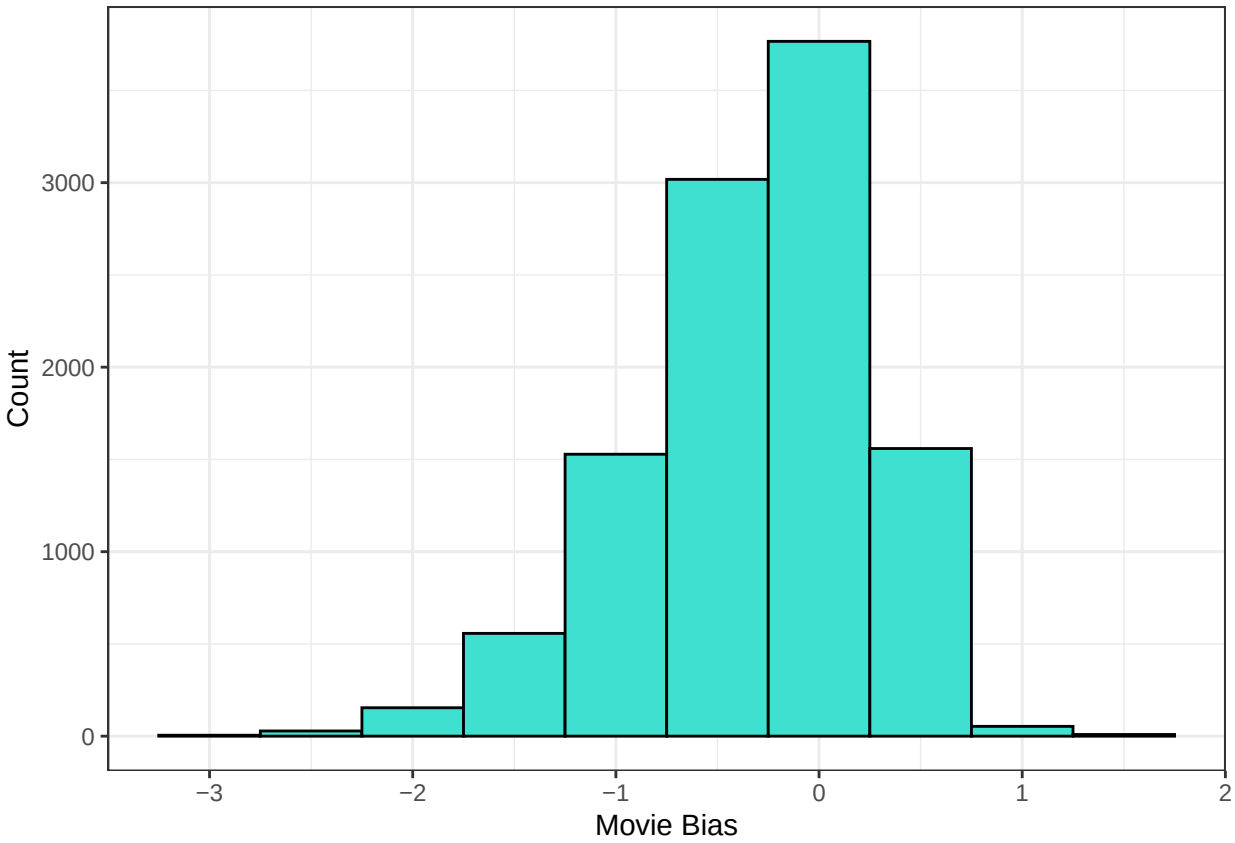
Method	RMSE
Model (1) Simply Mean	1.059

6.2 Model 2

Improve first model by taking into account movie bias i.e. popular movies receive higher ratings. We can prove there is bias by analyzing the distribution of ratings vs total.

```
bias_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bim = mean(rating-mu_hat))
```

```
bias_movie %>% ggplot(aes(bim))+
  geom_histogram(color="black", fill = "turquoise", bins = 10)+
  xlab("Movie Bias")+
  ylab("Count")+
  theme_bw()
```



Here is the impact of adding this bias to our model by running a RMSE test:

```
predict_ratings <- mu_hat + test_set %>%
  left_join(bias_movie, by="movieId") %>%
  pull(bim)

bias_movie_rmse <- RMSE(test_set$rating, predict_ratings)
results <- bind_rows(results, tibble(Method = "Model (2) Mean + Movie bias", RMSE =bias_movie_rmse))
results %>% knitr::kable()
```

Method	RMSE
Model (1) Simply Mean	1.0590002
Model (2) Mean + Movie bias	0.9426564

6.3 Model 3

This model improves the first model by taking into account movie bias and the user bias i.e. popular movies receive higher ratings.

```
bias_user <- train_set %>%
  left_join(bias_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(biu = mean(rating - mu_hat - bim))
```

A RMSE test will show how much we can reduce our typical error by adding this bias:

```
predict_ratings <- test_set %>%
  left_join(bias_movie, by = "movieId") %>%
  left_join(bias_user, by = "userId") %>%
  mutate(pred = mu_hat + bim + biu) %>%
  pull(pred)

bias_movie_user_rmse <- RMSE(test_set$rating, predict_ratings)
results <- bind_rows(results, tibble(Method = "Model (3) Mean + Movie bias + User bias", RMSE = bias_mov
results %>% knitr::kable()
```

Method	RMSE
Model (1) Simply Mean	1.0590002
Model (2) Mean + Movie bias	0.9426564
Model (3) Mean + Movie bias + User bias	0.8646047

6.4 Model 4

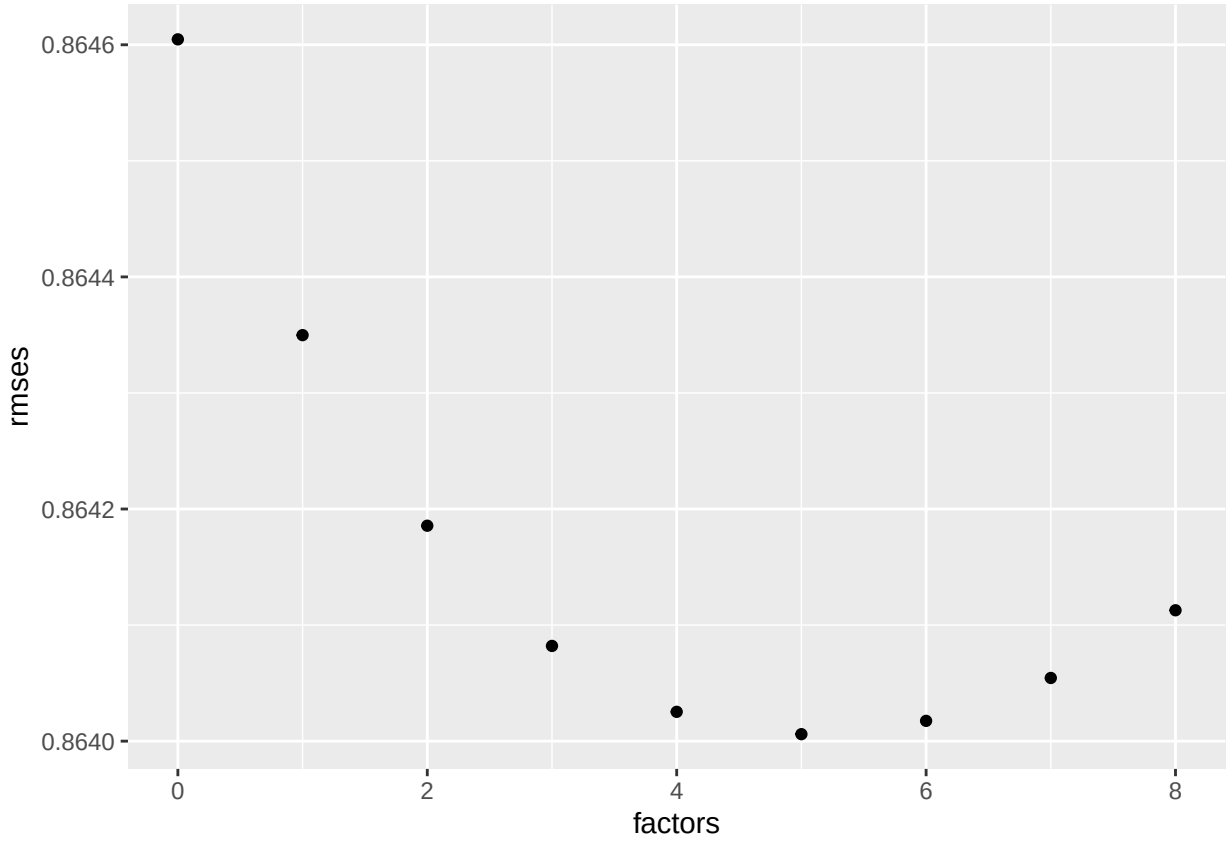
Some movies are rated by a few users and others by many adding variability and increasing RMSE. This proves that the model could benefit from regularization.

```
factors <- seq(0,8,1)

rmses <- sapply(factors, function(x){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+x))
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+x))
  predict_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
  return(RMSE(test_set$rating, predict_ratings))
})
```

Plot that shows a range of lambdas VS RMSE. The optimal setting provides the lowest error.

```
qplot(factors, rmses)
```



The factor with smallest RMSE is:

```
factor <- factors[which.min(rmses)]
factor
```

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

RMSE result:

```
results <- bind_rows(results, tibble(Method = "Model (4) Regularized Movie & User effects", RMSE = min(
results %>% knitr::kable()
```

Method	RMSE
Model (1) Simply Mean	1.0590002
Model (2) Mean + Movie bias	0.9426564
Model (3) Mean + Movie bias + User bias	0.8646047
Model (4) Regularized Movie & User effects	0.8640060

The RMSE has improved however it is a very small gain in accuracy. It will take a different approach to improve it significantly.

6.5 Model 5

Recommender systems use historical data to make predictions. One of the most popular approaches in achieving this is collaborative filtering. It is based on historical behavior by its users. So far we have approached a dataset that features sparsity and biases with models that account for these effects with decent accuracy. To get better results we turn to a more advanced method called matrix factorization. Our user

data is processed as a large and sparse matrix, then decomposed into two smaller dimensional matrices with latent features and less sparsity. To make the process more efficient the recosystem package will be used. For more information on it click here. We start by converting data into the recosystem format, find the best tuning parameters, train and finally test it.

```
library(recosystem)
set.seed(1)
train_rec <- with(train_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_rec <- with(test_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
rec_model <- Reco()

parameters <- rec_model$tune(train_rec, opts = list(dim = c(20,30),
                                                    costp_l2 = c(0.01, 0.1),
                                                    costq_l2 = c(0.01, 0.1),
                                                    lrate = c(0.01, 0.1),
                                                    nthread = 4,
                                                    niter = 10))

rec_model$train(train_rec, opts = c(parameters$min, nthread = 4, niter =30))
results_rec <- rec_model$predict(test_rec, out_memory())
```

With the algorithm trained we now test it to see the resulting RMSE:

```
factorization_rmse <- RMSE(test_set$rating, results_rec)
results <- bind_rows(results, tibble(Method = "Model (5) Matrix Factorization with Ricosystem", RMSE = f
results %>% knitr::kable()
```

Method	RMSE
Model (1) Simply Mean	1.0590002
Model (2) Mean + Movie bias	0.9426564
Model (3) Mean + Movie bias + User bias	0.8646047
Model (4) Regularized Movie & User effects	0.8640060
Model (5) Matrix Factorization with Ricosystem	0.7845607

The improvement is significant and therefore we will proceed with model 5 to our final validation.

7. Final Validation

Having found our model with the lowest RMSE using matrix factorization, the final step is to train it using the edx set and then test its accuracy on the validation set:

```
set.seed(1)
edx_reco <- with (edx, data_memory(user_index = userId, item_index = movieId, rating = rating))
validation_reco <- with(validation, data_memory(user_index = userId, item_index = movieId, rating = rati
r <- Reco()

para_reco <- r$tune(edx_reco, opts = list(dim = c(20,30),
                                          costp_l2 = c(0.01, 0.1),
                                          costq_l2 = c(0.01, 0.1),
                                          lrate = c(0.01, 0.1),
                                          nthread = 4,
                                          niter = 10))

r$train(edx_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
```

```
final_reco <- r$predict(validation_reco, out_memory())
```

The final RMSE result using the validation set is shown below:

```
final_rmse <- RMSE(validation$rating, final_reco)
results <- bind_rows(results, tibble(Method = "Final validation: Matrix factorization with Recosystem",
results %>% knitr::kable()
```

Method	RMSE
Model (1) Simply Mean	1.0590002
Model (2) Mean + Movie bias	0.9426564
Model (3) Mean + Movie bias + User bias	0.8646047
Model (4) Regularized Movie & User effects	0.8640060
Model (5) Matrix Factorization with Ricosystem	0.7845607
Final validation: Matrix factorization with Recosystem	0.7815331

8. Conclusion

We built and tested several models and achieved our best accuracy using matrix factorization which was simplified through the recosystem package. The model development started the linear model with a very simple model which is just the mean of the observed ratings. From there, we added movie and user effects, that models the user behavior and movie distribution. With regularization we added a penalty value for the movies and users with few number of ratings. Finally, we evaluated the matrix factorization with the recosystem package.

The final RMSE for the validation set is 0.7840. This is significantly below the target of 0.8649 for this project. Therefore the model is matrix factorization.