MovieLens Rating Predictions

Agustin Gallo

2/4/2021

Introduction

We can start this project by asking several questions about our surrounding world such as, How does Google give us options when we are typing a search, and we haven´t even search for anything like that before? How is it that Spotify create the famous "Your daily mix" for every single day? Or youTube and Netflix suggest a new video in order to keep you seeing more and more thing up to late hours at night?

Well, all of them are recommendation systems, and its name says the recommend YOU new things given a specific context, but, what is this context? Well, it could be how you rate previous things, or which videos you've been watching past week, or even if according to your profile there are items other users with a profile similar to yours have a great rating, then, is probably you will like them too, therefor its a good recommendation and a good chance to consume the suggested product.

In this project we will create a movie recommendation system using the dataset located in http://files.grouplens.org/datasets/movielens. the data used for this consist in two tables, the first of them "movies" consist in a list of movies containing their ID, title and genre. The second table "rating" consist in the rating provided per user to a given movie and the time when this rating was issued.

Next you can see a glance of both tables:

movies table

```
##   movieId                              title
## 1       1                   Toy Story (1995)
## 2       2                     Jumanji (1995)
## 3       3            Grumpier Old Men (1995)
## 4       4           Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                        Heat (1995)
##                                        genres
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                  Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                       Action|Crime|Thriller
```

rating table

```
##    userId movieId rating timestamp
## 1:      1     122      5 838985046
## 2:      1     185      5 838983525
## 3:      1     231      5 838983392
## 4:      1     292      5 838983421
## 5:      1     316      5 838983392
## 6:      1     329      5 838983392
```

The main objective of the project is to create a pipeline able to create a rating prediction using the user ID and movie ID, but other information could be used for this prediction, as time, genre, etc. The prediction should be given in a range from 0 to 5 in a continuous range, so this is a estimation problem, therefor RMSE (Round Mean Squared Error) will be used to assess the results obtained.

The project is develop in 4 chapters, Introduction (this one), Method, Results and Conclusions. In method is explained code and logic behind the analysis; in results, plots and results are shown, obtained from the method and finally in Conclusions, I discuss other approach used and inconvenience behind those as other solutions used in the industry for recommendations systems.

Method

I will divide the structure of this project in 4 sections in this section I will describe each part and in the next chapter (Results) I will show the results obtained for every section described here. So here you might expect to see the code used in the project and the description of the logic behind each section. The 4 sections in which this chapter is divided are the following.

1. Get movie and rating data. Download and create movies and rating datasets, join dataframes and create test and validation dataframes.
2. First dummy attempt. Evaluation Criteria(RMSE) and first exploration of a dummy prediction.
3. Data exploration. Explore dataframe data to create a model hypothesis.
4. Model implementation. Based on data exploration create a prediction model approach and final result.

Get Movies and Rating data

The first step of everything is to get our data, in this case we will get this from the link http://files.grouplens.org/datasets/movielens.

After we download and load our data into the environment, we need to perform the join between the two tables, set our random seed in order for reproducibility and create the proper data partitions. * Train: 90% * Validation: 10%

Then, in order to ensure we have all the users and movies we perform a check and in case is in the validation but no in training we move this cases from validation to training.

Finally, all information that we will not be further needed, we erase it for memory sake.

```
# If file already exist avoid download
if (!file.exists("ml-10M100K/ratings.dat")){
  dl <- tempfile()
  download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

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

  # Getting movies data
  movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
  colnames(movies) <- c("movieId", "title", "genres")

}else{
  # if there is a rating dataset in the environment variable avoid
  if (!exists('ratings')){
    ratings <- fread(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
                   col.names = c("userId", "movieId", "rating", "timestamp"))
  }
   # if there is a movies dataset in the environment variable avoid
  if (!exists('movies')){
     movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)
     colnames(movies) <- c("movieId", "title", "genres")
    }

}

# Putting info in the desired format
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

```r
# -------------------------------------------------------------------
# Joining tables and data partition
# -------------------------------------------------------------------

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

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

# Validation set will be 10% of MovieLens data
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```r
edx <- rbind(edx, removed)

# Removing stg datasets to save space
#rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Dummy attempt

To evaluate if everything we make is going to make sense and get any improvement, we will try a dummy attempt to predict the rating to a given movie, to make this attempt we will simply average all movie rating and this average will be predicted rating for all movies.

Other way to perform something similar is to normalize the rating, get the average and use this average to predict the movie rating.

Using this aproaches we get:

- No normalize ratings = 1.06 of RMSE
- Normalize ratings = 0.99 of RMSE

```r
# Dummy evaluation without normalization
dumy_predict <- mean(edx$rating)
Error_NoNorm <- rmse( edx$rating , dumy_predict)
Error_NoNorm ## Non normalize error, 1.06 aprx
```

```
## [1] 1.060331
```

```r
# Creating Normalized rating
train_set <- edx %>% mutate(ratingN = (rating - mean(rating))/sd(rating) )

# Making our prediction
dumy_predict <- mean(train_set$ratingN)
sd_rating <- sd(train_set$rating)

Error_Norm <- rmse( train_set$ratingN , dumy_predict) ## Around 0.99 of error
Error_Norm ## Normalize error, 0.99 aprx
```

```
## [1] 0.9999999
```

Data Exploration

Now that we have a dummy approach, we will use it as a way to measure our improvement for the predictions make on any future model.

For this we will check what parameters have an influence in the movie rating. To tackle this we will explore the influence each variable has on the rating. With this we will be able to define the appropiate model for a movie rating prediction.

We start by assuming the model prediction, taking all the variables, is in the form of:

$$y = u + b(movieId) + b(userId) + b(genre) + b(time) + error$$

Where y is our movie prediction and u is an inherent bias of the model. So, next we will show a set of plots showing if there is any influence on the rating given by the parameter evaluated.

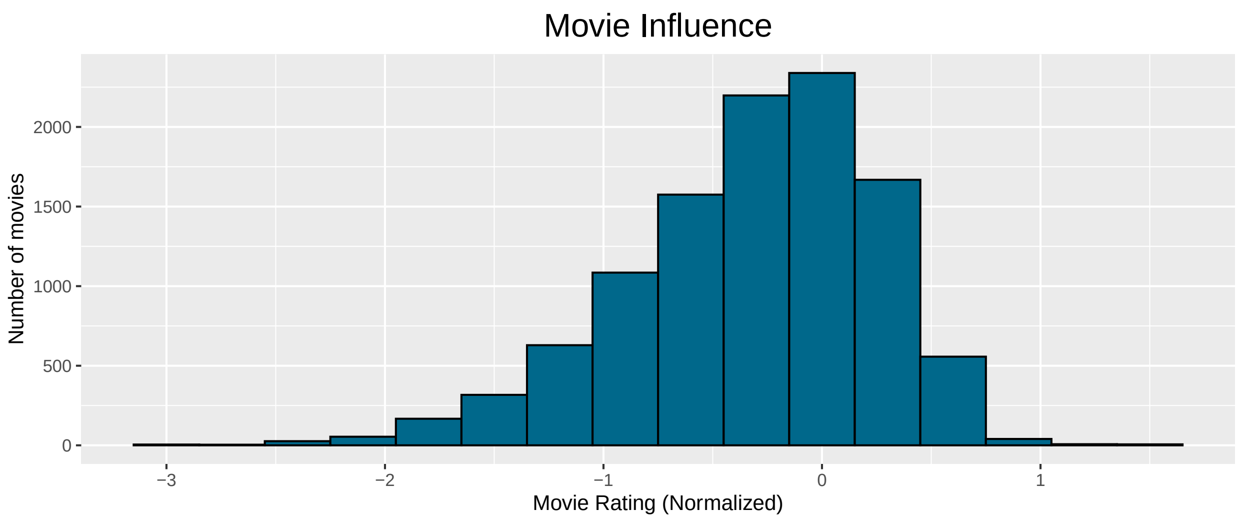

Figure 1. Influence of movieId in its rating. Some movies shown a preference to be evaluated higher

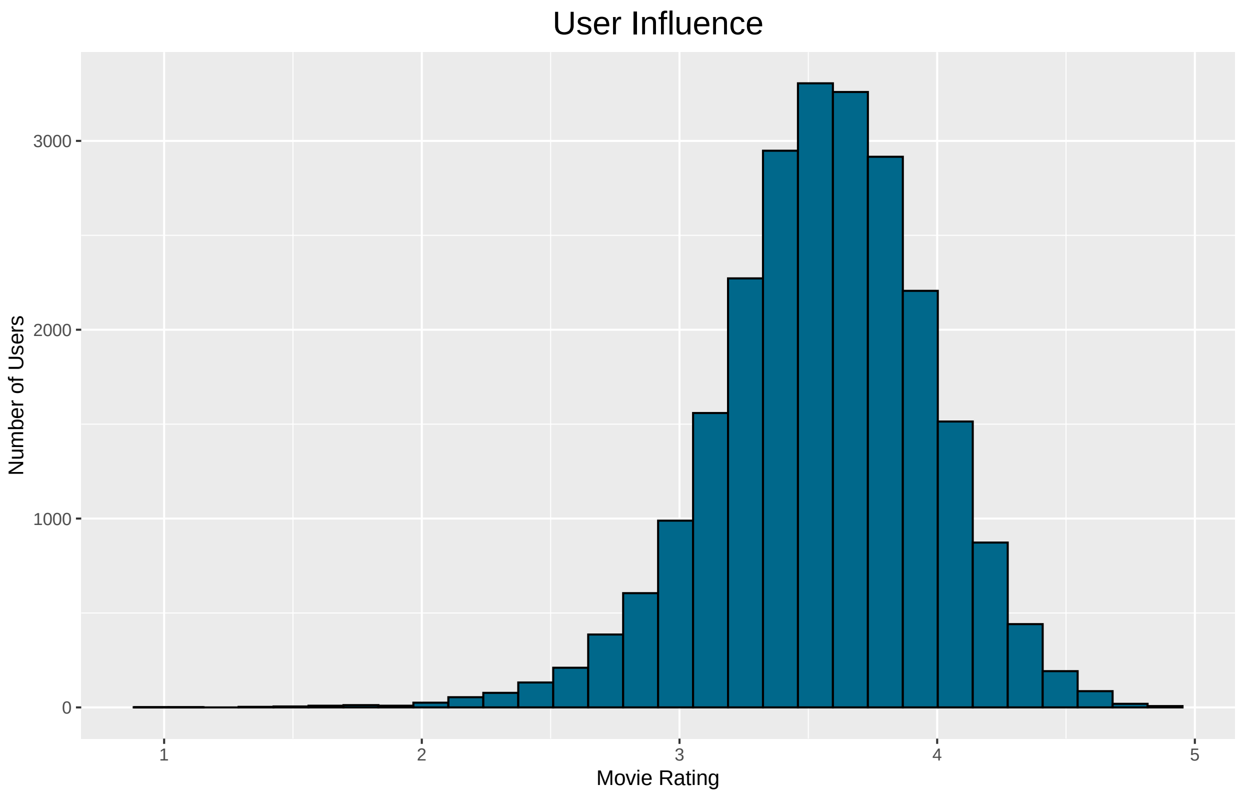

Figure 2. Influence of userId in its rating. Some users appear to rate higher the movies than others

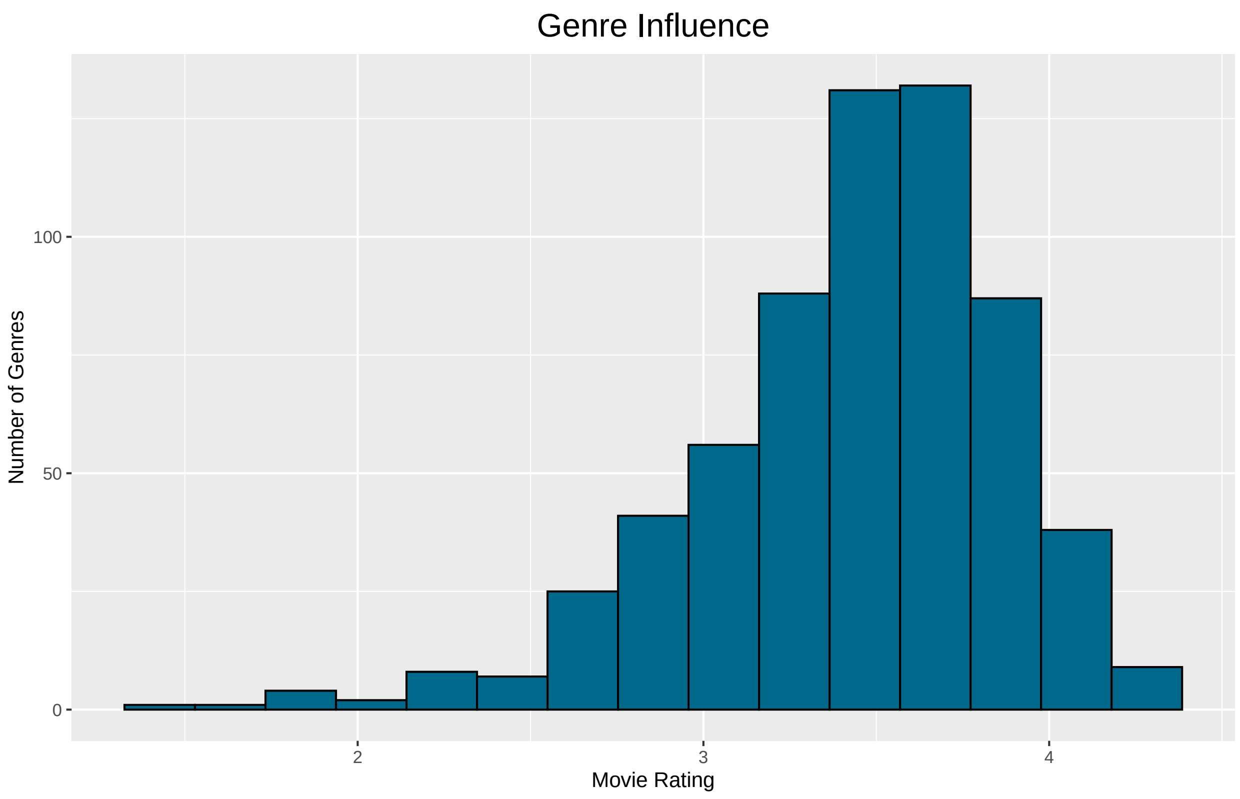

Figure 3. Some genres are prefered than others

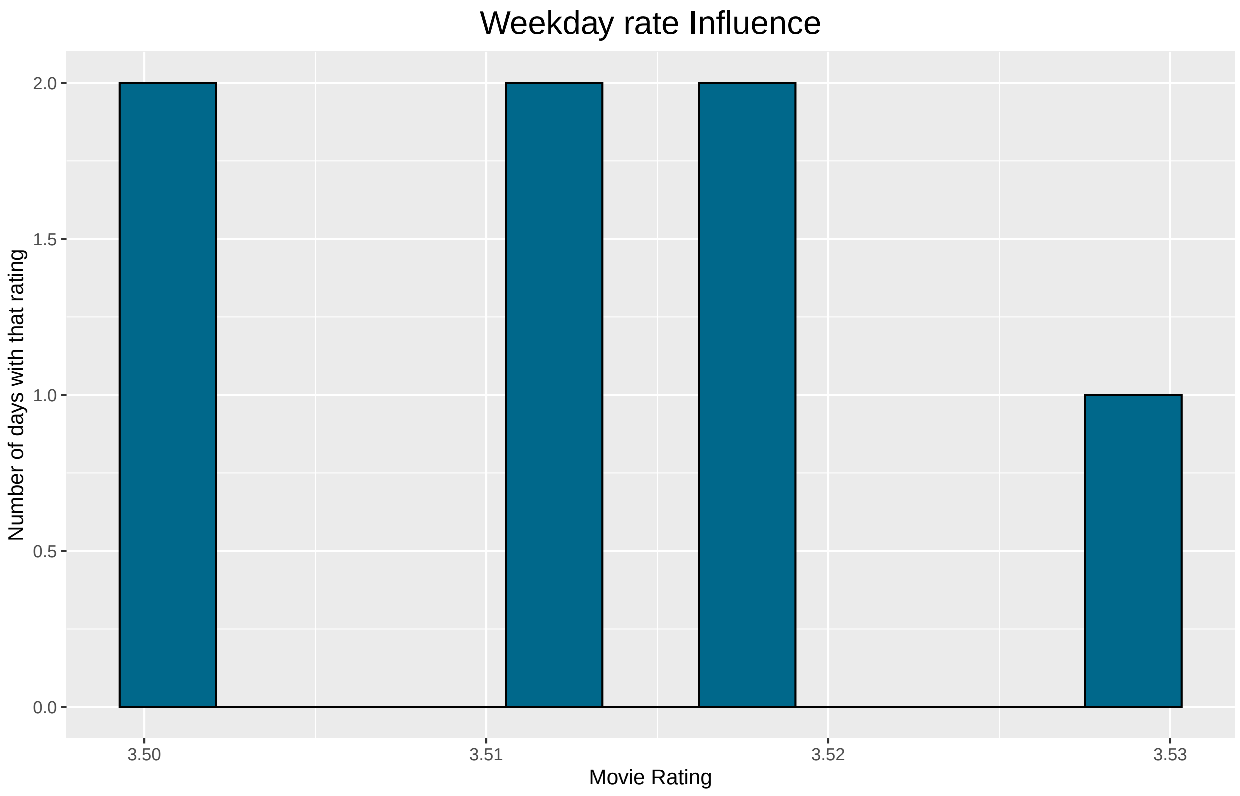

Figure 4. Not a clear indication of the weekday making a difference in ratings

Model Implementation

Based on the data exploration saw in the past point, we know we can implement a model similar to the shown before but taking out the time variable, so we get the model as follows:

$$y = mu + b(movieId) + b(userId) + b(genre) + error$$

After this model, we will also use the method of penalized least squared with the idea to adjust even more our prediction, this is made to the idea to weight the values of the ratings when we are grouping by the variable. This will change our model to be on the following form:

$$y = mu + \frac{b(movieId) + b(userId) + b(genre)}{n() + lambda} + error$$

where n() is the number of member of n group.

With this we get to our final model and the results will be display in the next section.

```r
# We get our bias, in this case or rating average
mu <- mean(train_set$rating)

# Our movie component, where long story short is the rating normalized as b_i
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# We get the user component as b_u
user_avg <- train_set %>%
  left_join(movie_avgs, by = 'movieId')%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating- mu - b_i))

# We get or genre component as b_g
genres_avg<-train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avg, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))


# Making prediction with our final model pred = mu + b_i + b_u + b_g
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genres_avg, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  select(pred)

# If by any mean the prediction is above 5 we set to that value,
# and similar to values below 0, we set them to 0
predicted_ratings_noPenalized <- predicted_ratings %>%
  mutate(pred = if_else(pred>5,5,if_else(pred<0,0,pred))) %>%
  pull(pred)
```

```r
# ----------------------------------------------------------------
## Now using penalized least squares, with penalty term Lambda
# ----------------------------------------------------------------
lambdas <- seq(0, .5, 0.05)


mu <- mean(train_set$rating)

# In order to evaluate the best lambda we need to iterate across many values of lambdas
# and get our final model. This may take a little if seq(0, 3, .05) or larger is used,
# so I cut it to 0-.5, to appreciate RMSE variation on future plot.

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

    # Pretty same as before but we and a division to n() + l
    movie_avgs <- train_set %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))

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

    genres_avg<-train_set %>%
      left_join(movie_avgs, by = 'movieId') %>%
      left_join(user_avg, by = 'userId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

    predicted_ratings <-
      train_set %>%
      left_join(movie_avgs, by='movieId') %>%
      left_join(user_avg, by='userId') %>%
      left_join(genres_avg, by='genres') %>%
      mutate(pred = mu + b_i + b_u + b_g) %>%
      select(pred)

    predicted_ratings <- predicted_ratings %>%
      mutate(pred = if_else(pred>5,5,if_else(pred<0,0,pred))) %>%
      pull(pred)

    # We use train rating, as in this part we are still training the model.
    return (rmse(train_set$rating,predicted_ratings))
})


lambda <- lambdas[which.min(rmses)] # Best Lambda is 4.75

# Now we construct or final model with the given lambda and apply it to
# the validation data.

#Final model will be
```

```
l <- lambda

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

# User Component
user_avg <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + l))

# Genres Component
genres_avg <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avg, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + l))

# Make our final prediction with the validation data
predicted_ratings <-
  validation_clean %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avg, by = 'userId') %>%
  left_join(genres_avg, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  select(pred)

predicted_ratings_penalized <- predicted_ratings %>%
  mutate(pred = if_else(pred > 5, 5, if_else(pred < 0, 0, pred))) %>%
  pull(pred)
```

Results

In this section we will display and discuss the results obtained using our final model. As we saw on the last piece of work we use a value named lambda to penalized the predicted ratings, to find the best lambda we iterate over a list of lambda values and get the predicted RMSE, next we can see the RSME obtained each of this values.

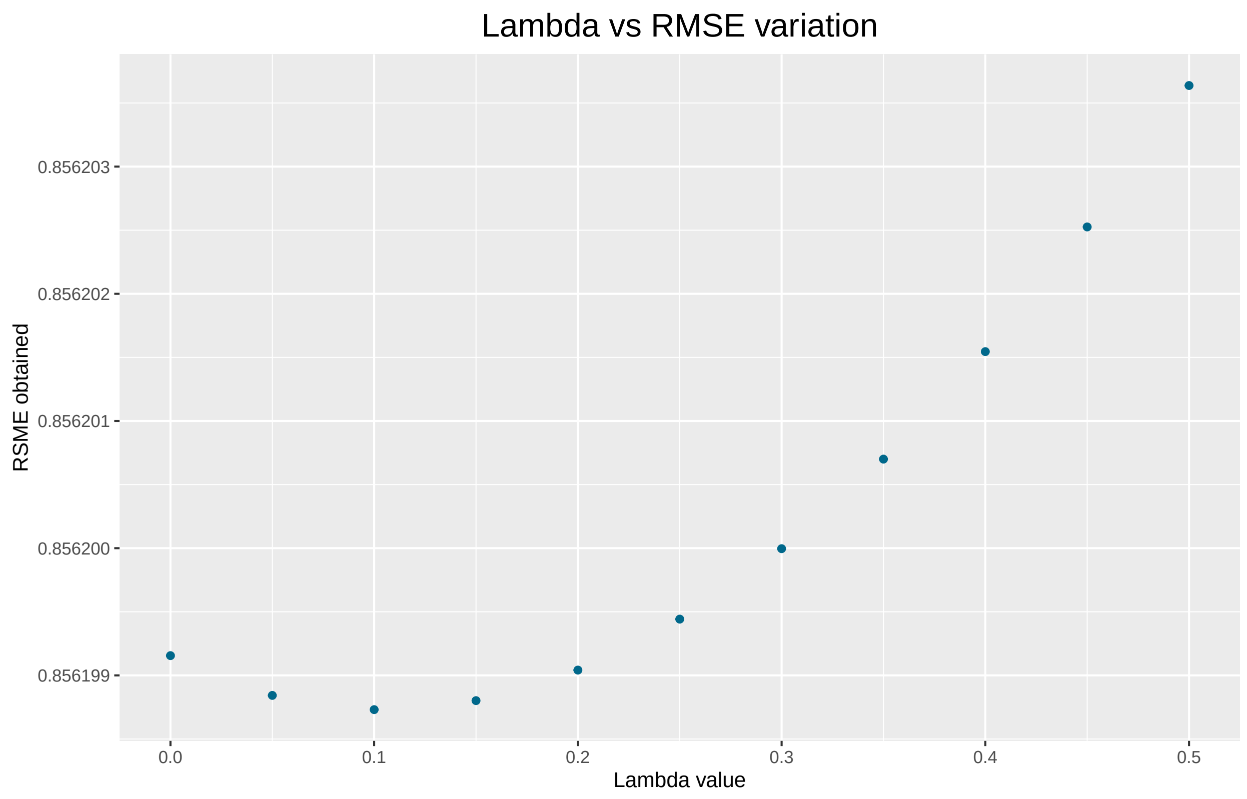

Figure 5. Variation of RMSE according to the penalized term lambda used

There we can see there is a little valley according to the best lambda values, this values corresponds to **0.1**. With this it mind, we apply this to our predictions, in this case to the validation data instead of the training data. We will use this to compare the results using a penalized term and not using it.

```
## Find RSME in not penalized model predictions
RSME_notPenalized <- rmse(validation_clean$rating, predicted_ratings_noPenalized)

## Find RSME in not penalized model predictions
RSME_penalized <- rmse(validation_clean$rating, predicted_ratings_penalized)

## Now showing the result
RSME_notPenalized
```

```
## [1] 0.8647678
```

```
RSME_penalized
```

```
## [1] 0.8647429
```

Summarizing all the results

Method	RMSE
Naive	1.060331
Naive Regularized	0.999999
Model no Penalized	0.864767
Model Penalized	0.864742

We can see, the main improvement was made when applying a model to our data, an finally a little improvement when using the penalized model. This significant improvement is expect as our first pridiction is very simple and just taking into account the average movie rating, when we use an approach which takes movie, user, and genre the prediction is much better.

We see a small improvent when penalizing terms because after all is just a fine tune value and appears that this terms do not significantly impact our predictions, this way we obtain a main square error of **0.86474** in our final model.

Conclusion

To conclude, this report shows a way to build a movie prediction model using information provided by the movie lens dataset, we did a parameter exploration in order to find the variables most impacting a movie rating. With this information we built a Penalized Model to make predictions in our test data getting a RMSE of 0.86474.

Our final model includes information regarding the user tendency to rate movies, the inherent nature of a movie to be scored higher and the popularity of the different genres. Nonetheless we do not include any kind of seasonality on the user preference, this meaning when a people for a short time experiments a great like or dislike for some kind of movies and after a short period of time, it disappears. This approach will be allso interest to perform and probably improve our model.

On this work also we did not use any kind of machine learning model nor deep learning model, which will be interesting to try, and see the improvements existing between those models and this one, if any.