HarvardX: PH125.9x Data Science
MovieLens Rating Prediction Project

Luis Sànchez

June 2020

Contents

1 Overview

The MovieLens project is a requirenment of the class HarvardX's PH125.9x Data Science: Capstone course. MovieLens is run by GroupLens, a research lab at the University of Minnesota. According to GroupLens' website, MovieLens uses "collaborative filtering" technology to make recommendations of movies that a user might enjoy, and also might help avoid the ones that users won't like.

According to wikipedia, a recommender system, or a recommendation system, is a subclass of information filtering system that seeks to predict the rating or preference a user would give to an item.

In our class, we learned that Netflix uses a recommendation system to forecast how many stars (their scoring system) a user will give a specific movie, with one star indicating the lowest score and five stars indicating the highest possible score for a movie. We learned how these recommendations are made, following some of the approaches taken by the winners of the Netflix challenge, a challenge set by Netflix in 2006 focused on the data science community. This challenge involved an improvement of their existing (then) recommendation algorithm by +10%, for the chance of winning a large dollar prize.

Altough the Netflix data is not publicly available, the GroupLens research lab mentioned above generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users.

For this project, we will use a smaller sample of the GroupLens dataset, the MovieLens10M, which contains apprximately 10 million ratings and over 95 thousand tags applied to over 10 thousand movies by apprximately 70 thousand users of the online movie recommender service MovieLens.

According to the documentation in the GroupLens website, users were selected at random for inclusion in their database. All users selected had rated at least 20 movies and no demographic information is included. Each user is represented by an id, and no other personal information is provided.

2 Aim of the project

I will start with a model that assumes the same rating for all movies and all users, with all the differences explained by random variation. This is the naive model.If μ represents the true rating for all movies and users and ϵ represents independent errors sampled from the same distribution centered at zero, then:

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

is the symbolic mathematical represenation of this assumption. In this case, the least squares estimate of μ — the estimate that minimizes the root mean squared error is the average rating of all movies across all users. We can improve this model by adding a term, b_i that represents the average rating for movie i:

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

This model could be further improved by adding b_u, a user-specific effect:

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

Furthermore, this last model could be further improved by adding b_{it}, a time-dependent movie bias:

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

Since there are many terms in the models proposed, any method using linear regression will be slow and prone to crash on an average machine. Therefore, we need to use a machine learning approach that can predict user ratings using the inputs given in the edx dataset to predict ratings in our validation set. The metric we will use is the Root Mean Square Error, or RMSE.

The RMSE metric is one of the most widely used measures in linear regression type of models. It meassures the variability between values predicted by a model and the values observed in reality. Here it will be used to compare forecasting errors of different models for the same dataset. A lower RMSE implies smaller errors on the aggregation of individua predictions, proportional to the size of the squared error. For this reason, RMSE is sensitive to outliers.

The dataset will be split into 2 subsets: a) a training subset, and b) a validation subset and the best model will be the best ranking model that forecasts movie ratings in the validation subset. As mentioned before, we will develop several models and compare them based on their RMSE score in order to determine their respective ranking.

The RMSE function that computes the fit between ratings and their corresponding predictors is the following:

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

3 Dataset

The MovieLens dataset is downloaded from the URLs below:

- MovieLens 10M dataset
- MovieLens 10M dataset - zip file

```
# 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")

# 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")
 movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[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)
```

In order to train and test the MovieLens dataset, again, we will split it into 2 subsets, a training subset to train the algorithms, and a validation subset to test our predicted ratings.

The table below indicates that there are no missing values, which simplfies things a little, since we do not have to make assumptions for missing data.

```
##      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
##
##
##
```

The total of unique movies and users in the edx subset is about 70 thousand users and about 10 thousand different movies:

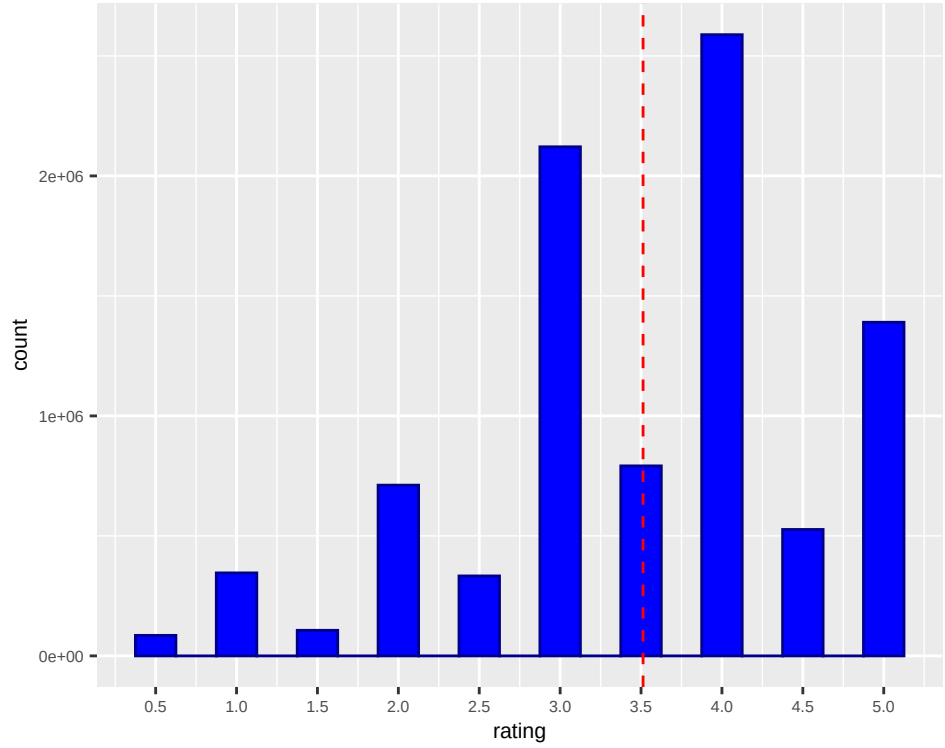
```
##   n_users n_movies
## 1   69878    10677
```

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

4 Data Analysis

As we can see, userId and movieId are numbers that represent unique users and movies. Timestamp is the number of seconds since January 1, 1970 in Unix time. Both title and genres are of type character, however, these features will not be tested in this analysis.

The rating variable has the following characteristics:

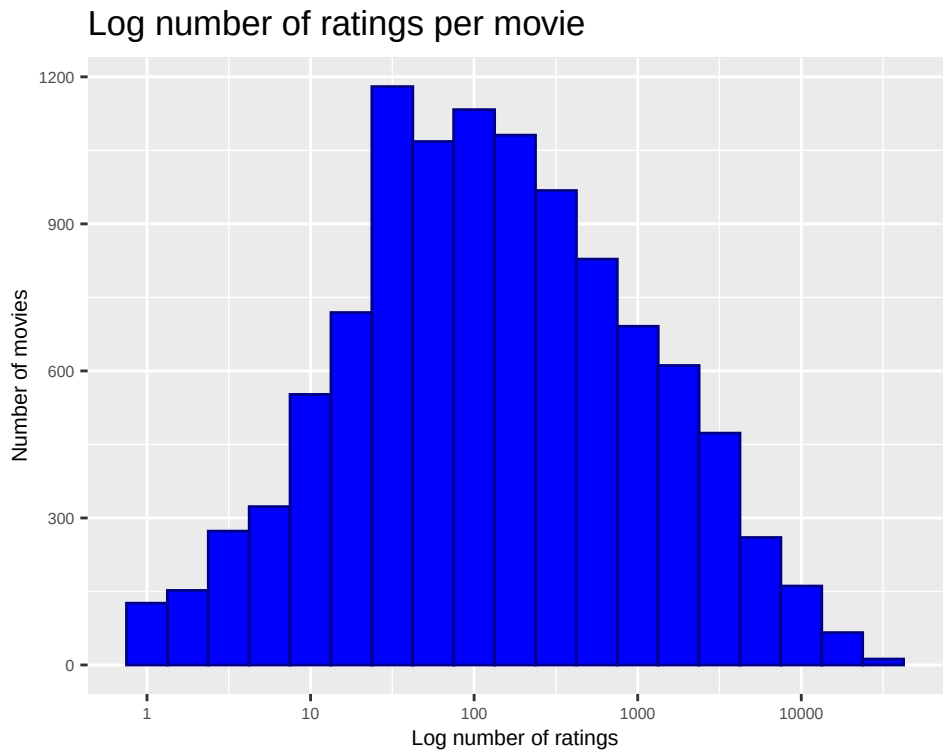


The red dashed line represents the average rating μ across all users and movies. As we can see, the ratings range from 0.5 stars to 5 stars.

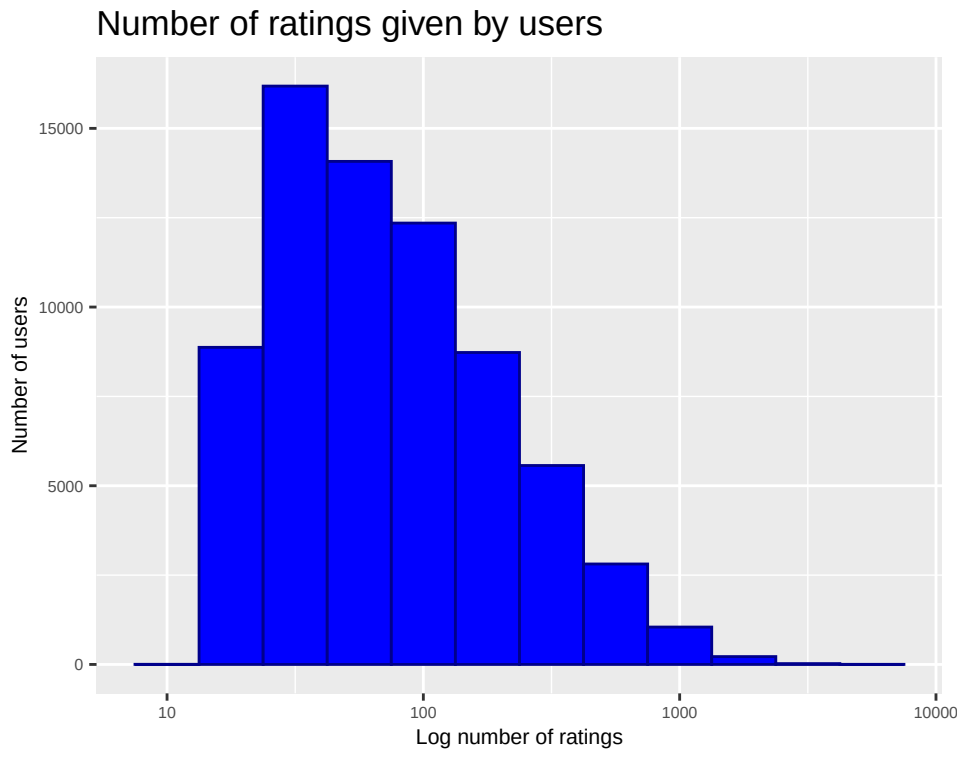
Users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below, with 4 being the median rating, followed by 3 and 5. 0.5 is the least common rating.

We can see that some movies are rated more than others, while some have very few ratings and sometimes only one rating. This asymetric distribution of ratings is important for our model as inbalances in the representation of scores for movies might distort results. For example, in our db, 125 movies have been rated only once.

For this reason, regularization will be applied to the models in this project. Regularization is a technique used to reduce the error by fitting a function on a given training set to avoid overfitting.

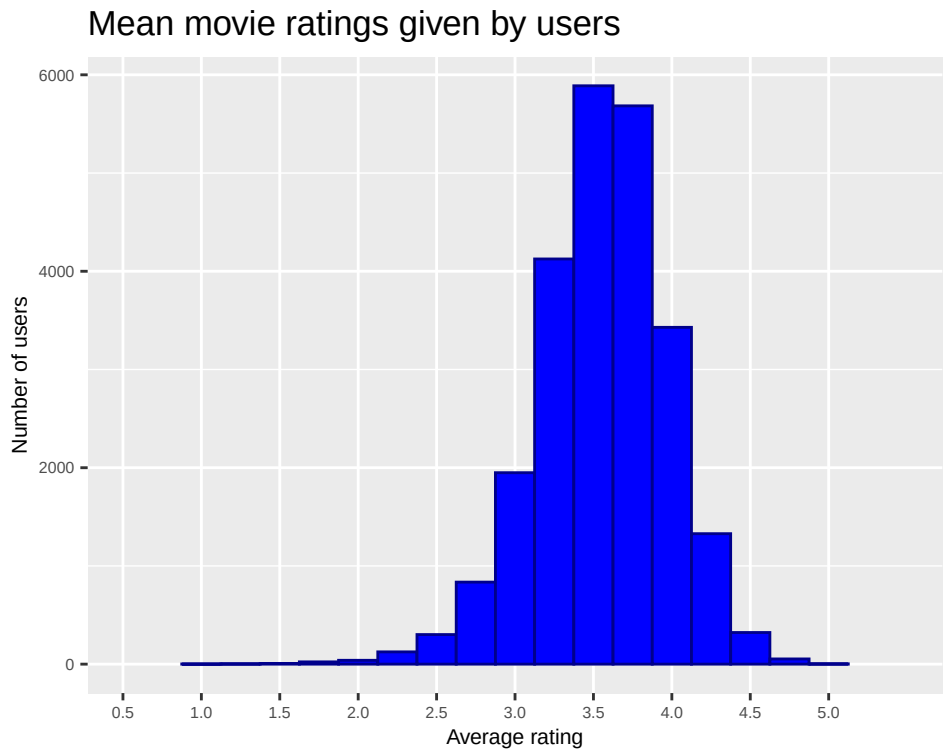


We can observe that the majority of users have rated between 30 and 100 movies. therefore, a user penalty term needs to be part of the modeling approach.



Furthermore, some users tend to give much lower ratings and some users tend to give much higher ratings to particular combination, compared to the average. Below we can see the distribution of ratings for users that

have rated at least 100 movies.



5 Modelling Approach

Below is the loss-function that computes the RMSE, defined as follows:

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

N is the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is our measure of model accuracy, and as mentioned before, the R function to compute the RMSE between ratings and our corresponding predictions is:

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

5.1 I. Average movie rating model

The first model, the naive or baseline model, predicts the same rating for all movies. For this, we simply compute the dataset's mean rating, which was already shown graphically in the "Data Analysis" section. With this model, we give the same rating to all movies regardless of users, time, etc, with any residual (the difference between the real rating and the predicted rating) assigned to a random variation:

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

We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings:

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

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

If we predict all unknown ratings with μ or mu, we obtain the first naive RMSE:

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

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

Below is the results table of the naive model:

```
rmse_results <- data_frame(method = "Average movie rating model", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.061202

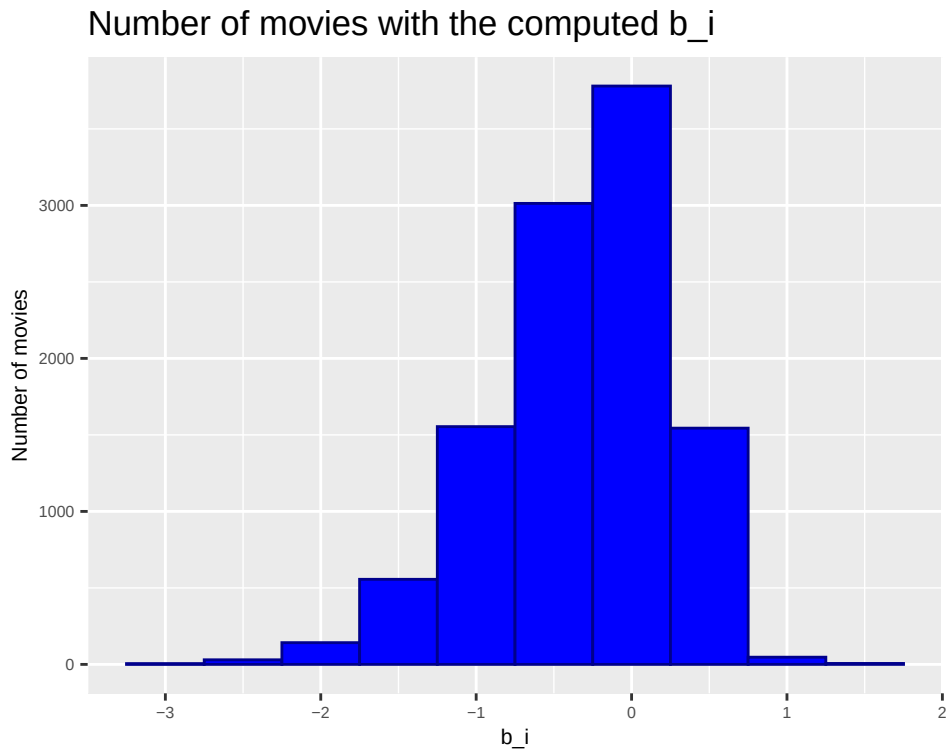
This give us our baseline RMSE to compare with other modelling approaches.

5.2 II. Movie effect model

To improve the naive model we will exploit the fact that higher ratings are generally assigned to popular movies among users, and the opposite is true. We calculate the deviation of each movie mean rating from the total mean of all movies μ. The resulting variable is called "b" (as in bias) for each movie "i" b_i. This represents the average ranking for movie i:

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

The histogram implies left skweness.



This is called the penalty term movie effect. Our prediction improve once we predict using this model.

```
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

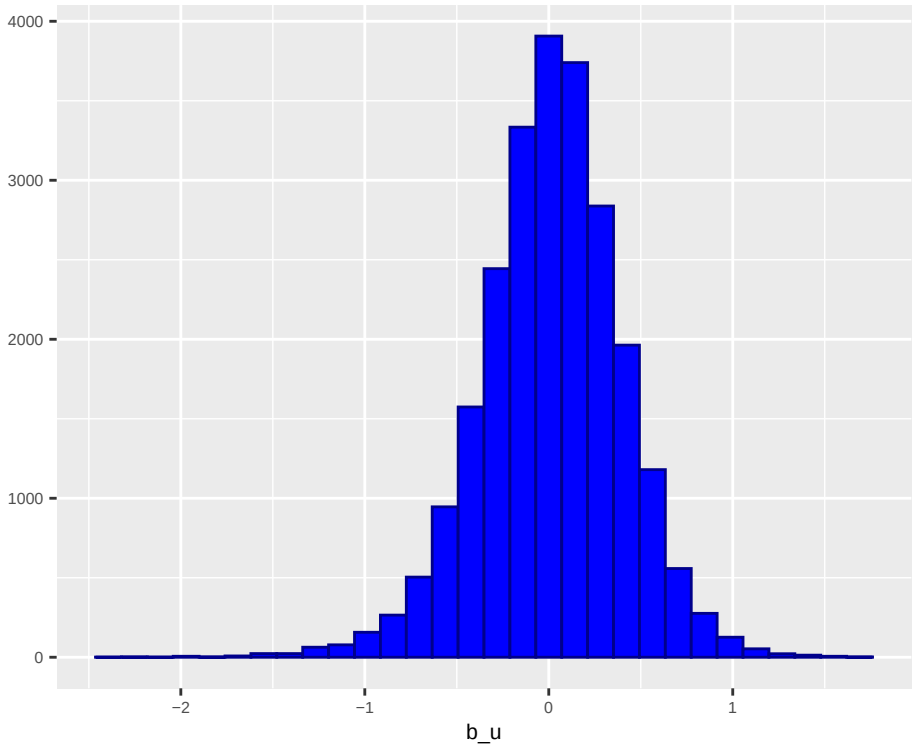
method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087

If a movie is on average rated worse that the average rating of all movies μ, we predict that it will rated lower that μ by b_i, the difference of the individual movie averages from the total average. There is an improvement, however, we are not yet considering any user rating effect.

5.3 III. Movie and user effect model

First, we compute the average rating for users that have rated over 100 movies. We can see that this term can also affect the ratings positively or negatively.

```
user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I('darkblue'), fill = I('blue'))
```



There is definitely variability across users. Some users are very negative about their ratings, while some others are very positive about their ratings. We can add this term to our model.

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

where b_u is a user-specific effect. If a user with a negative bias in b_u rates a great movie, the effects counter each other and we might be able to correctly predict that this user gave this great movie a 1 or 2 instead of a 4 or 5.

We approximate this by computing μ and b_i, and estimating b_u, as the average of:

$$Y_{u,i} - \mu - b_i$$

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

We can now see if our RMSE has improved:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

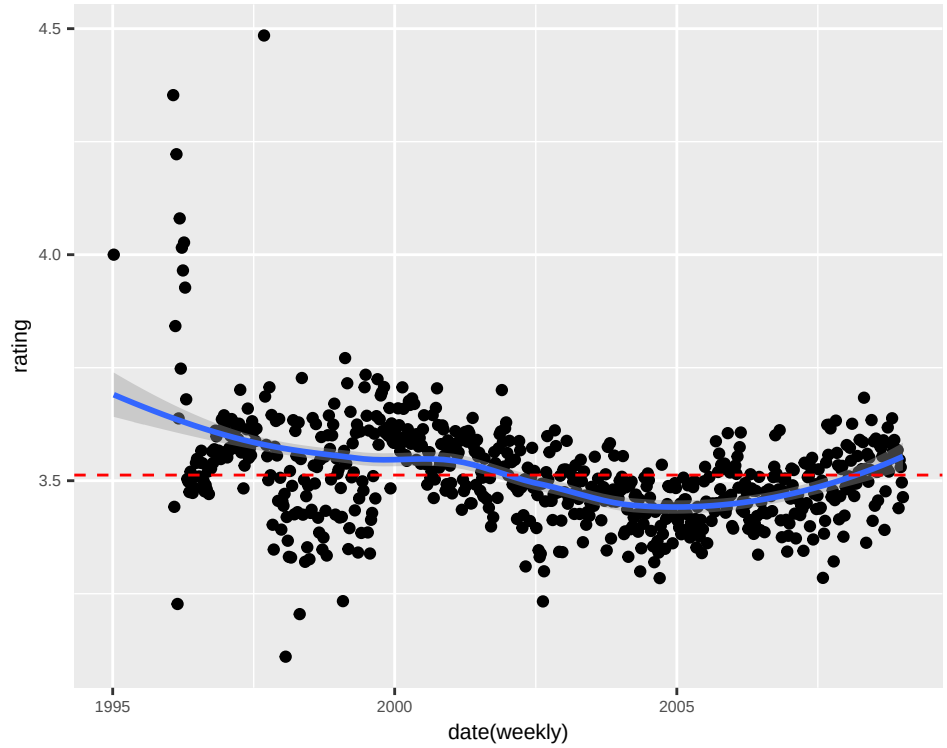
method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

This approach reduced the RMSE, but there is a built-in bias in using it: bottom popularity movies were rated by few users, in many cases just one user. For this reason, large negative or positive estimates of b_i are possible, which can distort our RMSE.

To solve this, we need to use regularization, which penalizes large estimates coming from small sample sizes, i.e. movies rated by just one user. We will add a penalty value for large values of b_i to the sum of squares equation, thus minimizing the chances of overfitting.

5.4 IV. Temporal movie effect

Let's visualize the effect of time on ratings:



The dotted red line is a time series of μ, rounded into intervals of weeks. Since this temporal movie effect shows very low volatility (with the exception of the period of approximately 1995 to 1998), we will not add

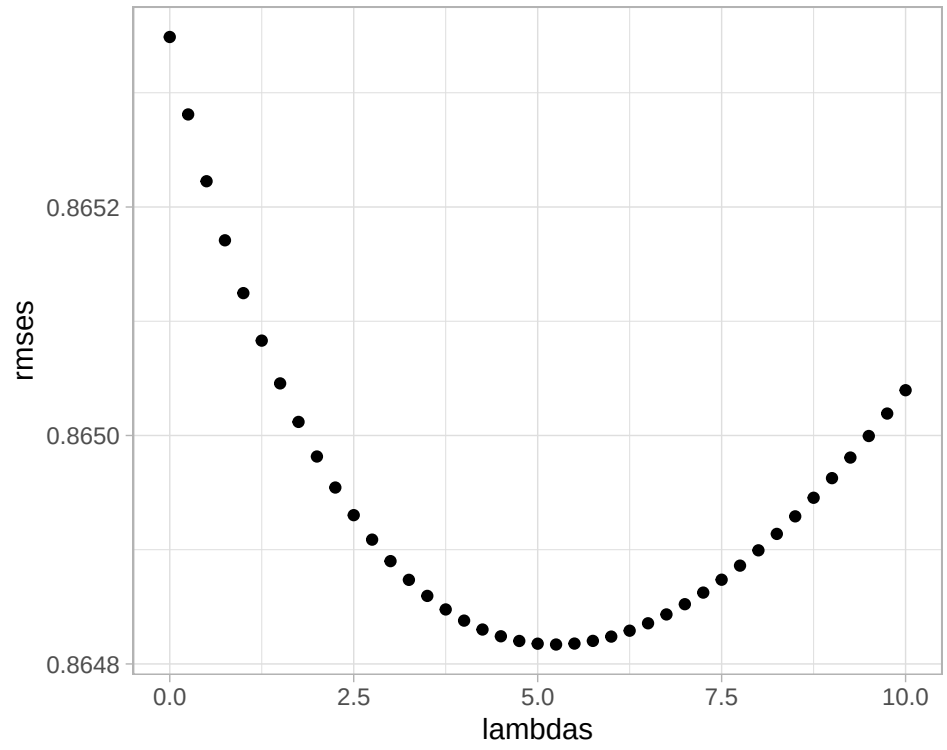
a temporal movie effect to this report since it might add additional levels of complexity to properly take temporal outliers into consideration.

5.5 V. Regularized movie and user effect model

Large individual estimates of b_i and b_u (i.e users that only rated a very small number of movies), can affect overall predictions. Using regularization via lambda, we can penalize these aspects by finding the lambda that minimizes RMSE.

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

  mu <- mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
```

Plotting RMSE vs lambdas to select the lambda that minimizes RMSE, we see:



The optimal lambda is:

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

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

For the full model, the optimal lambda is: 5.5

6 Results & conclusions

The results are:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8648170

The lowest value of RMSE found was 0.8648, which correspons to the "Regularized movie and user effect model". This is also lower than the threshold of 0.8649 for the highest score in the RMSE scoring section of the assignment (25 points for RMSE < 0.86490). The final model for our project is the following:

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

With these, we sucessfully built a machine learning algorithm to predict movie ratings with the MovieLens dataset. The regularized model including the effect of users shows the lowest RMSE value and it is the final one submited for the current assigment.

Improvements in this RMSE score could be achieved by adding other effect (genre, time since release, etc.), which were not explored in this assigment. Other different machine learning models (i.e. ensembles) could also improve the results, however, this option was not explored.

7 Appendix - Enviroment

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```