HarvardX Data Science Capstone MovieLens Project

Weichen Zhou

1/8/2021

Introduction

This is the first final project in the HarvardX Data Science course series on Edx. In this project, my goal is to build a movie recommendation system. The idea is that the model should be predicting movie ratings based on already existing movie ratings so that reasonable recommendations could be made when this algorithm is used by someone who needs movie recommendation.

The dataset I'll be using, called "MovieLens 10M Dataset", consists of data collected by Grouplens. And this dataset consists of 10 million ratings of 10,000 movies and the movies were rated by 72,000 users. The dataset can be found here: "https://grouplens.org/datasets/movielens/10m/".

Overview

Here, I will give a quick overview of this report. I will start by downloading the dataset and manipulating it to the format that will work the best for my analysis. Then I split the dataset into two piece, one is the training set called "edx" and the other is the testing set called "validation". Some tailoring will be done at the begining to both sets to prepare them for future analysis, then I will hold the validation set, which contains 10% of totall data on hold until the final test stage. The set "edx" will be the set I work on to come up with the model.

Then, I will extract information from the training set "edx" and make educated guesses on what could be the possible predictors. After making a list of predictors, the average rating induced by those predictors are computed. And based on those computations, I will decide which predictors to include in the model.

There are three models I came up with and the corresponding RMSE were computed. As I was testing the models, it became clearer what predictors to drop and why regularization is needed when constructing the final model.

Now, we'll start by looking at the analysis step by step.

Data setup and analysis

Data setup

Package installation and loading

We start by installing necessary data processing packages if they are not already installed, then load the packages.

```r
# Install necessary packages:
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(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")

# Load packages
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(ggplot2)
library(lubridate)
```

Data download

Now we are ready to download the "MovieLens-10M" dataset. It comes in a zip file and we will unzip it to extract the information in R.

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

Raw data extraction and manipulation

Then, we extract the rating information and movie list from the dataset after unzipping it.

```r
# Unzip the downloaded data and create the ratings data frame
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

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

Then, I used the following set of code to manipulate the so-called "raw datasets" movies and ratings and combine them into a data frame that I can easily work with. One thing that is worth noting here is that I am running R 4.0.3 so I am using code that works for this version of R.

```r
# Convert the movie list to data frame and format the columns
movies <- as.data.frame(movies)  %>% mutate(movieId = as.numeric(movieId),
                                            title = as.character(title),
                                            genres = as.character(genres))

# Combine the rating data to the movie list
movielens <- left_join(ratings, movies, by = "movieId")
```

It is always to good idea to check if the left_join is successfully performed. It is sufficient to check the first few rows of the data frame movielens and we can do that with the following code.

```r
head(movielens)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
```

```
## 2:      1     185      5 838983525                Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421                Outbreak (1995)
## 5:      1     316      5 838983392                Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:               Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                       Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

Partition into training and test sets

The data frame named "movielens" is the set containing all the information. Now, we randomly partition this data frame into two sets, one training set edx that is used to train the algorith, and a test set called validation. The idea is that the algorithm will predict ratings of movies in the validation set and we can see how accurate it is by comparing the prediction with the record in the validation set.

```
# Create a test set (edx) and validation set,
# require that the validation set contains 10% of data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

One important thing to check here is that we want to make sure all movies in the validation set also appear in the training set edx. The intuition is that the accuracy of we our prediction on something we have no information about is very low.

```
# Create a test set (edx) and validation set,
# require that the validation set contains 10% of data
# Make sure the movies in the validation set are all included in the edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add the removed rows from the validation set back to the train set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

The last step of pre-processing is to remove all intermediate files we created along the way. This will save some RAM on our computer and helps with processing speed.

```
# Remove all unnecessary files
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Analysis

We start our analysis by exploring the data frames that we created. There are a lot more information to extract and I'll only include a few in there.

```
# Number of rows and columns in edx
nrow(edx)
```

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

```
ncol(edx)
```

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

```
# Number of movies in edx
N_movies <- edx %>% distinct(movieId) %>% summarise(n=n())
N_movies
```

```
##       n
## 1 10677
```

```
# Number of users in edx
N_users <- edx %>% distinct(userId) %>% summarise(n=n())
N_users
```

```
##       n
## 1 69878
```

When observing the titles of the movies, we see that each movie title has a year that it is released in parenthesis in the title. We can extract that as that might be a useful predictor.

```
# Observe that there's a year related to each movie in the title col, we can extract that
# Do the same thing for validation set.
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

We can also interpret the timestamp column in the data frames as well. The timestamps column marks the date and time when the rating is made. We can interpret that column so it is more understandable.

```
# Interpret the time stamp as the time the rating occurred,
# for both testing and validation set
edx <- edx %>% mutate(rating_year = year(as_datetime(timestamp)))
validation <-validation %>% mutate(rating_year = year(as_datetime(timestamp)))
```

There are errors in this. We cannot rate a movie before it comes out.

```
error <- edx %>% filter(year > rating_year)
nrow(error)
```

```
## [1] 175
```

And we can fix it. The fix might not be very optimal here. I assumed that information were incorrected collected the these errored movies are rated the year they came out. Here's the fix:

```
edx$rating_year <- ifelse(edx$rating_year<edx$year,edx$year,edx$rating_year)
validation$rating_year <- ifelse(validation$rating_year<validation$year,
                                 validation$year,validation$rating_year)
```

Then, we create a column recording the "age" of the movie when it is rated, that is the number of years from releasing to rating.

```
# Define a movie_age to be the number of years the movie has been out when it is rated
# Compute the movie age, time different between rating and production.
edx <- edx %>% mutate(movie_age = rating_year-year)
validation <- validation %>% mutate(movie_age = rating_year-year)
```

Since we already interpreted the timestamp column, we can just have it removed.

```r
# Remove timestamp
edx <- edx %>% select(-timestamp)
validation <- validation %>% select(-timestamp)
```

Note: For all the above actions, I did to both training and testing sets because the newly added columns are potential predictors that we need for final step predicting. So we need them all in both data frames.

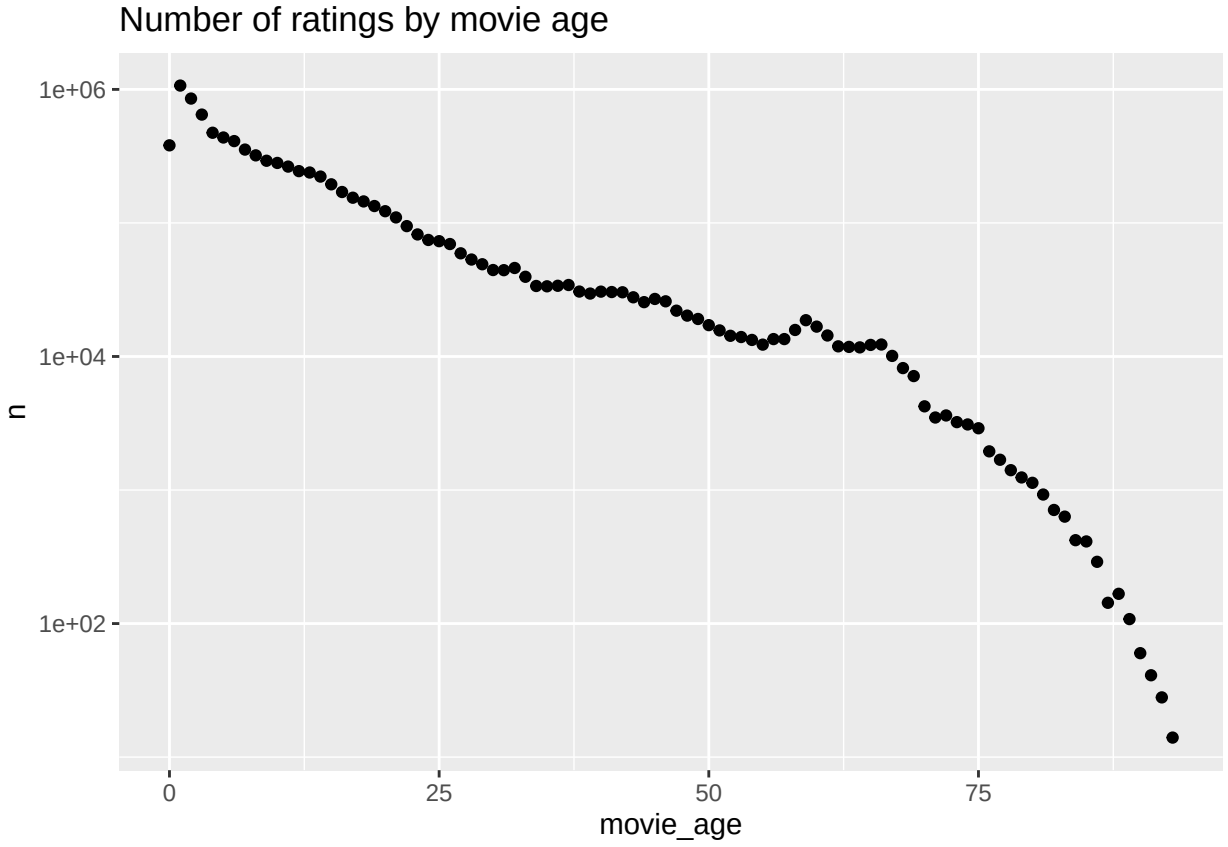
Analysis begin

Here are the list of candidates for predictors: Average rating for the movie, User effect,year the movie was produced, Movie age = rating_year-year and Genre. We will explore their affect on average rating in the training set "edx". It is very important that we only use the training set starting at this point.

Rating and movie_age

First, we can plot the movie_age against the number of observations for each age.

```r
edx %>% group_by(movie_age) %>% summarise(n=n()) %>%  ggplot(aes(movie_age,n)) +
  scale_y_log10() +geom_point() + ggtitle("Number of ratings by movie age")
```

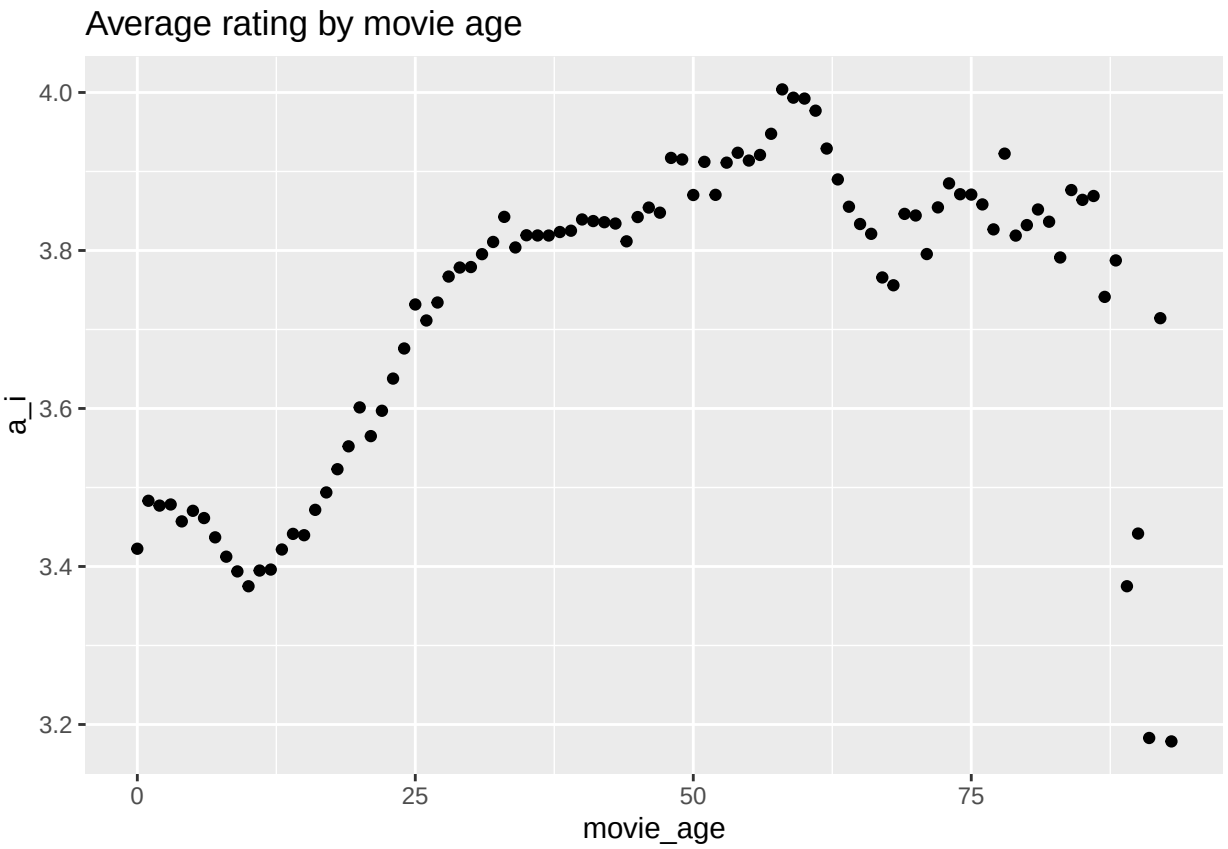


Then create data recording average rating by movie_age, and plot for visualization.

```r
# Average rating and age
age_avg <- edx %>% group_by(movie_age) %>% summarise(a_i = mean(rating))

# Plot for visualization:
```

```
age_avg %>% ggplot(aes(movie_age,a_i)) + geom_point() +
  ggtitle("Average rating by movie age")
```



One thing worth noticing is that very old movies do not have low ratings. This might be caused by them having very few ratings. We can explore a bit more on that:

```
age_avg %>% arrange(-movie_age)
```

```
## # A tibble: 94 x 2
##    movie_age   a_i
##        <dbl> <dbl>
##  1        93  3.18
##  2        92  3.71
##  3        91  3.18
##  4        90  3.44
##  5        89  3.38
##  6        88  3.79
##  7        87  3.74
##  8        86  3.87
##  9        85  3.86
## 10        84  3.88
## # ... with 84 more rows
```

```
edx %>% filter(movie_age>80) %>% group_by(movie_age) %>% summarise(n=n())
```

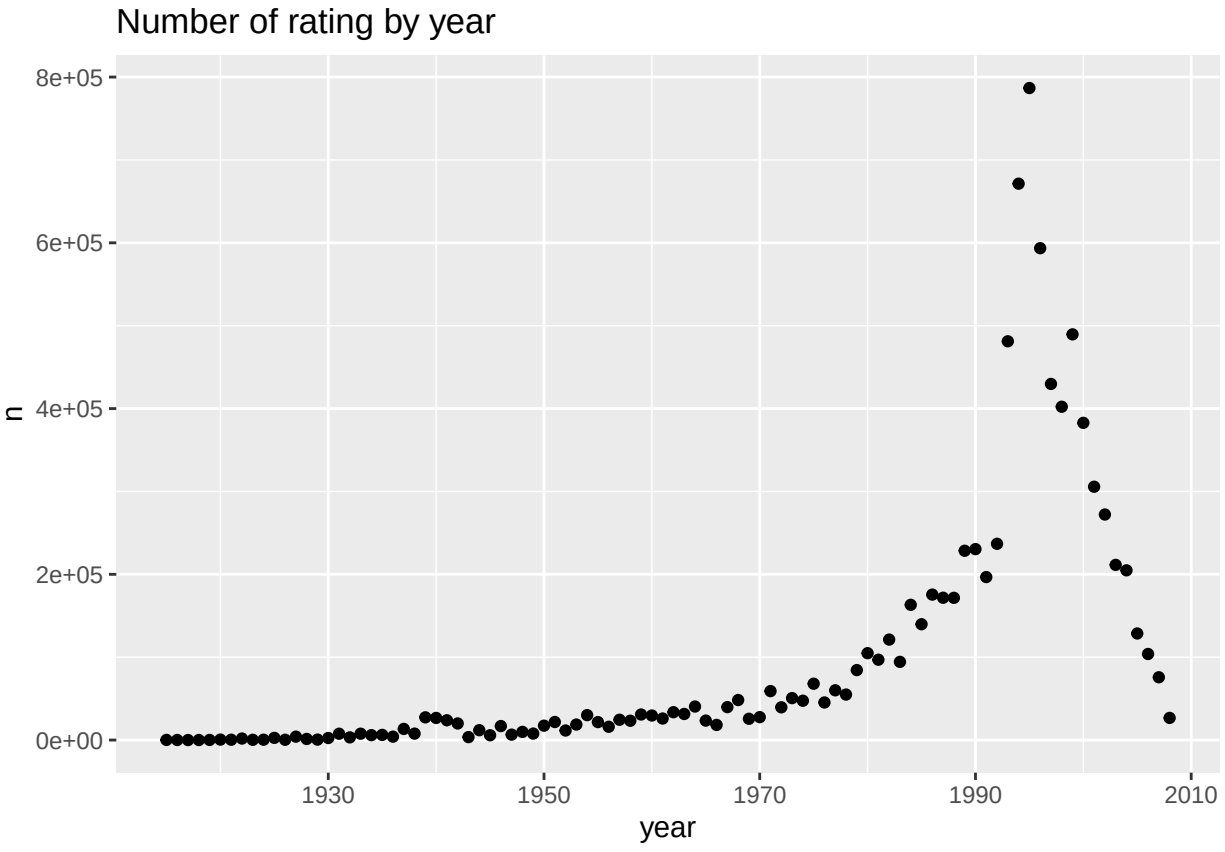
```
## # A tibble: 13 x 2
##    movie_age     n
##        <dbl> <int>
```

```
##  1         81   925
##  2         82   709
##  3         83   632
##  4         84   421
##  5         85   412
##  6         86   290
##  7         87   143
##  8         88   167
##  9         89   108
## 10         90    60
## 11         91    41
## 12         92    28
## 13         93    14
```

Rating and year

Now we explore the effect on rating by year, the year which the movies are produced. We first look at how many movies are produced in the documented years and plot for visualization.

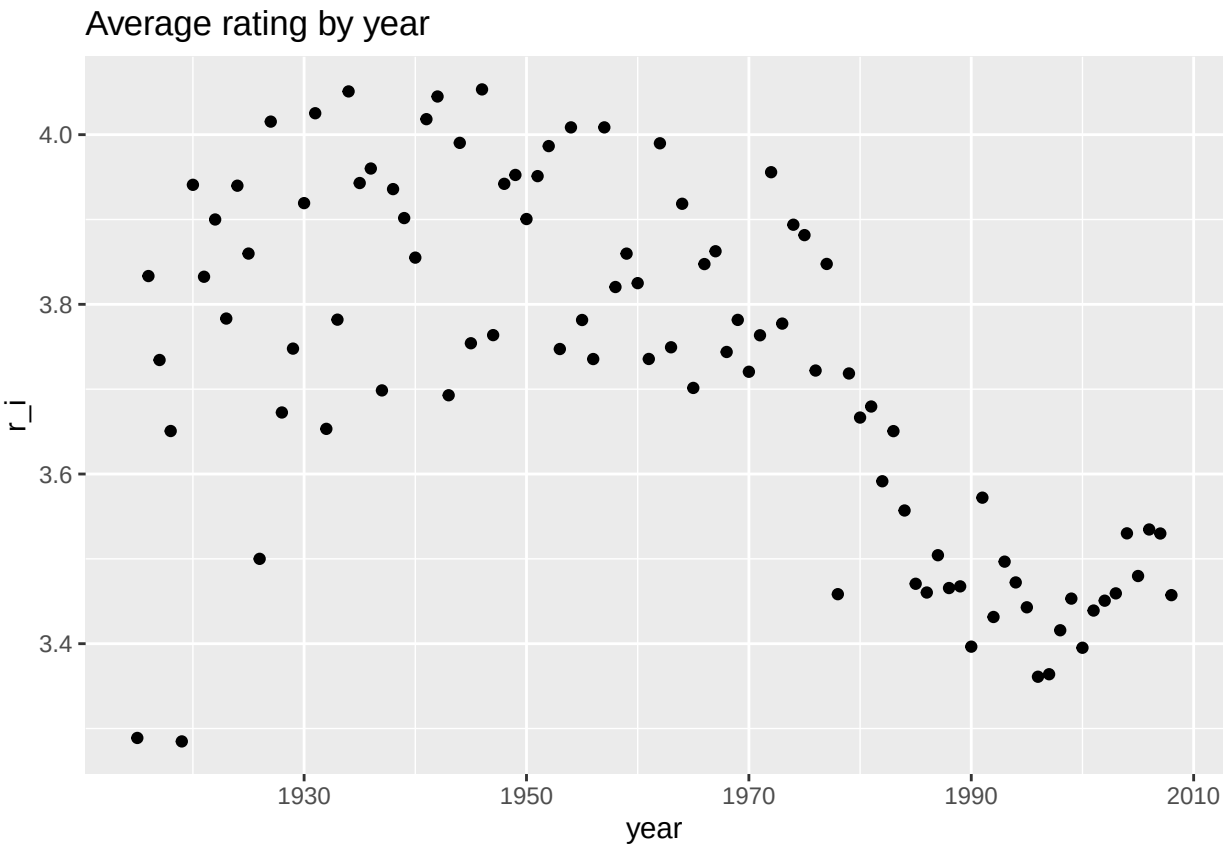
```
# Number of ratings in each year
edx %>% group_by(year) %>%summarise(n=n()) %>%ggplot(aes(year,n)) + geom_point() +
  ggtitle("Number of rating by year")
```



Then, we compute the average rating by year and plot to see better.

```
# Average rating by year
year_avg <- edx %>% group_by(year) %>% summarise(r_i = mean(rating))
```

```
year_avg %>% ggplot(aes(year,r_i)) + geom_point() +
  ggtitle("Average rating by year")
```



Rating and genre

Another possible predictor is genre. To explore this, notice that in our edx dataset, there are movies that are categorized into many genres. So we need to split them first. This piece of code takes several minutes to run. It took on average 7min on my laptop. This is due to the sizes of dataset we are dealing with. Personally I think 7min is not a horible runtime, so I decided not to take any subsets when running this part of analysis.

```
edx_split <- edx %>% separate_rows(genres,sep="[\\|]")
```

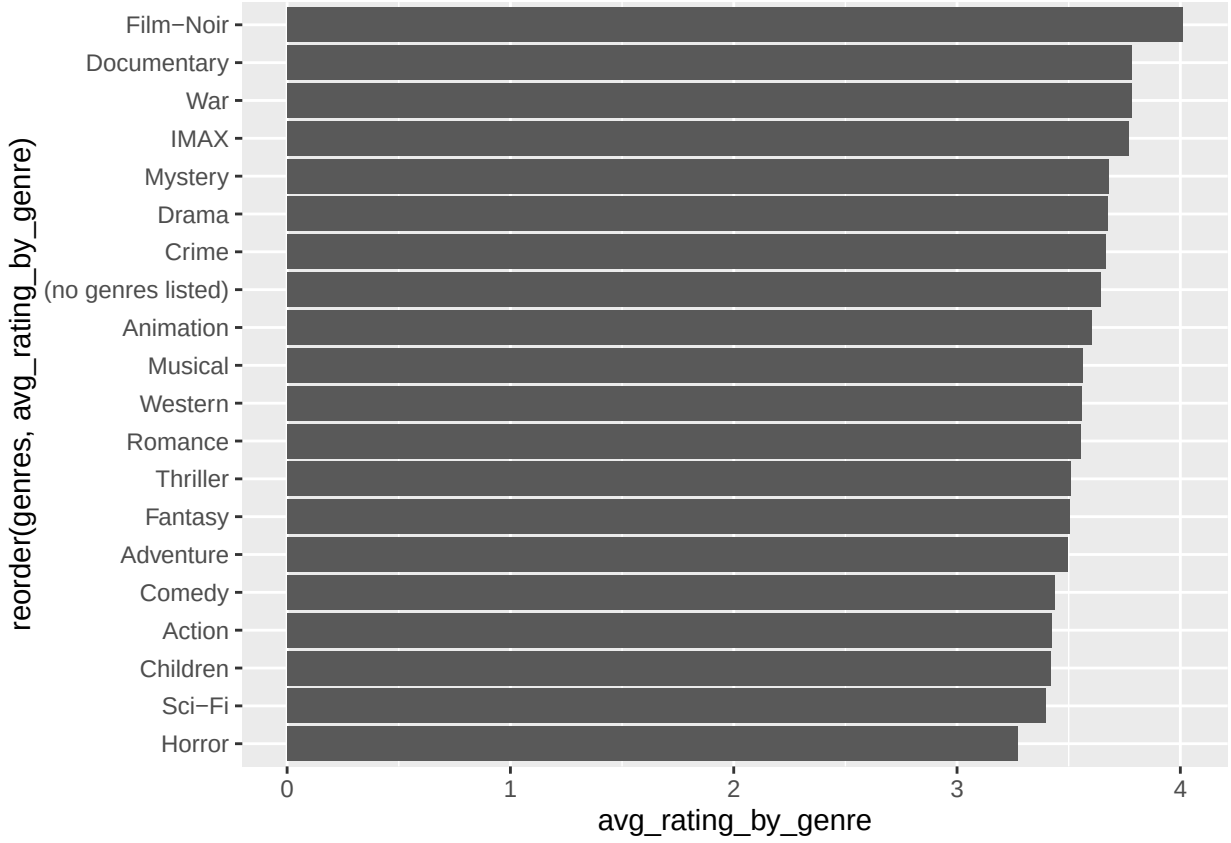
We count the number of ratings for each genre. Then compute and plot average ratings in each genre.

```
# Number of rating for each genre
rating_num_genre <- edx_split %>% group_by(genres) %>% summarise(n=n()) %>%
  arrange(n)
rating_num_genre
```

```
## # A tibble: 20 x 2
##    genres                 n
##    <chr>              <int>
##  1 (no genres listed)     7
##  2 IMAX                8181
##  3 Documentary        93066
##  4 Film-Noir         118541
##  5 Western           189394
```

```
##  6 Musical             433080
##  7 Animation           467168
##  8 War                 511147
##  9 Mystery             568332
## 10 Horror              691485
## 11 Children            737994
## 12 Fantasy             925637
## 13 Crime              1327715
## 14 Sci-Fi             1341183
## 15 Romance            1712100
## 16 Adventure          1908892
## 17 Thriller           2325899
## 18 Action             2560545
## 19 Comedy             3540930
## 20 Drama              3910127
```

```
# Average rating and Genres
edx_split %>% group_by(genres) %>% summarise(avg_rating_by_genre = mean(rating)) %>%
  ggplot(aes(reorder(genres,avg_rating_by_genre),avg_rating_by_genre)) +
  geom_bar(stat = "identity") +coord_flip()
```



From this plot, we see that genres in general do not affect the average rating much and there's no strong evidence in correlation. So we will drop this now and not consider genre in the model.

We know that different movies are rated differently and different users rate movies differently. So movie and user effect are two predictors that will definitely be considered in our model. And in order to minimize the effect on accuract by incidents like "absurd movie have only one high rating", we will use regularization by introducing a variable λ.

Before moving on to talk about the result, we'll include the year and rating average info into table

```
# Gather year and age induced rating average info into table
edx <- edx %>% left_join(age_avg,by = "movie_age")  %>%
  left_join(year_avg,by="year")
```

Result

Model 1

This model considers effects of all 4 predictor candidates and it looks like this

$$Y_{u,i} = \mu + m_{avg} + u_{avg} + a_i + r_i,$$

where μ is the overall average of rating, m_{avg} is the movie effect, u_avg accounts the user effect, a_i accounts for movie age effect and r_i accounts for year effect. \ To use regularization, we need to find an optimized λ that minimizes the RMSE. Before including the code to test for this model and show the result, we need to notice one thing: year and movie_age are heavily correlated.

```
edx %>% summarise(cor = cor(year,movie_age))
```

```
##          cor
## 1 -0.9632677
```

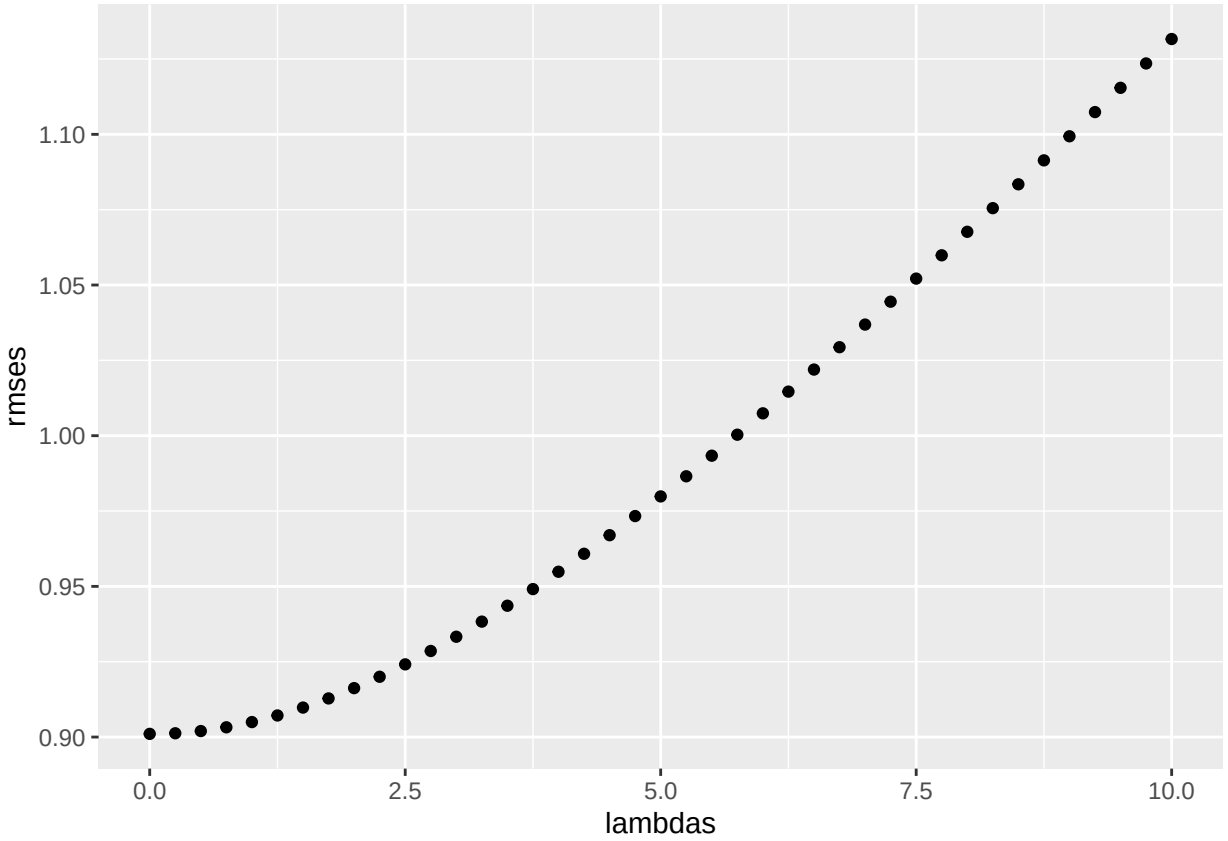
That's a sign that this may not be a great model, but we'll still test it and see. This will take a few minutes to run. Then we can plot the RMSE versus the λ's. Find the optimap λ and the minimal RMSE using this model.

```
# Function for RMSE
RMSE <- function(rating_true, rating_pred){
  sqrt(mean((rating_true-rating_pred)^2))
}

# Consider a sequence of lambda's
lambdas <- seq(0,10,0.25)
# Compute RMSE with different lambdas and store RMSE results
rmses <-sapply(lambdas,function(lambda){
  # Average mu of the given rating observations in the test set.
  mu <- mean(edx$rating)
  # Rating and effect by movie average, with regularization
  movie_avg<- edx %>% group_by(movieId) %>%
    dplyr::summarise(m_avg = sum(rating-mu)/(n()+lambda))
  # Average rating by user, with regularization
  user_avg<- edx %>% left_join(movie_avg,by="movieId") %>%
    group_by(userId) %>%
    dplyr::summarise(u_avg = sum(rating-(mu+m_avg+a_i+r_i))/(n()+lambda))
  pred_data <- validation %>% left_join(movie_avg,by="movieId") %>%
    left_join(user_avg,by = "userId") %>%
    left_join(year_avg,by = "year") %>%
    left_join(age_avg,by="movie_age")
  # Prediction
  rating_pred_data <- pred_data %>% mutate(pred = mu+m_avg+u_avg + a_i+r_i)
  rating_pred <- rating_pred_data$pred
  # Compute the RMSE using function defined above
  RMSE(validation$rating,rating_pred)
```

```
})

# plot the lambda values and rmses
qplot(lambdas,rmses)
```




```
# find the optimal lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0
```

```
rmse <- min(rmses)
rmse
```

```
## [1] 0.9010584
```

The RMSE here is about 0.9, which is not so good. And this result supported the earlier guess on affect of correlation between movie age and year. So we move on to Model 2 and make some adjustments.

Model 2

This model considers effects of 3 predictor candidates and it looks like this

$$Y_{u,i} = \mu + m_{avg} + u_{avg} + a_i,$$

where μ is the overall average of rating, m_{avg} is the movie effect, u_avg accounts the user effect and a_i accounts for movie age effect. Notice that we drop the year effect here. The reason we drop year instead of movie age is because movie age to some extend incoorporates the year information.

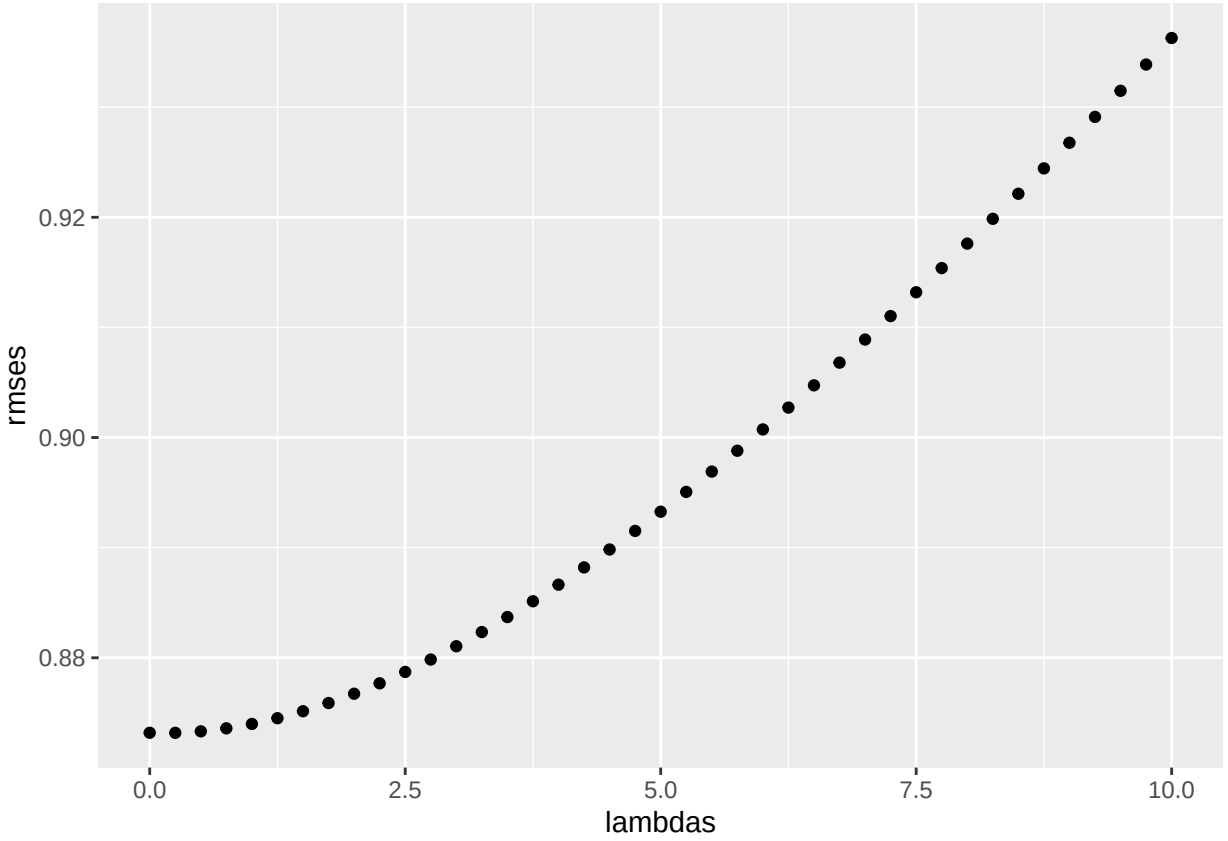
We shall test this model by doing the same computation on RMSE with the sequence of λ values. We then graph the RMSE versus the λ's, locate the optimal λ and report the minimal RMSE for this model.

```
# Function for RMSE
RMSE <- function(rating_true, rating_pred){
  sqrt(mean((rating_true-rating_pred)^2))
}

# Consider a sequence of lambda's
lambdas <- seq(0,10,0.25)

# Compute RMSE with different lambdas and store RMSE results
rmses <-sapply(lambdas,function(lambda){
  # Average mu of the given rating observations in the test set.
  mu <- mean(edx$rating)
  # Rating and effect by movie average, with regularization
  movie_avg<- edx %>% group_by(movieId) %>%
    dplyr::summarise(m_avg = sum(rating-mu)/(n()+lambda))
  # Average rating by user, with regularization
  user_avg<- edx %>% left_join(movie_avg,by="movieId") %>%
    group_by(userId) %>%
    dplyr::summarise(u_avg = sum(rating-(mu+m_avg+a_i))/(n()+lambda))
  pred_data <- validation %>% left_join(movie_avg,by="movieId") %>%
    left_join(user_avg,by = "userId") %>%
    left_join(year_avg,by = "year") %>%
    left_join(age_avg,by="movie_age")
  # Prediction
  rating_pred_data <- pred_data %>% mutate(pred = mu+m_avg+u_avg + a_i)
  rating_pred <- rating_pred_data$pred
  # Compute the RMSE using function defined above
  RMSE(validation$rating,rating_pred)
})

# plot the lambda values and rmses
qplot(lambdas,rmses)
```




```
# find the optimal lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0.25
```

```
rmse <- min(rmses)
rmse
```

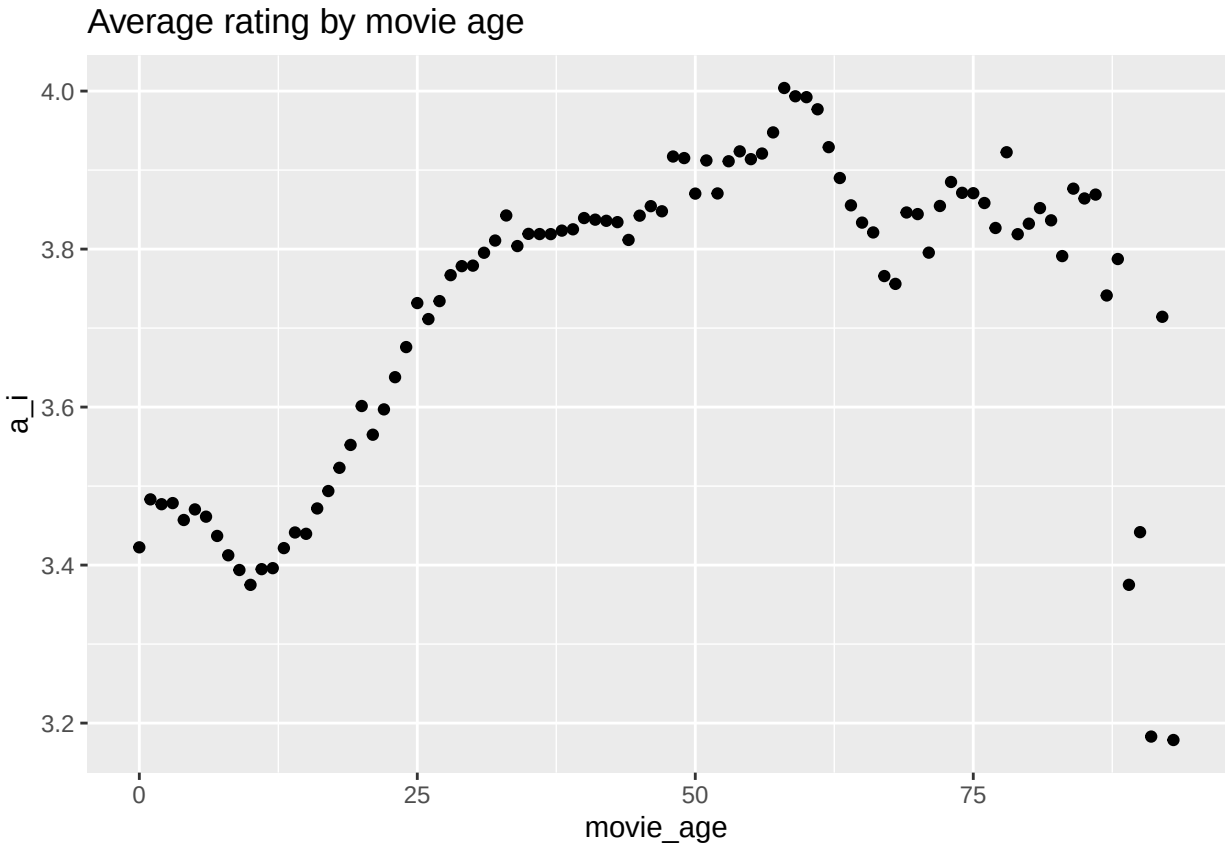
```
## [1] 0.8731822
```

We can see that the RMSE improved to around 0.87 but still not that good. Let's move on to the third model.

Model 3

Before defining the actual model, let's recall something from a previous section. We created a plot when we were looking at the effects of movie age on average ratings. Here's the plot that we created.

```
age_avg %>% ggplot(aes(movie_age,a_i)) + geom_point() + ggtitle("Average rating by movie age")
```



The average rating doesn't seem to affect too much by the movie age, and the range where most points are is only about 0.4. Having this intuition in mind, we are tempted to drop movie-age as a predictor, since it doesn't seem to help much with the prediction, instead, it introduced more variability. Note that movie age is heavy affected by the general moving making trend, there are years where many movies are made, and the users have limited time of watching movies but they have a vast choice pool. These are all factors that are noises that will affect the rating, but will not help with the prediction we want.

Here, we define the model this way:

$$Y_{u,i} = \mu + m_{avg} + u_{avg}$$

where m_{avg} accounts for movie effect and u_{avg} account for user effect. Then, apply the same approach as before:

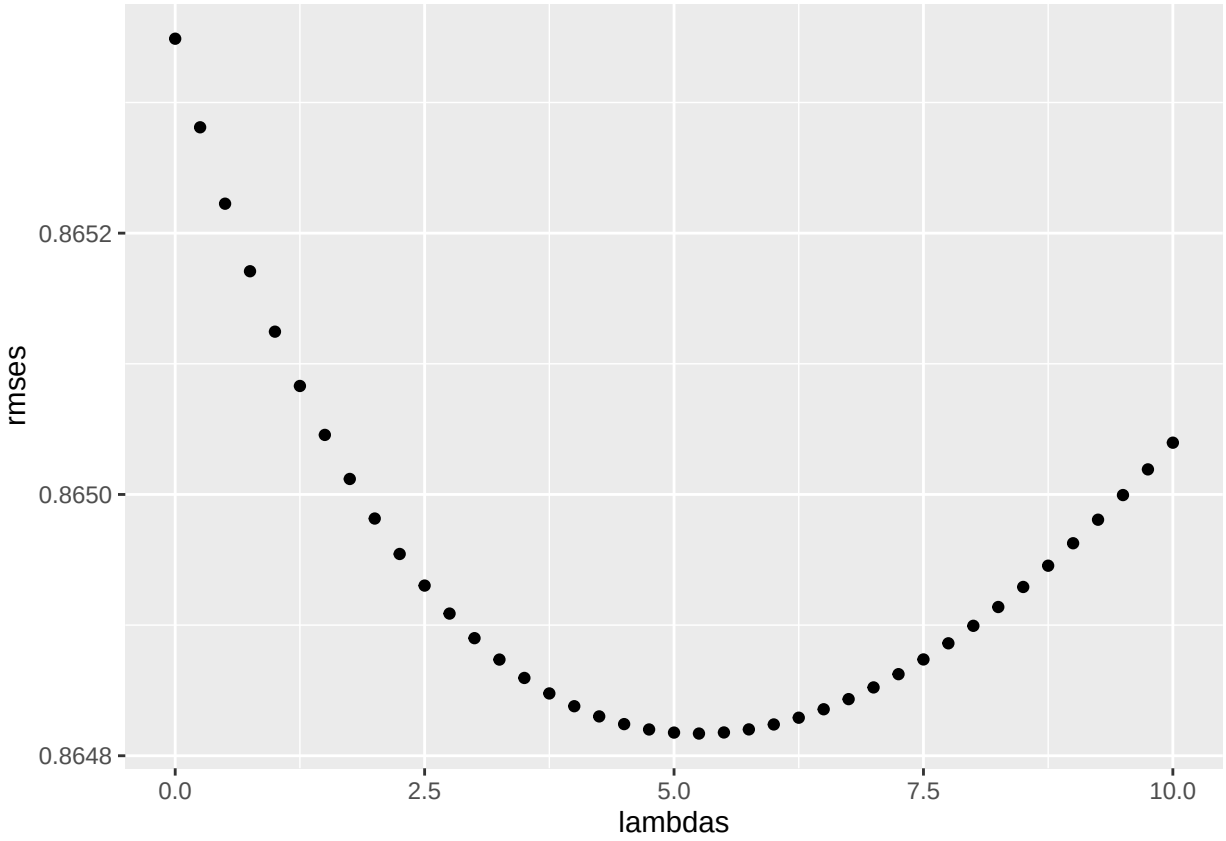
```
# Function for RMSE
RMSE <- function(rating_true, rating_pred){
  sqrt(mean((rating_true-rating_pred)^2))
}

# Consider a sequence of lambda's
lambdas <- seq(0,10,0.25)

# Compute RMSE with different lambdas and store RMSE results
rmses <-sapply(lambdas,function(lambda){
  # Average mu of the given rating observations in the test set.
  mu <- mean(edx$rating)
  # Rating and effect by movie average, with regularization
  movie_avg<- edx %>% group_by(movieId) %>%
    dplyr::summarise(m_avg = sum(rating-mu)/(n()+lambda))
  # Average rating by user, with regularization
```

```
  user_avg<- edx %>% left_join(movie_avg,by="movieId") %>%
    group_by(userId) %>%
    dplyr::summarise(u_avg = sum(rating-(mu+m_avg))/(n()+lambda))
  # Join the predictors m_avg and u_avg to the validation set
  pred_data <- validation %>% left_join(movie_avg,by="movieId") %>%
    left_join(user_avg,by = "userId")
  # Prediction
  rating_pred_data <- pred_data %>% mutate(pred = mu+m_avg+u_avg)
  rating_pred <- rating_pred_data$pred
  # Compute the RMSE using function defined above
  RMSE(validation$rating,rating_pred)
})

# plot the lambda values and rmses
qplot(lambdas,rmses)
```




```
# find the optimal lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

```
rmse <- min(rmses)
rmse
```

```
## [1] 0.864817
```

Here we get the minimal RMSE $= 0.864817$ and the optimal $\lambda = 5.25$. The RMSE is now under 0.87. For the model given, we should be happy about this result.

Conclusion

Among three models that we tested above. We see that Model 3 produces the smallest Residual Mean Square Error of 0.864817 when $\lambda = 5.25$. If given these three models to choose from, Model 3 is clearly the optimal choice.

Some thoughts

The above analysis shows that there are times we are given many data and many of them seemed useful when making prediction. But in fact, they are just noises! The fun part of data science and data analysis is probably figuring out which ones are the "useful" ones!

A part of me is positive that there should somehow be a model/algorithm that improves the prediction and lower the RMSE more, but given the time allowed, I don't have more time to work on a forth model. I hope that I can come back to it and work on it a some more in the future if possible.