HarvardX: PH125.9x Data Science: Capstone. MovieLens Prediction Project

Carlos Mayora

April 29, 2020

Contents

1 Introduction

Recommendation systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

Recommendation systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. Recommendation systems have also proved to improve decision making process and quality. In e-commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. In October 2006, Netflix offered a challenge to the data science community to improve their recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced.

The aim of this project is to create a movie recommendation system using the MovieLens dataset from GroupLens research lab, training a machine learning algotithm using the inputs in one subset to predict movie ratings in the validation set.

1.1 MovieLens Dataset

GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

GroupLens Research has collected and made available rating data sets from the MovieLens web site with 25 million ratings and one million tag applications applied to 62,000 movies by 162,000 users. This project is based in a subset of this dataset with 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000.

The 10 million dataset can be found:

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

2 Data Analysis

2.1 Data Ingestion

We will download and prepare the dataset to be used in the analysis using the code provided in the edx HarvardX: PH125.9x Data Science: Capstone course - Create Train and Validation Sets.

First we download and split the 10M MovieLens dataset with 90% for the training set, called *edx* and 10% for the evaluation set, called *validation*. We will use *edx* subset for algorithm training as *validation* subset will be used only for evaluating the RMSE of the final algorithm.

```
################################
# 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
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)
```

2.2 Data Exploration

Before start building the model, we need to get familiar and understand the data structure of the dataset in order to build a better model. First let's get the first 6 rows of the *edx* subset.

```
#6 first rows of edx dataset including column names
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
```

We can see that *edx* dataset has 6 columns:

- userId: user identifier.
- movieId: movie identifier.
- title: contains the movie name.
- rating: this is the rating given by the user to the movie, this is the column we will evaluate.
- timestamp: the date when the rating was given, it is in timestamp format which is the total seconds since January 1st, 1970 at UTC to the date the rating was given.
- genres: pipe-separated list containing all the genres for the movie.

We can see that the dataset is in the tidy format ready for exploration and analysis, where each row represents a rating given by one user to one movie and the column names are the features.

From the summary we can see that the lowest rating is 0.5 and the highest is 5.0.

```
# Basic summary statistics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
```

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

So just to confirm how many rows and columns are in the *edx* dataset:

```
# edx rows and columns
dim(edx)
```

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

We can compare dimmension against *validation* dataset:

```
# validation rows and columns
dim(validation)
```

```
## [1] 999999      6
```

We can see the number of unique users that provided ratings and how many unique movies were rated in the *edx* dataset:

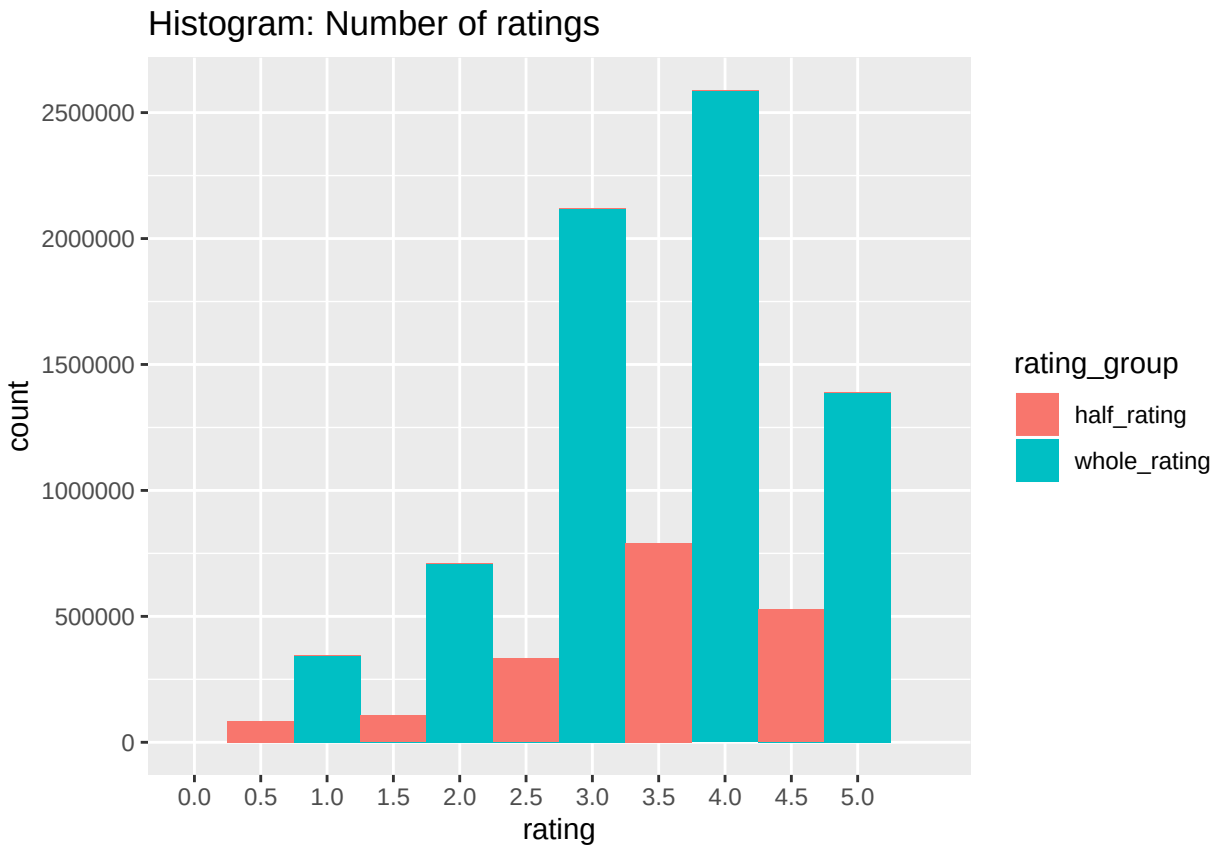
```
# Number of unique users and movies in the edx dataset
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

2.2.1 Ratings

Now let's explore the ratings in the *edx* dataset, let's use a histogram showing the count of ratings per each rating value:

```
# Histogram: Number of ratings
edx %>% mutate(rating_group = ifelse(rating %in% c(1,2,3,4,5), "whole_rating", "half_rating")) %>%
  ggplot(aes(x=rating, fill=rating_group)) +
  geom_histogram(binwidth = 0.5) +
  scale_x_discrete(limits = c(seq(0, 5, 0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  xlab("rating") +
  ylab("count") +
  ggtitle("Histogram: Number of ratings")
```



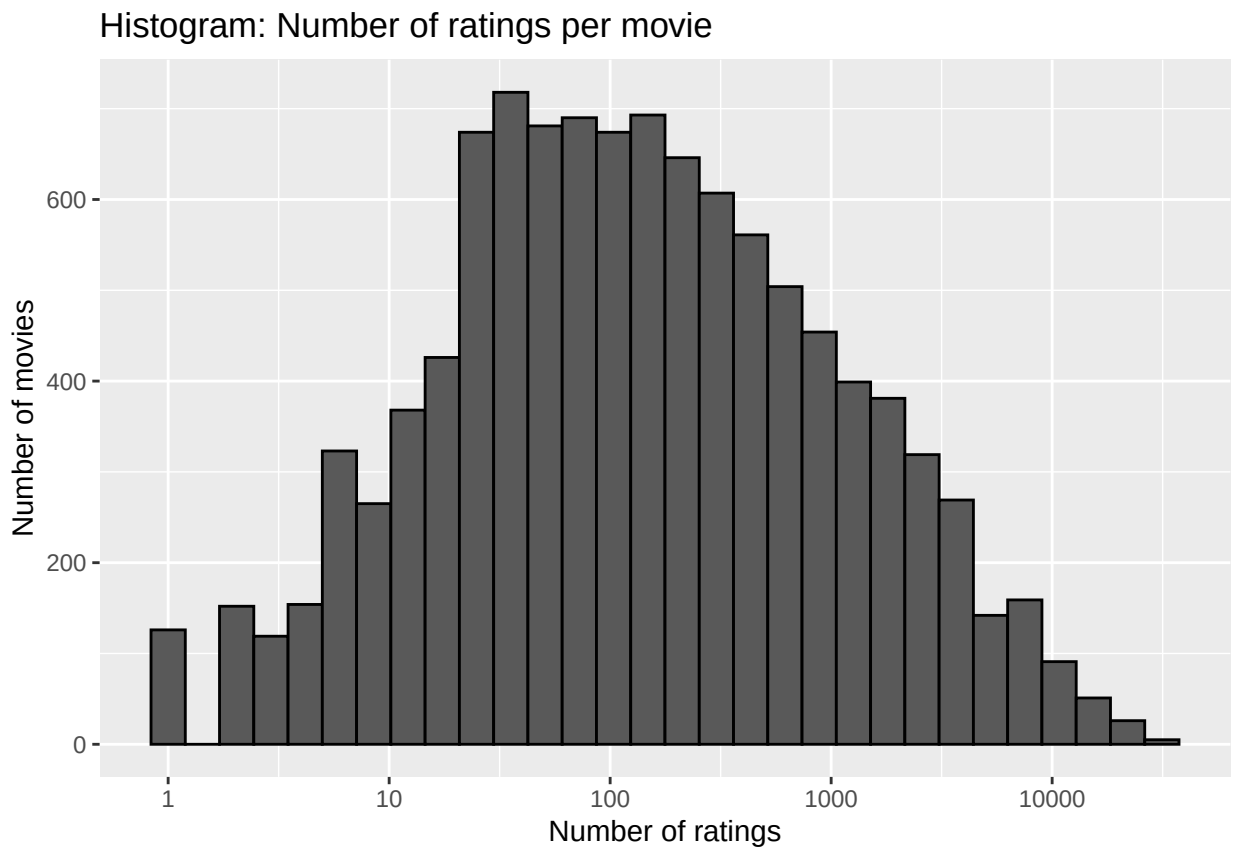
The above rating distributions shows that the users tend to rate movies rather higher than lower, being 4 the most common rating, followed by 3 and 5. We can also notice that most of the ratings are rounded, whole start rating, so we could say that half star rating is not the preferred option for users.

2.2.2 Movies

We know from intuition or based on our own experience that some movies get rated more than others, this makes sense when we think about the popularity of blockbuster movies watched by millions and on the other side we may have independent movies watched by just a few.

The number of ratings for each movie are shown below in the histogram. Some movies have been rated very few number of times which will make predicting future ratings more difficult.

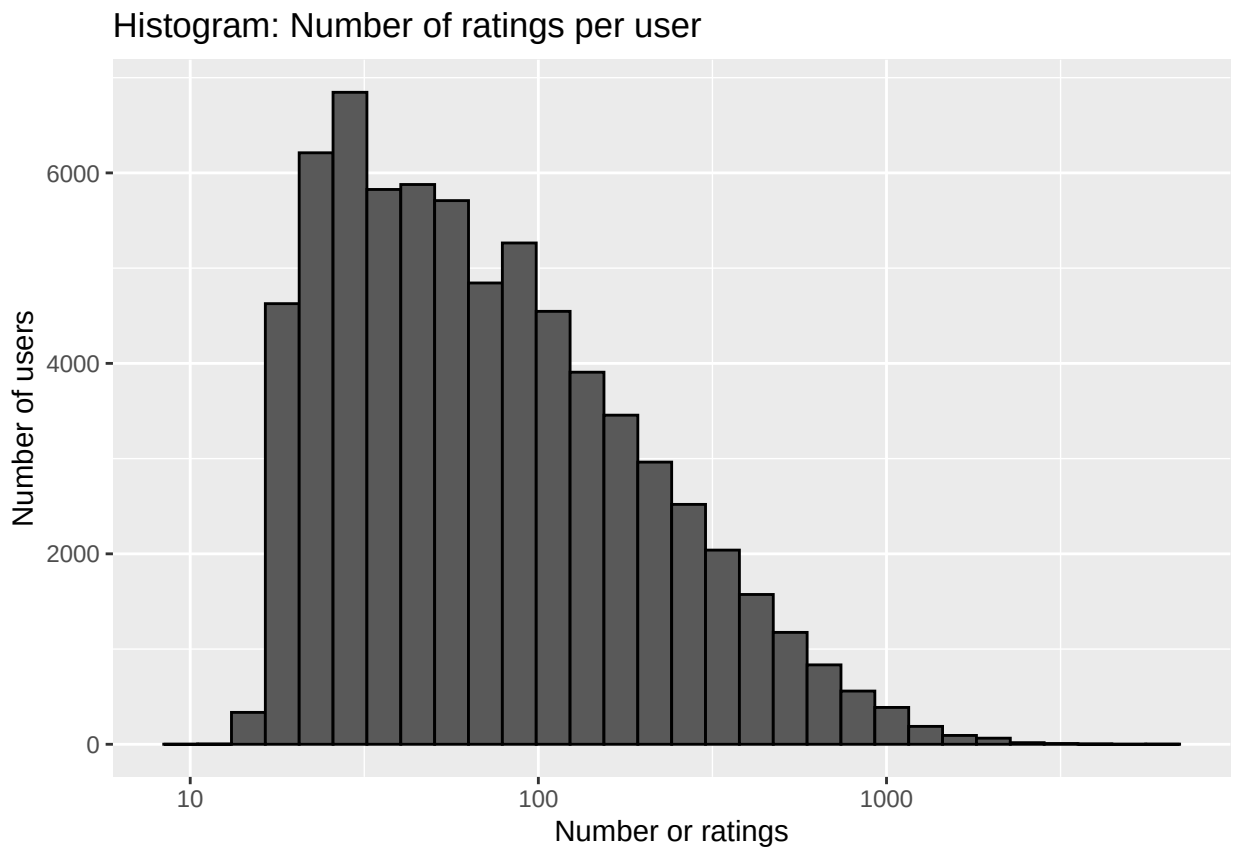
```
# Histogram: Number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins = 30) +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Histogram: Number of ratings per movie")
```



2.2.3 Users

In our next observation we can see that some users are more active than others at rating movies, most of the users rate between 30 and 100 movies, while a few users rate more than a thousand movies.

```
# Histogram: Number of ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", bins = 30) +
  scale_x_log10() +
  xlab("Number or ratings") +
  ylab("Number of users") +
  ggtitle("Histogram: Number of ratings per user")
```



2.3 Data Preparation

We will train and test our algorithm using the *edx* dataset. For a final test of the algorithm, we will use the *validation* dataset to predict movie ratings, simaulating new data.

Let's split the *edx* dataset in two, the train and the test set, the model building is done in the train set, and the test set is used to test the model.

Using same procedure used to create *edx* and *validation* sets, the train set will be 90% of *edx* data and the test set will be the remaining 10%.

```
# Let's split the edx dataset in two, train and test sets
# validation set will be used to calculate the final RMSE
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-edx_test_index,]
edx_temp <- edx[edx_test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- edx_temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
edx_removed <- anti_join(edx_temp, test_set)
```

```r
train_set <- rbind(train_set, edx_removed)

rm(edx_test_index, edx_temp, edx_removed)
```

3 Modeling

3.1 Algorithm Selection

The evaluation of machine learning algorithms consists in comparing the predicted value with the actual outcome. The loss function measures the difference between both values. Root mean square error (RMSE) is one of the most used loss functions to measure the differences between values predicted by a model and the values observed.

Root mean square error computes the mean value of all the differences squared between the true and the predicted ratings and then proceeds to calculate the square root out of the result. As a consequence, large errors may dramatically affect the RMSE rating, rendering the RMSE metric most valuable when significantly large errors are unwanted. The root mean square error between the true ratings and predicted ratings is given by:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

Remember that we can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good. The goal of this project is to create a recommendation system with RMSE lower than 0.86490.

Since RMSE will be used frequently, let's write a function that computes the RMSE for vectors of ratings and their corresponding predictors::

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

3.2 Just the Average Model

Let's start by building the simplest possible recommendation system, we predict all users will give the same rating to all movies regardless of user and movie. The initial prediction is just the average of all observed ratings, as described in this formula:

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0, and μ is the mean of the observed data (the "true" rating for all movies). Any value other than the mean increases the RMSE, so this is a good initial estimation.

We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

```
# Average of all ratings
mu <- mean(train_set$rating)
mu
```

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

Predicting all unkknown ratings with the mean gives the following RMSE:

```
# Just average model RMSE
avg_rmse <- RMSE(test_set$rating, mu)
avg_rmse
```

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

Let's present the results in a table including the expected RMSE, so we can compare the obtained results.

```
# Save results in Data Frame, including the project goal RMSE
rmse_results = data_frame(Method = "Project Goal", RMSE = 0.86490)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Just the Average Model", RMSE = avg_rmse ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.864900
Just the Average Model	1.060054

We get a RMSE of about 1, which is too high, but this give us our baseline RMSE to compare with next modelling approaches. We can do better than simply predicing the avarage rating, in next models we will include some of the insights we observed during the data exploration.

3.3 The Movie Effect Model

We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data, where higher ratings are mostly related to popular movies or blockbusters and lower ratings to unpopular movies, let's check the rating of the most rated movies and less rated movies:

```
# Most rated movies and their rating
edx %>% group_by(movieId, title) %>%
  summarize(count = n(), mean(rating)) %>%
  arrange(desc(count)) %>% head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   movieId [10]
##    movieId title                                     count `mean(rating)`
##      <dbl> <chr>                                     <int>          <dbl>
##  1     296 Pulp Fiction (1994)                       31362           4.15
##  2     356 Forrest Gump (1994)                       31079           4.01
##  3     593 Silence of the Lambs, The (1991)          30382           4.20
##  4     480 Jurassic Park (1993)                      29360           3.66
##  5     318 Shawshank Redemption, The (1994)          28015           4.46
##  6     110 Braveheart (1995)                         26212           4.08
##  7     457 Fugitive, The (1993)                      25998           4.01
##  8     589 Terminator 2: Judgment Day (1991)         25984           3.93
```

```
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star~ 25672          4.22
## 10     150 Apollo 13 (1995)                                  24284          3.89
```

```r
# Less rated movies and their rating
edx %>% group_by(movieId, title) %>%
  summarize(count = n(), mean(rating)) %>%
  arrange(count) %>% head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   movieId [10]
##    movieId title                                                count `mean(rating)`
##      <dbl> <chr>                                                <int>          <dbl>
##  1    3191 Quarry, The (1998)                                       1            3.5
##  2    3226 Hellhounds on My Trail (1999)                            1            5
##  3    3234 Train Ride to Hollywood (1978)                           1            3
##  4    3356 Condo Painting (2000)                                    1            3
##  5    3383 Big Fella (1937)                                         1            3
##  6    3561 Stacy's Knights (1982)                                   1            1
##  7    3583 Black Tights (1-2-3-4 ou Les Collants noirs) (1~         1            3
##  8    4071 Dog Run (1996)                                           1            1
##  9    4075 Monkey's Tale, A (Les Château des singes) (1999)         1            1
## 10    4820 Won't Anybody Listen? (2000)                             1            2
```

From the above data we can confirm that most rated movies usually get a higher rating and you can recognize the title since most of them are popular movies, the opposite happens with less rated movies.

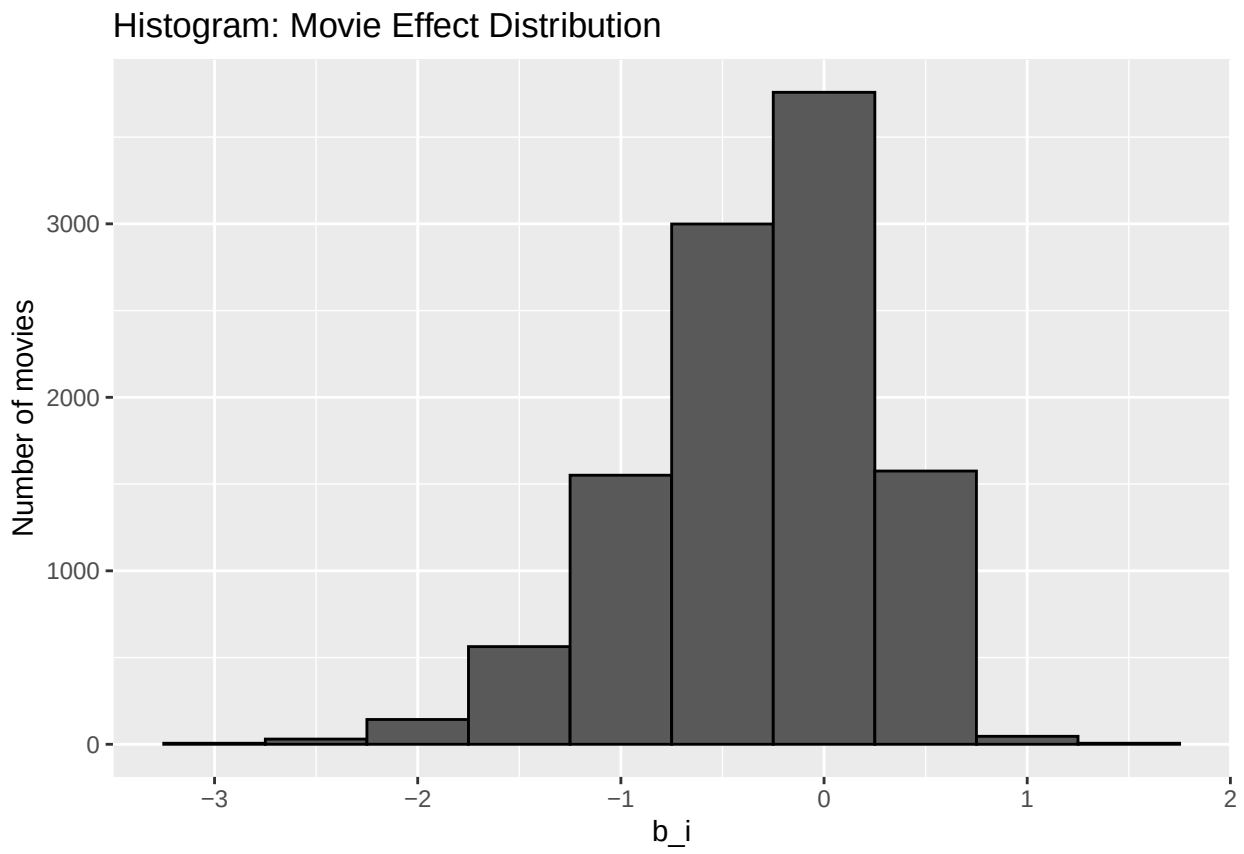
We can augment our previous model by adding the term b_i to represent average ranking for movie i, so the Movie Effect Model calculates a bias term for each movie based on the difference between the movies mean rating and the total mean rating of all movies, μ, calculated in the previous model.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, and b_i is the bias for each movie i.

The movie effect histogram is normally left skewed distributed, implying that more movies have negative effects:

```r
# Calculating b_i per movie
movies_mean <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Histogram of movie effect distribution
movies_mean %>% ggplot(aes(b_i)) +
  geom_histogram(color = "black", bins = 10) +
  xlab("b_i") +
  ylab("Number of movies") +
  ggtitle("Histogram: Movie Effect Distribution")
```



Remember μ is 3.5 so a b_i of 1.5 implies that movie has a perfect five star rating.

Let's see how much our prediction improves once we use this model:

```
# Movie Effect Model RMSE
predicted_ratings <- test_set %>%
  left_join(movies_mean, by='movieId') %>%
  mutate(pred_rating = mu + b_i) %>%
  pull(pred_rating)

movie_rmse <- RMSE(test_set$rating, predicted_ratings)

# Add the Movie Effect Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Effect Model", RMSE = movie_rmse ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615

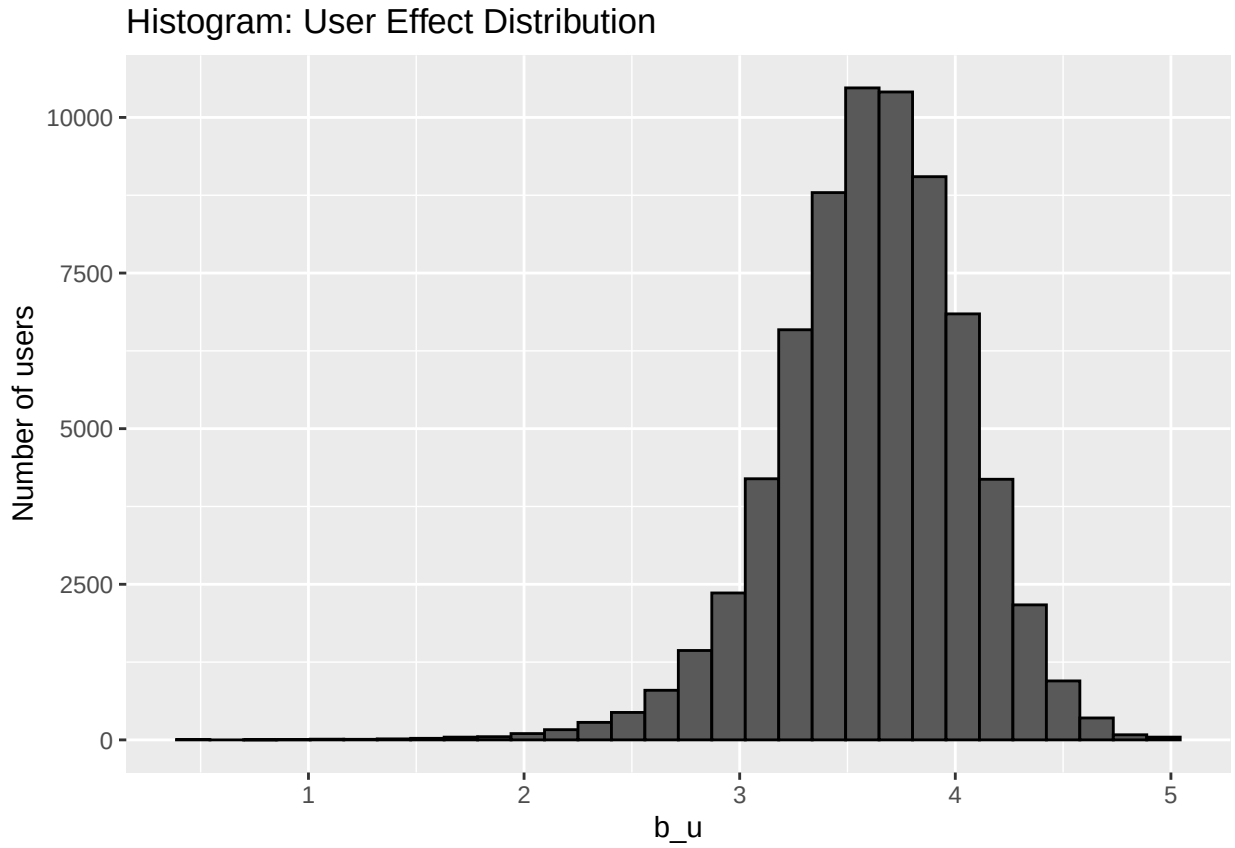
With the Movie Effect Model we have predicted the movie rating, taking into consideration that movies are rated differently, we can see an improvement with a lower RMSE value.

3.4 The Movie and User Effect Model

We also know from experience that users are different in how they rate the movies, some users are very critical and may rate a good movie lower or some others are very generous always giving high rates, or simply, we all have different movie tastes. As we already see in our data exploration we also have users more active than others at rating movies.

Let's compute the average rating for user u for those users that have rated over 100 movies:

```
# Histogram of user effect distribution
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n() >= 100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(color = "black", bins = 30) +
  xlab("b_u") +
  ylab("Number of users") +
  ggtitle("Histogram: User Effect Distribution")
```



We have confirmed the variability accorss users, some users are very cranky and others love every movie. The next step is to incorporate the User effect, b_u, in to the model:

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, b_i is the bias for each movie i, and b_u is the bias for each user u.

Now if a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

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

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

```
# Calculating b_u per user
users_mean <- train_set %>%
  left_join(movies_mean, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We can now construct predictors and see how much the RMSE improves:

```
# User Effect Model RMSE
predicted_ratings <- test_set %>%
  left_join(movies_mean, by='movieId') %>%
  left_join(users_mean, by='userId') %>%
  mutate(pred_rating = mu + b_i + b_u) %>%
  pull(pred_rating)

user_rmse <- RMSE(test_set$rating, predicted_ratings)

# Add the Movie Effect Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie and User Effect Model", RMSE = user_rmse ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615
Movie and User Effect Model	0.8646843

Incorporating the user bias into the model resulted in a good RMSE improvement from our last model.

3.5 Regularization

So far, we have noticed that some movies are rated very few times, just one rating, also we have some users that are really active at rating movies and others which have rated few movies. Let's explore further and see how we can improve our model, here are the 10 largest mistakes from our previous Movie Effect model:

```
# 10 largest mistakes from Movie Effect Model
test_set %>%
  left_join(movies_mean, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
```

```
  select(movieId, title, residual) %>%
  knitr::kable()
```

movieId	title	residual
6483	From Justin to Kelly (2003)	4.125683
318	Shawshank Redemption, The (1994)	-3.956567
318	Shawshank Redemption, The (1994)	-3.956567
858	Godfather, The (1972)	-3.916651
858	Godfather, The (1972)	-3.916651
858	Godfather, The (1972)	-3.916651
858	Godfather, The (1972)	-3.916651
50	Usual Suspects, The (1995)	-3.866552
527	Schindler's List (1993)	-3.864085
527	Schindler's List (1993)	-3.864085

Now let's look at the top 10 worst and best movies based on b_i, we will also add how often they are rated, but first let's create a database that connects movieId to movie title:

```
# Movies ids and titles
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
```

This are the 10 best movies according to our estimate:

```
# 10 best movies adding number of ratings
train_set %>% count(movieId) %>%
  left_join(movies_mean, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(movieId, title, b_i, n) %>%
  head(10) %>%
  knitr::kable()
```

movieId	title	b_i	n
3226	Hellhounds on My Trail (1999)	1.487544	1
33264	Satan's Tango (Sátántangó) (1994)	1.487544	1
42783	Shadows of Forgotten Ancestors (1964)	1.487544	1
51209	Fighting Elegy (Kenka erejii) (1966)	1.487544	1
53355	Sun Alley (Sonnenallee) (1999)	1.487544	1
64275	Blue Light, The (Das Blaue Licht) (1932)	1.487544	1
5194	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544	4
25975	Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544	2
26048	Human Condition II, The (Ningen no joken II) (1959)	1.237544	4
26073	Human Condition III, The (Ningen no joken III) (1961)	1.237544	4

The 10 worst:

```r
# 10 worst movies adding number of ratings
train_set %>% count(movieId) %>%
  left_join(movies_mean, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(movieId, title, b_i, n) %>%
  head(10) %>%
  knitr::kable()
```

movieId	title	b_i	n
5805	Besotted (2001)	-3.012456	1
8394	Hi-Line, The (1999)	-3.012456	1
61768	Accused (Anklaget) (2005)	-3.012456	1
63828	Confessions of a Superhero (2007)	-3.012456	1
64999	War of the Worlds 2: The Next Wave (2008)	-3.012456	2
8859	SuperBabies: Baby Geniuses 2 (2004)	-2.767775	47
61348	Disaster Movie (2008)	-2.745789	30
6483	From Justin to Kelly (2003)	-2.638139	183
7282	Hip Hop Witch, Da (2000)	-2.603365	11
604	Criminals (1996)	-2.512456	1

Note that both the best and worst movies are quite unknown, and were rated by very few users, in most cases just 1.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

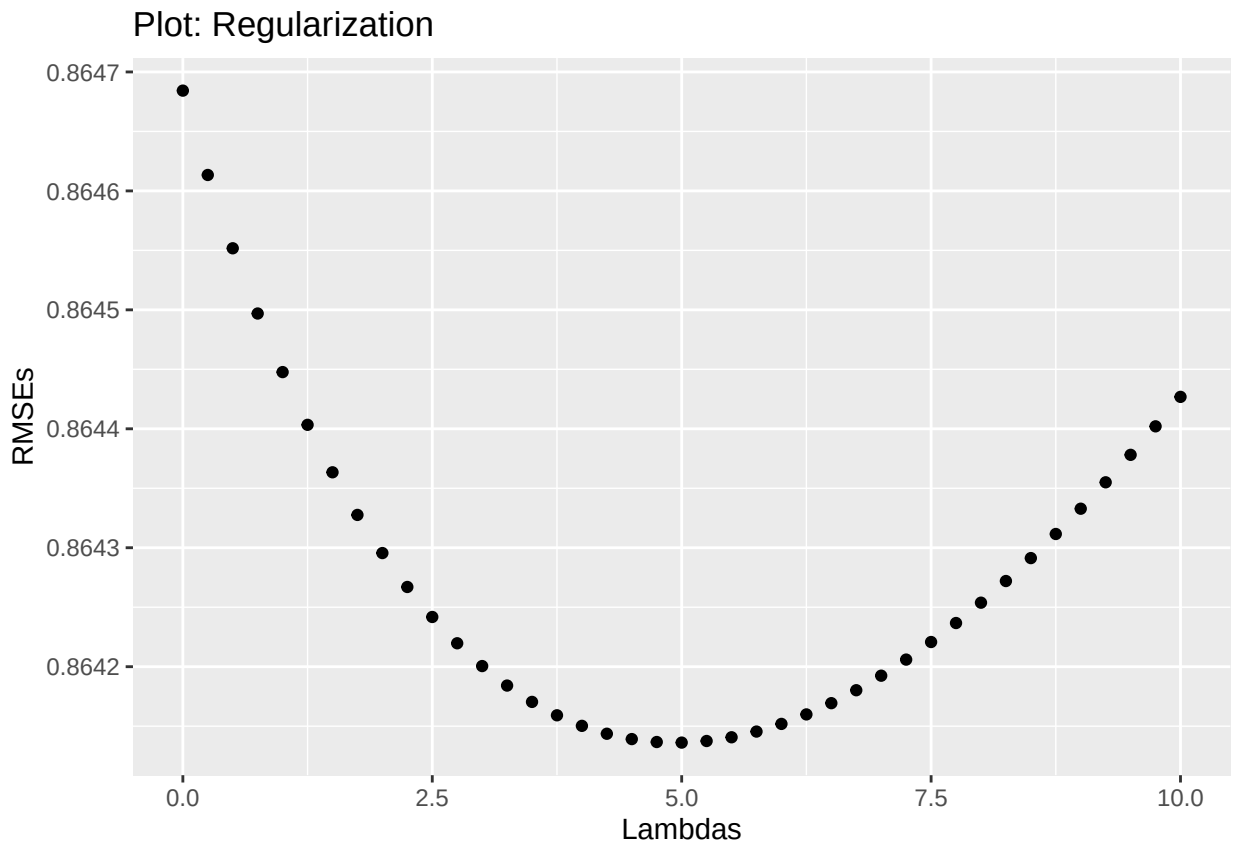
The use of regularization permits to penalize the noisy estimates, we should find the value that will minimize the RMSE, this tuning parameter is known as λ.

```r
# lambda is the tuning parameter
# We use cross-validation to choose it.
lambdas <- seq(0, 10, 0.25)

# for each lambda we find the movie (b_i) and user (b_u) effect and predict
rmses <- sapply(lambdas, function(l){

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

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

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

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

Let's plot RMSEs vs lambdas to visualize the optimal lambda:

```
# Plot rmses vs lambdas to visualize the optimal lambda
qplot(lambdas, rmses, main = "Plot: Regularization", xlab = "Lambdas", ylab = "RMSEs")
```



So, let's get the value for the optimal lambda:

```
# The optimal lambda
lambda <- lambdas[which.min(rmses)]
cat("The optimal lambda is: ", lambda)
```

```
## The optimal lambda is:  5
```

```
# Add the Regularization Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie and User Effect Model",
                                     RMSE = min(rmses) ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615
Movie and User Effect Model	0.8646843
Regularized Movie and User Effect Model	0.8641362

Regularization of a Movie and User Effect model has give us the lowest RMSE of the prediction methods for the MovieLens ratings system.

3.6 Matrix Factorization

Matrix factorization is a widely used concept in machine learning. It is very much related to factor analysis, singular value decomposition (SVD), and principal component analysis (PCA). Matrix factorization method works by approximating the whole user-movie matrix into the product of two matrices of lower dimensions.

To use matrix factorization we need to convert the train set from tidy format to user-movie matrix, we can do that using the following code:

```
train_matrix <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
```

Since the above piece of code is too heavy to run, we can use the recosystem package which is specifically use for recommender system using matrix factorization.

The usage of *recosystem* is quite simple, mainly consisting of the following steps:

1. Create a model object (a Reference Class object in R) by calling *Reco*().
2. (Optionally) call the $*tune*() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $*train*() method. A number of parameters can be set inside the function, possibly coming from the result of $*tune*().
4. (Optionally) output the model, i.e. write the factorized P and Q matrices info files.
5. Use the $predict() method to compute predictions and write results into a file.

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

set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Create the reco model object
r = Reco()

# Convert train and test sets
train_reco <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId, rating = rating))
test_reco <- with(test_set,  data_memory(user_index = userId,
                                          item_index = movieId, rating = rating))

# Select best tuning parameters
```

```
opts <- r$tune(train_reco, opts = list(dim = c(10, 20, 30), lrate = c(0.01, 0.1),
                                       costp_l1 = 0, costp_l2 = c(0.01, 0.1),  # user cost
                                       costq_l1 = 0, costq_l2 = c(0.01, 0.1),  # movie cost
                                       nthread  = 4, niter = 10)) # threads and iterations

# Train the model
r$train(train_reco, opts = c(opts$min, nthread = 4, niter = 40))
```

```
## iter      tr_rmse          obj
##    0       0.9825   1.1043e+07
##    1       0.8757   8.9931e+06
##    2       0.8422   8.3379e+06
##    3       0.8197   7.9508e+06
##    4       0.8037   7.6827e+06
##    5       0.7917   7.4976e+06
##    6       0.7817   7.3536e+06
##    7       0.7731   7.2370e+06
##    8       0.7659   7.1437e+06
##    9       0.7596   7.0652e+06
##   10       0.7541   7.0016e+06
##   11       0.7491   6.9429e+06
##   12       0.7447   6.8952e+06
##   13       0.7406   6.8495e+06
##   14       0.7369   6.8116e+06
##   15       0.7334   6.7770e+06
##   16       0.7302   6.7455e+06
##   17       0.7275   6.7202e+06
##   18       0.7247   6.6954e+06
##   19       0.7222   6.6707e+06
##   20       0.7199   6.6483e+06
##   21       0.7177   6.6300e+06
##   22       0.7156   6.6133e+06
##   23       0.7137   6.5958e+06
##   24       0.7119   6.5812e+06
##   25       0.7103   6.5666e+06
##   26       0.7087   6.5521e+06
##   27       0.7073   6.5409e+06
##   28       0.7059   6.5302e+06
##   29       0.7046   6.5191e+06
##   30       0.7033   6.5090e+06
##   31       0.7021   6.4994e+06
##   32       0.7010   6.4921e+06
##   33       0.6999   6.4821e+06
##   34       0.6989   6.4755e+06
##   35       0.6979   6.4669e+06
##   36       0.6971   6.4600e+06
##   37       0.6962   6.4534e+06
##   38       0.6954   6.4485e+06
##   39       0.6946   6.4424e+06
```

Finally we will compute the prediction and include the result in our table:

```
# Compute the prediction
predict_reco <- r$predict(test_reco, out_memory())

fact_rmse = RMSE(test_set$rating, predict_reco)

# Add the Matrix Factorization Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Matrix Factorization - Recosystem",
                                     RMSE = fact_rmse ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615
Movie and User Effect Model	0.8646843
Regularized Movie and User Effect Model	0.8641362
Matrix Factorization - Recosystem	0.7845036

We can see that Matrix Factorization has improved significantly the RMSE.

4 Results Analysis

From the results table above we can see that Regularized Movie and User Effect, and Matrix Factorizacion - Recosystem models got the lower RMSE, below the target. So now we are going to apply both models using *edx* set to train the model and the *validation* set to test it and calculate the RMSE, finally we will check if we still achieve the target which is RMSE lower than 0.86490.

4.1 Regularization with Validation Set

Let's calculate the regularization including the movie and user effects and using the same tuning parameter lambda λ.

```
# edx rating mean
mu_edx <- mean(edx$rating)

# Movie effect using edx
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

# User effect using edx
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

predicted_ratings <-
  validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  .$pred

reg_validation_rmse <- RMSE(validation$rating, predicted_ratings)

# Add the Regularization with Validation Set Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                data_frame(Method="Regularized Movie and User Effect Model with Validation Set",
                                    RMSE = reg_validation_rmse))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615
Movie and User Effect Model	0.8646843
Regularized Movie and User Effect Model	0.8641362
Matrix Factorization - Recosystem	0.7845036
Regularized Movie and User Effect Model with Validation Set	0.8648177

We can see that the RMSE caluculated using the *validation* set is slightly lower than the project target 0.86490, but higher than the RMSE calculated in the Regularization model using the test set.

4.2 Matrix Factorization with Validation Set

Matrix Factorization was our best model using the test set, now let's compute using *edx* and *validation* sets.

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Create the reco model object
r = Reco()

# Convert edx and validation sets
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 = rating))

# Select best tuning parameters
opts <- r$tune(edx_reco, opts = list(dim = c(10, 20, 30), lrate = c(0.01, 0.1),
                                     costp_l1 = 0, costp_l2 = c(0.01, 0.1),  # user cost
                                     costq_l1 = 0, costq_l2 = c(0.01, 0.1),  # movie cost
                                     nthread  = 4, niter = 10)) # threads and iterations

# Train the model using same tuning parameters
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 40))
```

```
## iter      tr_rmse          obj
##    0       0.9729   1.2030e+07
##    1       0.8729   9.8935e+06
##    2       0.8387   9.1832e+06
##    3       0.8158   8.7446e+06
##    4       0.8005   8.4653e+06
##    5       0.7889   8.2700e+06
##    6       0.7794   8.1192e+06
##    7       0.7717   8.0010e+06
##    8       0.7652   7.9084e+06
##    9       0.7596   7.8284e+06
##   10       0.7546   7.7651e+06
##   11       0.7501   7.7073e+06
##   12       0.7460   7.6585e+06
##   13       0.7424   7.6157e+06
##   14       0.7390   7.5782e+06
##   15       0.7358   7.5432e+06
##   16       0.7328   7.5110e+06
##   17       0.7301   7.4813e+06
##   18       0.7276   7.4559e+06
##   19       0.7252   7.4345e+06
##   20       0.7230   7.4119e+06
##   21       0.7209   7.3913e+06
##   22       0.7190   7.3740e+06
##   23       0.7172   7.3574e+06
```

```
##   24       0.7155   7.3425e+06
##   25       0.7139   7.3276e+06
##   26       0.7125   7.3147e+06
##   27       0.7111   7.3019e+06
##   28       0.7098   7.2901e+06
##   29       0.7085   7.2779e+06
##   30       0.7074   7.2682e+06
##   31       0.7062   7.2591e+06
##   32       0.7052   7.2505e+06
##   33       0.7042   7.2420e+06
##   34       0.7033   7.2342e+06
##   35       0.7024   7.2276e+06
##   36       0.7015   7.2196e+06
##   37       0.7007   7.2130e+06
##   38       0.7000   7.2072e+06
##   39       0.6992   7.2014e+06
```

```r
# Compute the prediction
predict_reco <- r$predict(validation_reco, out_memory())

fact_validation_rmse <- RMSE(validation$rating, predict_reco)

# Add the Matrix Factorization Model RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Matrix Factorization - Recosystem with Validation Set",
                                     RMSE = fact_validation_rmse ))
# Check the results
rmse_results %>% knitr::kable()
```

Method	RMSE
Project Goal	0.8649000
Just the Average Model	1.0600537
Movie Effect Model	0.9429615
Movie and User Effect Model	0.8646843
Regularized Movie and User Effect Model	0.8641362
Matrix Factorization - Recosystem	0.7845036
Regularized Movie and User Effect Model with Validation Set	0.8648177
Matrix Factorization - Recosystem with Validation Set	0.7804031

We can see that Matrix Factorization Model RMSE is significantly better than Regularization model, having an improvement of 10.26%.

5 Conclusion

From the RMSE table we can see how we have improved our model using different algorithms, having in mind the project goal which is 0.86490.

To start, we calculated the RMSE using just the average of all the ratings, this resulted in a RMSE higher than 1 (1.0600537), which is not good since we might be missing the rating by one star. Then we incorporated the Movie Effect (0.9429615) and Movie and User Effect (0.8646843) which gives a great improvement of 20.3% from initial model.

We could do it better, analysing the data revealed that some features have large effect on errors, so a regularization model was needed to penalize this data. We added a penalty value for movies and users with few number of ratings, this model returned a RMSE of 0.8648177 which is lower than our project target.

Finally, we evaluated Matrix Factorizacion algorithm using the Recosystem package and significantly improved our RMSE getting 0.7804031 which represents an improvement of 30.39% compared to our initial model.

With these results we can say that we have successfully built a machine learning algorithm to predict movie ratings with MovieLens dataset.

6 References

- Rafael A. Irizarry. (2020). Introduction to Data Science: Data Analysis and Prediction Algorithms with R.
- F.O.Isinkaye, Y.O.Folajimi & B.A.Ojokohc. (2015). Recommendation systems: Principles, methods and evaluation. https://www.sciencedirect.com/science/article/pii/S1110866515000341#b0020
- Safir Najafi and Ziad Salam. (2016). Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems. https://kth.diva-portal.org/smash/get/diva2:927356/FULLTEXT01.pdf
- http://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/
- https://www.netflixprize.com/
- https://grouplens.org/
- https://movielens.org/
- https://grouplens.org/datasets/movielens/
- https://www.rdocumentation.org/packages/recosystem/versions/0.3