MovieLens Capstone Project HarvardX

Farid Nazarov

3 December 2019

1.Introduction and a quick summary of the Project

This is the Capstone project for the Data Analysis Certificate from Harvard University.The MovieLens data set was collected by GroupLens Research. In this project 10 Million Movie Rating will be analyzed and make a new recommendation for the new film that will be graded by users. In order to do that I will develop a model. I will use penalized least square method after making some data preprocessing on MovieLens data set and find the best suitable model that reduce my forecast error. The aim of the model is to minimise the errors so called Root Square of the Mean Errors.We can show *RMSE* mathematically as follow:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations. 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 main idea is to find the model that minimize *RMSE* and predict better y.

1.1 Importing Essential Libraries

In my Data Science project, I will make use of these packages: `recommenderlab`, `ggplot2`, `data.table` , `caret` , `dplyr` and `tidyverse`. Let's download the package if necessary and make it ready to use

```
#Download the package if necessary
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")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(recommenderlab)) install.packages("recommenderlab", repos = "http://cran.us.r-project.org")
```

Loading required libraries

```
#Open the library to use
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(tinytex)
library(ggplot2)
```

```
library(recommenderlab)
library(reshape2)
```

2. Dataset

MovieLens Dataset consists of 10.000.054 ratings applied over 10.677 movies. We get out data from GroupLens internet site. If you want to download data manually here is the link. After downloading the data and clean data into tidy format. Finally we need to get the data `movielens` with column names `userId`, `movieId`, `rating`, `genres`, `title` and `timestamp` which will e usefull to build my model.

2.1 Data Visualisation

Let's explore our data that we have dowloaded from Grouplens intetnet site.We can overview the summary of the movies using the summary() function. We will also use the head() function to print the first six lines of movie_data

```
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35740   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The movielens dataset has more than 10 million ratings. Each rating comes with a `userId`, a `movieId`, the `rating`, a `timestamp` and information about the movie like `title` and `genre`. As we see users also can give 0.5 rating. It means that ratings is not as we used to 1 to 5 rather it is increment of 0.5. Min rating is 0.5 and also there is no rating 0.

Let's set seed to 1 and make `edx` set from the `movielens` that we have downloaded. We can split data 90% to `edx` and 10% to `validation`. Of course we can take more than 10% of the data in order to make `edx` set but 10% is enough to make a good model from 10 million data.

```
#Creating of edx set which will be 90% of the movielens
set.seed(1)
edx_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-edx_index, ]
temp <- movielens[edx_index, ]
```

So using `semi_join` function we put userId and movieId in validation set also in edx set and get rid of the unnecessary data to reduce the capacity.

```
# Create validation set which will be 10% of movielens and remove unnecessary data
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

2.2 Edx Data Summary

Let's see some charachteristics of *edx* data.

```
# Unique User ID
n_distinct(edx$userId)
```

```
## [1] 69878
```

```
#Unique Movie IDs
n_distinct(edx$movieId)
```

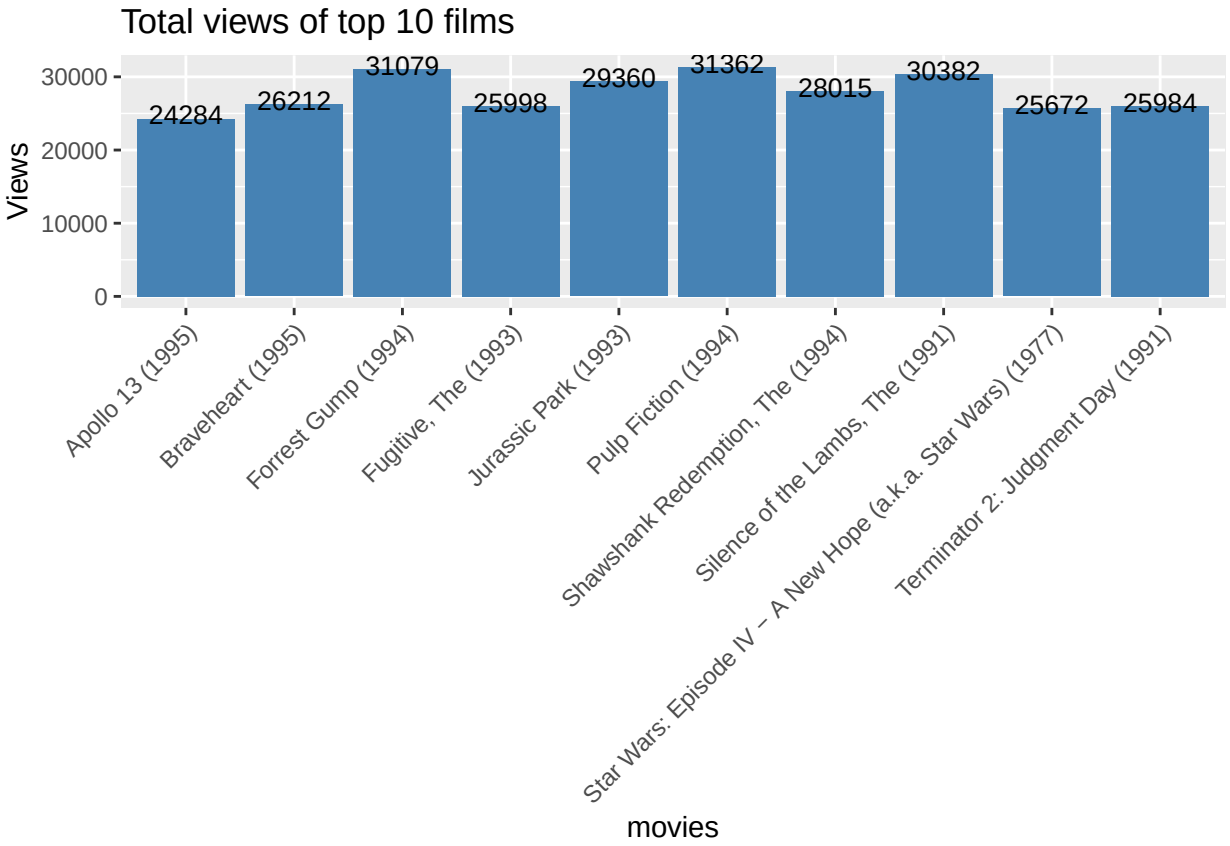
```
## [1] 10677
```

As we see from the output above there are 69.878 and 10.669 movies. But there are some users only gave one rating. So in there next data pre-preration system I will filter users that at least gave 50 ratings.

2.2.1 Most Viewed Movies

Let's explore the most viewed movies in our dataset. We will first count the number of views in a film and then organize them in a table that would group them in descending order.

```
# Most rated top 10 films
most_viewed<- edx %>% group_by(movieId,title )%>%
  summarise(views=sum(n())) %>%
  arrange(desc(views)) %>%
  ungroup()%>%
  top_n(10)
# Graph of top ten rated films
most_viewed %>% ggplot(aes(x=title,y = views))+
  geom_bar(stat = "identity", fill = "steelblue")+
  geom_text(aes(label=views), vjust = 0.3, size = 3.5)+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
  xlab ("movies") + ylab("Views") + ggtitle("Total views of top 10 films")
```



From the above bar-plot, we observe that Pulp Fiction is the most-watched film followed by Forrest Gump.

2.3 Data Preparation

For finding useful data in our dataset, I have set the threshold for the minimum number of users who have rated a film at lease 50 times. This is also same for minimum number of views that are per film. This way, I have filtered a list of watched films from least-watched ones. We can see the number of unique users that provided ratings and how many unique movies were rated:

```
# Nr of unique movies in edx set
edx %>%
summarize(n_users = n_distinct(userId),
n_movies = n_distinct(movieId))
```

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

We want to select users who has rated more than 50 times and films which also at least rated more than 50 times.

```
# Select Users who has given more than 50 movies
users_over_50 <- edx %>%
  group_by(userId) %>%
  summarise(n = n()) %>%
  filter(n > 50) %>%
  ungroup() %>%
```

```
  left_join(edx, by = "userId") %>%
  group_by(userId) %>%
  summarise(User_Rated = n())

# Select movies, which at least rated more than 50 films
movies_over_50 <- edx %>%
  group_by(movieId) %>%
  summarise(n = n()) %>%
  filter(n > 50) %>%
  ungroup() %>%
  left_join(edx, by = "movieId") %>%
  group_by(movieId) %>%
  summarise(Movies_Rated = n())

#We filter only users who gave more than 50 rating and movies which at least rated 51 times
edx <- edx %>%
  left_join(users_over_50, by = "userId") %>%
  left_join(movies_over_50, by = "movieId") %>%
  filter(!is.na(User_Rated), !is.na(Movies_Rated))

# How many users rated more than 50 times
unique_users = n_distinct(users_over_50$userId)
unique_movies = n_distinct(movies_over_50$movieId)
```

From the above output of *users_over_50*, we observe that there are 40151 users and `unique_movies` movies as opposed to the previous 69878 users and 10669 films.

2.4 Premier Year of the Movies

We see the movies has its premier year in it is titel column.

```
head(edx)
```

```
##   userId movieId rating timestamp
## 1      5       1      1 857911264
## 2      5       7      3 857911357
## 3      5      25      3 857911265
## 4      5      28      3 857913507
## 5      5      30      5 857911752
## 6      5      32      5 857911264
##                                                  title
## 1                                     Toy Story (1995)
## 2                                       Sabrina (1995)
## 3                                Leaving Las Vegas (1995)
## 4                                      Persuasion (1995)
## 5 Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
## 6                      12 Monkeys (Twelve Monkeys) (1995)
##                                        genres User_Rated Movies_Rated
## 1 Adventure|Animation|Children|Comedy|Fantasy         74        23790
## 2                             Comedy|Romance         74         7259
## 3                              Drama|Romance         74        11851
## 4                              Drama|Romance         74         1654
```

```
## 5                                Crime|Drama         74          772
## 6                              Sci-Fi|Thriller         74        21891
```

So in order to detect this year and put it into a seperate column I will use regex functions to detect the premier years. After detecting premier year we can easily find the age of the movie.

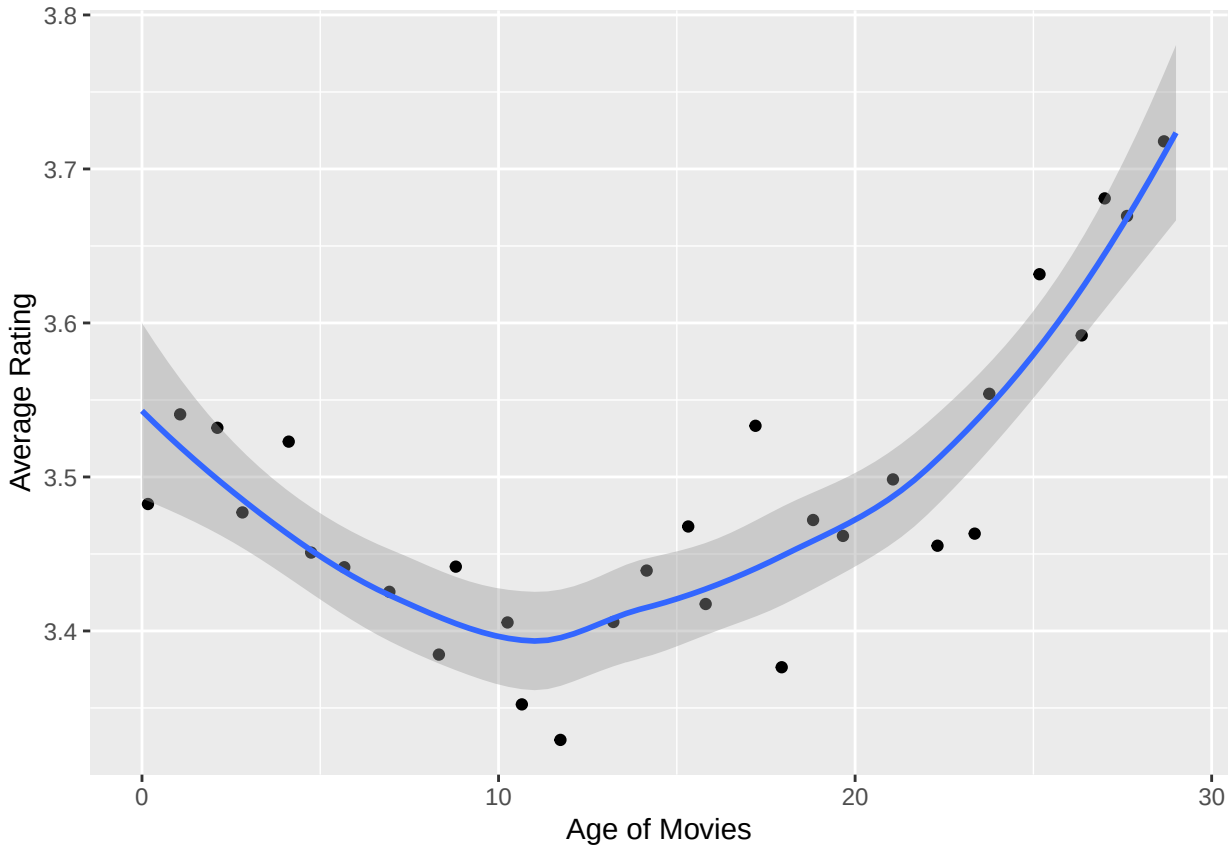
```
# First patterct detect premier year with bracket and second years in this bracket
pattern_1 <- "\\(\\d{4}"
pattern_2 <- "\\d{4}"
strings <- regmatches(edx$title, regexpr(pattern_1, edx$title))
premier_year <- as.numeric(regmatches(strings, regexpr(pattern_2, strings)))
edx <- edx %>% mutate(premier_year = premier_year,
                                  age_of_movie = max(premier_year) - premier_year)
head(edx)
```

```
##   userId movieId rating timestamp
## 1      5       1      1 857911264
## 2      5       7      3 857911357
## 3      5      25      3 857911265
## 4      5      28      3 857913507
## 5      5      30      5 857911752
## 6      5      32      5 857911264
##                                                 title
## 1                                   Toy Story (1995)
## 2                                     Sabrina (1995)
## 3                               Leaving Las Vegas (1995)
## 4                                     Persuasion (1995)
## 5 Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
## 6                    12 Monkeys (Twelve Monkeys) (1995)
##                                       genres User_Rated Movies_Rated
## 1 Adventure|Animation|Children|Comedy|Fantasy         74        23790
## 2                              Comedy|Romance         74         7259
## 3                               Drama|Romance         74        11851
## 4                               Drama|Romance         74         1654
## 5                                 Crime|Drama         74          772
## 6                              Sci-Fi|Thriller         74        21891
##   premier_year age_of_movie
## 1         1995           13
## 2         1995           13
## 3         1995           13
## 4         1995           13
## 5         1995           13
## 6         1995           13
```

So we see in the column `age_of_movies` the age of the movies. Let's see if there is a correlation between age of the year and the rating of the movies. There are some movies that is very old and it can make some noise in our analysis. So I will filter the movies that are less than 40 years old.

```
#avg ratings vs avg of movies
avg_age_rating <- edx %>%
  filter(age_of_movie < 30) %>%
  group_by(age_of_movie) %>%
  summarise(mean_rated=mean(rating))
```

```r
#Graph of age of movie and rating relationship
avg_age_rating%>%
  ggplot(aes(age_of_movie,mean_rated))+
  geom_jitter()+
  geom_smooth()+
  xlab("Age of Movies") + ylab("Average Rating")
```




```r
  ggtitle("Average Rating")
```

```
## $title
## [1] "Average Rating"
##
## attr(,"class")
## [1] "labels"
```

So what we see there is a clear relationship between the age of the movie and the rating. More recent movies get more user ratings. Much of the old movies get few ratings, whereas newer movies, especially in the 90s get far more ratings. I will try to use this relationship in my model. To make sure we need to look a simple linear correlation between movie age and mean value of the movies.

```r
# Linear Regression of age of movie against rating
lm_mean_age= lm(mean_rated~age_of_movie,data=avg_age_rating)
summary(lm_mean_age)
```

```
##
```

```
## Call:
## lm(formula = mean_rated ~ age_of_movie, data = avg_age_rating)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.14401 -0.06381 -0.01734  0.07107  0.14886
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.405860   0.029814 114.237  < 2e-16 ***
## age_of_movie 0.005625   0.001766   3.186  0.00353 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0837 on 28 degrees of freedom
## Multiple R-squared:  0.266,  Adjusted R-squared:  0.2398
## F-statistic: 10.15 on 1 and 28 DF,  p-value: 0.003529
```

So we see `p-value` is statistically significant.

3. Methods and Analysis

In Data Science we make in generall three data sets one is test one is training and another is validation set. So let's make a new test and training set from `edx` set that we have created.

3.1 Splitting Test and Train Set

```
#Splitting edx set into test and train set
set.seed(1)
test_index_edx <- createDataPartition(y = edx$rating, p = 0.2, list = FALSE)
test_set <- edx[ test_index_edx, ]
train_set <- edx[ -test_index_edx, ]
# Make sure userId and movieId in validation set are also in edx set
test_set <- test_set%>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3.2 A first model

So in the next part of this paper I will try to find a best suitable model to minimize the RMSE. In order to find the best model I will use train set that we have created from `edx` set. Our best starting point is always mean value of the data. It means that if we do not have any model on hand and can not model it the best prediction always would be the μ value of the data. So the μ value of the data would be:

$$\hat{\mu} = \frac{1}{N}\sum_{i} y_i$$

Where y_i are our train data. So our best simple guess would be:

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

Which $\varepsilon u, i$ is the error term that we can not predict with mean value even with our best model. So let's apply it to our data.

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. We can augment our previous model by adding the term b_i to represent average ranking for movie i:

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

Our best starting point is always mean value of the data. It means that if we do not have any model on hand and can not model it the best prediction always would be the μ value of the data. So the μ value of the data would be:

$$\hat{\mu} = \frac{1}{N} \sum_i y_i$$

Where y_i are our train data. So our best simple guess would be:

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

Which $\varepsilon u, i$ is the error term that we can not predict with mean value even with our best model. So let's apply it to our data.

```
#Find the RMSE of mean value of the train set
mu = mean(train_set$rating)
mean_rmse = mean((test_set$rating-mu)^2)^0.5
mean_rmse
```

```
## [1] 1.054638
```

So it was not a bad guess. But we need a sophisticated model to predict more precisely and reduce RMSE. To do that I will use some technique that we have seen some pattern above data visualisations such as some users and movies effect.

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. We can augment our previous model by adding the term b_i to represent average ranking for movie i:

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

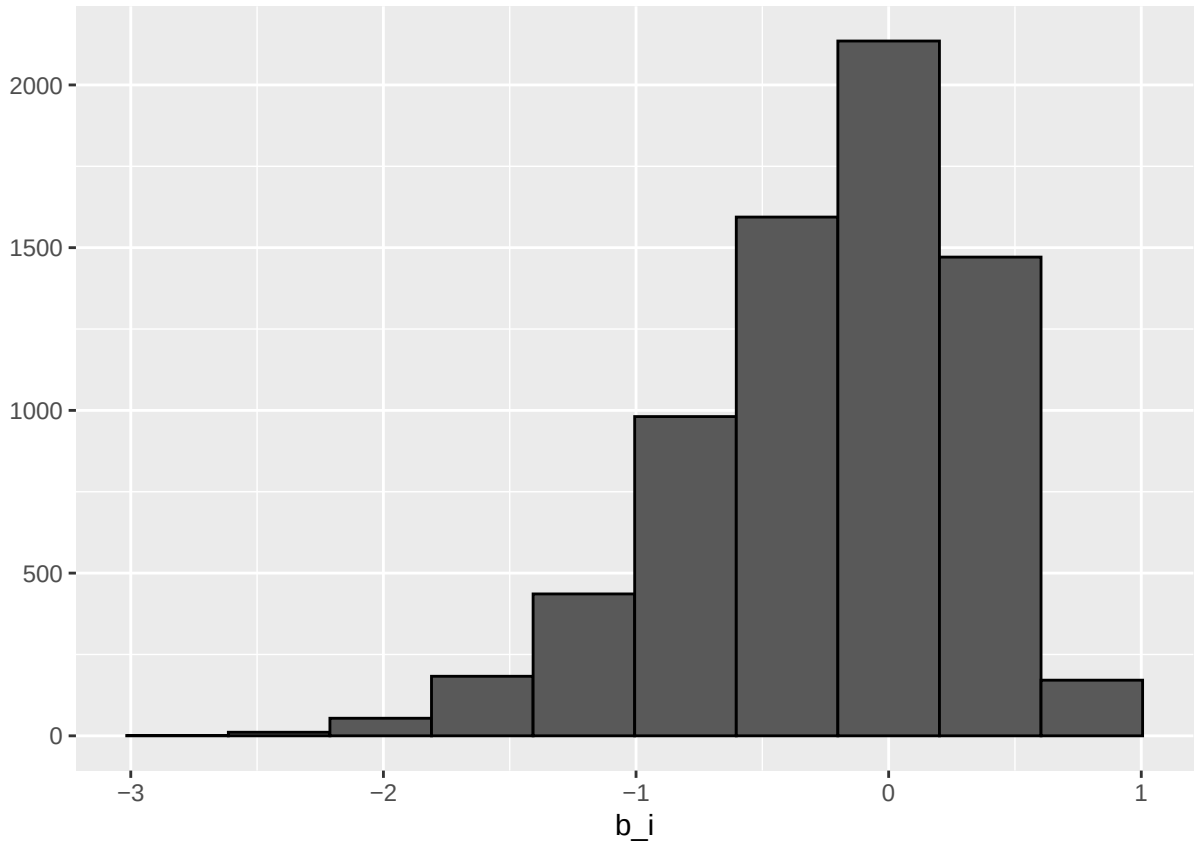
Let's look at some of the general properties of the data to better understand the challenges.

The first thing we notice is that some movies get rated more than others. Here is the distribution:

```
#Movies dispertion from mean value
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We can see that these estimates vary substantially:

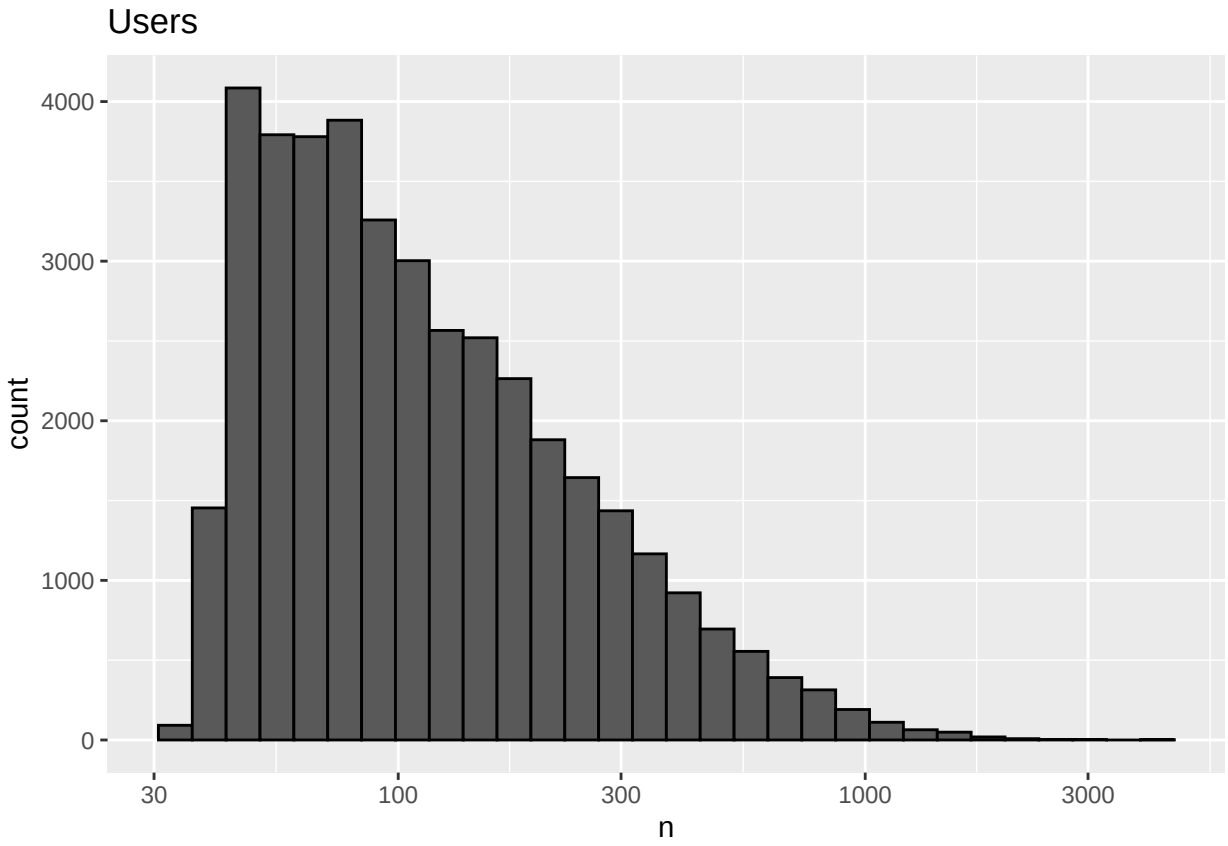
```
#Plot of the movie dispertion distribution
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



This distribution tell us that there are popular movies watched by millions and artsy, independent movies watched by just a few.

Our second observation is that some users are more active than others at rating movies:

```
#Plot distribution of users dispertion
train_set %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



This could be a usefull information in our model.

3.3 Modeling movie effects

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. We can augment our previous model by adding the term b_i to represent average ranking for movie i:

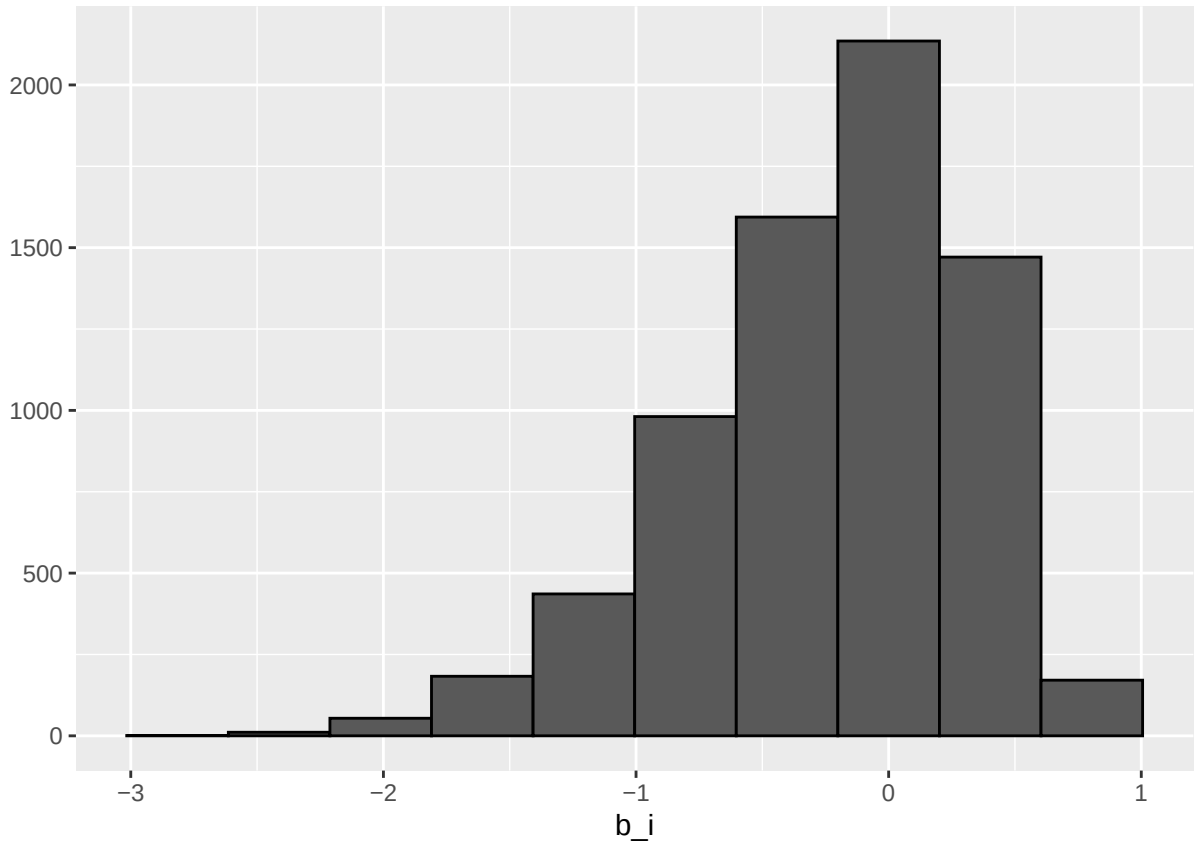
$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

So let us model it and see the result how it reduce our RMSE.

```
#Find the b_i which is movie effect
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i =mean(rating - mu))
```

We can see that these estimates vary substantially:

```
#Plot of Movie Effect
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Remember $\hat{\mu} = 3.5$ so a $b_i = 1.5$ implies a perfect five star rating.

Let's see how much our prediction improves once we use $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$:

```
#RMSE with Movie Effect
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

Movie_Effect_rmse <- RMSE(predicted_ratings, test_set$rating)

Movie_Effect_rmse
```

```
## [1] 0.9346297
```

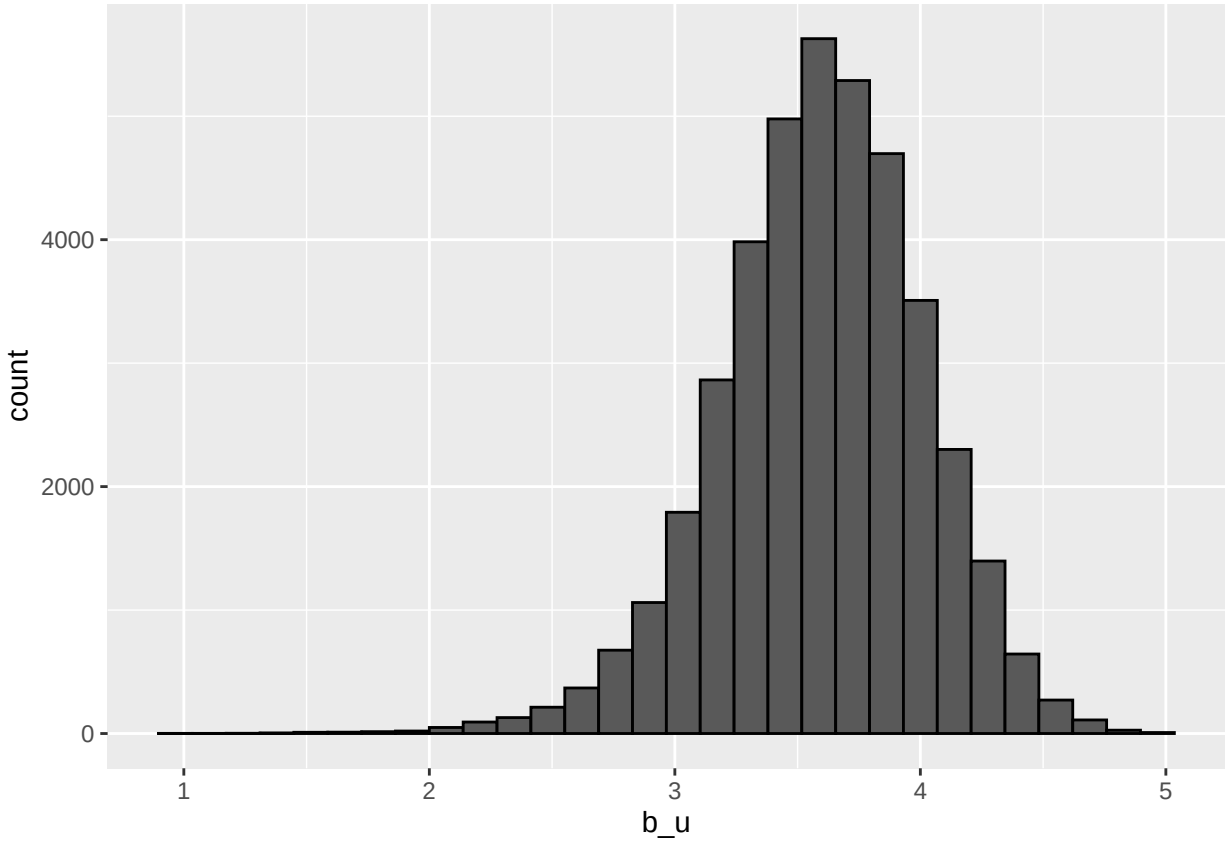
So it has already improved our RMSE substantially. Now let's model users effects.

3.4 Modelling User effects

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

```
#Users effect distribution graph
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
```

```
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



Notice that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

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

where b_u is a user-specific effect. 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.

3.5 Users Effect

```
#b_u which show the users effect
user_avgs <- train_set %>%
  left_join(movie_avgs, 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:

```
# Find RMSE together with Movie and Users effect
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)


RMSE_Movie_Users <- RMSE(predicted_ratings, test_set$rating)

RMSE_Movie_Users
```

```
## [1] 0.855556
```

3.6 Modelling Age of the Movie

What we see it has also substantially decrease our RMSE. So in the next step let's model the age of the movies and see how it effect our recommendation system.

```
 #Adding age of movie effect two movie and users effect
  mu <- mean(train_set$rating)

# find movie effect as b_i
  b_i <- train_set %>%
  group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()))

  #find users effect as b_u
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()))
  #adding users to movie
  b_t <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(premier_year) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n()))
  # Adding age of movie to the users and movie
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "premier_year") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    pull(pred)

 rmse_Movie_User_Premier <- RMSE(predicted_ratings, test_set$rating)
 rmse_Movie_User_Premier
```

```
## [1] 0.8553099
```

So the age of the movie has a minimal effect on our recommendation. But what we can do? We can use a very powerful technique penalized least square. The general idea behind regularization is to constrain the

total variability of the effect sizes. Why does this help? Consider a case in which we have movie $i = 1$ with 100 user ratings and 4 movies $i = 2, 3, 4, 5$ with just one user rating. In this case of course the movie 1 can be predicted using its mean value but movies 2-5 unfortunetly not. So there should be a system that "*penalize*" for these movies. So penalized least square deal this problem. For more mathematical information and intuitions please refer to Wikipedia. Here is the link

So in the following code I will use this tenchnique with a guessed number. Let's guess the a penalized number 4 and see what will be the result. Here is the code:

```
# Implement regularization with an arbitraty lambda to the algorithm we just applied
 lambdas=4
  mu <- mean(train_set$rating)

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

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

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

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

 RMSE_lambda <-RMSE(predicted_ratings, test_set$rating)
 RMSE_lambda
```

```
## [1] 0.8551505
```

Good job! It was a good guess. But isn't it a random guess? What we can do a sophistiacted guess. To do that we can imply *sapply* function to find the best *error term* that minimise our RMSE to minimum.So let's do it. Note that λ is a tuning parameter. We can use cross-validation to choose it.

3.7 Lambda Tuning

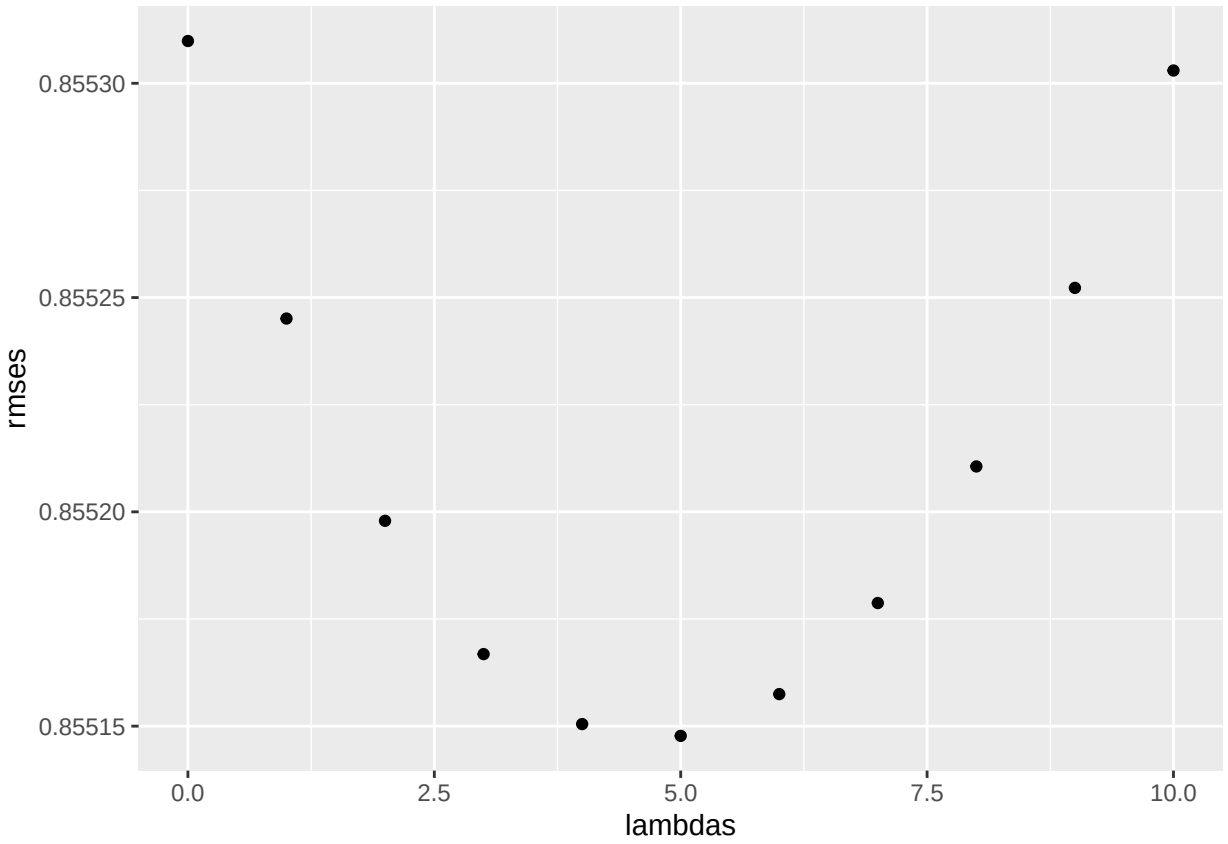
```
#Tuning the lamnda over the algortihm we have just implemented
lambdas <- seq(0, 10, 1)

rmses <- sapply(lambdas, function(l){

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

```
  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))

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

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

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

```
## [1] 0.8551477
```

So let's see what is the omptimal lamda.

```
#Graph of the RMSE with tuning lamnda parameters
qplot(lambdas, rmses)
```




```
min(rmses)
```

```
## [1] 0.8551477
```

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

For the full model, the optimal λ is 5 which reduce our RMSE substantially to 0.85. That is the great result.

4. Results

After finding best lambda that reduce our RMSE we can use it to check against validation set.

```
# Check the RMSE with the best tuned lambda over validation set
l= lambda
  mu <- mean(train_set$rating)

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

RMSE_final = RMSE(validation$rating, predicted_ratings)
RMSE_final
```

```
## [1] 0.8563175
```

So the challenge in this type of machine learning projects is to find the model that minimise the error so called RMSE, what I was already explained in the beginning sections. So what I found is the RMSE is minimum if we use user, movie and age of the movie effects. Of course one can develop further models, using some other technique such as the movies effect, who has already gained its first series popularity such as Star Wars, God Father etc. These type of movies has a clear relationship if the first series was sucessful then the next serie will come and if that also would be a successful the next will come and it will continue so. I did not want to go so deep rather to make a simple movie as possible to be understandible and practical.