MovieLens System Report

Made Ots

07/03/2022

INTRODUCTION

The following is a report based on a movie recommendation system using the MovieLens dataset. The report and analysis is part of an assessment for an edX PH125.9x Data Science Capstone Project. Recommendation systems use ratings which users have given to the movie based on their personal prefernces. Companies selling products to different client segments allow their customers to give ratings to their products. This results in creation of huge data sets like the MovieLens set, which can then be used to predict what type of product to recommended to specific and similar users. The recommendations are based either on content or on collaborative filtering system. Recommendation systems are one of the most used models in machine learning algorithms. Spotify, Netflix and Amazon are all using different types of recommendation systems based on customer registered behavioural preferences.

Github - https://github.com/Veronicaots/Harvard-CAPSTONE

GOAL

The aim of this assessment is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. Reference code has been provided to generate the training and validation sets that should be leveraged for this analysis. RMSE, known as the Root Mean Square Error, will be used to evaluate how close the predictions made by our model are to the actual/true values contained in the validation set. The desired RMSE should be a value less than 0.86490.

OVERVIEW

1. Explore the Movielens 10M dataset
2. Split the edX data set to Train and Test sets for cross validation
3. Train and tune your model on parameters using the Train set, validate on Test set
4. Apply your trained model to Validation data set and control your results (final RMSE)
5. Conclusion

The data set given for this project with the initial set-up code.

Next we can explore the Structure of the given data set.

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Summary of the Movielens 10M data set

Table 1: Summary of Edx Data Set

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

The above information is helping us to define the parameters to use for creating our ML algorithm for building the recommendation system. While we can work directly with userId, movieID and rating, we would need to extract the release date from the movie title and convert it into date format. # User # Movie # Release Year

```r
# Extract the date from edx and validation sets
edx <- edx %>%
  mutate(year = as.integer(str_extract(str_extract(title,"\\((\\d{4})\\)$"),"\\d{4}")))
validation <- validation %>%
  mutate(year = as.integer(str_extract(str_extract(title,"\\((\\d{4})\\)$"),"\\d{4}")))
```

We can explore further to see what types of genres are included in the data set

genres	rated
(no genres listed)	7
Action	24482
Action|Adventure	68688
Action|Adventure|Animation|Children|Comedy	7467
Action|Adventure|Animation|Children|Comedy|Fantasy	187
Action|Adventure|Animation|Children|Comedy|IMAX	66

There is in total 6 genres noted in the data set, one of which is not defined.

What is the most presented genre in this data set

count	genres	nr
7	Action|Adventure|Comedy|Drama|Fantasy|Horror|Sci-Fi|Thriller	256
6	Adventure|Animation|Children|Comedy|Crime|Fantasy|Mystery	10975
6	Adventure|Animation|Children|Comedy|Drama|Fantasy|Mystery	355
6	Adventure|Animation|Children|Comedy|Fantasy|Musical|Romance	515
5	Action|Adventure|Animation|Children|Comedy|Fantasy	187
5	Action|Adventure|Animation|Children|Comedy|IMAX	66

From the above is clear that the most common movies are the Action & Adventure.

We should also see to the ratings summary to help us to better understand the way how users rate the movies.

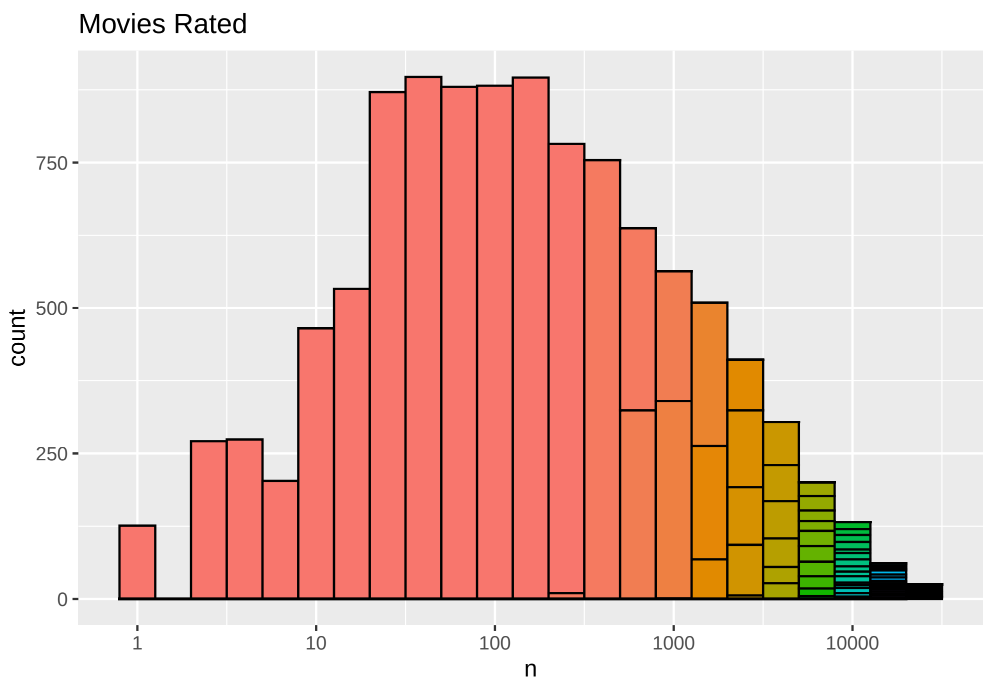
rating	given
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

Establish the unique movies and unique users

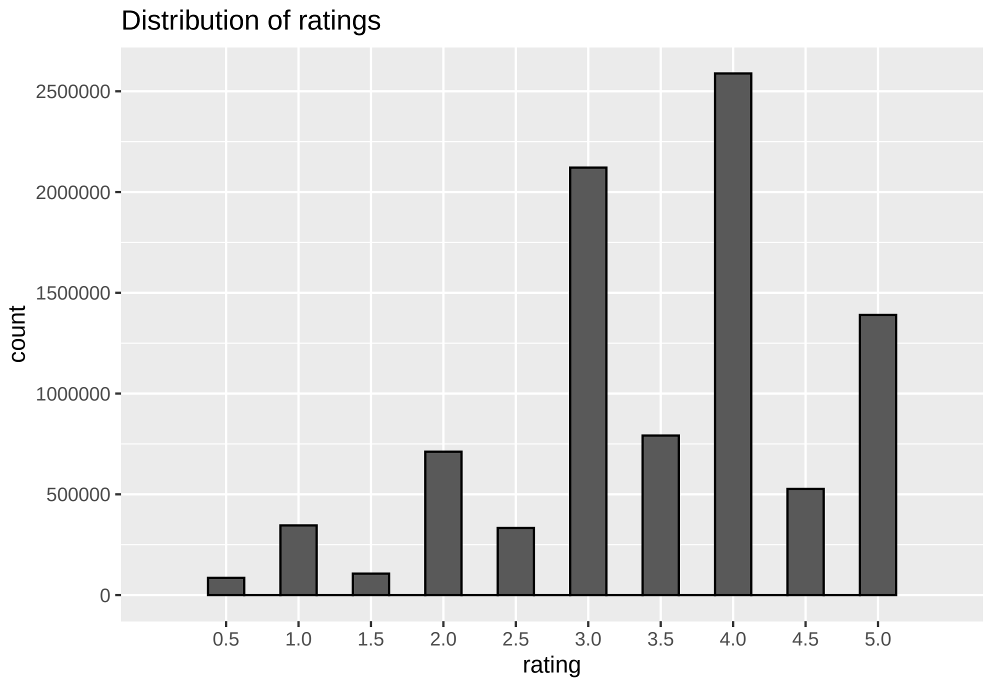
n_users	n_movies
69878	10677

There is almost 70000 users who have given their rating for total of 11000 movies. If all users rated every movie title, the dataset should contain 746 Million rows but there are only 9 Million records in our set, just a little more than 1%. This shows the low number of points we can work with and poses a difficulty to develop a recommendation system algorithm giving an accurate rating predictions for a given movie and a given user.

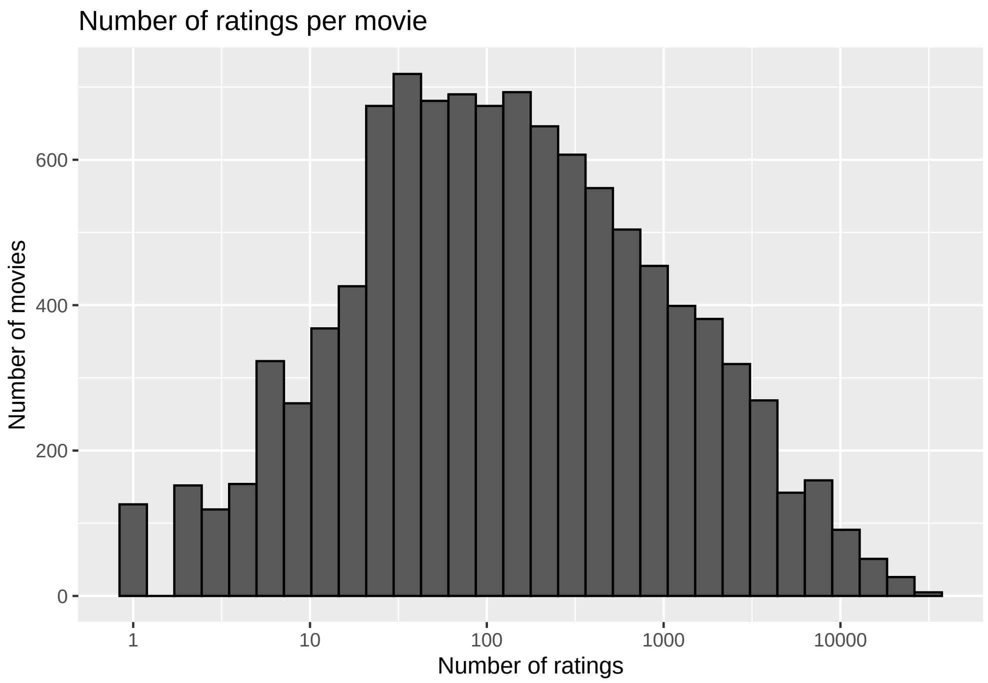
Most rated movies



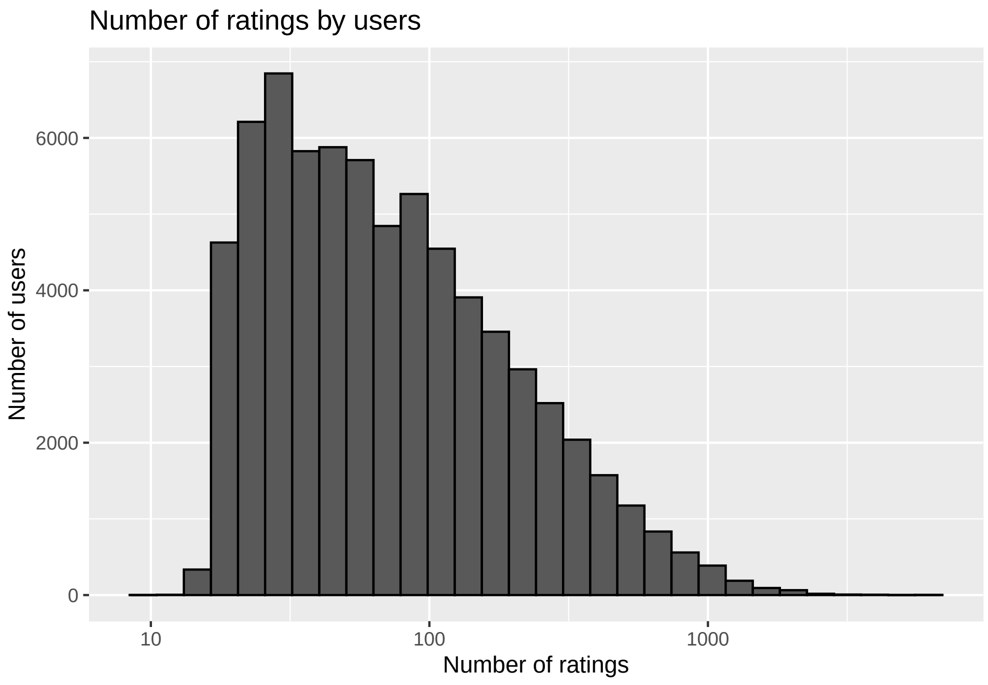
Distribution of the ratings looks a bit off because most users have been giving an even rating to the movie.



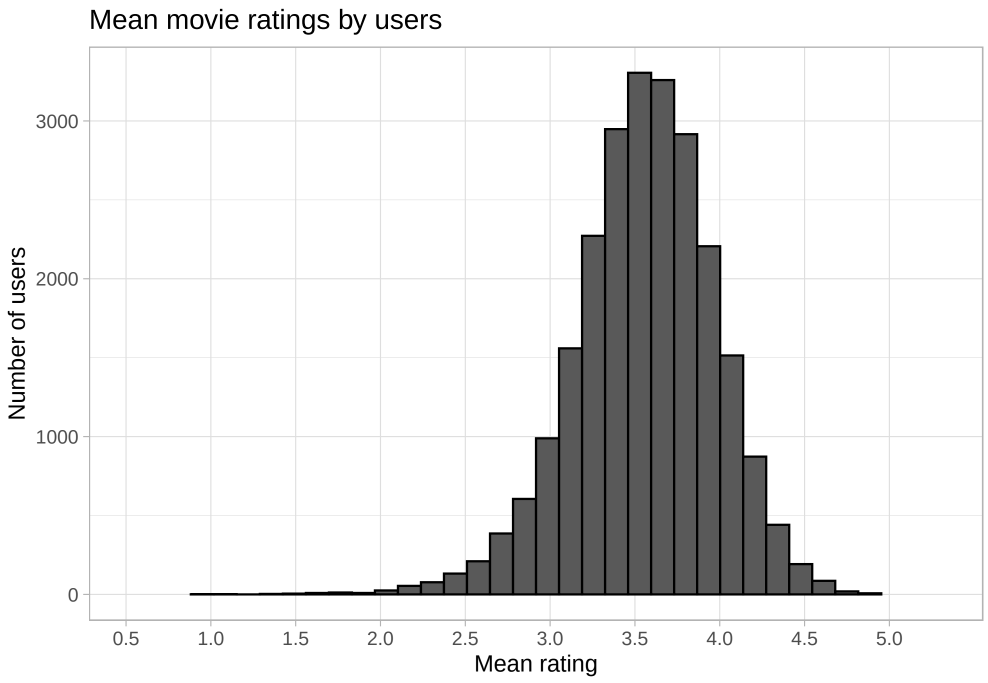
Next we will investigate the number of ratings per movie



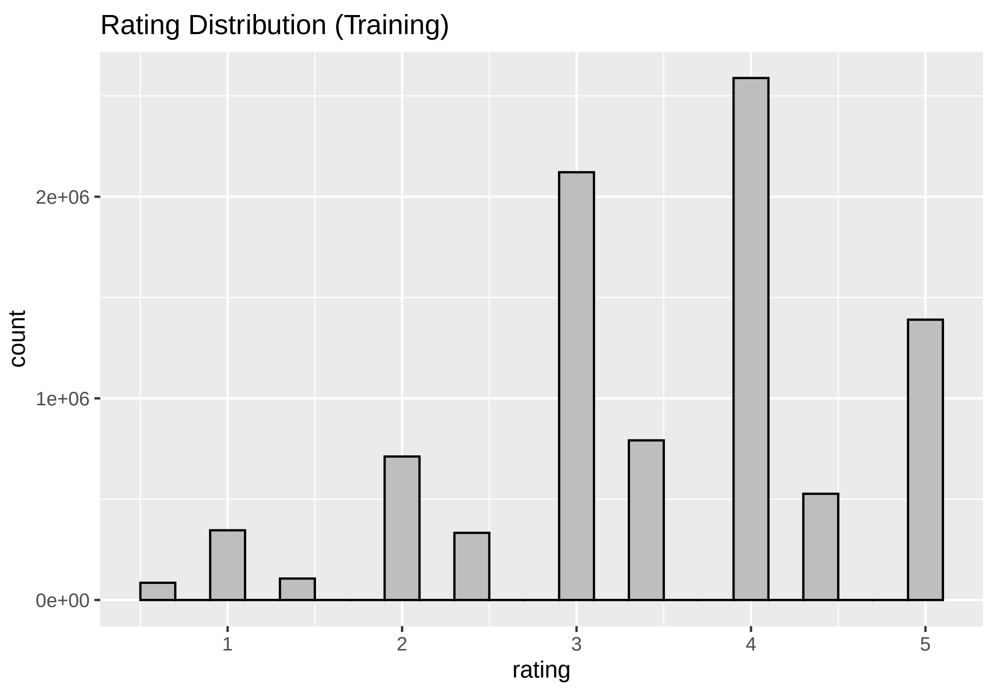
Having a better view now of the movie ratings, let's see what we can find about the users. This would help us to build our prediction model. The following plot shows the ratings per user.

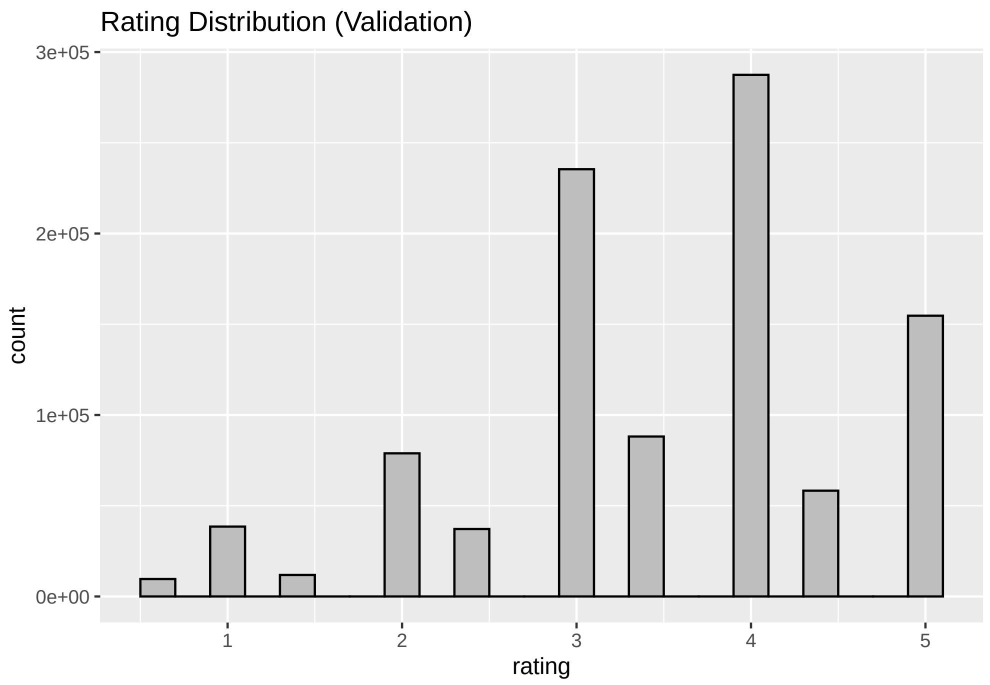


We have to see the mean to be able to draw the conclusion of which methods would be the best to use to train our recommendation algorithms.



Before starting to work on models of the recommendation system, we need to make sure that the distribution is similar in both the edX Training set and edX Validation set





Both data sets are similar in their ratings so we can assume that if our Machine learning algorithm is working on training data set, it will be applicable also on Validation set.

2. SPLIT THE EDX SET TO TEST SET & TRAIN SET

```
set.seed(1996)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,list = FALSE)
train_set <- edx[-test_index,]
test_set_tmp <- edx[test_index,]
test_set <- test_set_tmp %>%
  semi_join(train_set, by = "movieId") %>% semi_join(train_set, by = "userId")%>% semi_join(train_set, 
train_set <- rbind(train_set, anti_join(test_set_tmp, test_set))
rm(test_set_tmp)
```

All the following training and modelling is made on Train Set and the results are cross validated on Test Set.

3. MODELING METHODS

3.1 Average Method

We can startwith finding about about the average rating in the Train Set

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

```
## [1] 3.512465
```

Based on our data set inspection in previous section, we know that there are very few data points to support our algorithm building and while inspecting the initial data set, there were missing ratings. We need to calculate for these unknown or "cold ratings" to be able to compensate for them in our model.

```
RMSE(train_set$rating, mu)
```

```
## [1] 1.060476
```

We applied the basic Average Model to our Train Data Set which resulted to robust RMSE far over the limnits set out for this assessment. We need more complex approach to this data set.

Evaluating the result with basic RMSE function:

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

The resulted RMSE value is too high to satisfy working recommendation system. We need to use other approaches and models to lower the RMSE

3.2 Using Naive - Bayes

Machine Learning is the "art of prediction". It is about predicting the future while learning from the past. The algorithm first creates a frequency table (similar to prior probability) of all classes and then creates a likelihood table. Finally, it calculates the posterior probability.

x
0.8659845

The RMSE has improved significantly, but we would like our recommendation system to be more accurate and are trying to reach the desired RMSE of 0.8649.

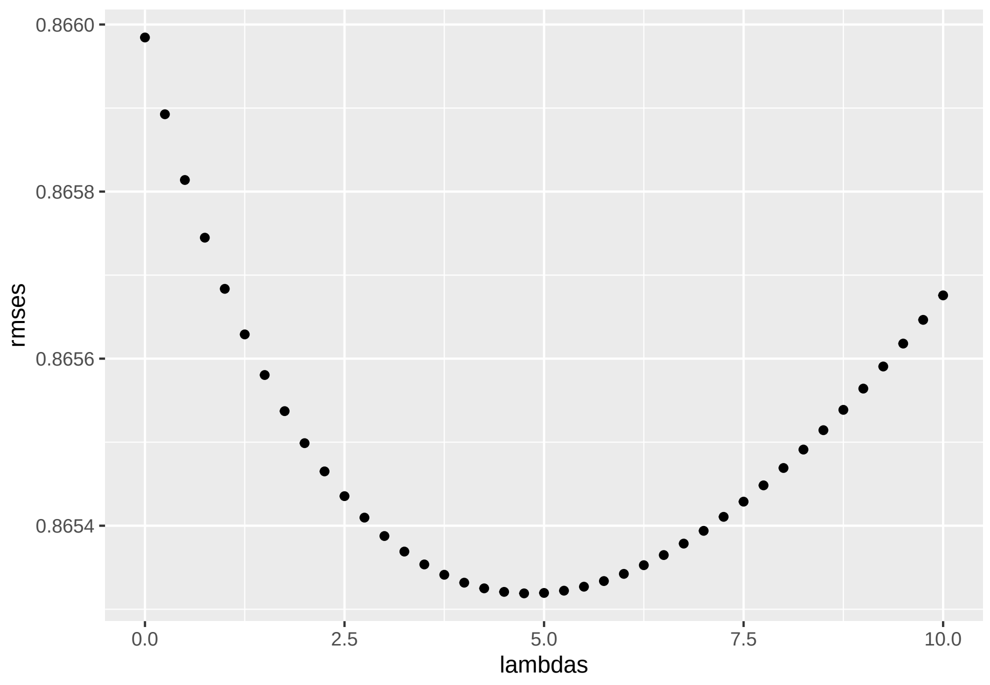
3.3. Regularized Movie and User Effect method

Below we are using a model with regularized 3 bias by a single parameter -> lambda. Regularization permits us to penalize large estimates that come from small sample sizes. It has commonalities with the Bayesian approach that "shrunk" predictions. The general idea is to minimize the sum of squares equation while penalizing for large values of b_i.

```
lambdas <- seq(0,10,0.25)
rmses <- sapply(lambdas, function(l){
  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_y <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_u - b_i - mu)/(n()+l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred = mu + b_i + b_u + b_y) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

To be sure that the Regularized Model is resulting in the best result, let's compare our RMSEs against lambda values:

```
qplot(lambdas, rmses)
```



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

```
[1] 4.75
```

```r
min(rmses)
```

```
[1] 0.865319
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Regularized movie + user + year effect",
                                     RMSE = min(rmses)))
```

The best lambda value is 4.75 and the RMSE is now 0.865319 and we are very close to our desired RMSE target

4. The last step would be to cross test and validate our best RMSE resulted model against the edX Data Set

```r
b_i_tuned <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_tuned = sum(rating - mu)/(n() + best_lambda))
b_u_tuned <- edx %>%
  left_join(b_i_tuned, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u_tuned = sum(rating - b_i_tuned - mu)/(n() + best_lambda))
b_y_tuned <- edx %>%
  left_join(b_i_tuned, by = "movieId") %>%
  left_join(b_u_tuned, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y_tuned = sum(rating - b_i_tuned - b_u_tuned - mu)/(n() + best_lambda))
```

Evaluating the model against the Validation Data Set:

```
predicted_ratings <- validation %>%
  left_join(b_i_tuned, by = "movieId") %>%
  left_join(b_u_tuned, by = "userId") %>%
  left_join(b_y_tuned, by = "year") %>%
  mutate(pred = mu + b_i_tuned + b_u_tuned + b_y_tuned) %>%
  pull(pred)
final_rmse <- RMSE(predicted_ratings, validation$rating)
final_rmse
```

```
## [1] 0.8645223
```

This results to the RMSE value below of 0.8649 which was the aim. There were significant differences in RMSE improvement between simple Average Model, Naive-Bayes and the Regularized Movie Effect + User Effect Model. We could try to fit it better, but because of the danger of over fitting and with sparse data points given in the system, the Regularized Effect method with 3 points seems to be the best to my knowledge in this situation.

5. CONCLUSION

The RMSE we reached at the end is almost 20% better than the one we achieved with Simple Model using the Mean as predictions. Our current best predicting model with bias is intuitive which means that it is easy to understand for everyone. Weakest point of this model is the small sample size. When using the regularization, we minimize the the weight of the sample and while it is good for RMSE result in this work, in general, we are not able to predict ratings for new film or ratings of a new user.

Machine learning has so many applications to business, science, and medicine that I believe it will be the most sought after profession of 21st century.