Netflix MovieLens Project

Article · May 2021

CITATIONS
0

READS
228

1 author:

Lucainson Raymond
Centre de Techniques de Planification et d'Economie Appliquée (CTPEA)
16 PUBLICATIONS **5** CITATIONS

SEE PROFILE

Netflix MovieLens Project

Lucainson RAYMOND [lucainson.raymond@gmail.com]
HarvardX Data Science Professional Certificate

2021-05-22

Abstract

Recommender system, also known as recommender engine, is a type of machine learning algorithm that aims to predict user rating. With the unprecedented development of online commerce, the virtualization of our daily economic activities, this algorithm is becoming more and more popular and taken very seriously. It therefore enables very efficient customer management by offering content and products for which the customer's probability of interest is high. At the same time, poor customer management (for example, inappropriate advertising) can negatively impact the company's turnover. Aware of this fact, the online streaming company Netflix launched in 2006 a challenge in the data science community by offering $1 million to the team which would increase the accuracy of their existing recommendation model by at least 10%. It provided data scientists with a dataset of 100 million anonymous users. And 3 years later, in 2009, the results were announced. The team named *BellKor's Pragmatic Chaos* was the winner. As part of the course HarvardX Data Science Professional Certificate, we sought to meet the challenge on a partition of the dataset, ie 10 million observations. Two (2) main methods were used to overcome said challenge. These are the "basic" and "advanced" methods. Regarding the basic method, we have trained several models where we take into account the global rating average, the individual effects of users, and movies. Further on, we presented the regularized version of said models in order to avoid, among other things, overfitting. As for the advanced method, we used the **Low-Rank Matrix factorization** (cf. model 7) based on the baseline model which achieves the lowest RMSE. That is **model 5: Regularized Movie & User Effect Model (Version 1)**. With this techniques, we were able to obtain a RMSE of *0.7865805*. And then our RMSE threshold goal— which is the asymptote *0.86490* — is therefore reduced by a little more than 9%.

Keywords: Recommender system, machine learning, rating, Netflix, model, low-rank matrix factorization, RMSE.

I. Introduction

The MovieLens dataset contains movie ratings. Each observation in the dataset corresponds to a movie rating by a user and it's uniquely identified by the numeric movieId and userId attributes. As for other variables in this dataset, there is the genre of the movies as well as the timestamps (which will be explained below). As a Feature engineering procedure, we divide the initial dataset which contains 10M observations into two (2) parts. And we make sure that the same variables are found in both. So, we will do some exploratory analysis on the first dataset — which represents 90%. We call it edx. We will start by

developing some graphics. This will allow us to already have a good idea of the overall behavior of the variables of interest.

After exploring the training dataset (edx) graphically, we move on to the most important part of the project, which is modeling; the latter should allow us to have a video recommendation system with satisfactory precision. Conversely, its RMSE should be as minimal as possible. In this sense, we divide the process into two parts in terms of methods. In other words, two (2) methods. A first so-called basic method where we build 6 models. From the simplest to the most elaborate. First, we start by building a naive model. The latter consists in predicting the same rating for any user and movie indifferently. Then, as we go along, we add other variables that we deem important a priori. These are what we call user and movie effects or biases. And finally, within the same framework of the basic method, we use the technique of regularization. Thus, 3 regularized models are trained. Note that we disregard other variables such as genres as well as timestamps, because on the one hand, in our test outside this project, we notice that they do not add much to the model in terms of improvement, on the other hand, taking them into account in the model causes our machine to crash every time. It is extremely heavy in terms of algorithmic computation. Therefore, the combination of these two (2) factors means that we take into account the only variables that really bring something significant to the model. As a reminder, this is the user and movie effects. Then, we continue with the advanced method which is based among other things on the most efficient baseline model. All this will be explained later. We therefore invite you to read the following lines to become acquainted with our project and leave us your various critics at the email address at the head of the document.

II. Methodology

As mentioned earlier, two main methods are used in our work to model the user ratings. In the first approach, called the basic method, the global average, movie and user biases or effects are modeled.

$$\hat{r}_{u,i} = \mu + \underbrace{b_i}_{\text{movie effect}} + \underbrace{b_u}_{\text{user effect}} + \epsilon_{u,i}$$

The loss function minimized for the basic method is the following:

$$LOSS_{\text{Basic Method}} = \frac{1}{N}\sum_{u,i}(r_{u,i} - (\mu + b_i + b_u))^2 + \lambda\left(\sum_i b_i^2 + \sum_u b_u^2\right)$$

The second approach, **the advanced method**, extends the basic with the *low-rank matrix factorization*[1] using Parallel Stochastic Gradient Descent (PSGD) to account for user-movie interactions:

$$\hat{r}_{u,i} = \mu + \underbrace{b_i}_{\text{movie effect}} + \underbrace{b_u}_{\text{user effect}} + P_u^T Q_i + \epsilon_{u,i}$$

1 See collaborative filtering paradigm.

Where $P_{(N \text{ users}, K \text{ latent})}$ is a matrix containing N rows that correspond to the unique users and K described as latent dimensions or principal components. $Q_{(N \text{ movies}, K \text{ latent})}$ is a matrix containing M rows that correspond to the unique movies and K. Matrix factorization tries to decompose the rating matrix into user matrix P and item (movie) matrix Q.The matrix factorization strategy is also used to model the residual of the baseline model.

The loss function minimized for the advanced approach is the following. Note that there are two separate lambdas, the advanced lambda penalizes large coefficients in P and Q:

$$LOSS_{\text{Advanced Method}} = \sum_{u,i} \left(r_{u,i} - (\underbrace{\hat{r}_{u,i}}_{\text{prediction using BM}} + P_u^T Q_i) \right)^2 + \lambda_{\text{AM}} \left(\sum_u \| P_u \|^2 + \sum_i \| Q_i \|^2 \right)$$

Where BM stands for Basic Method, and AM for Advanced Method.

We implement models of the basic method fully in caret package. For the advanced, we've used an additional package: *recosystem*. The latter is used to perform the matrix factorization calculations. Note that this R package is a wrapper of a C++ open-source library: LIMBF. Instead of using RAM memory, it proceeds by storing input, model and output informations in the hard disk. This process improves significantly the time computing.

Note that in the process of building our models, it will be trained as mentioned above on 90% of the initial data set. The remaining 10% will be used to assess the degree of accuracy of said models in terms of predictions. For comparison between the models, the RMSE metrics will be used.

III. Data Wrangling and Feature engineering

- **Loading relevant libraries for our data analysis**

```
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(irlba))
suppressPackageStartupMessages(library(recommenderlab))
suppressPackageStartupMessages(library(recosystem))
suppressPackageStartupMessages(library(data.table))
suppressPackageStartupMessages(library(RColorBrewer))
suppressPackageStartupMessages(library(ggthemes))
suppressPackageStartupMessages(library(kableExtra))
suppressPackageStartupMessages(library(lubridate))
```

```
suppressPackageStartupMessages(library(Matrix.utils))
suppressPackageStartupMessages(library(DT))
```

- **Importing the dataset**

```
#Let's setup the working directory
setwd("C:\\Users\\Lucainson Raymond\\Desktop\\Capstone")

#Let's import the data from our working directory

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

movies <- str_split_fixed(readLines("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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.see
d(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# 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, removed)
```

IV. Exploratory Data Analysis (EDA)

In this part, we intend to explore our dataset graphically. We will proceed variable by variable. Thus, we hope to have a very good knowledge of the data involved. This will help us a lot in the modeling procedure, in other words, in the construction of the recommendation engine.

Now let's take a look at the composition of the dataset. By this we mean to display the number of variables, the names of the variables (labels), as well as the number of cases. In short, after using the code provided to us as part of the project, we see, as shown below, that the edx dataset is made up of 6 variables and 9,000,0005 observations. Note that it is on this data partition that our different models will be trained. The validation set will be used as test data.In other words, it will serve to test the performance of our final model.

```
# Data overview
edx%>%
  glimpse()
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 4
20, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 8
3898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)"
, "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama
|Sci~
```

Rightly so, we also want to give an overview of the set validation. Note that this represents 10% of the 10M Movielens dataset. The validation set contains logically the same features than the edx set, but with a total of 999,999 occurrences. We made sure that validation set contains exactly the same features with the edx dataset.

As part of our exploration, below are the variables that we will have to consider. Note that it is the edx dataset that we will consider only in this context.

Response variable

- rating : takes value between 0 and 5 for the movie i (continuous variable)

Quantitative covariates

- timestamp : Date and time the rating was given (discrete feature)
- movieId: Unique ID for the movie (discrete feature)
- userId : Unique ID for the user (discrete feature)

Qualitative covariates

- title: movie title (not unique)
- genres: genres associated with the movie

a. Response variable: rating

Processing the data…

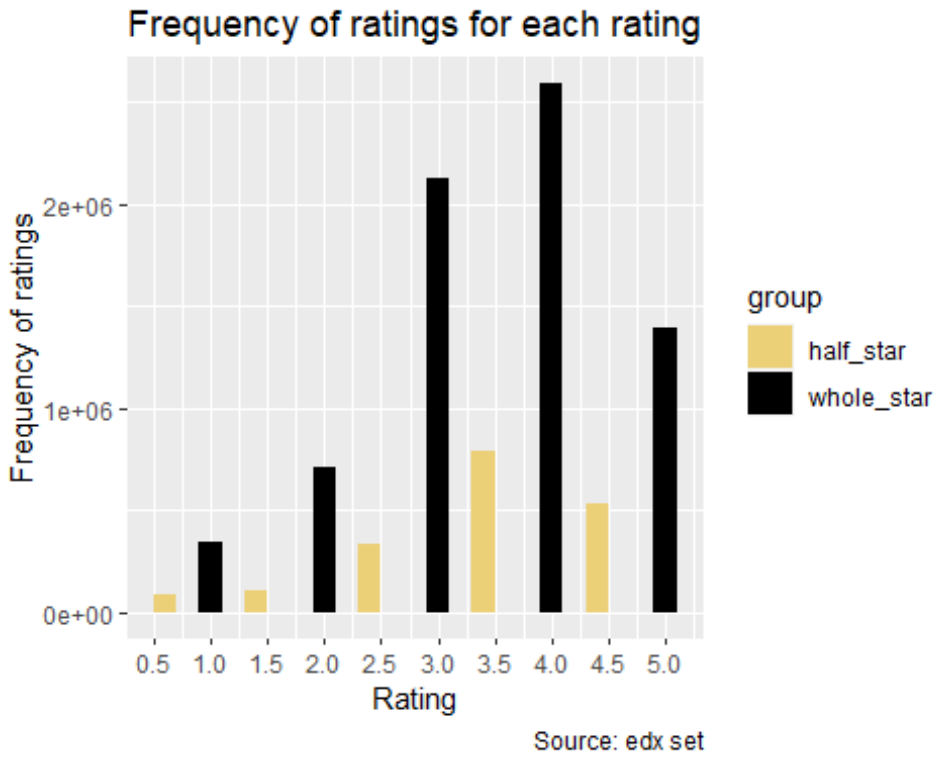
```
#i create a dataframe "explore_ratings" which contains half star and whole st
ar ratings  from the edx set :

group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                  edx$rating == 4 | edx$rating == 5) ,
                   "whole_star",
                   "half_star")

explore_ratings <- data.frame(edx$rating, group)
```

Projection of the graph

Exploring ratings of the edx set , we notice the following facts:

- the average user's activity reveals that no user gives 0 as rating;
- the top 5 ratings from most to least are : 4, 3, 5, 3.5 and 2;
- the histogram shows that the half star ratings are less common than whole star ratings.

```
ggplot(explore_ratings, aes(x= edx.rating, fill = group)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  scale_fill_manual(values = c("half_star"="#ebd078", "whole_star"="black"))
+
  labs(x="Rating", y="Frequency of ratings", caption = "Source: edx set") +
  ggtitle("Frequency of ratings for each rating")
```



b. Qualitative covariates: genres, titles

- **Genres**

The aim here is to explore the genres of movies contained in the edX dataset. First, we will proceed to extract the genre. Knowing that some are given several genres.

```
top_genres <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

Statistics of Genres

We notice that the "Drama" genre has the top number of movies ratings, followed by the "Comedy" and the "Action" genres. There's a last category (where the genre is not listed) which contains only 7 movies ratings.

Titles

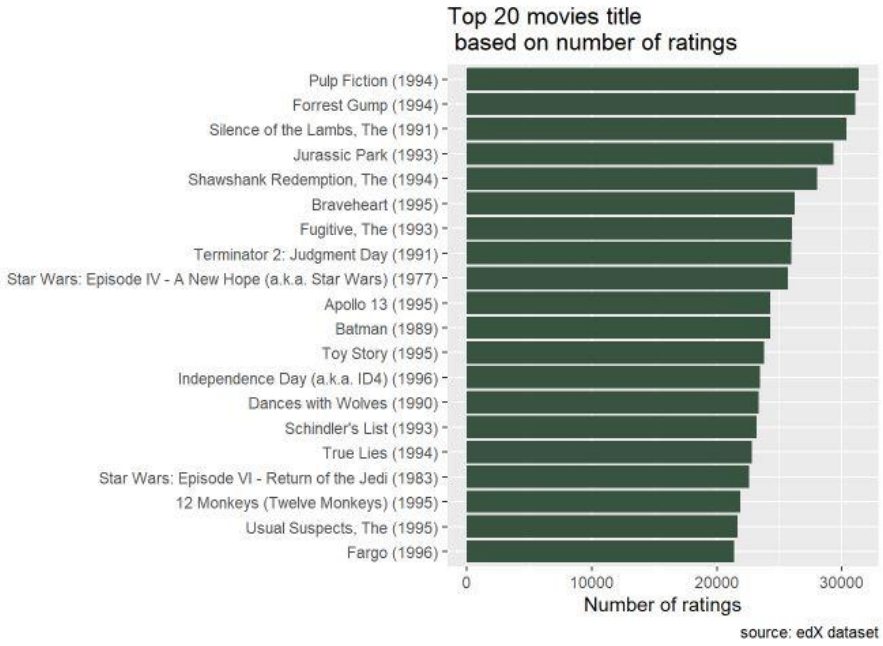
Now it's time to explore the title of the movies.

```r
#Data processing
top_title <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(20,count) %>%
  arrange(desc(count))
```

Projection of the graph

Exploring movie titles of the edx set , we notice that the movie named *Pulp fiction* (released in 1984) has the major ratings, followed by *Forest gump* released the same year.

```r
top_title %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="#395341") +
  coord_flip() +
  labs(x="", y="Number of ratings") +
  labs(title="Top 20 movies title \n based on number of ratings" , caption =
"source: edX dataset")
```



c. Quantitative covariates: UserId, movieId, timestamp

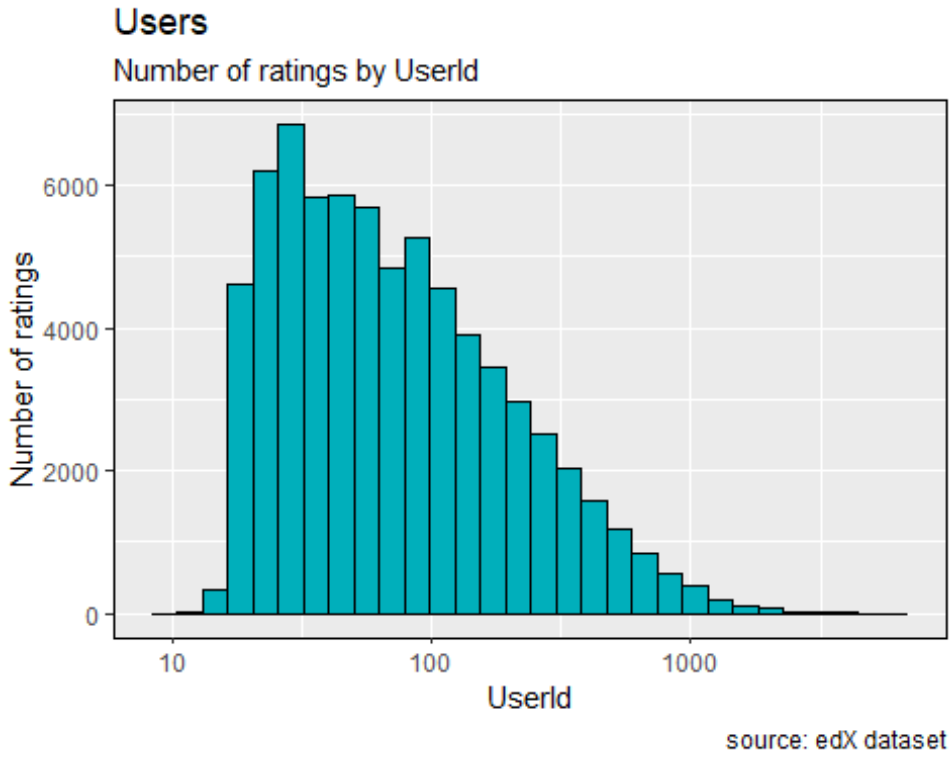
- **UserId**

Number of unique users

```
edx %>%
  summarize(n_users = n_distinct(userId))

##   n_users
## 1   69878
```

Graphic representation

```
#Plot of number of ratings by userId
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color="black",
           fill="#00AFBB") +
  scale_x_log10() +
  ggtitle("Users") +
  labs(subtitle ="Number of ratings by UserId",
       x="UserId" ,
       y="Number of ratings", caption = "source: edX dataset") +
  theme(panel.border = element_rect(colour="black", fill=NA))
```



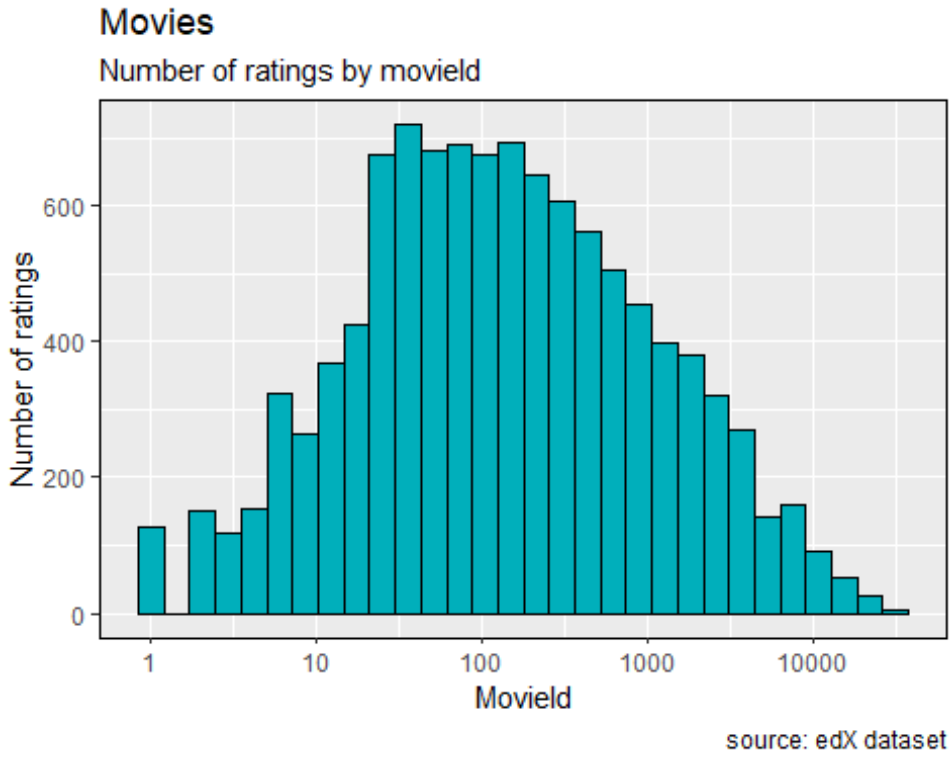
- **MovieId**

```
edx %>%
  summarize(n_movies = n_distinct(movieId))
```

```
##   n_movies
## 1    10677
```

Projection of the graph

```
#Plot of number of ratings by movieId
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color="black",
           fill="#00AFBB") +
  scale_x_log10() +
  ggtitle("Movies") +
  labs(subtitle  ="Number of ratings by movieId",
       x="MovieId" ,
       y="Number of ratings",
       caption = "source: edX dataset") +
  theme(panel.border = element_rect(colour="black", fill=NA))
```



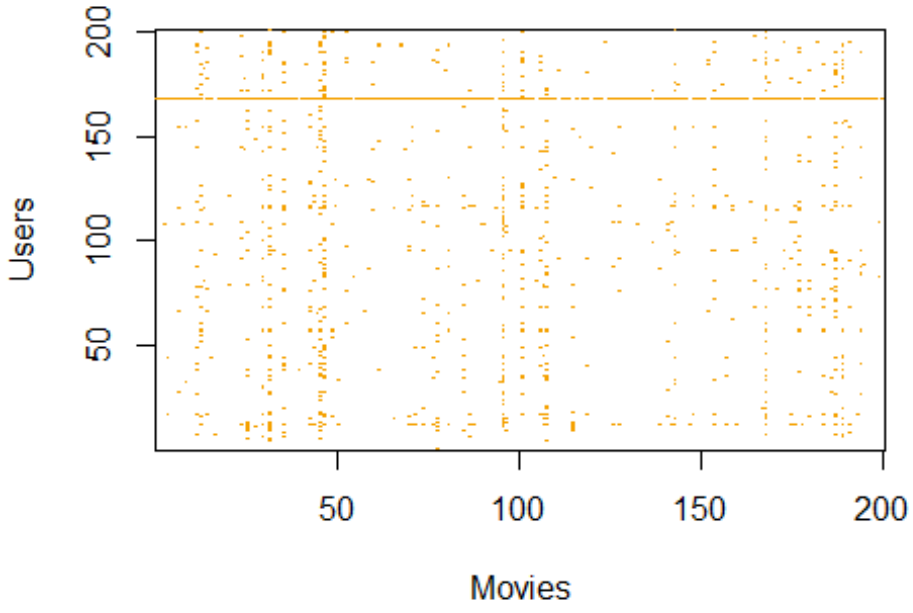
Matrix of User Ratings by Movies

How many random movies and users for sparseness chart?

Since we have the finite number of users (Q = 69 878), movies (P = 10 677), and the rating system indexes pair userId and movieId, the maximum possible size of the ratings matrix R is dim(R) = 69 878 × 10 677 = 746 087 406.

In order to examine the sparsity of the user-movie matrix, we randomly selected 200 movies. By projecting the image of the matrix, we can see many 0's that the said matrix contains below:

```
s <- 200
users <- sample(unique(edx$userId), s)
edx %>% filter(userId %in% users) %>%
select(userId, movieId, rating) %>%
mutate(rating = 1) %>%
spread(movieId, rating) %>% select(sample(ncol(.), s)) %>%
as.matrix() %>% t(.) %>%
image(1:s, 1:s,. , xlab="Movies", ylab="Users")
```



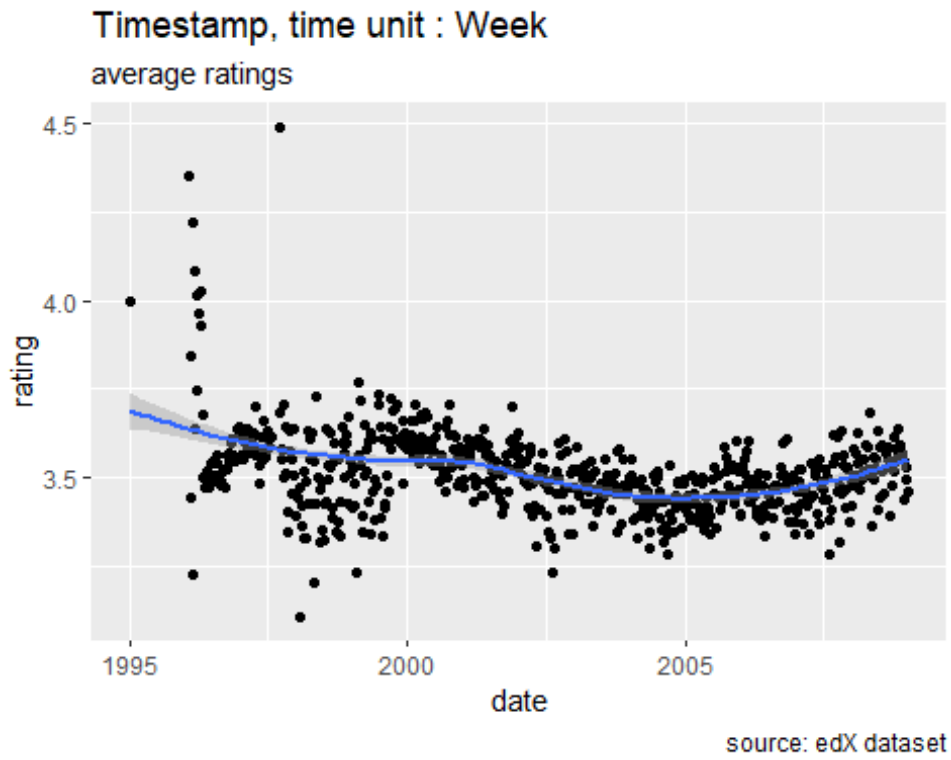
- **Timestamps**

The timestamp variable represents the time and data in which the rating was provided. The units are seconds since January 1, 1970. So we use the as_datetime ( ) function among others to make it in the right format which is ready for analysis. For seeing patterns in rating behaviors overtime, we use a smoothing procedure. Below, we will visualize this successively by week, month, and year.

1. Average Ratings during week

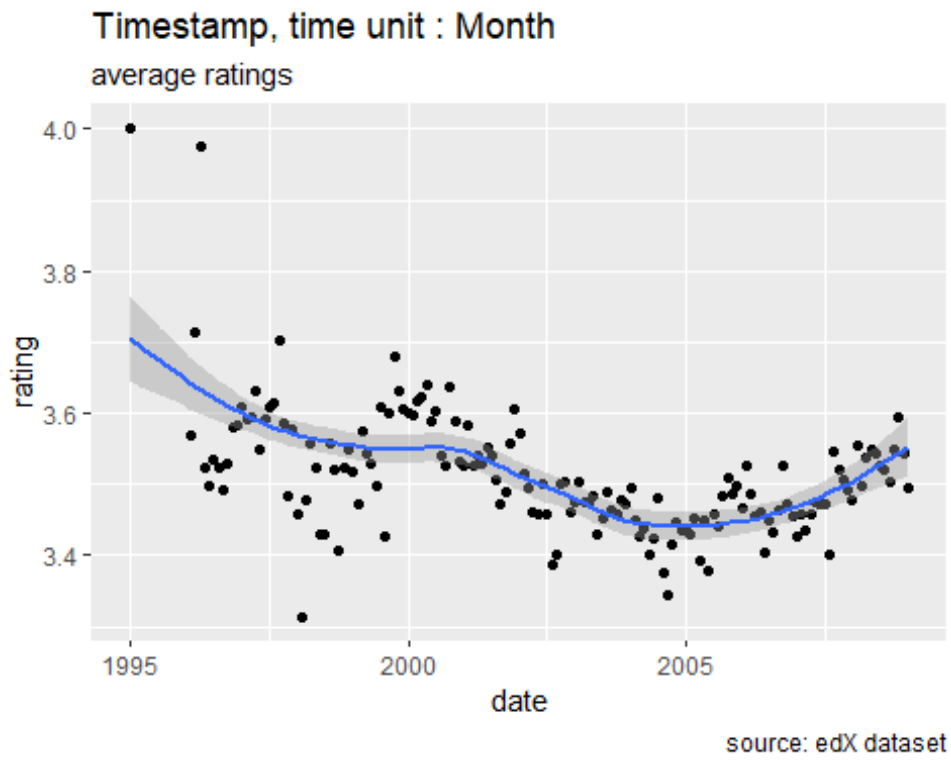
```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
```

```
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : Week")+
  labs(subtitle = "average ratings",
       caption = "source: edX dataset")
```



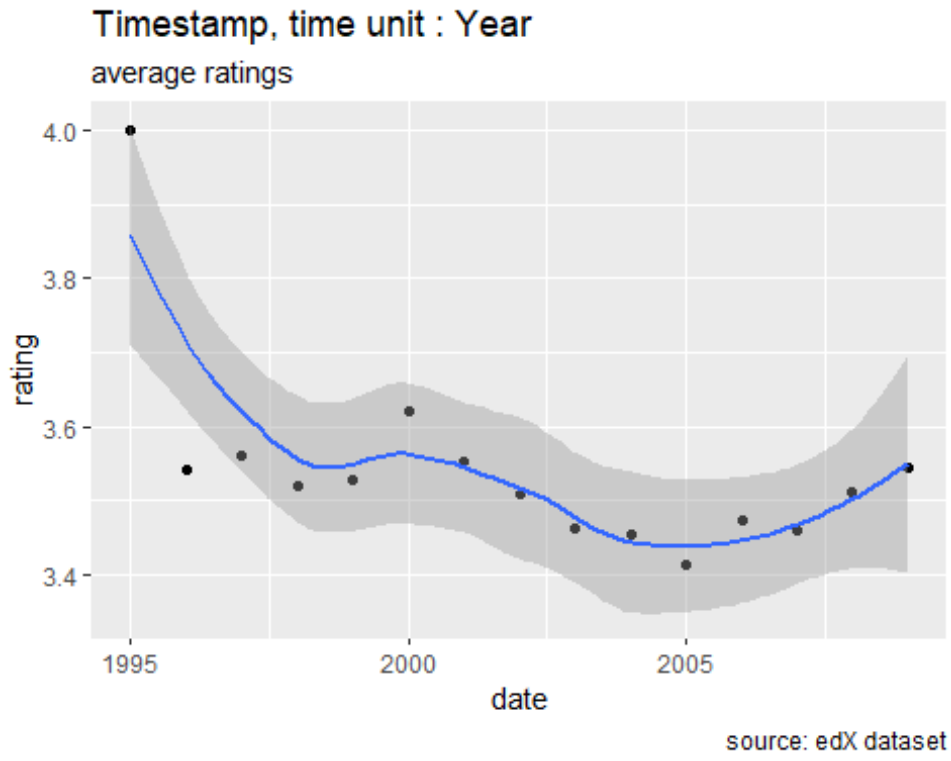
2. Average Ratings during month

```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "month")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : Month")+
  labs(subtitle = "average ratings",
       caption = "source: edX dataset")
```



3. Average Ratings during year

```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "year")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : Year")+
  labs(subtitle = "average ratings",
       caption = "source: edX dataset")
```



V. Modeling

Basic Method

As part of this method, we will build a whole set of models. In a way, this will be a kind of trial and error where the approach will consist in observing the effects or bias of certain variables on the rating prediction. Thus, we will add the effects as we go. Likewise, the regularized version of certain models of this kind will preferably be presented in order to really have a model that is much more robust than the simple version, in other words unregularized.

First of all, we want to define the RMSE function which will serve as a springboard for evaluating the discriminatory or predictive performance of basic models and thereby to choose the one which will prove to be relatively optimal. Mathematically the function is defined as the root mean square of the difference between the individual observations and the model output (predictions).

Defining RMSE function

On R, we codify it as follows:

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

a. Model 1: Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. We start by building the simplest possible recommender system by predicting the same rating for all movies

regardless of user who give it. A model based approach assumes the same rating for all movie with all differences explained by random variation :

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. And note that, in a linear regression, the BLUE[2] estimate that minimizes the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings: *the expected rating*.

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

avg_rmse <- RMSE(validation$rating, mu)
avg_rmse

## [1] 1.061202
```

In the first model, just based on the ratings itself, to minimize the RMSE, the best prediction of ratings for each movie will be the overall average of all ratings. The average rating is mu = 3.51247, and the naive RMSE = 1.0612.

```
rmse_table <- data_frame(Model = "Just the average", RMSE = avg_rmse)

rmse_table %>%
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.061202

b. Model 2: Movie effect model

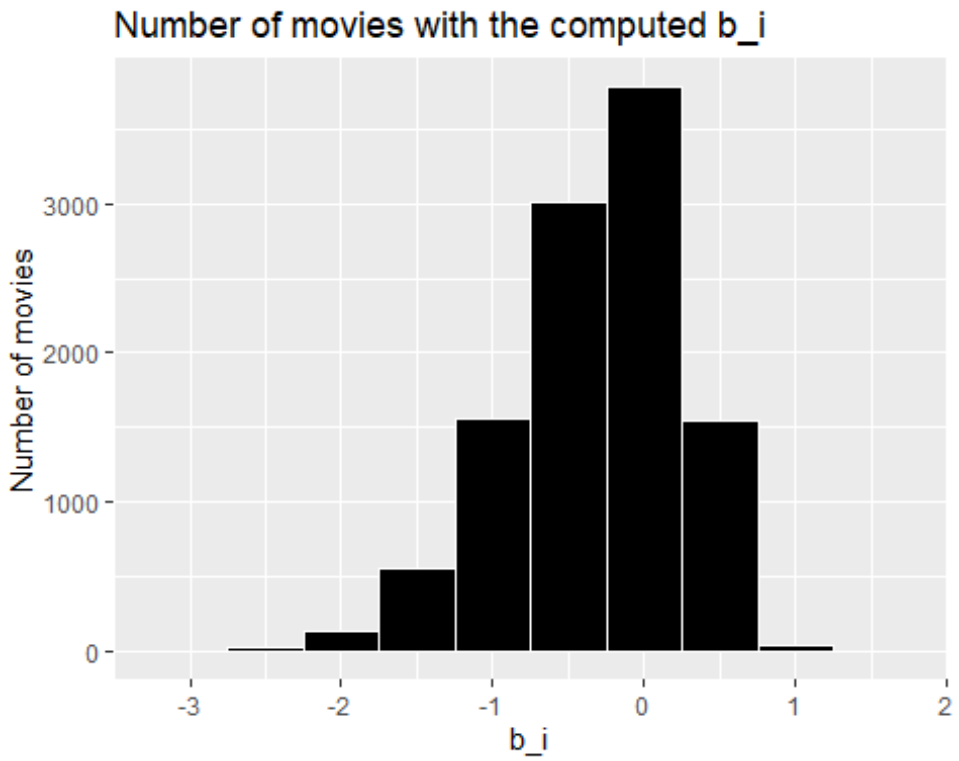
In order to improve the performance of Model 1, we rely on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies 'mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

[2] Best Linear Unbiaised Estimator

Let's just look at the graph below. We see that the histogram is left skewed, implying that more movies have negative effects.

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

#Graph
movie_average %>%
  qplot(b_i, geom ="histogram", bins = 10, data = .,fill= I("black"), color=I
("white"),
ylab = "Number of movies", main = "Number of movies with the computed b_i")
```



Now, let's see what it looks like in terms of prediction. This will allow us to compare with the results obtained above in the first model.

```
predicted_ratings <- mu +  validation %>%
  left_join(movie_average, by='movieId') %>%
  pull(b_i)
model_rmse_ME <- RMSE(predicted_ratings, validation$rating)
rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Movie effect model",
                                   RMSE = model_rmse_ME))
rmse_table %>%
```

```
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

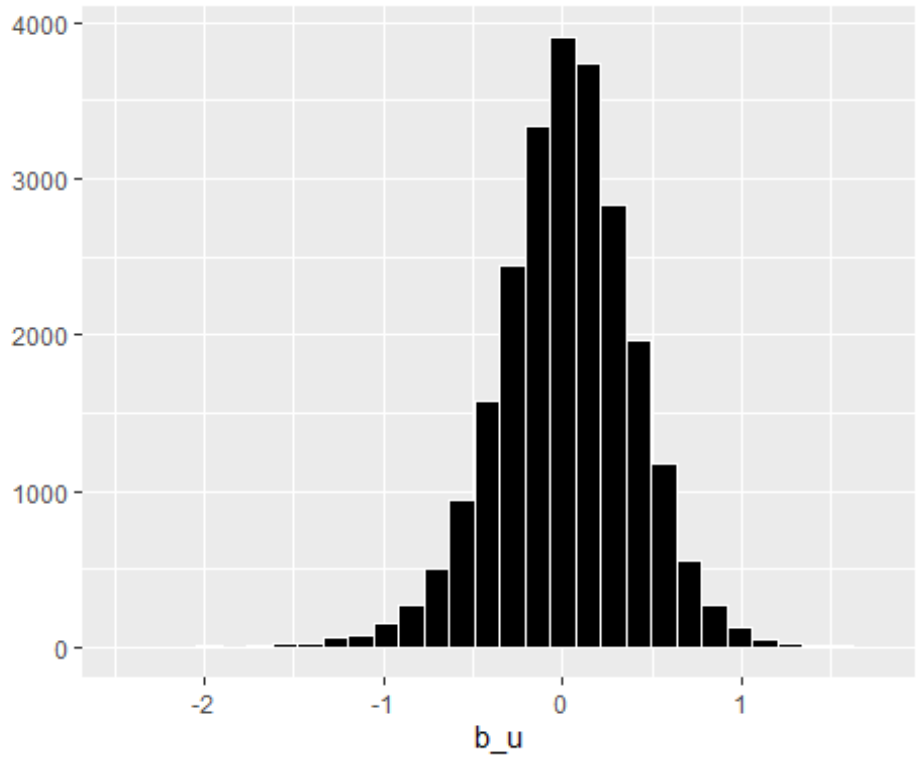
Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , it makes sense to predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average. We can see an improvement but this model does not consider the individual user rating effect.

c. Model 3: Movie and user effect model

We compute the average rating for user μ, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the ratings positively or negatively.

```
user_average<- edx %>%
  left_join(movie_average, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))

#Graph
user_average%>%
  qplot(b_u, geom ="histogram", bins = 30, data = ., fill= I("black"), color=
I("white"))
```



There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be:

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

where b_u is a user-specific effect. 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 2 rather than a 4.

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

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

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

We can now construct predictors and see RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_average, by='movieId') %>%
  left_join(user_average, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_rmse_MUE <- RMSE(predicted_ratings, validation$rating)
rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Movie and user effect model",
```

```
                                    RMSE = model_rmse_MUE))
rmse_table %>%
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

d. Model 4: Regularized Movie Effect Model

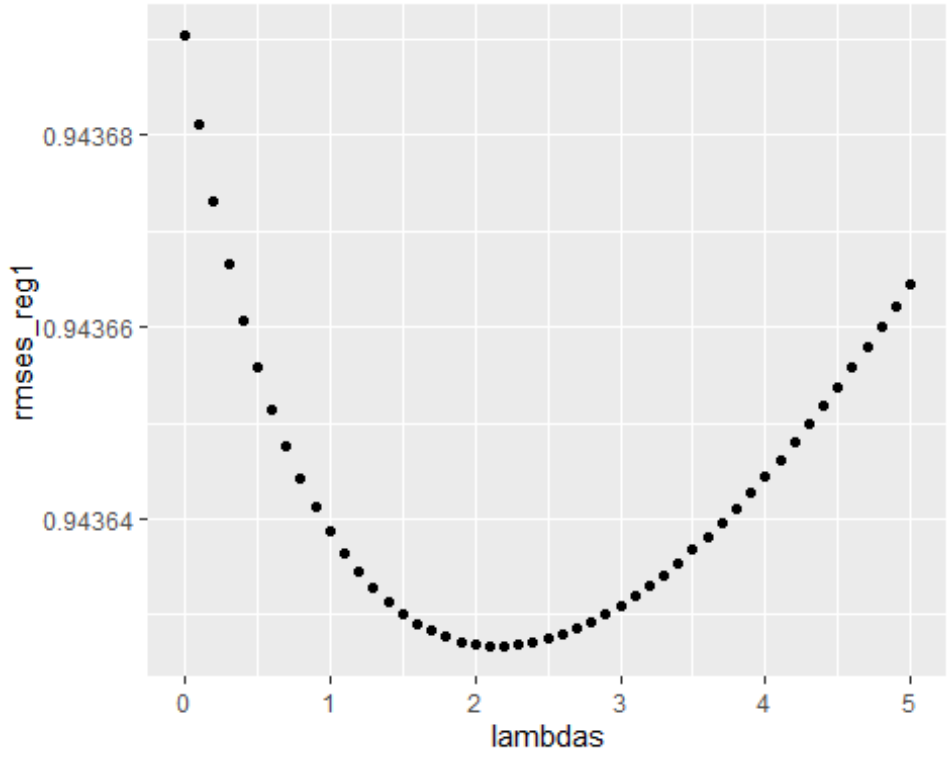
Previously, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we're interested, most of the time, at point estimate, not an interval. For this, we introduce the concept of **regularization**.

The aim here is to control the total variability of the movie effects. A machine learning model could be over-trained if some estimates were from a very small sample size. Regularization technique should be used to take into account the number of ratings made for a specific movie, by adding a larger penalty to estimates from smaller samples. To do this, a parameter lambda will be used. Cross validation within the test set can be performed to optimize this parameter before being applied to the validation set.

So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

```
#Splitting the data into 5 parts
set.seed(1996)

cv_splits <- caret::createFolds(edx$rating, k=5, returnTrain =TRUE)
lambdas <- seq(0, 5, 0.1)
rmses <- matrix(nrow=5,ncol=length(lambdas))

#Perform 5-fold cross validation to determine the optimal lambda
for(k in 1:5) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

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

```
  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)
  just_the_sum <- train_final %>%
    group_by(movieId) %>%
    summarize(s = sum(rating - mu), n_i = n())

  rmses[k,] <- sapply(lambdas, function(l){
    predicted_ratings <- test_final %>%
      left_join(just_the_sum, by='movieId') %>%
      mutate(b_i = s/(n_i+l)) %>%
      mutate(pred = mu + b_i) %>%
      pull(pred)
    return(RMSE(predicted_ratings, test_final$rating))
  })
}

rmses_reg1 <- colMeans(rmses)
qplot(lambdas,rmses_reg1)
```




```
lambda <- lambdas[which.min(rmses_reg1)]

lambda

## [1] 2.2
```

Using the optimized lambda, we can now perform prediction and evaluate the RMSE in the validation set.

```
mu <- mean(edx$rating)
movie_reg_average <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
predicted_ratings_5 <- validation %>%
  left_join(movie_reg_average, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
model_rmse_reg1 <- RMSE(predicted_ratings_5, validation$rating)
rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Regularized Movie Effect Model",
                                   RMSE = model_rmse_reg1))
rmse_table %>%
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized Movie Effect Model	0.9438522

e. Model 5: Regularized Movie & User Effect Model (Version 1)

```
lambdas <- seq(0, 5, 0.1)
rmses_reg <- matrix(nrow=5,ncol=length(lambdas))
#Performing 5-fold cross validation to determine the optimal lambda
for(k in 1:5) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

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

  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

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

```
 rmses_reg[k,] <- sapply(lambdas, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)
    return(RMSE(predicted_ratings, test_final$rating))
  })
}

rmses_reg
```

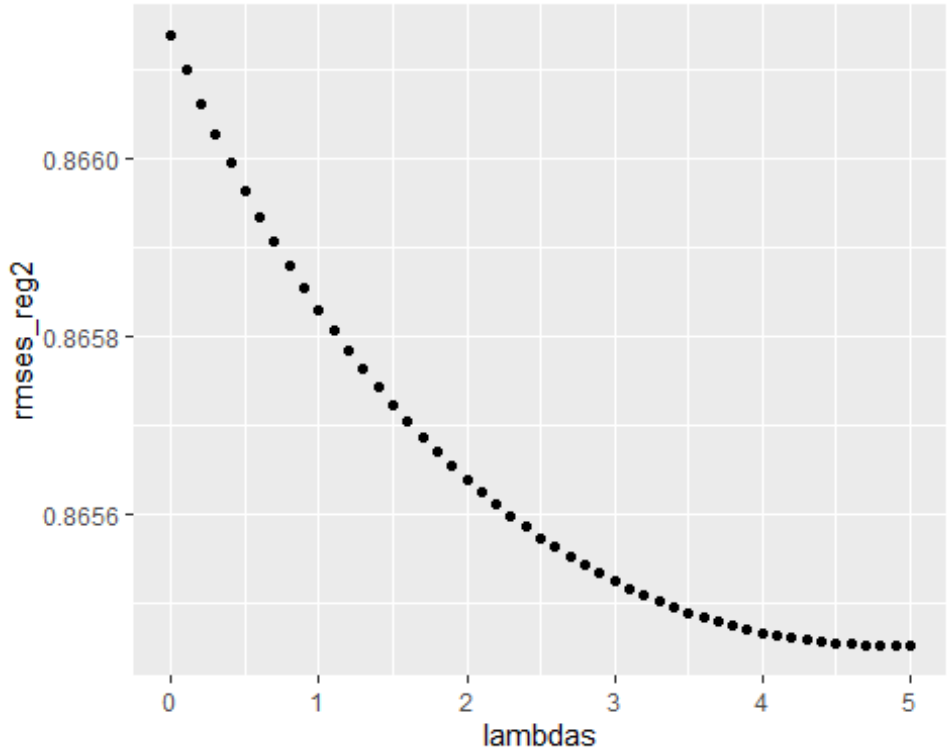
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.8667208 0.8666807 0.8666433 0.8666081 0.8665749 0.8665433 0.8665132
## [2,] 0.8659124 0.8658707 0.8658321 0.8657961 0.8657622 0.8657303 0.8657000
## [3,] 0.8659278 0.8658857 0.8658463 0.8658093 0.8657743 0.8657412 0.8657097
## [4,] 0.8661191 0.8660827 0.8660485 0.8660162 0.8659856 0.8659564 0.8659286
## [5,] 0.8660179 0.8659796 0.8659438 0.8659103 0.8658787 0.8658488 0.8658205
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 0.8664846 0.8664573 0.8664311 0.8664062 0.8663823 0.8663594 0.8663376
## [2,] 0.8656713 0.8656441 0.8656181 0.8655934 0.8655698 0.8655473 0.8655259
## [3,] 0.8656798 0.8656512 0.8656240 0.8655980 0.8655731 0.8655494 0.8655267
## [4,] 0.8659020 0.8658767 0.8658525 0.8658294 0.8658072 0.8657861 0.8657659
## [5,] 0.8657936 0.8657680 0.8657436 0.8657204 0.8656983 0.8656772 0.8656572
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## [1,] 0.8663167 0.8662967 0.8662776 0.8662593 0.8662419 0.8662252 0.8662093
## [2,] 0.8655054 0.8654859 0.8654673 0.8654496 0.8654327 0.8654166 0.8654013
## [3,] 0.8655050 0.8654842 0.8654644 0.8654455 0.8654274 0.8654102 0.8653937
## [4,] 0.8657466 0.8657282 0.8657106 0.8656938 0.8656779 0.8656626 0.8656482
## [5,] 0.8656380 0.8656198 0.8656025 0.8655860 0.8655703 0.8655554 0.8655413
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
## [1,] 0.8661942 0.8661797 0.8661660 0.8661530 0.8661406 0.8661288 0.8661177
## [2,] 0.8653867 0.8653729 0.8653598 0.8653474 0.8653356 0.8653245 0.8653140
## [3,] 0.8653781 0.8653632 0.8653490 0.8653355 0.8653227 0.8653106 0.8652991
## [4,] 0.8656344 0.8656213 0.8656089 0.8655972 0.8655861 0.8655756 0.8655658
## [5,] 0.8655279 0.8655152 0.8655033 0.8654920 0.8654813 0.8654713 0.8654619
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
## [1,] 0.8661072 0.8660973 0.8660879 0.8660792 0.8660709 0.8660633 0.8660561
## [2,] 0.8653041 0.8652948 0.8652861 0.8652780 0.8652704 0.8652634 0.8652569
## [3,] 0.8652883 0.8652780 0.8652684 0.8652593 0.8652508 0.8652429 0.8652355
## [4,] 0.8655565 0.8655478 0.8655397 0.8655321 0.8655251 0.8655186 0.8655126
## [5,] 0.8654531 0.8654449 0.8654372 0.8654301 0.8654236 0.8654176 0.8654120
```

```
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
## [1,] 0.8660494 0.8660433 0.8660376 0.8660324 0.8660277 0.8660234 0.8660196
## [2,] 0.8652508 0.8652453 0.8652403 0.8652357 0.8652317 0.8652280 0.8652248
## [3,] 0.8652286 0.8652223 0.8652164 0.8652110 0.8652061 0.8652017 0.8651978
## [4,] 0.8655071 0.8655020 0.8654975 0.8654934 0.8654898 0.8654866 0.8654838
## [5,] 0.8654070 0.8654025 0.8653985 0.8653949 0.8653918 0.8653891 0.8653869
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
## [1,] 0.8660162 0.8660132 0.8660107 0.8660085 0.8660068 0.8660054 0.8660045
## [2,] 0.8652221 0.8652197 0.8652178 0.8652163 0.8652152 0.8652144 0.8652141
## [3,] 0.8651942 0.8651912 0.8651885 0.8651863 0.8651844 0.8651830 0.8651820
## [4,] 0.8654815 0.8654796 0.8654781 0.8654770 0.8654763 0.8654759 0.8654760
## [5,] 0.8653851 0.8653837 0.8653827 0.8653822 0.8653820 0.8653822 0.8653827
##          [,50]     [,51]
## [1,] 0.8660039 0.8660036
## [2,] 0.8652141 0.8652145
## [3,] 0.8651813 0.8651811
## [4,] 0.8654764 0.8654771
## [5,] 0.8653837 0.8653849
```

```
rmses_reg2 <- colMeans(rmses_reg)
rmses_reg2
```

```
##  [1] 0.8661396 0.8660999 0.8660628 0.8660280 0.8659951 0.8659640 0.8659344
##  [8] 0.8659063 0.8658794 0.8658539 0.8658295 0.8658061 0.8657839 0.8657626
## [15] 0.8657423 0.8657230 0.8657045 0.8656868 0.8656700 0.8656540 0.8656388
## [22] 0.8656243 0.8656105 0.8655974 0.8655850 0.8655733 0.8655622 0.8655517
## [29] 0.8655418 0.8655326 0.8655239 0.8655158 0.8655082 0.8655011 0.8654946
## [36] 0.8654886 0.8654831 0.8654781 0.8654735 0.8654694 0.8654658 0.8654626
## [43] 0.8654598 0.8654575 0.8654556 0.8654540 0.8654529 0.8654522 0.8654518
## [50] 0.8654519 0.8654522
```

```
qplot(lambdas,rmses_reg2)
```




```
lambda <- lambdas[which.min(rmses_reg2)]

lambda

## [1] 4.8
```

Now we use this parameter lambda to predict the validation dataset and evaluate the RMSE.

```
mu <- mean(edx$rating)
b_i_reg1 <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg1 <- edx %>%
    left_join(b_i_reg1, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_reg2 <-
    validation %>%
    left_join(b_i_reg1, by = "movieId") %>%
    left_join(b_u_reg1, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
model_rmse_reg2 <- RMSE(predicted_ratings_reg2, validation$rating)
rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Regularized Movie & User Effect M
odel (Version 1)",
                                   RMSE = model_rmse_reg2))
rmse_table %>%
```

```
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized Movie Effect Model	0.9438522
Regularized Movie & User Effect Model (Version 1)	0.8648195

f. Model 6: Regularized Movie & User Effect Model (Version 2)

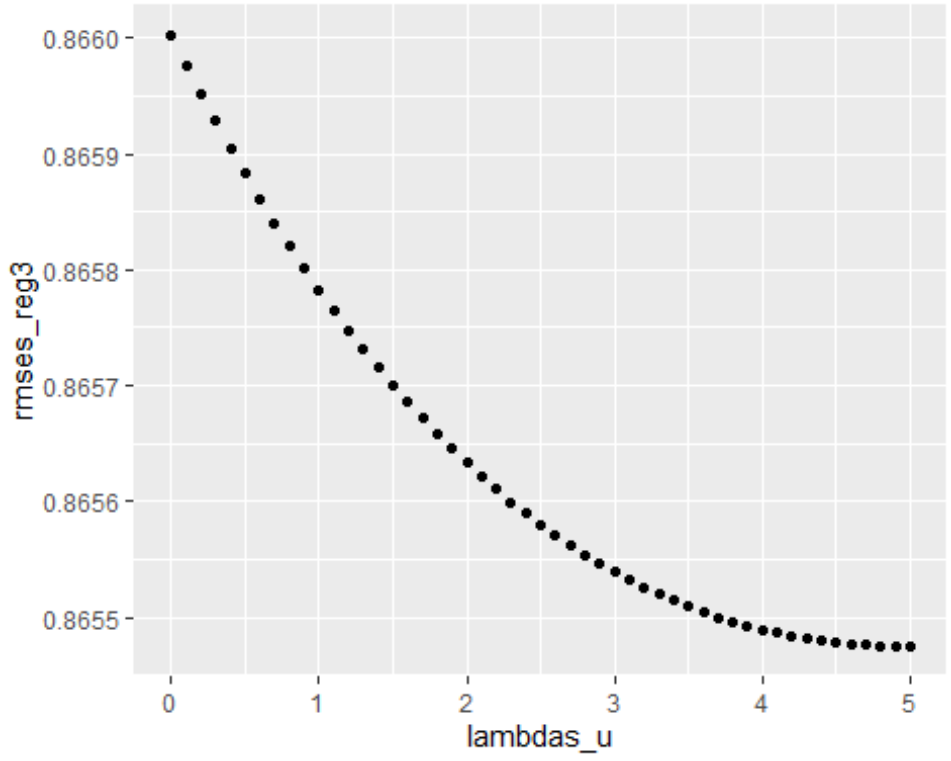
In this model, instead of optimizing the same lambda for both user and movie effects, we use differents lambdas. We fix a lambda for movie using the value we got in model 5 (lambda_i=2.2), and optimize the lambda by a cross-validation process for user (lambda_u).

```
lambda_i <- 2.2
lambdas_u <- seq(0, 5, 0.1)
rmses_op <- matrix(nrow=5,ncol=length(lambdas_u))

#Performing 5-fold cross validation to determine the optimal lambda
for(k in 1:5) {
  train_set <- edx[cv_splits[[k]],]
  test_set <- edx[-cv_splits[[k]],]

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

  removed <- anti_join(test_set, test_final)
  train_final <- rbind(train_set, removed)

  mu <- mean(train_final$rating)

  rmses_op[k,] <- sapply(lambdas_u, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+lambda_i))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
```

```
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)
    return(RMSE(predicted_ratings, test_final$rating))
  })
}
rmses_op
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.8665889 0.8665624 0.8665368 0.8665121 0.8664882 0.8664651 0.8664428
## [2,] 0.8657638 0.8657383 0.8657136 0.8656897 0.8656667 0.8656445 0.8656231
## [3,] 0.8657848 0.8657575 0.8657312 0.8657057 0.8656811 0.8656573 0.8656343
## [4,] 0.8659918 0.8659672 0.8659435 0.8659207 0.8658986 0.8658774 0.8658570
## [5,] 0.8658820 0.8658577 0.8658342 0.8658115 0.8657897 0.8657686 0.8657483
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 0.8664213 0.8664005 0.8663805 0.8663612 0.8663426 0.8663246 0.8663074
## [2,] 0.8656024 0.8655825 0.8655633 0.8655449 0.8655271 0.8655100 0.8654936
## [3,] 0.8656121 0.8655908 0.8655701 0.8655502 0.8655311 0.8655126 0.8654948
## [4,] 0.8658373 0.8658183 0.8658001 0.8657826 0.8657657 0.8657496 0.8657340
## [5,] 0.8657288 0.8657100 0.8656920 0.8656746 0.8656579 0.8656419 0.8656266
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## [1,] 0.8662908 0.8662748 0.8662595 0.8662448 0.8662306 0.8662171 0.8662041
## [2,] 0.8654778 0.8654626 0.8654481 0.8654341 0.8654207 0.8654080 0.8653957
## [3,] 0.8654777 0.8654612 0.8654454 0.8654302 0.8654157 0.8654017 0.8653883
## [4,] 0.8657192 0.8657049 0.8656913 0.8656782 0.8656657 0.8656538 0.8656424
## [5,] 0.8656118 0.8655977 0.8655843 0.8655713 0.8655590 0.8655473 0.8655361
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
## [1,] 0.8661916 0.8661797 0.8661684 0.8661575 0.8661472 0.8661374 0.8661280
## [2,] 0.8653840 0.8653729 0.8653623 0.8653522 0.8653426 0.8653334 0.8653248
## [3,] 0.8653754 0.8653632 0.8653514 0.8653402 0.8653296 0.8653194 0.8653097
## [4,] 0.8656316 0.8656213 0.8656115 0.8656023 0.8655935 0.8655852 0.8655773
## [5,] 0.8655254 0.8655152 0.8655056 0.8654965 0.8654879 0.8654797 0.8654720
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
## [1,] 0.8661191 0.8661107 0.8661027 0.8660952 0.8660881 0.8660815 0.8660752
## [2,] 0.8653166 0.8653089 0.8653016 0.8652948 0.8652884 0.8652824 0.8652768
## [3,] 0.8653005 0.8652918 0.8652836 0.8652758 0.8652685 0.8652616 0.8652551
## [4,] 0.8655700 0.8655631 0.8655566 0.8655505 0.8655449 0.8655397 0.8655349
## [5,] 0.8654648 0.8654581 0.8654517 0.8654458 0.8654404 0.8654353 0.8654306
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
## [1,] 0.8660694 0.8660639 0.8660589 0.8660542 0.8660499 0.8660460 0.8660425
## [2,] 0.8652716 0.8652668 0.8652624 0.8652584 0.8652547 0.8652514 0.8652485
## [3,] 0.8652490 0.8652434 0.8652382 0.8652333 0.8652288 0.8652247 0.8652210
## [4,] 0.8655304 0.8655264 0.8655228 0.8655195 0.8655166 0.8655140 0.8655118
## [5,] 0.8654264 0.8654225 0.8654190 0.8654159 0.8654131 0.8654107 0.8654087
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
```

```
## [1,] 0.8660393 0.8660364 0.8660339 0.8660317 0.8660298 0.8660283 0.8660270
## [2,] 0.8652459 0.8652437 0.8652417 0.8652402 0.8652389 0.8652379 0.8652373
## [3,] 0.8652177 0.8652147 0.8652120 0.8652097 0.8652077 0.8652061 0.8652047
## [4,] 0.8655099 0.8655084 0.8655072 0.8655063 0.8655057 0.8655054 0.8655055
## [5,] 0.8654070 0.8654056 0.8654046 0.8654038 0.8654034 0.8654033 0.8654035
##          [,50]     [,51]
## [1,] 0.8660261 0.8660255
## [2,] 0.8652369 0.8652369
## [3,] 0.8652037 0.8652030
## [4,] 0.8655058 0.8655064
## [5,] 0.8654040 0.8654048

rmses_reg3 <- colMeans(rmses_op)
rmses_reg3

##  [1] 0.8660023 0.8659766 0.8659519 0.8659279 0.8659049 0.8658826 0.8658611
##  [8] 0.8658404 0.8658204 0.8658012 0.8657827 0.8657649 0.8657477 0.8657313
## [15] 0.8657155 0.8657003 0.8656857 0.8656717 0.8656584 0.8656456 0.8656333
## [22] 0.8656216 0.8656105 0.8655999 0.8655897 0.8655801 0.8655710 0.8655624
## [29] 0.8655542 0.8655465 0.8655392 0.8655324 0.8655260 0.8655201 0.8655145
## [36] 0.8655094 0.8655046 0.8655002 0.8654963 0.8654926 0.8654894 0.8654865
## [43] 0.8654839 0.8654817 0.8654799 0.8654783 0.8654771 0.8654762 0.8654756
## [50] 0.8654753 0.8654753

qplot(lambdas_u,rmses_reg3)
```



```
lambda_u <-lambdas_u[which.min(rmses_reg3)]
lambda_u

## [1] 4.9
```

Using the lambda_i (fixed lambda) and lambda_u we determined, we generated the prediction model and evaluated the RMSE on the validation set.

```
lambda_i <- 2.2
lambda_u <- 5
mu <- mean(edx$rating)
b_i_reg2 <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_i))
b_u_reg2 <- edx %>%
  left_join(b_i_reg2, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_u))
predicted_ratings_reg3 <-
  validation %>%
  left_join(b_i_reg2, by = "movieId") %>%
  left_join(b_u_reg2, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_rmse_reg3 <- RMSE(predicted_ratings_reg3, validation$rating)
rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Regularized Movie & User Effect M
odel (Version 2)",
                                   RMSE = model_rmse_reg3 ))
rmse_table %>%
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized Movie Effect Model	0.9438522
Regularized Movie & User Effect Model (Version 1)	0.8648195
Regularized Movie & User Effect Model (Version 2)	0.8648503

Advanced Method

Model 7: Low-rank Matrix Factorization

Below, we will use the mathematical model exposed in the methodology part of our work. This is the advanced method model built partly on the basis of the basic method. Let us recall rightly that the advanced method tries to explain the user-movie interactions only. It does that by first covering all the discussed effects using the basic model prediction $\widehat{r_{i,j}}$. The implemented low-rank matrix factorization PSGD algorithm works as follows:

- First, we will use the Matrix Factorization techniques to model the residual of the baseline model which achieves the lowest RMSE;
- Then, the matrices P and Q will be created and initialized to standard normal values with zero mean and σ standard deviation parameter.
- For niter(argument which represents the number of iterations) random samples compute the gradient update derived below and update P and Q accordingly;
- Store final P and Q as part of the fit object and use them to make predictions.

$$\epsilon_{u,i} = r_{u,i} - (\underbrace{\hat{r}_{u,i}}_{\text{prediction using BM}} + P_u^T Q_i)$$

$$LOSS_{\text{AM}} = \sum_{u,i} \epsilon_{u,i}^2 + \lambda_{\text{AM}} \left(\sum_u \| P_u \|^2 + \sum_i \| Q_i \|^2 \right)$$

$$\underset{P,Q}{\text{argmin}} \sum_{u,i} \epsilon_{u,i}^2 + \lambda_{\text{AM}} \left(\sum_u \| P_u \|^2 + \sum_i \| Q_i \|^2 \right)$$

$$\frac{\partial LOSS_{\text{AM}}}{\partial P_u} = 2\epsilon_{u,i} Q_i + 2\lambda P_u = 2(\epsilon_{u,i} Q_i + \lambda P_u) \Rightarrow \Delta P_u = \gamma(\epsilon_{u,i} Q_i + \lambda P_u)$$

$$\frac{\partial LOSS_{\text{AM}}}{\partial Q_i} = 2\epsilon_{u,i} P_u + 2\lambda Q_i = 2(\epsilon_{u,i} P_u + \lambda Q_i) \Rightarrow \Delta Q_i = \gamma(\epsilon_{u,i} P_u + \lambda Q_i)$$

Where AM stands for Advanced Method, and BM for Basic Method. Note that γ is the learning rate. Therefore, in every SGD step the following P and Q updates are executed:

$$P_u = P_u + \gamma(\epsilon_{u,i} Q_i + \lambda P_u)$$
$$Q_i = Q_i + \gamma(\epsilon_{u,i} P_u + \lambda Q_i)$$

We see that this model has the following possible hyper-parameters: K number of latent dimensions, λ regularization that penalizes large values for P and Q, γ the learning rate (lrate as argument in our function) and possibly σ the standard deviation corresponding to the normal distribution used to initialize P and Q.

Best baseline model

We compare the RMSE of all the baseline models and decide to use the 5th (Regularized Movie & User Effect Model) prior to further modeling because it achieves the least RMSE.

As said earlier, we're about to calculate the residuals of the baseline model now, and then perform matrix factorization on that. And then we will continue with the whole procedure detailed above. And in the end, we will use the set validation to evaluate the final model's performance.

```
edx_residual <- edx %>%
  left_join(b_i_reg1, by = "movieId") %>%
  left_join(b_u_reg1, by = "userId") %>%
  mutate(residual = rating - mu - b_i - b_u) %>%
  select(userId, movieId, residual)
head(edx_residual)
```

```
##    userId movieId  residual
## 1:      1     122 0.7995875
## 2:      1     185 0.5301408
## 3:      1     292 0.2415692
## 4:      1     316 0.3098884
## 5:      1     329 0.3220965
## 6:      1     355 1.1708070
```

Performing the matrix factorization

Now let's use the recosystem library to perform the Low-Rank Matrix Factorization on the residuals. Both training and validation sets need to be organized to 3 columns: user, item (movies), value (ratings or residuals). Then they need to be transformed into matrix format. Next we write these datasets into hard disk, which will later be assigned to train_set and valid_set to build the "recosystem". A recommender object r will be built using Reco() in the recosystem package and parameters trained using the train_set.

While modeling the residuals of Model 5, we add up the base prediction of said model and the residuals predicted here to get the final prediction for the validation set. Next, the parameters mentioned above will be used to build the prediction model.

```
#Matrix format
edx_for_mf <- as.matrix(edx_residual)
validation_for_mf <- validation %>%
  select(userId, movieId, rating)
validation_for_mf <- as.matrix(validation_for_mf)
```

```r
#Writing edx_for_mf and validation_for_mf tables on disk
write.table(edx_for_mf , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(validation_for_mf, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

#Using data_file() function to specify a data set from a file in the hard disk.
set.seed(1996)
train_set <- data_file("trainset.txt")
valid_set <- data_file("validset.txt")

#Building a recommender object
r <-Reco()

#Tuning training set
opts <- r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                      costp_l1 = 0, costq_l1 = 0,
                                      nthread = 4, 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] 0.7933294
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8076132
## 2   20        0     0.01        0     0.01   0.1 0.8112085
## 3   30        0     0.01        0     0.01   0.1 0.8204287
```

```
## 4   10        0     0.10        0     0.01   0.1 0.8051662
## 5   20        0     0.10        0     0.01   0.1 0.8013011
## 6   30        0     0.10        0     0.01   0.1 0.8024689
## 7   10        0     0.01        0     0.10   0.1 0.8038664
## 8   20        0     0.01        0     0.10   0.1 0.7953881
## 9   30        0     0.01        0     0.10   0.1 0.7933294
## 10  10        0     0.10        0     0.10   0.1 0.8237863
## 11  20        0     0.10        0     0.10   0.1 0.8239370
## 12  30        0     0.10        0     0.10   0.1 0.8238403
## 13  10        0     0.01        0     0.01   0.2 0.8108484
## 14  20        0     0.01        0     0.01   0.2 0.8229232
## 15  30        0     0.01        0     0.01   0.2 0.8393999
## 16  10        0     0.10        0     0.01   0.2 0.8063319
## 17  20        0     0.10        0     0.01   0.2 0.8053925
## 18  30        0     0.10        0     0.01   0.2 0.8071878
## 19  10        0     0.01        0     0.10   0.2 0.8030703
## 20  20        0     0.01        0     0.10   0.2 0.7991247
## 21  30        0     0.01        0     0.10   0.2 0.7990403
## 22  10        0     0.10        0     0.10   0.2 0.8222556
## 23  20        0     0.10        0     0.10   0.2 0.8224637
## 24  30        0     0.10        0     0.10   0.2 0.8211814
```

```
#Training the recommender model
r$train(train_set, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8564   6.9327e+06
##    1       0.8338   6.4404e+06
##    2       0.8165   6.2691e+06
##    3       0.7992   6.1031e+06
##    4       0.7841   5.9603e+06
##    5       0.7715   5.8445e+06
##    6       0.7612   5.7555e+06
##    7       0.7528   5.6835e+06
##    8       0.7456   5.6246e+06
##    9       0.7393   5.5757e+06
##   10       0.7338   5.5317e+06
##   11       0.7288   5.4943e+06
##   12       0.7245   5.4622e+06
##   13       0.7206   5.4324e+06
##   14       0.7170   5.4067e+06
##   15       0.7138   5.3838e+06
##   16       0.7108   5.3615e+06
##   17       0.7082   5.3435e+06
##   18       0.7058   5.3271e+06
##   19       0.7036   5.3106e+06
```

```
# Making prediction on validation set and calculating RMSE:
pred_file <- tempfile()
r$predict(valid_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\LUCAIN~1\AppData\Local\Temp\RtmpaI
vsey\file65c025a1d86

predicted_residuals_mtx_fact <- scan(pred_file)

predicted_ratings_mtx_fact <- predicted_ratings_reg2 + predicted_residuals_mt
x_fact
rmse_mtx_fact <- RMSE(predicted_ratings_mtx_fact,validation$rating)

rmse_table <- bind_rows(rmse_table,
                        data_frame(Model="Matrix Factorization",
                                   RMSE = rmse_mtx_fact))


rmse_table %>%
  knitr::kable()%>%
    kable_styling(bootstrap_options = "striped" , full_width = F , position =
"center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="c
enter") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="black")
```

Model	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized Movie Effect Model	0.9438522
Regularized Movie & User Effect Model (Version 1)	0.8648195
Regularized Movie & User Effect Model (Version 2)	0.8648503
Matrix Factorization	0.7868063

VI. Conclusion

After having trained 6 different models as part of the basic method, we retained the one which had the smallest RMSE. This RMSE stands at *0.8648195*. This is **Model 5** which was, like all the others for that matter, trained on the training dataset we call *edx set*, then tested on the test dataset we call *validation set*. This model combines average ratings, user and movies effects. It has been regularized as the name suggests in order to avoid the recurring problem of overtraining. With this model alone, we have obtained a smallest RMSE than the threshold — which was fixed at *0.86490*. Further, by applying the technique of Low-Rank Matrix Factorization based on the baseline model, we got an even smallest RMSE. This gives us an extraordinary one equal to *0.7865805*. If we calculate the percentage decrease of RMSE values between the two methods, we could see that the advanced method (Low-Rank Matrix Factorization) shows a decrease of more than 9%. In the end, the advanced method is a very powerful technique. His

performance further improves the rating predictions. We therefore retain it as the mathematical model of our recommendation engine for this project. However, we believe that there may be much better performing models that can decrease RMSE to a very low level. We only tried 7, actually. Moreover, these are very time-consuming models at the R infrastructure level. We hope to resolve all that issues in a further project.

References

- Irizzary,R., 2018, Introduction to Data Science, github page, https://rafalab.github.io/dsbook/
- Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In Recommender systems handbook (pp. 257-297). Springer, Boston, MA.
- Saranya, K. G., Sadasivam, G. S., & Chandralekha, M. (2016). Performance comparison of different similarity measures for collaborative filtering technique. Indian J. Sci. Technol, 9(29), 1-8.
- Aggarwal, C. C. (2016). Recommender systems (pp. 1-28). Cham: Springer International Publishing.
- Koren, Y. (2009). The bellkor solution to the netflix grand prize. Netflix prize documentation, 81, 1-10.
- http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/
- https://medium.com/sfu-cspmp/recommendation-systems-collaborative-filtering-using-matrix-factorization-simplified-2118f4ef2cd3
- https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b