MovieLens Project

Jose Velasco

03/Sep/2020

Movie Recommendation System

This report is elaborated for the Capstone project of the HarvardX: PH125.9X Data Science course.

The motivation of this excercise is to generate a machine learning algorithm that recommends movies based on movie rating predictions. First of all, a general idea to recommendation systems is presented. Then, the data will be explored and summarised in order to get familiarised with it. In the third place, a machine learning algorithm is gradually developed to predict movie ratings. Finally, results and conclusions will be presented.

1 Introduction:

Section that describes the dataset and summarizes the goal of the project and key steps that were performed.

Recommendation systems use ratings that users have given to make specific recommendations. Specifically, a movie recommendation system predicts how many stars a user will give to a movie. One star suggests that the user did not like the movie and five stars imply that the user loved the movie.

A dataset was provided by Edx. It includes more than 10 millions of movie ratings for around 70,000 movies and 10,700 users. The data wrangling process was also provided by Edx. The starting dataset includes movie rating, user id, movie id, title. In this analysis the year is extracted from the title to further expand the model and achieve a better result.

In this report, a movie recommendation system is constructed gradually, starting from a baseline model which predicts any movie rating by simply taking the average rating of all movies. The second model includes a movie effect and the third model a user effect. The third model incorporates a year effect. The final model is a regularized model which takes into account the previous models plus a penalization. The model will be evaluated using RMSE.

2 Methods and Analysis:

Section that explains the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and your modeling approach.

2.1 Data Exploration:

In order to get familiarized with the data, we briefly present it. The following piece of code was provided by Edx to create a two separate datasets: the train set and the test set:

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# Note: this process could take a couple of minutes

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(data.table)) install.packages("stringr", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

#the stringr library is added to expand the model
library(stringr)

# 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)` instead
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)
```

First, the required packages are downloaded from the provided link and loaded into R. The stringr package is also included in this analysis to further expand the model as will be shown later.

Second, the dataset is downloaded and a data wrangling process is performed in order to get a clean dataset from which we can derive the model.

The dataset is split in a training set (Edx), which comprises 90% of the data, and a test set (validation), the remaining 10% of the data.

Furthermore, a year of release column is created for each movie.

```
edx <- edx %>%
  mutate(year = as.numeric(str_sub(title,start= -5,end= -2)))
validation <- validation %>%
  mutate(year = as.numeric(str_sub(title,start= -5,end= -2)))
```

It is also validated that the year column does not have any missing values.

```
summary(edx$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1915    1987    1994    1990    1998    2008
```

```
summary(validation$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1915    1987    1994    1990    1998    2008
```

A quick summary of the data shows a matrix of the following characteristics. Each column represents a variable and each row represents a different rating given by one user to one movie in particular.

```
dim(as.matrix(edx))
```

```
## [1] 9000055       7
```

```
names(edx)
```

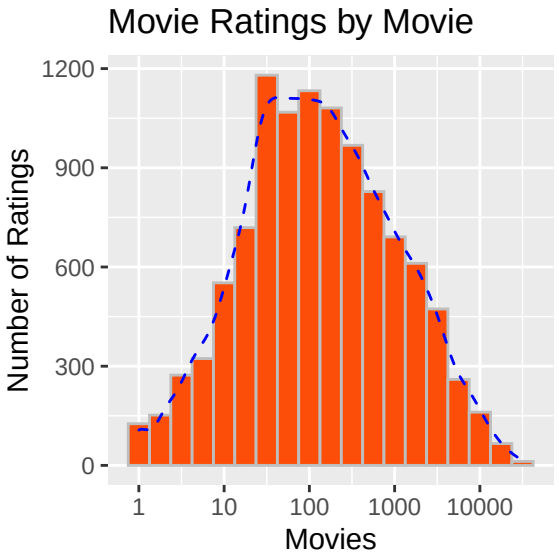
```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
## [7] "year"
```

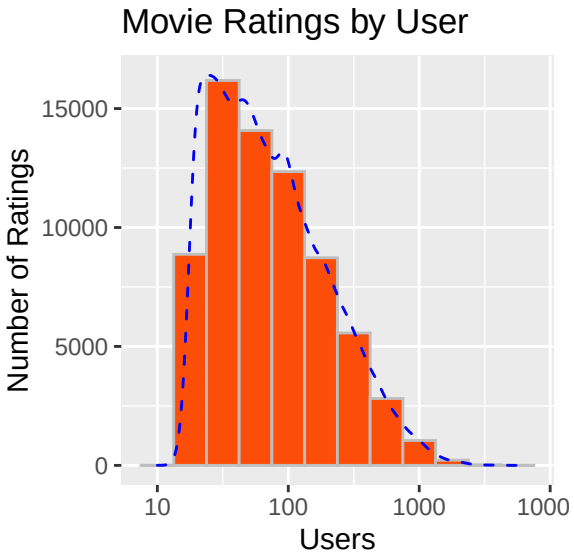
Furthermore, we can see that there are around 70,000 different users and 10,700 different movies.

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

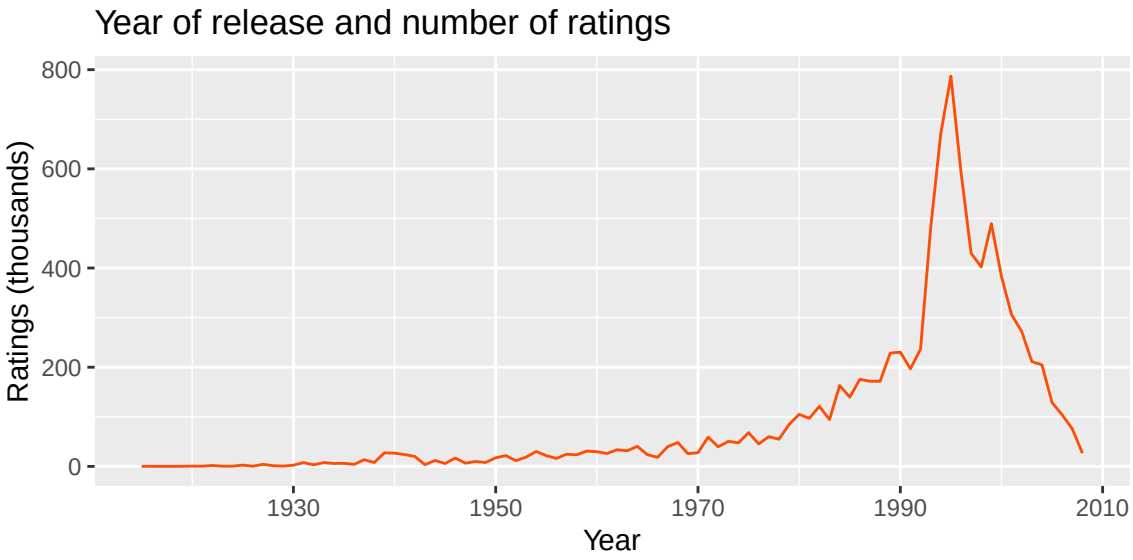
We examine now that not all movies are rated equally and not all users are frequent raters. There are movies that are generally acclaimed by the public and some movies are just not that good. On the other hand, there are users who are very strict (like movie critics) and some users that love every movie they watch.





Additionally, movies released after 1990 tend to have more ratings than movies released before that date. This could be explained by the assumption that most people tend to view recent movies and only a handful would watch old movies (sometimes considered classic or art movies).

It is also noted that recently released movies tend to have fewer ratings than movies around 1990-2000. This could be explained by the fact that in order to get more ratings more time is needed.



After exploring the data, it is reasonable to think that a model can be derived using movie ratings, users and year of release as explanatory variables.

As can be noted, not all users rate every movie, the most active users rate slightly more than 15,000 movies, whereas the dataset has almost 70,000 movies. The dataset has many blank spaces and the machine learning algorithm can be thought of as if the blank spaces were going to be filled by it, thus predicting ratings for each movie and user in order to recommend movies for users.

2.2 Model Evaluation:

The model performance is going to be evaluated through the RMSE (residual mean square error). The function is defined as follows:

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

Let $y_{u,i}$ be defined as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$, with N being the number of user/movie combinations and the sum occurring over all these combinations.

The RMSE is similar to a standard deviation: it is the typical error that is made when predicting a movie rating. If this number is larger than one, it means our typical error is larger than one star, which is not good.

The following function computes the RMSE for vectors of ratings and their corresponding predictors:

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

2.3 Deriving the Model

In order to tackle the problem, a similar approach as this one is used: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/.

It is noted that the prediction can be decomposed in three factors:

a) baseline rating (the mean of all rows)
b) user effect
c) movie effect

This factorization is useful since there are better movies that tend to be rated better and there are different users that could tend to rate lower or higher on average.

Furthermore, a fourth factor is added:

d) year of release effect

The rationale behind this decision is that different trends characterizes different decades for movies. We can find artsy films mostly in the first decades, action films during the 80s and 90s and a lot of special movie effect after the 2000.

2.3.1 Baseline Model This model is going to be useful as it will be the starting point from which we will reduce our RMSE. This model takes the naïve approach of predicting every movie with the mean of all ratings across all users. Everything else is explained by randomness, so the model looks like this:

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

The estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

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

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

This model predicts all unknown ratings with $\hat{\mu}$ and obtains the following RMSE:

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

This result can be interpreted as the typical error it is made when the model fills a blank space with no rating.

The results are stored in the following table:

```
rmse_results <- tibble(method = "Baseline model", RMSE = naive_rmse)
```

2.3.2 Modeling Movie Effects Some movies are rated higher than others, as noted before. The previous model can be augmented by adding the term b_i to represent average ranking for movie i:

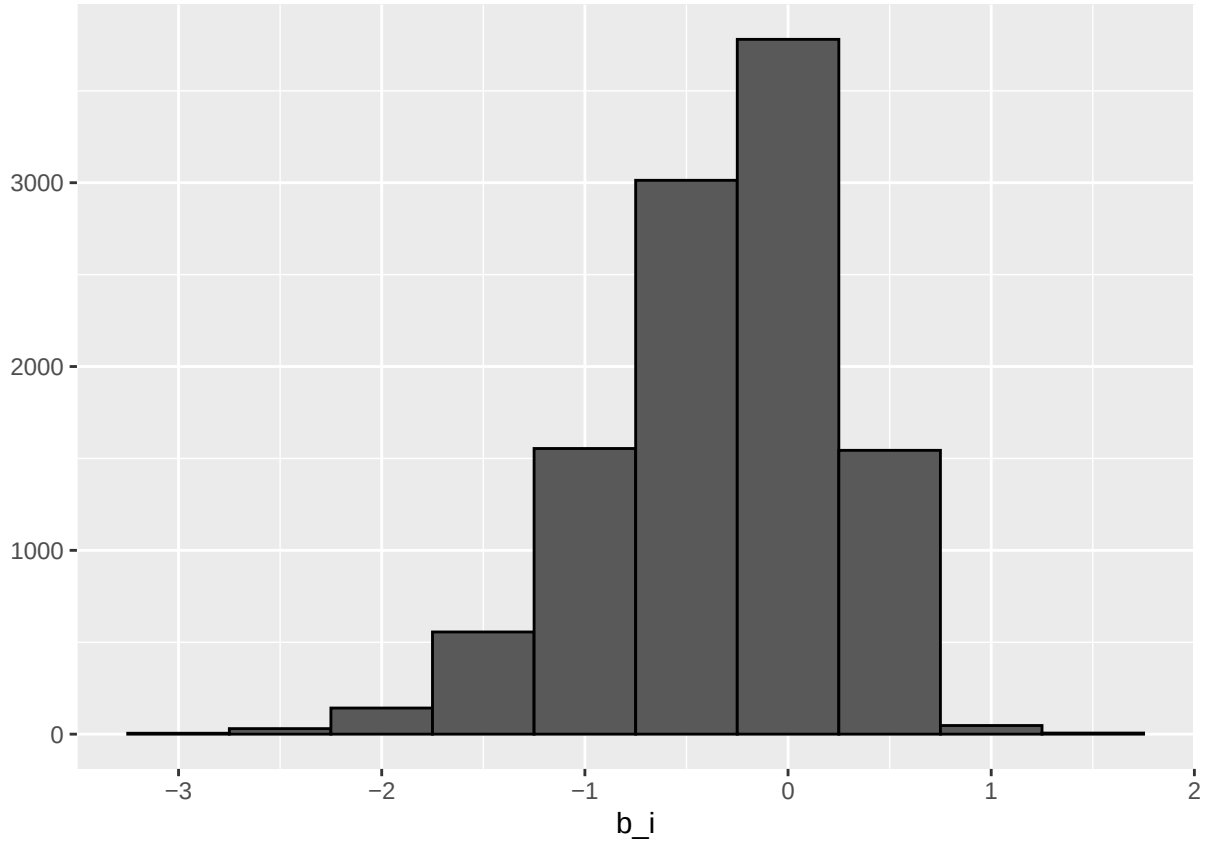
$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

In this particular situation, the least squares estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So it is possible to compute that term instead of computing the least squares model:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

These estimates vary substantially and are centered around the mean:

```
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Now, the new predictions are calculated using the expanded model $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$ and the RMSE is also computed:

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.9439087
```

Then, the results are stored:

```
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = model_1_rmse))
```

The RMSE drops by 12% to 0.9439, a significant improvement from the naïve approach.

2.3.3 User Effects Model Since there is substantial variability across users, the model can still be improved by adding a user effect:

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

Where b_u is a user-specific effect.

An approximation is computed for $\hat{\mu}$ and $\hat{b}_i$ and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$:

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Then the predictors are constructed and the RMSE again diminishes:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8653488
```

```
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                 RMSE = model_2_rmse))
```

2.3.4 Year Effects Model For the third model, we add the folowing term: b_y which represents the year effect:

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

An approximation is computed for $\hat{\mu}$ and $\hat{b}_i$ and $\hat{b}_u$. Let $\hat{b}_y$ be estimated as the average of $y_{u,i,y} - \hat{\mu} - \hat{b}_i - \hat{b}_u$:

```
year_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))
```

The predicted ratings are computed and a new RMSE estimated:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8650043
```

The results are stored:

```
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Year Effects Model",
                                 RMSE = model_3_rmse))
```

2.3.5 Regularized Model In order to improve the results, a regularization method based on penalized least squares is applied. Instead of minimizing the least squares equation, an equation that adds a penalty term λ (lambda) is minimized:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

The first term is just least squares and the second is a penalty that gets larger when many b_i are large. The values of b_i that minimize this equation are:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

Where n_i is the number of ratings made for movie i. This approach will have the desired effect: when the sample size n_i is very large, a case which will result in a stable estimate, then the penalty λ is effectively ignored since $n_i+\lambda\approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0.

For the generalized model with three explanatory variables we consider the following equation:

$$\sum_{u,i,y}(y_{u,i,y}-\mu-b_i-b_u-b_y)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\ +\sum_y b_y^2\right)$$

2.3.5.1 Cross Validation In order to find the optimal λ, a process of cross-validation is needed. First, the Edx dataset is split in train and test sets:

```
set.seed(1998)
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

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

rm(test_index, temp, removed)
```

Then, different values of λ are tested across the train set in order to find the value of λ that minimizes RMSE.

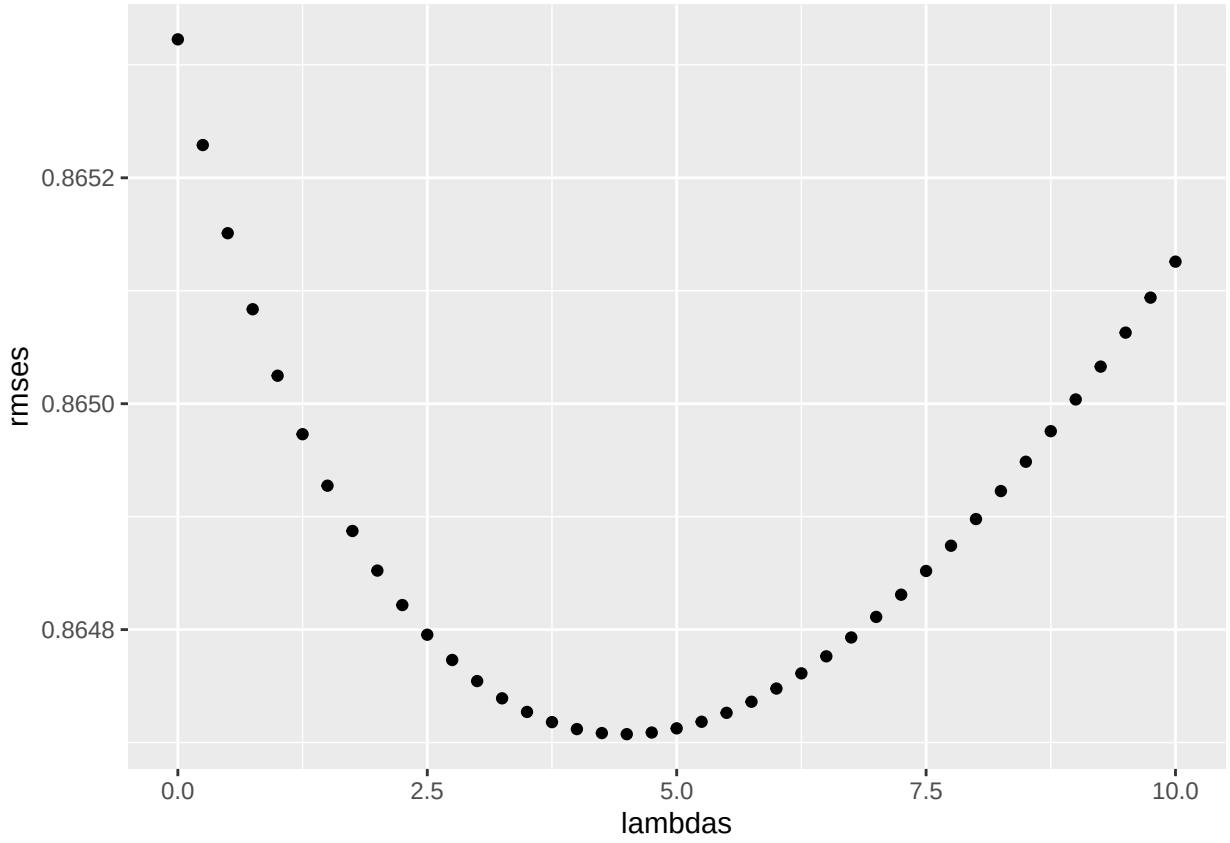
```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mu_train <- mean(train_set$rating)

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

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_train)/(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_i - b_u - mu_train)/(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_train + b_i + b_u + b_y) %>%
    pull(pred)

  return(RMSE(as.matrix(test_set$rating), as.matrix(predicted_ratings)))
})
```

Now, it is noted that the best value of lambda is between 4 and 5:

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

```
## [1] 4.5
```

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



Now, the results are stored:

```
model_4_rmse <- min(rmses)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User + Year Effects Model",
                                 RMSE = model_4_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                          RMSE
##   <chr>                                          <dbl>
## 1 Baseline model                                 1.06
## 2 Movie Effect Model                             0.944
## 3 Movie + User Effects Model                     0.865
## 4 Movie + User + Year Effects Model              0.865
## 5 Regularized Movie + User + Year Effects Model 0.865
```

3 Results

Section that presents the modeling results and discusses the model performance.

As confirmed by the final model, there is evidence that movie ratings, users and the year of release help to improve prediction over a a simple recommendation system analysis based solely on the average of all movies.

```
## # A tibble: 5 x 2
```

```
##   method                                     RMSE
##   <chr>                                     <dbl>
## 1 Baseline model                            1.06
## 2 Movie Effect Model                        0.944
## 3 Movie + User Effects Model                0.865
## 4 Movie + User + Year Effects Model         0.865
## 5 Regularized Movie + User + Year Effects Model 0.865
```

method	RMSE
Baseline model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Movie + User + Year Effects Model	0.8650043
Regularized Movie + User + Year Effects Model	0.8647074

Furthermore, the RMSE was improved from an initial 1.22 to 0.8647.

4 Conclusion

Section that gives a brief summary of the report, its limitations and future work.

The model developed in this report was successful in explaining how a rating from a certain movie from a single user could be interpreted.

This rating was decomposed in the following manner:

a general mean rating + a movie effect + a user effect + a year effect + randomness

There is still room for improvement. For example, this model could be further improved by using genres effect, which would be based on a user preference for some genres. Another path could be to use the timestamp to improve the model. This could be explained by the hypothesis that if some users are fans of certain actors or franchises, they would tend to watch and rate the movie as soon as it is released and they would give higher ratings to these movies.