# Movie Lens Recommendation

HarvardXPH125.9x Data Science Capstone

Sajini Arumugam

12/7/2020

## Contents

**1 Introduction** **2**

**2 Analysis** **3**

2.1 Loading Data from EDX . . . . . 3

2.2 Exploratory Analysis . . . . . 4

**3 Modelling Methods** **9**

3.1 Model 1: Mean . . . . . 10

3.2 Model 2: Movie effect . . . . . 11

3.3 Model 3: User effect . . . . . 11

3.4 Model 4: Release year effect . . . . . 12

3.5 Prediction on Validation set . . . . . 12

**4 Regularization** **13**

**5 Results** **14**

**6 Conclusion** **14**

# 1 Introduction

This is a part of the HarvardX:PH125.9x Data Science: Capstone course. The goal of the project is to develop a Movie recommendation system that predicts the user ratings (from 0.5 to 5 stars) using the data from the Movie lens 10M data set. It's provided by the Group lens research lab from the University of Minnesota which contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users.

To develop the recommendation we are going to split the data into training and validation sets using the code provided in the course. The final object is to obtain a Root mean square error (RMSE) of less than 0.86490. We are going to be making analysis using different methods and training our model to achieve the goal of minimal RMSE which will then be applied to the validation set.

# 2 Analysis

## 2.1 Loading Data from EDX

```
# 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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

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

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

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

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

## 2.2 Exploratory Analysis

As you can see the edx training set has 9,000,055 rows and 6 columns. From the summary we can see that there are no missing values.

```
##      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
##
##
##

## [1] 0
```

The columns contain userId, movieId, rating, timestamp, title and genre details. Here is a short preview of the data.

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 1 | 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy\|Romance |
| 2 | 1 | 185 | 5 | 838983525 | Net, The (1995) | Action\|Crime\|Thriller |
| 4 | 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action\|Drama\|Sci-Fi\|Thriller |
| 5 | 1 | 316 | 5 | 838983392 | Stargate (1994) | Action\|Adventure\|Sci-Fi |
| 6 | 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action\|Adventure\|Drama\|Sci-Fi |
| 7 | 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children\|Comedy\|Fantasy |

If we examine further we can see that there are 69,878 distinct users and 10,977 distinct movies in the edx data set. Since we are trying to predict the ratings, let's look at the distribution. 4 is the most common rating with more than 2.5 million ratings followed by 3. The mean is 3.51 among all the ratings given by users.

Some movies are rated more than others which is obvious since people tend to rate blockbuster movies more often than others. 'Pulp fiction', 'Forest gump' are some of the most rated movies where as there are certain movies that are rated only once. We need to address this since it will affect our model while training, hence we need to introduce some kind of regularization to reduce the errors. We will look into that in the later parts of analysis.

| title | n |
|---|---|
| Pulp Fiction (1994) | 31362 |
| Forrest Gump (1994) | 31079 |
| Silence of the Lambs, The (1991) | 30382 |
| Jurassic Park (1993) | 29360 |
| Shawshank Redemption, The (1994) | 28015 |

We can see from the figure(Figure 1) that 4 and 3 are the most awarded ratings in the Movie Lens data set and very few are rated 1. The mean of the dataset is ***3.51***. The half ratings were only introduced in the year 2003, so there are less number of half ratings compared to the whole ones. Figure 2 shows the distribution of the rating.

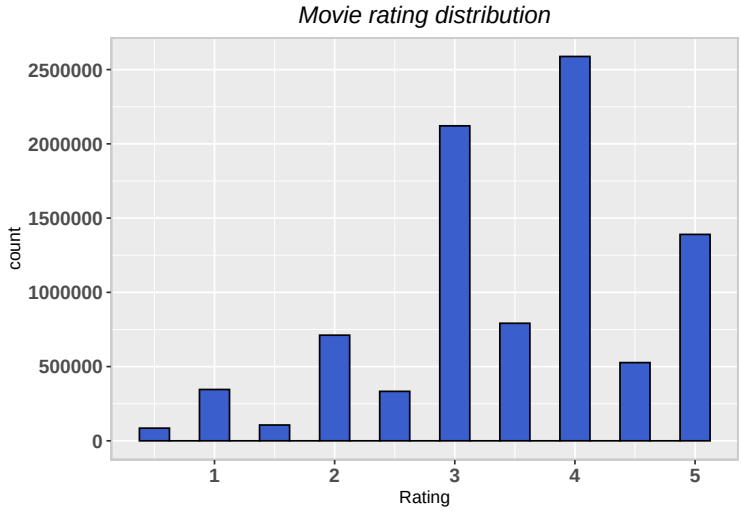


Figure 1: Rating

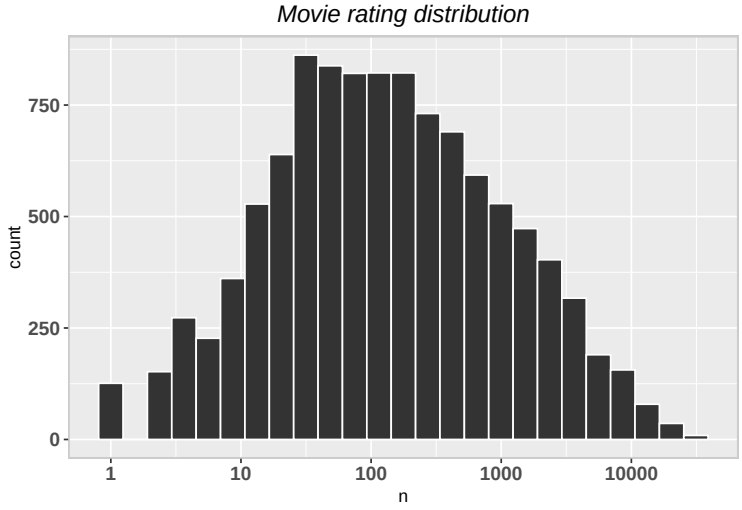


Figure 2: Movie Rating distribution

The figure below shows the sparseness of our data set for a sample of 100 movies and 100 users and

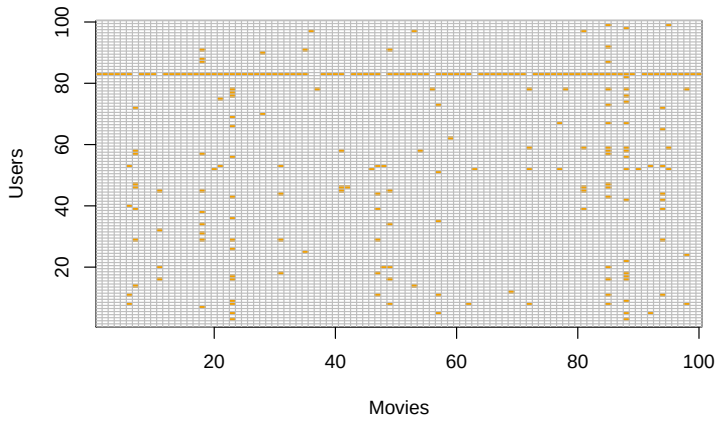


Figure 3: Sparse chart

also the rating distribution per movie.

Continuing on the user & rating combination, different users predict a different number of movies and different genre of movies. Some users are more active than the others, some have rated more than 1000 movies whereas some have only rated a handful. As we can see from the figure, the ratings of most users are anywhere from 20 to 100 movies.

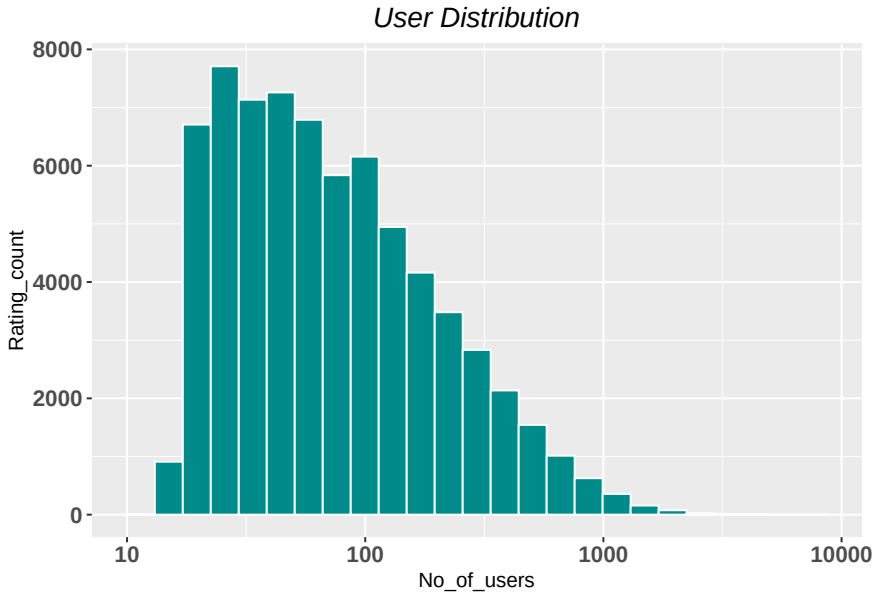


Figure 4: User ratings

Moving onto the genre column, A movie can be grouped into any number of genre and there are 797 distinct genre combinations in our data set. Here is a table of individual genres. If we look into those titles with more then 100,000 ratings we can see that Drama|War has the highest rating where as Comedy movies have the least (Figure 5).

| Genre | Total rating |
|---|---|
| Drama | 3,910,127 |
| Comedy | 3,540,930 |
| Action | 2,560,545 |
| Thriller | 2,325,899 |
| Adventure | 1,908,892 |
| Romance | 1,712,100 |
| Sci-Fi | 1,341,183 |
| Crime | 1,327,715 |
| Fantasy | 925,637 |
| Children | 737,994 |
| Horror | 691,485 |
| Mystery | 568,332 |
| War | 511,147 |
| Animation | 467,168 |
| Musical | 433,080 |
| Western | 189,394 |
| Film-Noir | 118,541 |
| Documentary | 93,066 |
| IMAX | 8,181 |
| (no genres listed) | 7 |

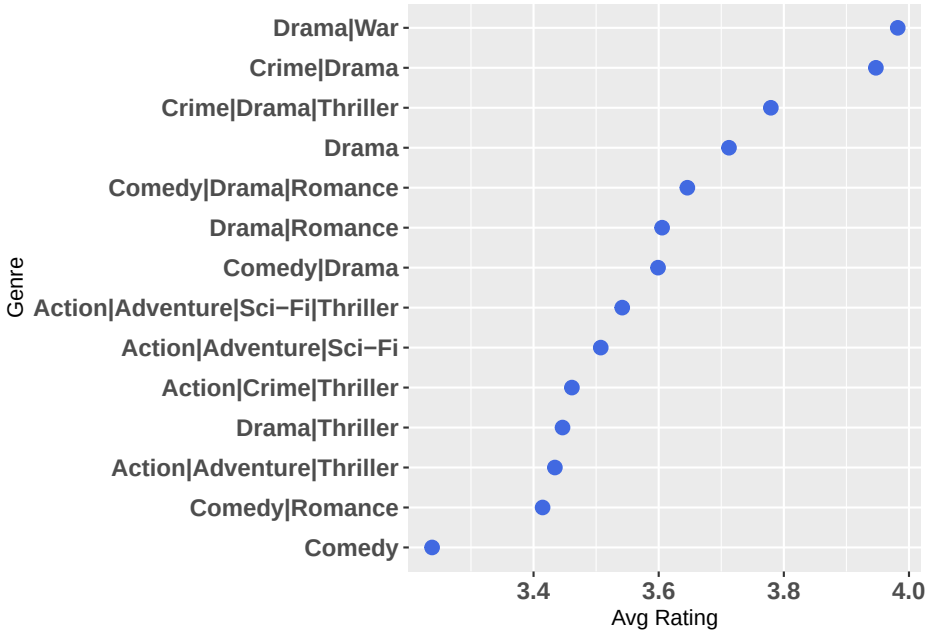


Figure 5: Genres

We are proceeding to look into the title and year of release to see if they have a significant importance in the rating. To better understand this, we are splitting the title into title and year. From the chart we can see that movies prior to 1990s were rated highly than the ones after with the peak between the years 1930 and 1970. After that, the ratings have gradually decreased.

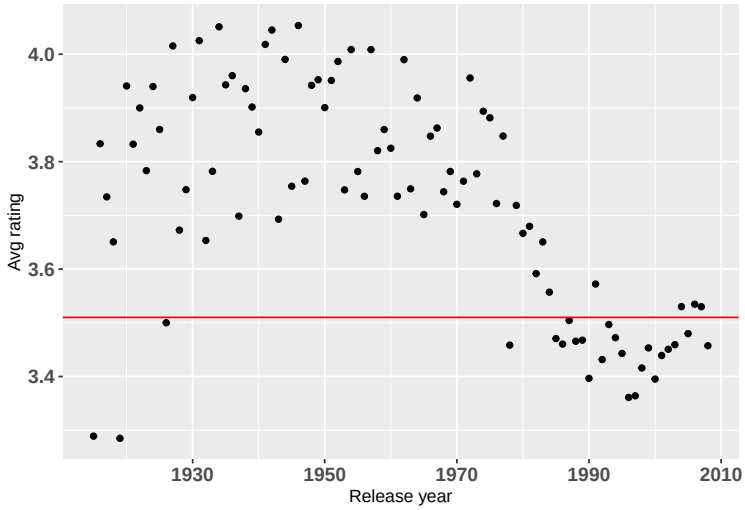


Figure 6: Avg rating throughout the years

Also, movies prior to 1980s didn't receive as many ratings as the ones after. Movies with most ratings are over the years of 1994 and 1995. Modern films are closer to the overall mean of the dataset and have tighter variability, which is shown in the figure. Thus adjusting for effects of release year should improve the accuracy of the model.

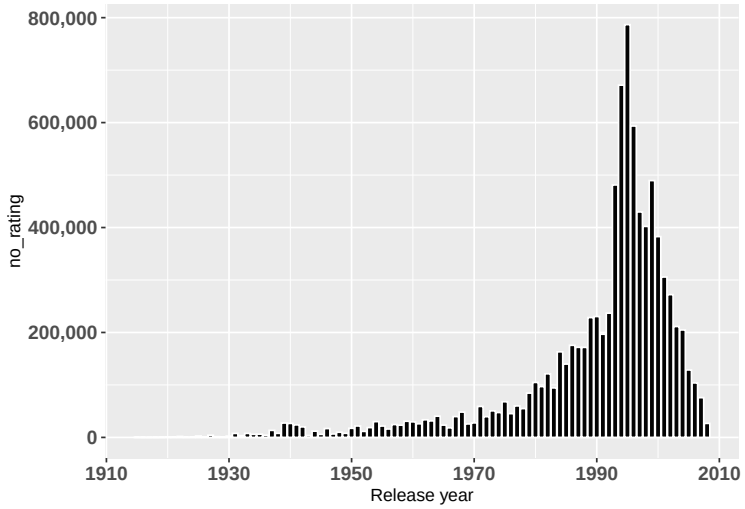


Figure 7: Rating over years in count

# 3 Modelling Methods

Let's move onto modeling now. We will start our predictions with the simple mean of all ratings. Before that we need to go over the concept of RMSE. Root mean Square Error is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are ; RMSE is a measure of how spread out these residuals are.

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

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

$y_{u,i}$ is the rating for movie $i$ by user $u$ and $\hat{y}_{u,i}$ is our prediction with N being the number of user/movie combinations and the sum occurring al over the 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 out typical error is larger than one star which is not good. Here are working towards reducing the error to **0.86490** or less.

First let us create the test and training sets off of **edx** as instructed

```
#creating test and training set from edx
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx_mod$rating, times = 1, p = 0.1, list = FALSE) #90-10

train_edx <- edx_mod[-test_index,]
temp <- edx_mod[test_index,]
nrow(edx_mod)
names(test_edx)
nrow(train_edx)


# Check to ensure all 'users' and 'movies' from test are in training set
test_edx <- temp %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")
```

Let's move on to our models now.

### 3.1 Model 1: Mean

Lets start by building the simplest possible recommendation system, we are going to predict same rating for all movies regardless of the user and the movie. So we are building a model that assumes the same rating for all movies and all users, with all the differences explained by random variation given as,

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

Where $\mu$ is the true rating and $\epsilon$ is the error sample.

```
#prediction
naive_rmse <- RMSE(test_edx$rating, mu_hat)
naive_rmse
```

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

| method | RMSE |
|---|---|
| Naive_RMSE | 1.060054 |

We get an RMSE value of 1.0600537 which is far from our goal.

### 3.2 Model 2: Movie effect

As we saw from the analysis earlier that different movies are rated differently. We can augment our previous model by adding the term $bi(bias)$ to represent average ranking for movie $i$. It's represented as,

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

Let's see how our prediction improves once we use the movie bias.

```
#prediction on test_edx
movie_rmse <- RMSE(predicted_b_i, test_edx$rating)
movie_rmse
```

```
## [1] 0.9429615
```

| method | RMSE |
|---|---|
| Naive_RMSE | 1.0600537 |
| Movie Effect Model | 0.9429615 |

RMSE with movie effect is 0.9429615, let's see if we can do better.

### 3.3 Model 3: User effect

Some users love every movie whereas some users give low rating even for a good movie. So including a user bias $b\_u$ along with the movie effect might prove helpful in achieving our desired RMSE.

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

```
#prediction on test set
user_rmse <- RMSE(predicted_b_u, test_edx$rating)
user_rmse
```

```
## [1] 0.8646844
```

| method | RMSE |
|---|---|
| Naive_RMSE | 1.0600537 |
| Movie Effect Model | 0.9429615 |
| Movie + User Effects Model | 0.8646844 |

Brings out an RMSE of 0.8646844, onto the next model.

### 3.4 Model 4: Release year effect

Let's build on the user model to see if there are any improvements to the RMSE. We already saw that movies prior to 1990s are rated highly than the others and most ratings were given to the movies between 1980 and 2000. Introducing a release year bias will hopefully reduce our RMSE.

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

```
#prediction on test set
release_year_rmse <- RMSE(predicted_b_yr, test_edx$rating)
release_year_rmse
```

```
## [1] 0.8643302
```

| method | RMSE |
|---|---|
| Naive_RMSE | 1.0600537 |
| Movie Effect Model | 0.9429615 |
| Movie + User Effects Model | 0.8646844 |
| Movie + user+Release year Effects Model | 0.8643302 |

we get an RMSE of 0.8643302. Let's focus on validation set now.

### 3.5 Prediction on Validation set

Let's use our model on the validation set now to see how it fairs. We get a RMSE of 0.8655155 which although is close, is not what we want. So we will look into regularization now.

```
# Predicting on the 'Validation' data
predicted_val <- validation_mod %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  left_join(year_bias, by ="r_year")%>%
  mutate(pred = mu_hat + b_i + b_u + b_yr) %>%
  .$pred
predicted_val[is.na(predicted_val)] <- 0 #setting 4NA's to 0
# prediction
rmse_val <- RMSE(predicted_val, validation_mod$rating)
rmse_val
```

```
## [1] 0.8655155
```

# 4 Regularization

Although our model proved valid, there are some discrepancies in the data.Rmse is not close to what we want. A lot of 5 star movies just were rated once. Here are the top 10 movies according to our estimate using b_i(movie bias).

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)"
##  [3] "Shadows of Forgotten Ancestors (1964)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Blue Light, The (Das Blaue Licht) (1932)"
##  [7] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
##  [8] "Life of Oharu, The (Saikaku ichidai onna) (1952)"
##  [9] "Human Condition II, The (Ningen no joken II) (1959)"
## [10] "Human Condition III, The (Ningen no joken III) (1961)"
```

The best movies were rated by very few users, in most cases just 1. This leads to uncertainty in our model. These noisy estimates in data should not be trusted, since they tend to increase our RMSE value. So we will be rather conservative than unsure. Hence we introduce regularization into the modeling approach.Regularization permits us to penalize large estimates that are formed using small sample sizes.

### 4.0.1 Penalized least square estimates

Penalized least squares estimates provide a way to balance fitting the data closely and avoiding excessive roughness or rapid variation.The general idea of penalized regression is to control the total variability of the effects.Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$\sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

The first term is just the sum of squares and the second is a penalty that gets larger when many $b\_i$ are large.

Let's move on to choosing the penalty. $\lambda$ is a tuning parameter, we can use cross validation to choose it. Local tests were done to choose the limits and settled on the one provided in the code below, to reduce run time.

```
#displaying which lambda yielded best result
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.6
```

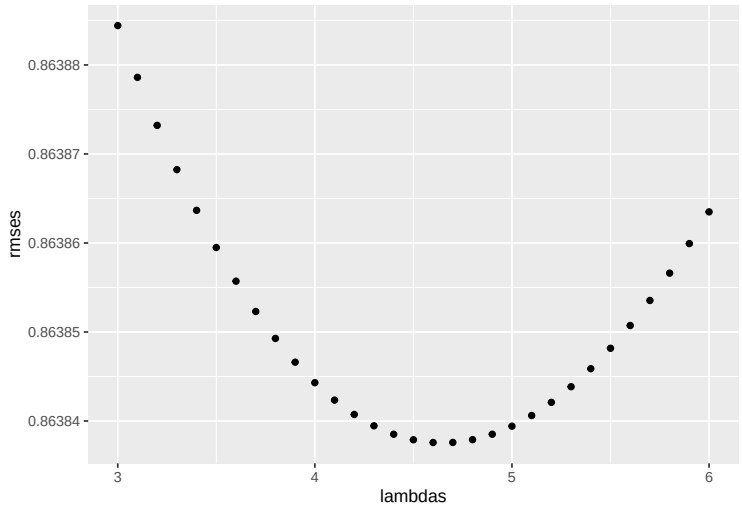


Figure 8: Plot of lambdas

# 5 Results

We are going to test our model along with the penalty term on the validation set(test out set). For this purpose we will use the model with movie bias, user bias and release year bias.Our penalty term is given as lambda which is 4.6.

```
#prediction on validation(hand out) set
validation_rmse <- RMSE(validation_mod$rating, predicted_ratings_regularized)
validation_rmse
```

```
## [1] 0.8645233
```

By using the three biases, we are able to bring down the error value to 0.8645223 which is under the acceptable limit given by this project.

# 6 Conclusion

The objective of this project is to analyze the Movies lens 10M dataset and build a recommendation with least errors. Through our analysis, we looked into every category and found out how they affected the ratings. We took into account 3 biases, Movie, User and release year to help reduce our prediction errors. The project statement is to reduce the RMSE to *0.86490* or less and our model produced an error of *0.8645223* on the final hold out set.

This particular recommendation is built on the ones that were taught in the course. Including another bias will also reduce the error by some value. Although, they didn't prove much effective in our model, we stuck to just 3 biases. Apart from this, we can also use Matrix factorization, XG boost or KNN.

Matrix factorization is a way to create a matrix when multiplying two different kinds of entities. The idea behind matrix factorization is to represent users and items in a lower dimensional latent

space. So, the user is matched against the movies that they have rated and the others will remain blank. Since there will be many missing values due to the fact that users don't rate every single movie, this is not much effective.

KNN does not make any assumptions but it relies on item feature similarity. When KNN makes inference about a movie, KNN will calculate the "distance" between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbor movies as the most similar movie recommendations.The data will be in a m*n array where m is the number of movies and n is the number of users. Again this will have a sparse data distribution as Matrix factorization. We won't go into much detail about this here considering the scope of the project.